Mining Partially Periodic Event Patterns With Unknown Periods*

Sheng Ma and Joseph L. Hellerstein
IBM T.J. Watson Research Center
Hawthorne, NY 10532

Abstract

Periodic behavior is common in real-world applications. However, in many cases, periodicities are partial in that they are present only intermittently. Herein, we study such intermittent patterns, which we refer to as **p-patterns**. Our formulation of p-patterns takes into account imprecise time information (e.g., due to unsynchronized clocks in distributed environments), noisy data (e.g., due to extraneous events), and shifts in phase and/or periods. We structure mining for p-patterns as two sub-tasks: (1) finding the periods of p-patterns and (2) mining temporal associations. For (2), a level-wise algorithm is used. For (1), we develop a novel approach based on a chi-squared test, and study its performance in the presence of noise. Further, we develop two algorithms for mining p-patterns based on the order in which the aforementioned sub-tasks are performed: the period-first algorithm and the association-first algorithm. Our results show that the association-first algorithm has a higher tolerance to noise; the period-first algorithm is more computationally efficient and provides flexibility as to the specification of support levels. In addition, we apply the period-first algorithm to mining data collected from two production computer networks, a process that led to several actionable insights.

1 Introduction

Periodic behavior is common in real-world applications [7][8][13] as evidenced by web logs, stock data, alarms of telecommunications, and event logs of computer networks. Indeed, in our studies of computer networks, over half of the events are in periodic patterns. Periodic patterns arise from many causes: daily behavior (e.g. arrival times at work), seasonal sales (e.g. increase sale of notebooks before a semester), and installation policies (e.g. rebooting print servers every morning). Some periodic patterns are complex. For example, in one network we discovered a pattern consisting of five repetitions every 10 minutes of a port-down event followed by a port-up event, which in turn is followed by a random gap until the next repetitions of these events.

Our experience in analyzing events of computer networks is that periodic patterns often lead to actionable insights. There are two reasons for this. First, a periodic pattern indicates something persistent and predictable. Thus, there is value in identifying and characterizing the periodicity. Second, the period itself often provides a signature of the underlying phenomena, thereby facilitating diagnosis. In either case, patterns with a very low support are often of considerable interest. For example, we found a one-day periodic pattern due to a periodic port-scan. Although this pattern only happens three times in a three-day log, it provides a strong indication of a security intrusion.

While periodic patterns have been studied, there are several characteristics of the patterns of interest to us that are not addressed in existing literature:

1. Periodic behavior is not necessarily persistent. For example, in complex networks, periodic monitoring is initiated when an exception occurs (e.g., CPU utilization exceeds a threshold) and stops once the exceptional situation is no longer present.
2. Time information may be imprecise due to lack of clock synchronization, rounding, and network delays.
3. Period lengths are not known in advance. Further, periods may span a wide range, from seconds to days. Thus, it is computationally infeasible to exhaustive consider all possible periods.
4. The number of occurrences of a periodic pattern typically depends on the period, and can vary drastically. For example, a pattern with a period of one day period has, at most, seven occurrences in a week, while one minute period may have as many as 1440 occurrences. Thus, support levels need to be adjusted by the period lengths.
5. Noise may disrupt periodicities. By noise, we mean that events may be missing from a periodic pattern and/or random events may be inserted.

This work develops algorithms for mining partially periodic patterns, or **p-patterns**, that consider the above five issues. We begin with establishing a looser definition of periodicity to deal with items (1) and (2). Specifically, we introduce an on-off model consisting of an "on" segment followed by an "off" segment. Periodic behavior is present only during the on segment. Our definition of p-patterns generalizes the partial periodicity defined in [7] by combining it with temporal associations (akin to episodes in [11]) and including a time tolerance to account for imperfections in the periodicities.

Finding p-patterns requires taking into account items (3)-(5) above. We structure this as two subtasks: (a) finding period lengths and (b) finding temporal associations. A level-wise algorithm can be employed for (b). For (a), we develop an algorithm based on a chi-squared test. Further, we develop two algorithms for discovering p-patterns based on whether periods or associations are discovered first. The performance trade-offs between these two approaches are studied. We conclude that the association-first algorithm has higher tolerance to noise while the period-first algorithm is more computationally efficient. We study effectiveness and efficiency of the algorithms using synthetic data. In addition, we apply the period-first algorithm to mining data collected from two production computer networks, a process that led to several actionable insights.

Sequential mining[11][5][18][15][12] has been studied extensively, especially in domains such as event correlation in telecommunication networks [11], web log analysis[6][17], and transactions processing[7][15]. All these work focus on discovering frequent temporal associations[11][15][18]; that is, finding a set of events that co-occur within a predefined time window. There has also been recent work in identifying periodic behaviors[7][8][13][16] closely related to our work. Ozden et. al.[13] study mining of cyclic association rules for full periodicities–patterns that are present at each cycle. As noted earlier, this is quite restrictive since periodicities may occur only intermittently. Han et. al.[7][8] study partially periodic patterns with the assumption that period lengths are known in advance or that it is reasonable to employ an exhaustive search to find the periods. Yang et. al.[16] extend Han's work by introducing concepts from information theory to address noisy symbols. We note that these latter efforts focus on symbol sequences, not time-based sequences. In addition, none of these studies study the effect of noise in the form of random occurrences of events of the same type as those in the periodic pattern. Nor do these studies address the problem of phase shifts or imperfections in the periodicity.

This sequel is organized as follows. Section 2 formulates the problem addressed. Section 3 presents our algorithm for finding possible periods. Section 4 describes our algorithms for finding p-patterns with unknown periods. Section 5 discusses empirical assessments of our algorithms. Our conclusions are contained in Section 6.

2 Definitions and Problem Statement

Fundamental to our work is the concept of an event. An event has two attributes: a type and an occurrence time.

Definition 1 *Let A be a set of event types. An event is a pair (a, t), where $a \in A$ is an event type and $t \in R$ is the occurrence time of the event. An* ***event sequence*** *S is an ordered collection of events, i.e. $\{(a_1, t_1), (a_2, t_2), \ldots, (a_N, t_N)\}$, where $a_i \in A$ is the event type of the i-th event. t_i represents the occurrence time of the event, and $t_j \leq t_i$ for $1 \leq j \leq i \leq N$. Throughout, we use D to denote the set of events being mined.*

An event type may have a complex structure. For example, we can encode host name and alarm type into event type and so a "port down" alarm sent from host X has a different event type than a "port down" alarm sent from host Y.

Often, it is desirable to consider just the times associated with a sequence of events.

Definition 2 *A* ***point sequence*** *is an ordered collection of occurrence times.*

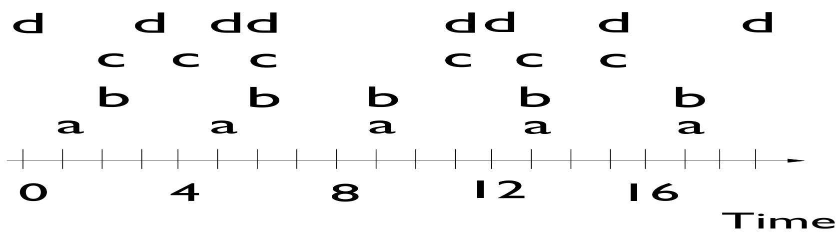

Figure 1: An illustrative event sequence. The event type of an event is labeled above its occurrence time.

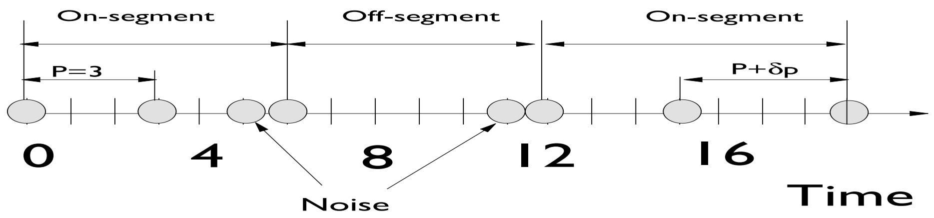

Figure 2: The point sequence for event type "d" in Figure 1. "o" represents a "d" event.

An event sequence can be viewed as a mixture of multiple point sequences of each event type. Figure 1 illustrates an event sequence, where the set of event types is $A = \{a, b, c, d\}$. The type of an event is labeled above its occurrence time. Figure 2 illustrates the point sequence of event type "d" $S_d^{'} = (0, 3, 5, 6, 11, 12, 15, 19)$. A point is plotted as "o" at its occurrence time on the time axis.

Now, we define a periodic point sequence. We include the concept of tolerance to account for factors such as phase shifts and lack of clock synchronization.

Definition 3 *$S' = (t_1, \cdots, t_N)$ is a* ***periodic point sequence*** *with period p and time tolerance δ if a point occurs repeatedly every $p \pm \delta$ time units.*

For example in Figure 1, the point sequence of event type "a" is $S_a' = (1, 5, 9, 13, 17)$. Thus, event type "a" is periodic with period 4. "b" is also periodic, if $\delta \geq 1$.

The above definition of periodicity requires periodic behavior throughout the entire sequence, which does not capture partial periodicities. To accomplish this, we consider a situation in which periodicities occur during on-segments and no periodicities are present during off-segments. This is illustrated in Figure 2, where periodicities have a duration of p.

δ	time tolerance of period length	Predefined
w	a time window defining temporal association	Predefined
A_1	a subset of event types A	To be found by algorithms
p	period length	To be found by algorithms
$minsup$	the minimum support for a p-pattern	Predefine; or computed based on p

Table 1: Summary of parameters of p-patterns

Definition 4 *A point sequence is* ***partially periodic*** *with period p and time tolerance δ, if points are periodic with p and δ during on-segments.*

Referring to the above example, "d" and "c" are partially periodic with periods 3 and 2, respectively, but not periodic. Figure 2 illustrates the point sequence for "d", and its "on" and "off" segments.

Thus far, we have described periodicities of point sequences. This can be generalized to a set of events that occur close in time.

Definition 5 *A set of event types $A_1 \subset A$ is a* ***partially periodic temporal association (p-pattern)*** *with parameters p, δ, w, and minsup, if the number of qualified instances of A_1 in D exceeds a support threshold minsup. A qualified instance $S_1 \subset D$ satisfies the following two conditions:*
(C1) The set of the event types of events in $S_1 = A_1$, and where there is a t such that for all $e_i \in S_1$, $t \leq t_i \leq t + w$.
(C2) The point sequences for each event type in S_1 occur partially periodically with the parameters p and δ.

To illustrate this definition, we refer again to Figure 1. Let $w = 1$, $\delta = 1$, and $minsup = 2$. Then, $\{a, b\}$ is a p-pattern with length 2 and period 4. Moreover, all non-null subsets of this pattern– $\{a\}, \{b\}$–are also p-patterns with length 2.

It can be easily verified that p-patterns are downward closed. That is, if A_1 is a p-pattern, then all non-null subsets of A_1 are also p-patterns. This means that a level-wise search can be used, thereby providing computational efficiencies.

2.1 Overview of Approach

Table 1 summarizes the parameters of a p-pattern. w specifies the time window for temporal associations. p and δ characterize a period length and its time tolerance.

Herein, we assume that δ, w, and $minsup$ are given. Thus, finding p-patterns includes two sub-tasks: (1) finding (all) possible periods $\{p\}$, and (2) discovering temporal patterns A_1 with w.

We proceed as follows. Section 3 develops algorithms for finding possible periods. Section 4 develops

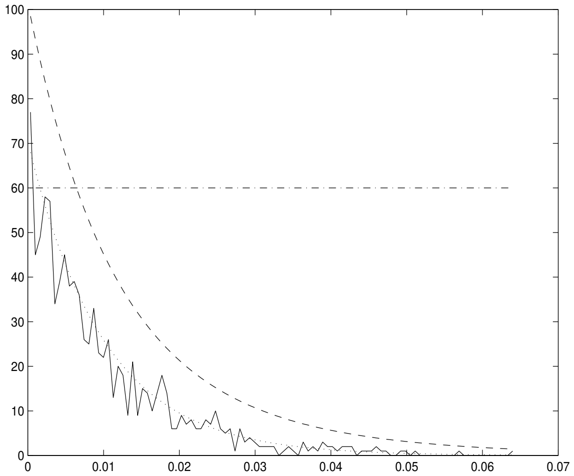

Figure 3: Distribution of inter-arrivals for a random point sequence. "-": empirical result. "..": theoretical distribution. "–": a threshold obtained by a chi-squared test. "-."': a constant threshold.

algorithms to discover p-pattern with unknown periods by combining the two sub-tasks.

3 Finding Unknown Periods

This section develops an effective and efficient approach to finding periods in point sequences. Effectiveness requires robustness to missing events and to random occurrence of additional events as well as dealing with partial periodicities. Efficiency demands that computations scale well as the number of events grows and the range of time-scales increases. Our approach to finding periods applies both to individual events and to sets of events.

One approach to finding unknown periods is to use the fast Fourier transform (FFT), a well developed technique for identifying periodicities. There are two problems with doing so. First, the FFT does not cope well with random off-segments in p-patterns. Further, the computational efficiency of FFT is $O(T \log T)$, where T is the number of time units. In our applications, T is large even though events are sparse. For example, although there may be several hundred thousands of events in a month at a medium-sized installation, there are over one billion milliseconds.

3.1 Chi-squared test for Finding Periods

We begin with some notation. Let $(t_1, \cdots, t_N)$ be a point sequence in a time window $[0, T]$, where N is the total number of points. We denote the i-th inter-arrival time by $\tau_i = t_{i+1} - t_i$, where $1 \leq i \leq N-1$. We define two extreme point sequences: an ideal partially periodic sequence and a random sequence. An ideal partially periodic sequence is modeled by a sequence of on-off segments, in which a point re-occurs periodically during an "on" segment, and no point occurs during an "off" segment (e.g., event "a" in Figure 1). A random point sequence is an ordered collection of random points generated randomly and

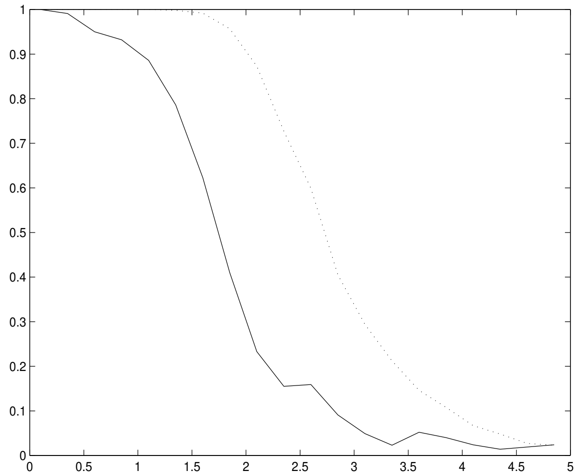

Figure 4: The effect of noise. y-axis: success percentage. x-axis: NSR ratio. "-": first-order inter-arrival. "..": second-order inter-arrival

uniformly in $[0, T]$.

Now, we characterize inter-arrivals for an ideal partial periodic point sequence with parameters p and δ. Let $\iota_i = 1$ if both t_i and t_{i+1} are in the same on-segments; otherwise, $\iota_i = 0$. When $\iota_i = 1$, i.e. the i-th and $(i+1)$-th arrivals are in the same on-segment, $\tau_i = p + \mathbf{n}_i$, where $-\delta \leq \mathbf{n}_i \leq \delta$, characterizes a random phase shift. When $\iota_i = 0$, i.e. successive arrivals are not in the same on-segment, $\tau_i = \mathbf{r}_i$, a random variable characterizing a off-segment. Put these two cases together, we obtain

$$\tau_i = (\iota_i)(p + \mathbf{n}_i) + (1 - \iota_i)\mathbf{r_i}. \tag{1}$$

Now consider an arbitrary inter-arrival time τ and a fixed δ. Let C_τ be the total number of arrivals with values in $[\tau - \delta, \tau + \delta]$. Intuitively, if τ is not equal to p, C_τ should be small; otherwise C_p should be large. Based on this observation, an naive algorithm for finding periods is to look for large values of C_τ. That is, an inter-arrival τ is declared to be a period, if

$$C_\tau > thres. \tag{2}$$

However, there is a problem. Intuitively, the number of partially periodic points with a large p is much smaller than that with a small p given the same on-segment length. Therefore, the naive algorithm favors small periods. To illustrate this, we randomly and uniformly generate 1000 points in the interval $[0, 100]$. The solid line of Figure 3 plots the distribution of inter-arrivals. A threshold used by the naive algorithm is illustrated by the dash-dotted line. The figure shows that the distribution of inter-arrivals decays as inter-arrivals increase. As the naive algorithm employs a fixed threshold regardless of period length, it tends to produce false positives for small periods and false negatives for large periods.

Clearly, we need a threshold that adjusts with the period under consideration. One approach is to use a chi-squared test. The strategy here is to compare C_τ with the number of inter-arrivals in $[\tau - \delta, \tau + \delta]$

that would be expected from a random sequence of inter-arrivals. The chi-squared statistic [9] is defined as

$$\chi_\tau^2 = \frac{(C_\tau - NP_\tau)^2}{NP_\tau(1 - P_\tau)}, \tag{3}$$

where N is the total number of observations and P_τ is the probability of an inter-arrival falling in $[\tau - \delta, \tau + \delta]$ for a random sequence. Thus, NP_τ and $NP_\tau(1 - P_\tau)$ are the expected number of occurrences and its standard deviation, respectively. χ_τ^2 is the normalized deviation from expectation. Intuitively, the chi-squared statistic measures the degree of independence by comparing the observed occurrence with the expected occurrence under the independence assumption. A high χ_τ^2 value indicates that the number of inter-arrivals close to τ cannot be explained by randomness, which is a necessary condition for periodicity.

χ^2 is usually defined in terms of a confidence level. For example, 95% confidence level leads to $\chi^2 = 3.84$. Then, the above equation can be changed to

$$C'_\tau = \sqrt{3.84NP_\tau(1 - P_\tau)} + NP_\tau. \tag{4}$$

C'_τ can be used as a threshold to find possible periods. That is, we say that τ is a possible period, if

$$C_\tau > C'_\tau. \tag{5}$$

To compute P_τ , we note that a random event sequence approaches a Poisson arrival sequence[14] as the number of points increases. Further, it is well known that the inter-arrival times of a Poisson process are exponentially distributed. Using this, we obtain

$$\begin{aligned} P_\tau &= \int_{p-\delta}^{p+\delta} \lambda \exp(-\lambda t) dt, & (6) \\ &\approx 2\delta\lambda exp(-\lambda p), & (7) \end{aligned}$$

where $\lambda = N/T$ is the mean arrival. The approximation is obtained based on the assumption that δ/p is small.

We conducted simulation experiments to assess the accuracy of this result. Figure 3 plots the results. The x-axis is the period (expected inter-arrival times), and the y-axis is the count of inter-arrival times for a period. These curves depict: (i) the empirical density of C_τ for a simulated exponential inter-arrival distribution (solid line), (ii) the expected value of C_τ in the simulated data (dotted line), and (iii) C'_p for 95% confidence level (dashed line). Observe that the χ^2 test results in a threshold that varies with period. Further observe that $C_\tau < C'_\tau$ in all cases, and hence there is no false positive.

3.2 Finding Periods in Noisy Data

Here we consider the ability of our test to detect partial periodicities in the presence of noise. By noise, we mean the addition of randomly occurring events in either an on-segment or an off-segment of the p-pattern.

This situation is best described with an example. Consider event "d" in Figure 2. This event has a period of 3 starting at time 0 and extending through time 18. However, there are also two noisy "d" events at times 5 and 11. Thus, the set of inter-arrival times are $\{3, 2, 1, 5, 1, 3, 3\}$. Clearly, noise events make it more difficult to detect the underlying period.

To gain more insight, we conducted a series of experiments. In each, there was a periodic point sequence with $p = 2$ in $[0, 200]$. We then gradually increase the number of noise points, which are generated uniformly and randomly. We quantify the noise by computing the ratio of the number of noise events to the number events in the partial periodicity. We refer to this as the **noise-to-signal ratio (NSR)**. For each NSR, 1000 runs were conducted. A run is successful if the period is correctly identified (i.e. above 95% confidence level), and no false period is found. Figure 4 plots the results. The x-axis is NSR, and the y-axis is the fraction of successful runs. The solid line displays the results for this experiment. Note that the success rate is above 90% if NSR does not exceed 1. However, as NSR increases above 1, the success rate degrades rapidly.

There is an intuitive explanation for the rapid drop-off in success rate as NSR increases beyond 1. As suggested by the example above, the performance of our test degrades rapidly when noise points lie between periodic events. With more noise events, this becomes increasingly likely. We note, however, that a noise point in an off-segment does less damage than one in an on-segment since the latter creates two inter-arrivals that do not have the period and deletes one that does. Thus, these experiments are in some sense a worst case since there is no off-segment.

Can we improve the performance of the above algorithm? Intuitively, we may increase the chance to detect the period if we take into consideration inter-arrivals between events separated by n other events. We refer to this as n-order inter-arrivals. Of course, this comes at a price—increased computational complexity. The dotted line in Figure 4 shows the performance of our test with second-order inter-arrivals. Note that the NSR value at which the success probability declines rapidly is close to 2, which is almost double the NSR value at which success probability declines rapidly when using first-order inter-arrivals.

3.3 Implementation Details

We now present an algorithm for finding periods based on the chi-squared test introduced earlier. Our discussion focuses on first-order inter-arrivals. The algorithm for higher-order inter-arrivals is essentially the same.

Let S be a point sequence. We assume that memory is larger than $|S|$ as we need the maximum of $|S|$ memory to hold counts in the worst case. This assumption is reasonable for a sequence of events of the same type since: (a) we only need to keep track of arrival instants and (b) this is a shorter sequence than raw data that contains a mixture of event types. When memory is problematic, indexing mechanism such as CF-tree[19] can be used to control the total number of buckets.

The algorithm takes as input the points s_i in S, the tolerance δ, and confidence-level (e.g. 95%).

1. For $i = 2$ to $|S|$
   (a) $\tau = s_i.time - s_{i-1}.time$
   (b) If C_τ does not exist, then $C_\tau = 1$
   (c) Else, $C_\tau = C_\tau + 1$
2. AdjustCounts($\{C_\tau\}, \delta$) /*Adjust counts to deal with time tolerance*/
3. For each τ for which a C_τ exists
   (a) Compute threshold C'_τ based on Equation 5
   (b) If $C_\tau > C'_\tau$, then insert τ into the set of periods being output

Step (1) counts the occurrence of each inter-arrival time. Step (2) groups inter-arrival times to account for time tolerance δ by merging into a single group counts whose τ values are within within δ of one another. Step (3) computes the test threshold. If the threshold is exceeded by the test statistic, a possible period is found, and inserted into results. This procedure is repeated until all event types are processed.

It is easy to see that the above algorithm scales linearly with respect to the number of events. Further, the algorithm does not depend on the range of time scales of the periods.

4 Mining P-patterns With Unknown Periods

As we discussed, mining p-patterns with unknown periods is combination of two tasks: (a) finding periods, and (b) finding temporal association. In this section, we first overview algorithms for (b), i.e. mining associations. We then develop algorithms that combine the two tasks in different ways: the period-first algorithm and the association-first algorithm.

4.1 Level-wise Algorithm, and Its Variation for Mining P-patterns with Known Period

Many algorithms have been developed to efficiently mine associations[3][1][4], most of which are variations of the Apriori algorithm[3][1]. Here, we summarize this algorithm and its variation for mining temporal associations ([11]). Then, we show how such an algorithm can be applied to mining p-patterns with known periods. This is done in a way that extends the algorithms in [8][7] so as to consider time tolerance and temporal data (rather than just sequence data). In next section, we show how to mine p-patterns with unknown periods.

[1] All algorithms that can find associations can be used for (b) in theory. Here, we focus on the level-wise algorithm for demonstration purpose.

We begin with level-wise search. Let L_k and C_k be, respectively, the set of qualified patterns and candidate patterns at level k. Given events D and $C_1 = \{c \in 2^A || c| = 1\}$, where A is the set of event types and C_1 is a set of patterns with length 1, the level-wise algorithm proceeds as follows.

(1) k =1;

(2) Count the occurrences in D for each pattern $v \in C_k$

(3) Compute the qualified candidate set: $L_k = \{v \in C_k | v.count > minsup\}$

(4) Compute the new candidate set C_{k+1} based on L_k

(5) if C_{k+1} is not empty, $k = k+1$ and goto (2)

Step (3) computes the qualified candidate set L_k from C_k based on the supports. Step (4) computes a new candidate set C_{k+1} based on the previous qualified patterns in L_k. This is typically implemented by a join operation followed by pruning[2]. Step (2) counts occurrences of patterns in D. Step (3) and (4) are similar for finding different patterns, while Step (2) varies based on the patterns to be found.

In mining associations, Step (2) augments the count of a pattern (called an item set) $v \in C_k$ by 1, if v is a subset of a transaction (see [3] for details). This can be generalized to an event sequence by introducing a time window to segment the temporal sequence into "transactions". In particular, Step (2) in mining temporal associations augments the count of a temporal association $v \in C_k$ (also called an Episode by [11]), if v is a subset of event types in a time window $[t-w, t]$[2]. Similarly, Step (2) in mining p-patterns with a known period augments the count of a p-pattern $v \in C_k$, if v satisfies two conditions: (a) v is a subset of event types in a time window $[t-w, t]$; and (b) v is a subset of those in the pervious window $[t-p-\delta'-w, t-p-\delta']$, where $0 \le \delta' \le \delta$ is corresponding to a possible time shift. Note that (a) corresponds to Condition 1 of Definition 5 (qualifying an instance of a temporal association), and (b) addresses Condition 2 (qualifying an instance with partial periodicity).

4.2 Discovering P-patterns with Unknown Periods

Now, we describe algorithms for mining p-patterns with unknown periods. Two steps are needed in achieving this goal: (1) finding the possible periodicities and (2) determining the temporal associations. These two steps can be combined in different orders resulting in two different algorithms: the period-first algorithm, and the association-first algorithm. The former does step (1) and then step (2), while the latter does step (2) and then step (1). A hybrid algorithm is also possible.

4.3 Period-first algorithm

This algorithm operates by first finding partial periodicities for each event type, and then constructing temporal associations for each period.

[2] Please refer [11] for subtle details.

Let D be the set of events (sorted by time). The period-first algorithm is described as follows:

Period-first algorithm

Step 1: Find all possible periods

1 Group D by event type

2 Find possible periods for each event type

Step 2: Find patterns

3 For each period p

1. Find related event types $A_p = \{a \in A | a \text{ has a period } p\}$
2. $C_1 = \{v \in 2^{A_p} || v| = 1\}$
3. Set support level $sup(p)$, and window size $w(p)$
4. Find p-patterns with p

Step 1 finds the possible periods for individual events using the algorithm described in the proceeding section. In particular, Line 1 groups events by type so that events in each group can be read out sequentially to find possible periods using algorithm discussed in Section 3.3. Alternatively, this step can be implemented in parallel to find periods for all event types simultaneously. However, tree indexing schemes, as those used in [19], are needed to handle a large number of events.

Step 2 finds all patterns for each period. It first finds event types $A_p = \{a \in A | a \text{ has a period } p\}$. It then seeds initial candidate set C_1 and specifies the minimum support and window size. Last, the level-wise mining (Section 3.3) can be performed to find p-patterns with known p. Further, we note that Step 2 can be done in parallel for each period to gain further reductions in data scans.

The period-first algorithm uses the periods discovered to reduce the set of event types considered for temporal mining and hence significantly reduce the computational complexity of Step 2. Our experience has been that $|A_p|$ is much smaller than $|A|$.

Another advantage of the period-first algorithm is that the minimum support and the window size can be assigned separately for each period based on the period length and its support. This makes it possible to find a pattern with a very low support because of a large p[3], while avoiding drastic increase of the search space.

The computational complexity of Step 1 is determined by the group operation[4]. For Step 2, complexity is the sum of the complexity of finding temporal associations for each A_p.

There is, however, a disadvantage to this algorithm. If noise events are present, then our ability to

[3] In our experience, we let minimum support be $minsup(p) = T/p\alpha$, and $w(p) = \beta p$, where $0 \leq \alpha, \beta << \leq 1$ are two user-controlled parameters.

[4] It is reasonable to assume that event logs are sorted by time before mining.

identify the correct periods is reduced. Without the correct periods, Step 2 cannot output the correct p-patterns. In particular, as NSR increases, we expect the effectiveness of this algorithm to decrease.

4.4 Association-first Algorithm

The association-first starts by mining for temporal associations. The results of this step are then used to determine p-patterns.

How do we determine p-patterns from temporal associations? One approach is to check partial periodicity for each event type in a temporal association and then select only those associations whose event types occur in the same phases of the same period. This approach does not offer much benefit over the period-first algorithm.

An alternative approach is to find periodicities for the sequence of the instances of a temporal association. The algorithm, as we will further discuss, can operate at a very high NSR.

Association-first algorithm

Step 1 Find temporal patterns

1 Set support level and window size

2 $C_1 = A$

3 Run temporal mining

Step 2: Find periods of patterns

1. For each temporal association pattern
   (a) Find sequence of pattern occurrence times
   (b) Find periods for the sequence

Step 1 mines temporal associations. This can be done using the level-wise algorithm discussed in Section 4.1. Step 2 finds periods of temporal association patterns found in step 1. It is accomplished by two sub-steps. First, we identify a sequence of instances of a temporal association pattern. Doing so requires defining the time of each occurrence of a pattern. We use the time of the first event in each occurrence. Second, we employ the chi-squared test discussed in Section 3 to find periods. As with the period-first algorithm, Step 2 can be implemented in parallel to find periods for multiple patterns simultaneously.

The association-first approach has a computational complexity comparable to that of temporal mining, which can be substantial for large patterns and low support levels. On the other hand, this algorithm has much better resilience to noise than the period-first algorithm for a p-pattern with at least 2 items. The reasoning is that when finding periods in Step 2b, difficulties arise if there is a random occurrence of a pattern between two periodic occurrences. However, the chances of a random occurrence of a temporal pattern should be much less than a random occurrence of an individual event, especially for larger patterns.

We note that there is a subtle consideration. If the window size (w) is large compared to the tolerance (δ) of a period, then we are actually finding a superset of the p-patterns. To understand this, we consider an example. Let $V = \{a, b, c\}$ be a temporal association. Suppose that we have the following occurrences:

$\{(a, 1), (b, 3), (c, 5)\}$

$\{(b, 31), (c, 33), (a, 35)\}$

$\{(c, 61), (a, 63), (b, 65)\}$

$\{(b, 91), (a, 93), (c, 95)\}$

Here, the inter-arrival times for a are $\{34, 28, 30\}$; for b, $\{28, 34, 28\}$; and for c, $\{28, 28, 34\}$. Let $\delta = 1$. The period-first algorithm finds that a is not partially periodic and in turn $\{a, b, c\}$ is not a p-pattern, while the aforementioned approach will claim it as a p-pattern because the occurrence time of a pattern is its first event.

4.5 Thoughts on a Hybrid Algorithm

Clearly, there is a trade-off between the period-first and association-first algorithms. The former provides efficiency while the latter provides robustness to noise.

Is it possible to trade robustness to noise for computational efficiency? One approach is to control the maximum size of the temporal patterns considered before checking for periodicities. Specifically, we can view period-first as looking for periodicities in temporal associations of size 1. Conversely, association-first looks for periodicities in temporal associations that are as large as possible. A hybrid algorithm could take as input a parameter m that specifies the maximum size of a temporal association prior to checking for periodicities. Since the temporal association step largely determines computational complexity, we gain some control here. Also, larger temporal associations provide more noise robustness.

This algorithm is summarized as follows.

- Run level-wise algorithm to find all temporal patterns up to the m-th level
- Scan data to find an instance sequence for each m level pattern, and find possible periods
- Use period information to group m-level patterns
- For each pattern group run level-wise search

5 Experiments

This section assesses the algorithms for discovering p-patterns. These assessments consider both effectiveness and efficiency.

5.1 Synthetic Data

We compare the period-first and the association-first algorithms using synthetic data, in which true p-patterns are known in advance. Our focus here is to evaluate their performance in the presence of noise.

Each synthetic data set consists of a mixture of random events and p-patterns. Data sets are characterized by their NSR, the ratio of the duration of on-segments to that of off-segments of a p-pattern, and the length of a p-pattern (i.e. the number of the event types of a p-pattern). We fix the following: ratio of off-segments to on-segments is 0.5; the number of p-patterns is 10; p-pattern length is 5. Further, the duration of an off-segment is exponentially distributed. This distribution is determined by the rate necessary to generate the number of p-patterns to achieve the NSR. Noise events are randomly and uniformly generated. The number of noise events is determined by NSR and the total number of events. Thus, a run is specified by the number of events and NSR.

We begin by assessing the scalability of the period-first and the association-first algorithms. NSR ratio is controlled between $0.5 \sim 1$. We vary the total number of events. A run is effective if there is no false negative, i.e. all p-patterns are found. The minimum support is set[5] so that there is no false positive above level 2. Our experiment consists of 5 runs done with the same value of NSR and different random event streams. Figure 5 plots the average CPU time of the runs in each experiment against the total number of events (in thousands). We can see that the period-first algorithm is $3 \sim 5$ times faster than the association-first algorithm. This is not surprising since the period-first algorithm uses the information about the periods found in its first step to effectively constrain the search space of the level-wise algorithm.

Now we fix the number of events to be 100,000 and vary NSR. This provides an opportunity to see effectiveness of the algorithms. The success rate is computed at the event type level. As shown in Table 2, both algorithms find all p-patterns at low NSRs, although the period-first algorithm is considerably more efficient. As NSR increases, the effectiveness of the period-first algorithm (with first order periodicities) degrades since fewer periodicities are detected. As shown in Figure 3, for an $NSR > 1$, the performance of this algorithm drops considerably. Even more extreme, at NSR in the range of 5 to 10, the period-first algorithm fails completely. Note that the CPU time drops for the period-first as NSR increases. This is because the period-first algorithm finds fewer periods and so there are fewer possible temporal

[5] Our results are not sensitive to *minsup*. A typical *minsup* ranges from $0.1\% \sim 1\%$ of the total number of events.

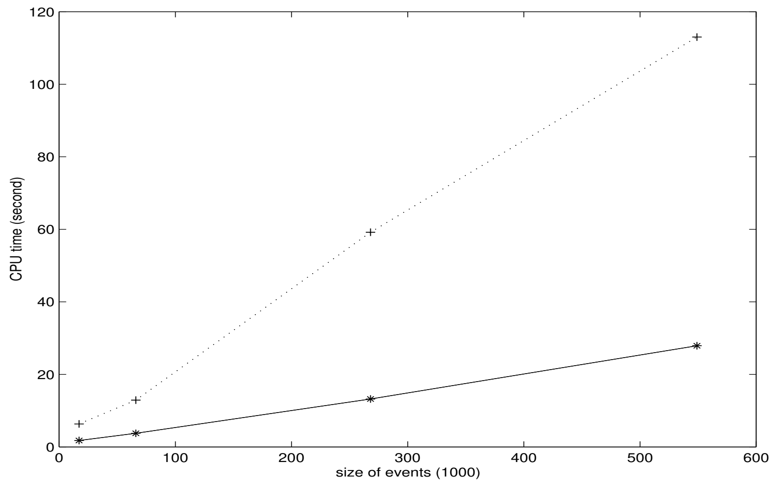

Figure 5: Average run time vs. the number of events

	Effectiveness		Run Time(second)	
NSR	Period-first	Association-first	Period-first	Association-first
$0.5 \sim 1$	100%	100%	4.3	14.7
$1 \sim 2$	40%	100%	1.5	14.5
$5 \sim 10$	0%	100%	0.9	14.9

Table 2: Experimental results

associations. In contrast, the association-first algorithm still finds 100% of the p-patterns.

One final comment is of interest here. This relates to the nature of the low effectiveness of the period-first algorithm if $NSR >> 1$. Under such circumstances, the problem is false negatives as a result of the period detection algorithm. That is, we fail to identify p-patterns that are present rather than falsely identify p-patterns that are not present.

5.2 Production Data

Now, we apply our algorithms to mine p-patterns in real data. Here, our evaluation criteria are more subjective than the last section in that we must rely on the operations staff to detect whether we have false positives or false negatives.

Two data sets are considered. The first data set was collected from an Intranet containing hundreds of network elements (e.g., routers, hubs, and servers). The second data set was collected from an outsourcing center that supports multiple application servers across a large geographical region. Events in the second data set are mostly application-oriented (e.g. the cpu utilization of a server is above threshold), whereas those in the first data set are both network-oriented events (e.g. a link is down for a router) and application-oriented. An event in both data sets consists of three key attributes: host name, which is the source of the event; alarm type, which specifies what happened (e.g., a connection was lost, port up); and the time stamp of when the event occurred. In our preprocessing, we map each distinct pair of host and alarm type into a unique event type. The first data set contains over 10,000 events with around 400 event types over a three-day period. The second data set contains over 100,000 events with around 3000 event types over a two-week period.

Table 3 and Table 4 report our results for the two data sets by pattern size. Column 1 indexes the search level of the level-wise algorithm. Column 2 shows the size of C_k (i.e. the number of qualified patterns at level k) at each level. Column 3 indicates the number of large p-patterns. Here, large p-patterns refer to p-patterns that are not a subset of other p-patterns. Three more columns are used to indicate the diversity of p-patterns found: column 4 shows the number of periods at each level; column 5 is the range (minimum to maximum) of the periods; and column 6 is the range (minimum to maximum) of the occurrence counts for the p-patterns at that level.

As shown in the two tables, many p-patterns are present. Indeed, over 50% of the events are in partially periodic patterns. Thus, $NSR < 1$ and so the period-first algorithm should be effective.

Why are periodic behaviors so common here? Two factors contribute to this. The first is a result of periodic monitoring that is initiated when a high severity event occurs (e.g., an interface-down event). The second factor is a consequence of installation policies, such as rebooting print servers every morning.

Further insights can be drawn from our results. Note that the range of periods is quite large–from less than a second to one day. This has a couple of implications. First, FFT would be extremely inefficient with such a range of time scales. A second implication is the importance of having algorithms that employ different minimum supports. For example, the daily patterns have a support no more than 3 in the three-day data, and no more than 14 in two-week data. Thus, daily patterns can only be discovered if there is a small minimum support.

We reviewed these patterns with the operations staff. It turned out that many of the p-patterns related to underlying problems. The p-patterns with period 1-day and lengths 10 and 11 were found to relate to a periodic port-scan, a possible indicator of a security intrusion. The pattern with period 5-minutes and length 1 resulted from an incorrect configuration of a hub. The pattern with period 60 seconds and 1 was caused by a router that did not get the most current routing information. More details on the data and out results can be found in [10].

6 Conclusion

This paper addresses the discovery of partially periodic temporal associations (p-patterns), a pattern that is common in many applications. An example of a p-pattern in a computer networks is five

Level	Candi-date size	p-patterns	Min:Max Periods	Min:Max count
1	100	28	0:1-day	6 : 680
2	307	22	0:300	3:689
3	938	5	0:30	3:8
4	1917	1	4	3
5	3010	5	4	3
6	3525	3	4	3
7	3104	0		
8	2057	2	4:1-day	3
9	1017	0		
10	366	1	1 day	5
11	91	2	1 day	5
12	14	1	20	20
13	1	1	10	21

Table 3: Experimental results of the first data set

Level	Candi-date size	p-patterns	Min:Max Periods	Min:Max count
1	1500	354	0:1-day	10:2258
2	804	340	0:1-day	10 : 1890
3	781	103	0:2700	12 : 132
4	378	63	$4 : 1 - day$	14 : 160
5	148	24	$30\ 1 - day$	14 : 51
6	39	16	300	21 : 51
7	4	4	300	18 : 54

Table 4: Experimental results of the Second data set

repetitions every 30 seconds of a port-down event followed by a port-up event, which in turn is followed by a random gap until the next five repetitions of these events. Mining such patterns can provide great value. Unfortunately, existing work does not address key characteristics of these patterns, especially the presence of noise, phase shifts, the fact that periods may not be known in advance, and the need to have computationally efficient schemes for finding large patterns with low support.

We begin by defining partially periodic patterns (p-patterns) in a way that includes on-off segments, and phase shifts (via the tolerance parameter δ). Next, we construct an efficient algorithm for finding the period of a partially periodic pattern using a chi-squared test, and we study the performance of the proposed algorithm in the presence of noise. Further, we develop two algorithms for discovering p-patterns based on whether periods or associations are discovered first, and we study trade-offs between these approaches. One result is that the association-first algorithm has higher tolerance to noise while the period-first algorithm is more computationally efficient. In particular, using synthetic data we find that the period-first algorithm is three to five times faster than the association-first algorithm. On the other hand, the effectiveness (i.e., robustness to false negatives and false positives) of the period-first algorithm degrades rapidly if the noise-to-signal ratio exceeds 1. Using these insights, we apply the period-first algorithm to two types of event logs. Many p-patterns are discovered in the log, some of which led to diagnostic and corrective actions.

One area of future work is to explore hybrid algorithms that provide a way to control the trade-off between computational efficiency and effectiveness. Our current work suggests a couple of possibilities in this regard. One is to use the association-first approach but to limit the size of the temporal associations discovered before switching to the period first algorithm. Another strategy is to focus on the algorithm for finding unknown periods. We observe that by considering n-order inter-arrivals (the time between events separated by n other events), we can increase robustness to noise. Thus, we could employ the period-first algorithm but control the parameter n.

References

[1] C. Aggarwal, C. Aggarwal, and V.V.V Parsad. Depth first generation of long patterns. In *Int'l Conf. on Knowledge Discovery and Data Mining*, 2000.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of VLDB*, pages 207–216, 1993.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of VLDB*, 1994.

[4] R.J. Bayardo. Efficiently mining long patterns from database. In *SIGMOD*, pages 85–93, 1998.

[5] C. Bettini, X. Wang, and S. Jajodia. Mining temporal relationships with multiple granularities in time sequences. *Data Engineering Bulletin*, 21:32–38, 1998.

[6] R. COOLEY, J. SRIVASTAVA, and B. MOBASHER. Web mining: Information and pattern discovery on the world wide web. In *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97)*, 1997.

[7] J. Han, G. Dong, and Y. Yin. Efficient mining of partially periodic patterns in time series database. In *Int. Conf. Data Engineering*, 1999.

[8] J. Han, W. Gong, and Y. Yin. Mining segment-wise periodic patterns in time-related database. In *Int'l Conf. on Knowledge Discovery and Data Mining*, 1998.

[9] H.O. Lancaster. *The Chi-squared distibution.* John Wiley & Sons, New York, 1969.

[10] S. Ma and J.L. Hellerstein. Eventbrowser: A flexible tool for scalable analysis of event data. In *DSOM'99*, 1999.

[11] H. Mannila, H. Toivonen, and A. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3), 1997.

[12] T. Oates, M. Schmill, D. Jensen, and P. Cohen. A family of algorithms for finding temporal structure in data. In *6th Intl. Workshop on AI and Statistics*, 1997.

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

[14] S.M. Roos. *Introduction to probability and statistics for engineers and scientists.* John Wiley & Sons, New York, 1987.

[15] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT). Avignon, France.*, 1996.

[16] J. Yang, W. Wang, and P. Yu. Mining asynchronous periodic pattern in time series. In *Int'l Conf. on Knowledge Discovery and Data Mining*, 2000.

[17] O. Zaane, M. Xin, and J. Han. Discovering web access patterns and trends by applying olap and data mining technology on web logs. In *Proc. Advances in Digital Libraries ADL'98*, pages 19–29, 1998.

[18] M. Zaki. Fast mining of sequential patterns in very large databases, 1997. Technical Report URCS TR 668, University of Rochester.

[19] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, pages 141–182, 1997.