InfoMiner: Mining Surprising Periodic Patterns

Jiong Yang
UIUC
jioyang@cs.uiuc.edu

Wei Wang
UNC Chapel Hill
weiwang@cs.unc.edu

Philip S. Yu
IBM T. J. Watson Research Center
psyu@us.ibm.com

September 1, 2003

Abstract

In this paper, we focus on mining surprising periodic patterns in a sequence of events. In many applications, e.g., computational biology, an infrequent pattern is still considered very significant if its actual occurrence frequency exceeds the prior expectation by a large margin. The traditional metric, such as *support*, is not necessarily the ideal model to measure this kind of *surprising patterns* because it treats all patterns equally in the sense that every occurrence carries the same weight towards the assessment of the significance of a pattern regardless of the probability of occurrence. A more suitable measurement, *information*, is introduced to naturally value the degree of surprise of each occurrence of a pattern as a continuous and monotonically decreasing function of its probability of occurrence. This would allow patterns with vastly different occurrence probabilities to be handled seamlessly. As the accumulated degree of surprise of all repetitions of a pattern, the concept of *information gain* is proposed to measure the overall degree of surprise of the pattern within a data sequence. The *bounded information gain* property is identified to tackle the predicament caused by the violation of the *downward closure* property by the information gain measure and in turn provides an efficient solution to this problem. Furthermore, the user has a choice between specifying a minimum information gain threshold and choosing the number of surprising patterns wanted. Empirical tests demonstrate the efficiency and the usefulness of the proposed model.

Jiong Yang is the contact author. His address is 1304 W. Springfield Ave, Urbana, IL 61801. His phone number is 217-333-1681, and email address jioyang@cs.uiuc.edu

1 Introduction

Periodic pattern discovery is an important problem in mining time series data and has wide application. A periodic pattern is an ordered list of events which repeats itself in the event sequence. It is useful in characterizing the cyclic behavior. As a newly developed research area, most previous work on mining sequential data addresses the issue by utilizing the support (number of occurrences) as the metric to identify important patterns from the rest [22, 23, 37, 56]. A qualified pattern in the support model must occur sufficient number of times. In some applications, e.g., market basket, such a model has proved to be meaningful and important. However, in other applications, the number of occurrences may not always represent the significance of a pattern. Consider the following examples.

- *Computational Biology*. A genome consists of a long sequence of nucleotides. Researchers are interested in motifs (i.e., (short) subsequences of nucleotides) which are statistically significant rather than those occurring frequently. The statistical significance of a motif is defined as how likely such a nucleotide permutation would occur in an equivalent random data sequence [10]. In other words, we want to find the motifs that occur at a frequency higher than their expected frequencies. It is obvious that a statistically significant motif may not necessarily occur frequently, thus, the support metric is not an appropriate measurement of the significance of a motif.

- *Web server load*. Consider a web server cluster consisting of 5 servers. The workload on each server is measured by 4 ranks: *low*, *relatively low*, *relatively high*, and *high*. Then there are $4^5 = 1024$ different events, one for each possible combination of server states. Some preliminary examination of the cluster states over time might show that the state fluctuation complies with some periodic behavior. Obtaining such knowledge would be very beneficial for understanding the cluster's behavior and improving its performance. Although having high workload on all servers may occur at a much lower frequency than other states, patterns involving it may be of more interest to administrators if the occurrence of such patterns contradicts the prior expectation.

- *Earthquake*. Earthquakes occur very often in California. It can be classified by its magnitude and type. Scientists may be interested in knowing whether there exists any inherent seismic period so that prediction can be made. Note that any unidentified seismic periodicity involving major earthquake is much more valuable even though it occurs at a much lower frequency than minor ones.

In above examples, we can see that users may be interested in not only the frequently occurred patterns, but also the surprising patterns (i.e., beyond prior expectation) as well. A large number of occurrences of an "expected" frequent pattern sometimes may not be as interesting as a few occurrences of an "expected" rare pattern. The support model is not ideal for these applications because, in the support model, the occurrence of a pattern carries the same weight (i.e., 1) towards its significance, regardless of its likelihood of occurrence. Intuitively, the assessment of significance of a pattern in a sequence should take into account the expectation of pattern occurrence (according to some prior knowledge). Recently, many research has been proposed [4, 9, 11, 12, 26, 28, 29, 33, 36, 39, 40, 50, 55, 59] towards this objective. We will furnish an overview in the next section. In this paper, a new model is proposed to characterize the class of so-called *surprising* patterns (instead of frequent patterns). We will show that our model not only has solid theoretical foundation but also allows an efficient mining algorithm.

The measure of *surprise* should have the following properties. (1) The surprise of a pattern occurrence is anti-monotonic with respect to the likelihood that the pattern may occur by chance (or by prior knowledge). (2) The metric should have some physical meaning, i.e., not arbitrary created. It is fortunate that the *information* metric [7] which is widely studied and used in the communication field can fulfill both requirements. Intuitively, information is a measurement of how likely a pattern will occur or the amount of "surprise" when a pattern actually occurs. If a pattern is expected to occur frequently based on some prior knowledge or by chance, then an occurrence of that pattern carries less information. Thus, we use information to measure the surprise of an occurrence of a pattern. The *information gain* metric is introduced to represent the accumulated information of a pattern in an event sequence and is used to assess the degree of surprise of the pattern. In the remainder of this paper, we refer to this model as the *information model*.

The information model is different from the support model. For a given minimum information gain threshold, let Ψ be the set of patterns that have information gain higher than this threshold. Under the support model, in order to find all patterns in Ψ when event occurrence frequencies are vastly different, the minimum support threshold has to be set very low. A major problem could rise from this: too many patterns. Table 1 shows a comparison between the support model and the information model. The test sequences are constructed from real traces. (The construction of the sequence is described in the experimental section.) In order to find the pattern with the highest information gain, the support threshold has to be set at 0.000234 and there are over 16,000 satisfied patterns in one sequence. It is obvious that the support threshold has to be set very low to discover a small number of patterns with high information gain. This means that patterns with very high information gain are buried in a sea of patterns with relatively low information gain. This could be a large burden for the end user to distinguish the surprising patterns (i.e., patterns with high information gain) from the rest. In addition, since a large number of patterns have to be generated, the support model may yield an inefficient algorithm.

Table 1: Support threshold vs. information gain threshold

Number of Patterns	Scour Trace		IBM Trace	
Satisfied Info. Thresh.	Support Thresh.	Num. of satisfied patterns	Support Thresh.	Num. of satisfied patterns
1	0.000234	16,123	0.0035	637
10	0.000212	16,953	0.0031	711
100	0.000198	17,876	0.0024	987

Although the information gain is a more meaningful metric for the problems addressed previously, it does not preserve the *downward closure* property (as the *support* does). For example, the pattern (a_1, a_2) may have enough information gain while neither $(a_1, *)$ nor $(*, a_2)$ does[1]. The $*$ symbol represents the "don't care" position, which is proposed in [22]. We cannot take advantage of the standard pruning technique (e.g., Apriori algorithm) developed for mining association rules [2] and temporal patterns [3, 23]. Fortunately, we are able to identify the *bounded information gain property* where patterns with inextensible prefixes could not be surprising (given some information gain threshold) and can be excluded from consideration at a very early stage. This motivates us to devise a recursive algorithm as the core of our pattern discovery tool, InfoMiner. In summary, this paper has the following contributions.

- We propose a new mining problem that is to find surprising periodic patterns in a sequence of data.
- The concepts of information and information gain are proposed to measure the degree of surprise of the pattern exhibited in a sequence of data.
- Since the downward closure does not hold with information gain, we devise an efficient algorithm to mine the surprising patterns and associated subsequences based on the *bounded information gain property* that is preserved by the information gain metric.
- The proposed algorithm is capable of finding not only the surprising patterns for a given information gain threshold, but also the K most surprising patterns.

The remainder of this paper is organized as follows. Some related work is discussed in Section 2. We present the information model in Section 3. Section 4 discusses the detailed algorithm of finding patterns whose information gain is

[1]We will explain it in more detail later in this paper.

above a certain threshold while Section 5 presents a simple modification to solve an alternative problem where the K most surprising patterns are wanted. Section 6 discusses some optimization technique. Experimental results are shown in Section 7. Finally, we draw the conclusion in Section 8.

2 Related Work

In this section, we provide a brief overview of recent advances that is closely related to our work presented in this paper.

2.1 Mining Sequence Data

Most previous work on mining sequence data fell into two categories: discovering sequential patterns [3, 5, 6, 14, 17, 20, 25, 32, 39, 52, 58] and mining periodic patterns[22, 23, 37, 56]. The primary difference between them is that the models of sequential pattern purely take into account the number of occurrences of the pattern while the frameworks for periodic patterns focus on characterizing cyclic behaviors. Our work belongs to the latter category.

2.1.1 Sequential Patterns

Discovering frequent sequential patterns was first introduced in [3]. The input data is a set of sequences, called data-sequences. Each data-sequence is a list of transactions and each transaction consists of a set of items. A sequential pattern also consists of a (fully ordered) list of transactions. The problem is to find all frequent sequential patterns with a user-specified minimum support, where the support of a sequential pattern is the percentage of data-sequences that contain the pattern. Apriori-based algorithms, such as AprioriALL [3] and GSP [52], were proposed to mine patterns with some minimum supports in a level-wise manner. To further improve the performance, projection-based algorithms such as FreeSpan [25] and PrefixSpan [43] were introduced to reduce the candidate patterns generated and hence reduce the number of scans through the data. Additional useful constraints (such as time constraint and regular expression constraint) and/or taxonomies were also studied extensively in [17, 52, 58] to enable more powerful models of sequential patterns.

As a more generative model, the problem of discovering frequent episodes from a sequence of events was presented in [32]. An episode is defined to be a collection of events that occur relatively close to each other in a given partial order. A time window is moved across the input sequence and all episodes that occur in some user-specified percentage of windows are reported. The model was further generalized by Padmanabhan et al. [38] to suit temporal logic patterns.

2.1.2 Periodic Patterns

Full cyclic pattern was first studied in [37]. The input data to [37] is a set of transactions, each of which consists a set of items. In addition, each transaction is tagged with an execution time. The goal is to find association rules that repeat themselves throughout the input data. In [22, 23], Han et. al. presented algorithms for efficiently mining partial periodic patterns. In practice, not every portion in the time series may contribute to the periodicity. For example, a company's stock may often gain a couple of points at the beginning of each trading session but it may not have much regularity at later time. This type of looser periodicity is often referred to as *partial periodicity*. We will see later that our model also allows partial periodicity. The difference between our model and [22, 23] is that we aim at mining surprising periodic patterns while Han et al. focused on frequent periodic patterns.

2.2 Models of Interestingness

Despite the difference in problem formulation, most work surveyed in the previous subsection adopted the support as the measure of interestingness/significance and aimed at discovering frequent patterns. Recently, many efforts have been carried out to address the potential disadvantages associated with the support model and to propose alternative solutions.

2.2.1 Refining Mined Results

As a well-known fact, the number of patterns/rules discovered under the support model can be very large. Many post-processing techniques have been developed to reduce the number of discovered patterns into a manageable size while preserving the discovered knowledge as much as possible. Human interaction is involved in [28, 48, 51] to specify the interestingness or beliefs to guide the process while others [8, 29, 31] focused on reducing redundant information possessed by the discovered rules. It is clear that these post-processing techniques are typically used as an additional pruning step after the normal mining procedure (which produces a large rule set). In contrast, our proposed scheme successfully avoids the generation of a large number of insignificant/uninteresting patterns from the beginning and enables a much more efficient solution.

Another approach to reduce redundancy is to return only closed frequent itemset [41, 42, 59]. Intuitively, an itemset is a closed itemset if all of its supersets have smaller support. While the set of frequent closed itemsets is typically much smaller, it has been proved that all frequent itemsets can be uniquely derived from the set of frequent closed itemsets. Again, this approach still focuses on mining frequent itemsets and fails to address the problem we mentioned in the previous section.

2.2.2 Multiple Supports Scheme

Multiple supports scheme was introduced by Liu et. al. [29] and later extended by Wang et al. [55] to find itemsets which do not occur frequently overall, but have high correlation to occur with some other items. The support threshold to qualify a frequent itemset can be specified as a fraction of the minimum support of all items [29] or subsets of items [55] in the itemset. This variable support has similar effect as the information gain introduced in this paper. However, there exists some fundamental difference between these two concepts. For example, if the support of item A, B, and C is 0.01, 0.02, 0.8, respectively, then the support threshold to qualify itemset AB and AC is the same. Nevertheless, the itemset AC is expected to occur more frequently than AB because the support of C is much larger than that of B. This aspect was not fully taken into account by the multiple support model[2]. In contrast, the information gain metric proposed in this paper would capture the difference of occurrences between B and C.

2.2.3 Statistically Significant Patterns

Mining patterns that are statistically significant (rather than frequent) becomes a popular topic. Brin et al. [9] first introduced the concept of correlation and it was shown that in many applications the correlation measurement can reveal some very important patterns. The Chi-squared test was used to test the correlation among items. Instead of explicitly enumerating all correlated itemsets, the border comprising the set of minimal correlated itemsets[3] is identified, and no further distinction is made on the degree of correlation of itemsets above the border (i.e., supersets of some itemset on the border). This model sometimes becomes sub-optimal. As shown in Figure 1, itemsets A and B are highly correlated but C is independent of them[4]. In addition, $\{A, B, C, D\}$ is also highly correlated. We can view that the degree of correlation of $\{A, B, C\}$ is not

[2]Theoretically, the model in [55] is capable of addressing this problem by explicitly enumerating all itemsets, which is unfortunately impractical in general.

[3]A minimal correlated itemset is a correlated itemset whose subsets are all independent.

[4]$Prob(AB) \times Prob(C) = \frac{1}{2} \times \frac{2}{3} = Prob(ABC)$.

as strong as that of $\{A, B\}$ and $\{A, B, C, D\}$. This observation can also be confirmed by the Chi-squared test [5]. In many applications, users are only interested in the itemsets such as $\{A, B\}$ and $\{A, B, C, D\}$, but not $\{A, B, C\}$. However, [9] cannot distinguish between $\{A, B, C\}$ and $\{A, B, C, D\}$ once $\{A, B\}$ is identified as a correlated itemset. Furthermore, if a user is interested in finding k itemsets with the highest correlation, then all itemsets in the lattice have to be examined before k highest ones can be determined. Another potential drawback of this model is the expensive computation required by this model. The running time of all patterns with i-correlated items is $O(n \times |CAND| \times \min\{n, 2^i\})$ where n and $|CAND|$ are the number of transactions and the number of candidates at the ith level, respectively. To overcome these drawbacks, Oates et al. [35, 36] proposed models for statistical dependencies using G statistic and devised randomized algorithms to produce approximate results. In contrast, our model not only can successfully identify $\{A, B\}$ and $\{A, B, C, D\}$ without including $\{A, B, C\}$ but also leads to a much more efficient deterministic algorithm.

Transaction ID	**Items**
1	**ABCD**
2	**ABFG**
3	**CEGF**
4	**ABCD**
5	**CEGH**
6	**CEFH**

Figure 1: An example of transaction set

More recently, Cohen et al. [12] and Fujiwara et al. [16] address the problem of identifying pairs of attributes with high confidence or similarity (in terms of probabilistic correlations in the database) in the absence of support requirement. Hashing based algorithms [12] are proposed to tackle the problem, which consist of three general phases: computing hash signature, generating candidates, and pruning candidates. To avoid both false negatives and false positives that may be yielded with the hashing based scheme, a family of so called *dynamic miss counting* algorithms are proposed in [16]. Instead of counting the number of hits (as most other algorithm does), the number of transactions where the given pair of attributes disagree is counted and this counter is deleted as soon as the number of misses exceeds the maximum number of allowed misses for that pair. This strategy is proved to be able to reduce the memory size significantly.

Another important advance is accomplished in mining so-called *unexpected patterns*. Berger et al. [4] proposed a probabilistic measure of interestingness based on unexpectedness in the context of temporal logic, whereby a pattern is deemed interesting if the ratio of the actual number of occurrences of the pattern exceeds the expected one by some user defined threshold. Solving the problem in its general frame is in nature NP-hard and hence some heuristics are proposed to produce an approximate answer. Our model presented in this paper can in fact achieve a similar objective but enables an efficient solution without sacrificing the accuracy. Padmanabhan et al. [39, 40, 50] define the unexpectedness of association rules relative to a system of prior beliefs. Specifically, the belief is of the form $X \to Y$ and a rule is said to be unexpected if it contradicts the belief. The set of beliefs (given by the user) are used to conduct the mining process efficiently so that an exhaustive search is avoided. The primary advantage of this model is that it can customize the mining process for the users who have fairly good prior knowledge and specific interests, and is particularly useful in refinements of user's beliefs.

A formal study of surprising patterns is furnished in [11], focusing on the analysis of variation of inter-item correlations along time. The surprise is defined in terms of the coding length in a carefully chosen encoding scheme and has solid theoretic foundation, but requires much more expensive computation comparing to other models.

[5] In general, the chi-squared test requires a large sample. For the demonstration purpose only, we assume that the chi-squared test is valid in this example.

2.3 Projection-based Approaches

Some research [1, 24, 25, 60] has been carried out on mining frequent patterns in a depth-first projection-based fashion as opposite to the traditional breadth-first level-wise traversal. As patterns are usually organized via a tree [1, 24, 25] (or lattice in [60]) structure, along with the discovery of frequent patterns when traversal through the structure, data projection onto each newly identified frequent pattern is also taken to facilitate the subsequent examination of its super-patterns. The Apriori property is (sometimes implicitly) used to efficiently prune the tree. In particular, Han et al. [24] propose a so-called frequent pattern tree (FP-tree) to organize the produced data projection in a concise and ingenious manner so that the generation of a huge number of candidate patterns can be avoided completely. While the FP-growth [24] is designed for mining frequent itemsets in general, the FreeSpan [25], PrefixSpan [43], and SPADE [60] are specifically tailored for mining sequential patterns. It is interesting to notice that, the depth-first approaches generally perform better than breadth-first ones, and the advantage becomes more substantial when the pattern is long.

However, the Apriori property does not hold on the information gain metric proposed in this paper and hence can not be employed to mine patterns satisfying a user specified information gain threshold. Looking ahead, we will introduce a so-called *bounded information gain* property and employ a depth-first, projection-based approach that starts from the patterns with only one filled position and then proceeds to more complex patterns gradually, and in the mean time utilizes the bounded information gain property to continuously refine the candidate event list associated with each unspecified position in the pattern.

3 Model of Surprising Patterns

In this paper, we adopt the general model for periodic patterns proposed in [23] with one exception: Instead of finding *frequent* patterns[6], our goal is to discover **surprising** patterns in an event sequence. Let $E = \{a_1, a_2, \ldots\}$ be a set of distinct events. The event sequence is a sequence of events in E. A periodic pattern is a list of l events that may occur recurrently in the sequence with period length l. The **information** carried by an event a_i ($a_i \in E$) is defined to be $I(a_i) = -\log_{|E|} Prob(a_i)$ where $\mid E \mid$ and $Prob(a_i)$ are the number of events in E and the probability that a_i occurs, respectively. The probability $Prob(a_i)$ can be assessed in many ways which include but are not limited to:

- *Uniform distribution*: $Prob(a_1) = Prob(a_2) = \ldots = Prob(a_i) = \ldots = \frac{1}{|E|}$;
- *Experimental evidence*: $Prob(a_i) = \frac{Num_D(a_i)}{N}$ for all $a_i \in E$ where $Num_D(a_i)$ and N are the number of occurrences of the event a_i in an event sequence D and the length of D, respectively;
- *Prior belief*: $Prob(a_i)$ is determined by some domain expert.

Without loss of generality, we adopt the second option to assess the information carried by an event, i.e. an occurrence of a frequent event carries less information/surprise than that of a rare event. Note that this also coincides with the original intention of *information* in the data communication community. We shall show later that this gives us the opportunity to handle patterns with divergent probabilities seamlessly. Theoretically speaking, the base of the logarithm function can be any real number that is greater than 1. Typically, $\mid E \mid$ is chosen to be the base to play a normalization role in the computation (i.e., $I(a_i) = 1$ if $Prob(a_i) = \frac{1}{|E|}$). For example, the sequence in Figure 2 contains 6 different events a_1, a_2, a_3, a_4, a_5, and a_6. Their probabilities of occurrence and information are shown in Table 2.

A **pattern** of **length** l is a tuple of l events, each of which is either an event in E, or the **eternal event** (represented by symbol $*$). An eternal event is a virtual event that matches any event in E and is used to represent the "don't care" position

[6]For a pattern s in a sequence $d_1, d_2, \ldots, d_N$, the frequency count is defined as $\mid \{i \mid 0 \leq i \leq \frac{N}{|s|}$, and the string s is true in $d_{i|s|+1}, \ldots, d_{i|s|+|s|}\} \mid$.

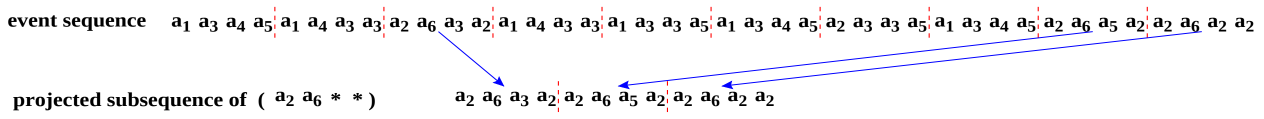

event sequence
a1 a3 a4 a5 a1 a4 a3 a3 a2 a6 a3 a2 a1 a4 a3 a3 a1 a3 a3 a5 a1 a3 a4 a5 a2 a3 a3 a5 a1 a3 a4 a5 a2 a6 a5 a2 a2 a6 a2 a2
projected subsequence of (a2 a6 * *)
a2 a6 a3 a2 a2 a6 a5 a2 a2 a6 a2 a2

Figure 2: Event sequence and projected subsequence

Table 2: Probability of Occurrence and Information

Event	Probability	Information
a_1	$\frac{6}{40} = 0.15$	$-\log_6(0.15) = 1.06$
a_2	$\frac{8}{40} = 0.20$	$-\log_6(0.20) = 0.90$
a_3	$\frac{12}{40} = 0.30$	$-\log_6(0.30) = 0.67$
a_4	$\frac{5}{40} = 0.125$	$-\log_6(0.125) = 1.16$
a_5	$\frac{6}{40} = 0.15$	$-\log_6(0.15) = 1.06$
a_6	$\frac{3}{40} = 0.075$	$-\log_6(0.075) = 1.45$

in a pattern. By definition, the information of the eternal event * is $I(*) = -log_{|E|}Prob(*) = 0$ since $Prob(*) = 1$. An intuitive interpretation is that the occurrence of an event that is known to be always true does not provide any "new information" or "surprise". A pattern P with length l is in the form of $(p_1, p_2, \ldots, p_l)$ where $p_i \in E \cup \{*\}$ $(1 \leq i \leq l)$ and at least one position has to be filled by an event in E[7]. P is called a **singular pattern** if only one position in P is filled by an event in E and the rest positions are filled by *. Otherwise, P is referred to as a **complex pattern**. For example, $(*, a_3, *)$ is a singular pattern of length 3 and $(a_2, a_6, *, a_2)$ is a complex pattern of length 4. Note that an event may have multiple occurrences in a pattern. As a permutation of a list of events, a pattern $P = (p_1, p_2, \ldots, p_l)$ will occur with a probability $Prob(P) = Prob(p_1) \times Prob(p_2) \times \ldots \times Prob(p_l)$ in a random event sequence if no advanced knowledge on correlation among these events is assumed. Then the information carried by P is $I(P) = -\log_{|E|} Prob(P) = I(p_1) + I(p_2) + \ldots + I(p_l)$. It follows directly that the information of a singular pattern always equals to the information of the event specified in the pattern. This property provides a natural bridge between events and patterns. For example, $I((*, a_6, *, *)) = I(a_6) = 1.45$ and $I((a_2, a_6, *, *)) = I(a_2) + I(a_6) = 0.90 + 1.45 = 2.35$ according to Table 2.

Given a pattern $P = (p_1, p_2, \ldots, p_l)$ and a segment S of l events $s_1, s_2, \ldots, s_l$, we say that S **supports** P if, for each event p_i $(1 \leq i \leq l)$ specified in P, either $p_i = *$ or $p_i = s_i$ is true. The segment a_2, a_6, a_3, a_2 supports the pattern $(a_2, a_6, *, *)$ while the segment a_1, a_6, a_4, a_5 does not. To assess whether a pattern of length l is surprising in an event sequence D, D is viewed as a list of disjoint contiguous segments, each of which consists of l events. The number of segments that support P is also called the **support** of P (denoted by $Support(P)$). The event subsequence[8] consisting of the list of segments that support P is called the **projected subsequence** of D on P. In Figure 2, the event sequence consists of 40 events. When mining periodic patterns with $l = 4$, it can be viewed as a list of 10 segments, each of which contains 4 events. The support of $(a_2, a_6, *, *)$ is 3 in the sequence and the projected subsequence on $(a_2, a_6, *, *)$ is a_2, a_6, a_3, a_2 a_2, a_6, a_5, a_2, a_2, a_6, a_2, a_2. As a measurement of the degree of surprise of a pattern P in an event sequence D, the **information gain** of P in D is defined as $G(P) = I(P) \times (Support(P) - 1)$. Since our objective is to mine surprising periodic patterns, an event combination appearing in the event sequence which never recurs is of little value in this problem domain. Therefore, in the proposed model, only recurrences of a pattern will have positive contribution to the information gain. $Support(P) - 1$ is indeed the number of recurrences of P. In the rest of the paper, we will use $Repetition(P)$ to denote it. For example, $Repetition((a_2, a_6, *, *)) = 3 - 1 = 2$ and $G((a_2, a_6, *, *)) = 2.35 \times 2 = 4.70$

[7] This requirement is employed to exclude the trivial pattern $(*, *, \ldots, *)$ from being considered.

[8] Given two sequences D and D', D is a **subsequence** of D' if D can be obtained by removing some events in D'.

in Figure 2.

Similar to the support model, an information gain threshold, min_gain, is specified by the user to define the minimum information gain to qualify a surprising pattern. Given an event sequence and an information gain threshold, the goal is to discover all patterns whose information gains in the event sequence exceed the threshold. Obviously, the proper value of this threshold is application dependent and may be specified by a domain expert. A user may use the following heuristic to choose the value of min_gain. If a pattern with probability p is regarded as a surprising pattern when it repeats itself by at least n times in the sequence. Then the min_gain can be set to $(-\log_{|E|} p) \times n$ where $-\log_{|E|} p$ is the information of the pattern. Alternatively, the user also has the opportunity to specify the number of (most) surprising patterns needed. We will show later that our proposed algorithm can efficiently produce desired results under both specifications.

It is conceivable that the information model can also be applied to define both surprising itemsets from transaction database and surprising sequential patterns from sequence database. Without loss of generality, we focus our discussion to the domain of mining periodic patterns in a sequence in this paper. To facilitate the explanation in the rest of the paper, we refer to a pattern, say P, as a **subpattern** of another pattern, say P', if P can be generated by replacing some event(s) in P' by the eternal event *. P' is called a **superpattern** of P. For example, $(a_2, a_6, *, *)$ and $(*, a_6, *, *)$ are subpatterns of $(a_2, a_6, *, a_2)$. The pattern-subpattern relationship essentially defines a partial order among patterns of the same length.

4 Projection-based Algorithm

Previous work on pattern discovery usually utilizes the Apriori property that can be stated as "if a pattern P is significant, then any subpattern of P is also significant". This property holds for the support model but is not true in the information model. For example, in Figure 2, the information gain $G((a_2, *, *, *)) = 0.90 \times 3 = 2.70$ which is less than the information gain of pattern $(a_2, a_6, *, *)$ (i.e., 4.70). If the threshold is set to be 4.5, then $(a_2, a_6, *, *)$ qualifies while $(a_2, *, *, *)$ does not. (Note that a_6 is an infrequent event which occurs only three times in the event sequence.) This implies that the algorithms developed for the support model are not applicable. The pruning power of the support model essentially comes from the fact that if we know a pattern is not valid, we do not need to examine its superpatterns. Can we achieve a similar pruning power under the information model? To answer this question, we first introduce a concept called *extensible prefix*.

Definition 4.1 *For a pattern $P = (p_1, p_2, \ldots, p_l)$, the tuple $(p_1, p_2, \ldots, p_i)$ is called a* **prefix** *of P where $1 \leq i \leq l$.*

A prefix is part of a pattern. A pattern can be generated by appending more events to the prefix. For instance, $(a_1, *, a_4)$ is a prefix of patterns $(a_1, *, a_4, a_3)$ $(a_1, *, a_4, a_2)$, $(a_1, *, a_4, a_4)$, and $(a_1, *, a_4, *)$, etc.

Definition 4.2 *Given an information gain threshold min_gain, a prefix is* **extensible** *if at least one pattern with this prefix is surprising (i.e., whose information gain meets min_gain), and is* **inextensible** *otherwise.*

It follows from the definition that, all prefixes of a surprising pattern are extensible, and any pattern with an inextensible prefix cannot be a surprising pattern. In order to find all surprising patterns, we only need to examine extensible prefixes. The challenge lies on how to recognize inextensible prefixes as early as possible so that they can be excluded from further investigation. However, if all patterns with a certain prefix have to be examined exhaustively before we are able to determine whether the prefix is inextensible, then we will not be able to save any computation. Fortunately, the assessment of extensibility can be done efficiently due to a so-called *bounded information gain* property discussed in the following context.

Lemma 4.1 *Let $P = (p_1, p_2, \ldots, p_l)$ be a pattern and $P_i = (p_1, p_2, \ldots, p_i)$ be a prefix of P. Given an event sequence, $G(P) \leq I(P) \times Repetition(P_i)$, where $I(P) = \sum_{k=1}^{l} I(p_k)$ is the information of the pattern P.*

Proof. Since P_i is a prefix of P, $Repetition(P) \leq Repetition(P_i)$ due to the Apriori property. Thus $G(P) = I(P) \times Repetition(P) \leq I(P) \times Repetition(P_i)$. □

Consider the pattern $P = (a_1, a_3, a_4, a_5)$ and its prefix $P_2 = (a_1, a_3)$ in the sequence in Figure 2. Clearly, we have $G(P) = I(P) \times Repetition(P) = 4.34 \times 2 = 8.68$ which is less than the value of $I(P) \times Repetition(P_2) = 4.34 \times 3 = 13.02$. This suggests that we may take advantage of this "bounding" effect when we assess the extensibility of a prefix. For a given prefix P_i, consider the set (denoted by $\Lambda(P_i)$) of patterns of period length l with the same prefix P_i and let P_{max} be the pattern with the highest information in $\Lambda(P_i)$. Then, for any pattern P with prefix P_i (i.e., $P \in \Lambda(P_i)$), the inequality $G(P) \leq I(P_{max}) \times Repetition(P_i)$ holds by Lemma 4.1. Therefore, we can determine whether P_i is extensible by estimating the maximum information $I(P_{max})$ that may be carried by any pattern with prefix P_i. The value of $I(P_{max})$ can be computed by, first, for each unspecified position (following the prefix P_i), identifying the highest information possessed by any potential event for that position; and then aggregating them together with the information possessed by P_i. It is conceivable that the prefix P_i can not be extensible if $I(P_{max}) \times Repetition(P_i)$ is below the information gain threshold min_gain. This leads to the following theorem.

Theorem 4.2 (Bounded information gain) *Given an information gain threshold min_gain and a period length l, a prefix $P_i = (p_1, p_2, \ldots, p_i)$, is not extensible iff $Repetition(P_i) < \frac{min\_gain}{max\_info}$ where $max\_info = I(P_i) + \sum_{k=i+1}^{l} f_k$ is the maximum information that can be carried by any pattern with prefix P_i and f_k is the highest information that can be carried by any potential event at the (unspecified) position k.*

Once a prefix is deemed to be inextensible, we will immediately eliminate it from any further examination. Only extensible prefixes will be used to extend to longer (extensible) prefixes and to construct candidate patterns. Furthermore, given a period length l, for any prefix $P_i = (p_1, p_2, \ldots, p_i)$, consider an unspecified position k where $i < k \leq l$. Not every event can potentially be at position k in a surprising pattern with prefix P_i. An event $e \in E$ can possibly be a candidate for position k only if e recurs on position k for a sufficient number of times. In particular, the minimum required number of repetitions is $min\_rep = \frac{min\_gain}{I(P_i)+\sum_{j=i+1}^{l} f_j}$. This indicates that only a (small) subset of events may serve as the candidates for each unspecified position and we can limit our search to these candidate events, and also leads to the following remark.

Remark 4.3 (Candidate refinement) *For any two prefixes $P_{i1} = (p_1, p_2, \ldots, p_{i1})$ and $P_{i2} = (p_1, p_2, \ldots, p_{i2})$ where $i1 < i2$, any candidate event e on position k ($i2 < k \leq l$) for prefix P_{i2} must also be a candidate on position k for prefix P_{i1}, where l is the period length.*

Proof. With prefix P_{i2}, an event has to recur at least $min\_rep_2 = \frac{min\_gain}{I(P_{i2})+\sum_{j=i2+1}^{l} f_j}$ times on position k to qualify as a candidate while the minimum required number of repetition with prefix P_{i1} is $min\_rep_1 = \frac{min\_gain}{I(P_{i1})+\sum_{j=i1+1}^{l} f_j} = \frac{min\_gain}{I(P_{i1})+\sum_{j=i1+1}^{i2} f_j+\sum_{j=i2+1}^{l} f_j} \leq \frac{min\_gain}{I(P_{i1})+\sum_{j=i1+1}^{i2} p_j+\sum_{j=i2+1}^{l} f_j} = \frac{min\_gain}{I(P_{i2})+\sum_{j=i2+1}^{l} f_j} = min\_rep_2$. Therefore, any candidate on position k for prefix P_{i2} must be a candidate on position k for prefix P_{i1}. □

Remark 4.3 states that, as the prefix grows longer by filling some unspecified positions, the candidate set of each remaining unspecified position will only shrink monotonically. This provides us with the opportunity to mine surprising patterns by only proceeding with candidate event for each position. Powered by this pruning technique, we develop a progressive approach by starting from extensible prefixes that contain only one filled position (the remaining positions are unspecified) and then proceeding to extensible prefixes with more filled positions gradually to achieve the maximum pruning effect. A candidate event list for each open (i.e., unspecified) position is maintained and continuously refined when more positions are filled. This process continues until all surprising patterns have been identified by examining and extending extensible prefixes. A depth first algorithm is then developed to generate all qualified patterns in a recursive manner.

Another observation we made is that a segment shall not support a pattern P, if it does not support one of P's prefixes. To expedite the process, when we examine a prefix Q, we may screen out those segments that do not support Q and only retain the projected subsequence of Q so that the evaluation of any prefix containing Q would not have to resort to the original sequence. Note that the projected subsequence will also be further refined every time the algorithm proceeds to a prefix with more specified positions. For a given period l, starting with a pattern frame of l slots (without any specific event assigned), potential patterns (or prefixes of potential patterns) are generated progressively by subsequently assigning a candidate event to a yet-unspecified position one at a time. Such assignment may lead to both a refinement of event candidates for the remaining position(s) by applying the above property and a further projection of the projected subsequence onto the remaining open positions. There certainly exist many different orders according to which the open positions are examined. For brevity, we assume the left-to-right order if not specified otherwise in the following discussion. Some heuristic on determining the optimal order is discussed in Section 6 and its effect is shown in Section 7.2.2.

4.1 Main Routine

As shown in Algorithm 4.1, the main procedure of mining patterns for a given pattern period is described in the procedure *InfoMiner*. *InfoMiner* is a recursive function. At the kth level of recursion, the patterns with k non-eternal events are examined. For example, all singular patterns (e.g., $(a_1, *, *, *)$) are examined at the initial invocation of *InfoMiner*; at the second level of invocations of *InfoMiner*, all candidate patterns with two non-eternal events (e.g., $(a_1, *, *, a_5)$) are evaluated; an so on. This is achieved by extending the extensible prefixes to include an additional event during each invocation of *InfoMiner* (Line 5-6) and passing the new prefixes to the next level of recursion (Line 8-10). Notice that at most l levels of recursion may be invoked to mine patterns of period l.

Being more specific, at each level of the recursion, we evaluate patterns with certain prefixes in a projected subsequence S. Starting from a null prefix and the sequence in Figure 2, a step-by-step example is shown in Figures 3(i), 3(ii), 4(i), and 4(ii) for prefix $null$, (a_1), (a_1, a_3), and (a_1, a_3, a_4), respectively. First, in the procedure $Repetition\_Calculation(l, S)$ (Line 1), the number of repetitions for each candidate event of each open position is collected from the projected subsequence S. Then the bounded information gain property is employed to refine the candidate list for each remaining open position (Line 2). Finally, for each open position i and each event e in the refined candidate list, a new prefix is created by extending the original one to include the event e on position i (Line 5). Note that this newly created prefix is guaranteed to be extensible and would have the same number of repetitions as the event e at position i (i.e., $repetition[i][e]$). A candidate pattern $P = (prefix, \overbrace{* \ldots *}^{l-i})$ is constructed by filling all remaining open positions following the new_prefix with the eternal event * (Line 6). A subroutine $Pattern\_Validation$ (Line 7) is then invoked to verify whether P has sufficient information gain. The projected subsequence on each new prefix is also generated (Line 8).

Algorithm 4.1 *InfoMiner*

```
/* Generate qualified patterns prefixed by $prefix$ and having
$l$ other undecided positions from the projected subsequence $S$
where $min\_gain$ is the required minimum information gain of a
qualified pattern. */

InfoMiner($min\_gain, l, S, prefix$)
{
1:Repetition_Calculation($l, S$)
2:Bounded_Information_Pruning($min\_gain, l, S, prefix$)
3:for $i \leftarrow 1$ to $l$ do
4:    for each candidate event $e$ at the $i$th open position do
5:        $new\_prefix \leftarrow (prefix, \overbrace{* \ldots *}^{i-1}, e)$
6:        $P \leftarrow (newprefix, \overbrace{* \ldots *}^{l-i})$
7:        Pattern_Validation($min\_gain, P, repetition[i][e]$)
8:        $S' \leftarrow$ Projected_Subsequence($S, new\_prefix$)
9:        if $S' \neq \emptyset$
10:       then InfoMiner($min\_gain, l-i, S', new\_prefix$)
}
```

Algorithm 4.2 *Repetition Calculation*

```
/* For each of these $l$ undecided positions, compute the number
of repetitions for each candidate event for this position from the
projected subsequence $S$. */

Repetition_Calculation($l, S$)
{
1:for $i \leftarrow 1$ to $l$ do
2:    for each candidate event $e$ of the $i$th position do
3:        calculate $repetition[i][e]$ from $S$
}
```

On the initial call of *InfoMiner*, the entire event sequence and $null$ are taken as the input sequence S and the pattern prefix, respectively. In addition, the candidate list of each position consists of all events initially (Figure 3(i)(a)). For each successive invocation of *InfoMiner*, a non-empty prefix and its corresponding projected subsequence are passed in as input arguments, and for each remaining open position, those retained events after $Bounded\_Information\_Pruning$ in the previous invocation are taken as the candidates for $Repetition\_Calculation$. Consider Figure 3(ii), where *InfoMiner* is invoked with $min\_gain = 4.5$, a prefix (a_1), and the projected subsequence given in Figure 3(ii)(a). The retained events for positions 2, 3, and 4 in Figure 3(i)(b) are taken as the candidates in Figure 3(ii)(b) to calculate the repetition in the projected subsequence of prefix (a_1). After the $Repetition\_Calculation$ subroutine, a further pruning of these candidates is carried out in the $Bounded\_Information\_Pruning$ subroutine. The refined candidate lists are given in Figure 3(ii)(c). It is obvious that the candidate list of each open position shrinks after each pruning. This observation can also be made by comparing Figures 3(i)(b), 3(ii)(c), 4(i)(c), and 4(ii)(c), and is demonstrated in Section 7.2. From the candidates in Figure 3(ii)(c), four new prefixes are generated by extending the previous prefix (a_1): (a_1, a_3), $(a_1, *, a_3)$, $(a_1, *, a_4)$, and $(a_1, *, *, a_5)$ in Figure 3(ii)(d). The same procedure (i.e., *InfoMiner*), if applicable, is performed recursively on the newly generated prefixes and their corresponding projected subsequences. In the following subsections, we will discuss each subroutine separately.

Algorithm 4.3 *Bounded Information Gain Pruning*

```
/* For each of these $l$ undecided positions, generate a refined list
of candidate events. */

Bounded_Information_Pruning($min\_gain, l, S, prefix$)
{
1:$max\_info \leftarrow$ information of $prefix$
2:for $i \leftarrow 1$ to $l$ do {
3:    $max[i] \leftarrow$ the maximum value of $I(e)$
4:            for all candidate event $e$ of the $i$th position
5:    $max\_info \leftarrow max\_info + max[i]$ }
6:$min\_rep \leftarrow \lceil \frac{min\_gain}{max\_info} \rceil$
7:for $i \leftarrow 1$ to $l$ do
8:    remove all events $e$ whose $repetition[i][e] < min\_rep$
9:            from the candidates list
}
```

Algorithm 4.4 *Pattern Validation*

```
/* Verify whether a given candidate pattern has sufficient infor-
mation gain. If so, add it to the $Result$. */

Pattern_Validation($min\_gain, P, repetition$)
{
1:$info \leftarrow$ information of $P$
2:$G(P) = repetition \times info$
3:if $G(P) \geq min\_gain$
4:then $Result \leftarrow Result \cup \{\langle P, G(P) \rangle\}$
}
```

4.2 Repetition Calculation

This subroutine in Algorithm 4.2 is responsible for collecting the number of repetitions for each candidate event of each position. At the first time this subroutine is invoked, the $prefix$ is specified as $null$ and every event is considered as a candidate for every position as illustrated in Figure 3(i)(a). The entire sequence is scanned to collect the repetition for each event-position combination. The repetitions shown in Figure 3(i)(a) is computed from the event sequence in Figure 2. During each subsequent invocation, a newly generated non-empty $prefix$ is specified and only the projected subsequence on this prefix is passed in as shown in Figure 3(ii)(a), 4(i)(a), and 4(ii)(a). Only the repetition of each retained candidate event (after pruning in preceding recursions) of each open position in the projected subsequence is collected (e.g., Figure 3(ii)(b), 4(i)(b), and 4(ii)(b)).

Algorithm 4.5 *Projected Subsequence Construction*

```
/* Construct the projected subsequence of a data sequence $S$ for
a given $prefix$. */

Projected_Subsequence($S, prefix$)
{
1:$S' \leftarrow \emptyset$
2:for each segment $s$ in $S$ do
3:    if $s$ supports $(prefix, * \ldots *)$
4:    then append $s$ to the end of $S'$
5:return $S'$
}
```

4.3 Bounded Information Gain Pruning

Arming with the bounded information gain property, the procedure *Bounded Information Pruning* aims to refine the candidate list for each open position (Algorithm 4.3). The general idea is that, by calculating the maximum information (denoted by max_info) that may be carried by a candidate pattern[9] (with a certain prefix) (Line1-4), we can obtain the minimum repetitions (denoted by min_rep) that is necessary to accumulate enough information gain (Line 5). Those events that do not have sufficient repetitions are then removed from the candidate list (Line 6-7). To calculate the value of max_info, for each open position, the event of the maximum information among all candidates for that position is identified (Line 3). The aggregation is taken on the information of each identified event and the information carried by the $prefix$ (Line 1, 4). The value of this aggregation serves as max_info, and min_rep can be calculated by $min\_rep = \lceil \frac{min\_gain}{max\_info} \rceil$. For example, events a_1, a_6, a_4, and a_5 are the events that have the maximum information among the candidates for position 1, 2, 3, and 4 in Figure 3(i)(a), respectively. (The information associated with each event is shown in Table 2.) Then the maximum information that a qualified pattern may carry can be computed as $max\_info = I(a_1) + I(a_6) + I(a_4) + I(a_5) = 1.06 + 1.45 + 1.16 + 1.06 = 4.73$. In turn, the required minimum repetition is $min\_rep = \lceil \frac{4.5}{4.73} \rceil = 1$ if $min\_gain = 4.5$. Finally, after removing all events whose repetition is less than min_rep, the refined candidate list for each position is shown in Figure 3(i)(b).

In the case that the prefix contains some non-eternal event (e.g., $prefix = (a_1)$ in Figure 3(ii)), the information of the prefix (a_1) is aggregated together with the maximum information from each open position (i.e., the information of a_4, a_4, and a_5, respectively). We have $max\_info = I(a_1) + I(a_4) + I(a_4) + I(a_5) = 1.06 + 1.16 + 1.16 + 1.06 = 4.44$ and $min\_rep = \lceil \frac{4.5}{4.44} \rceil = 2$ if $min\_gain = 4.5$. Figure 3(ii)(c) shows the refined candidate lists.

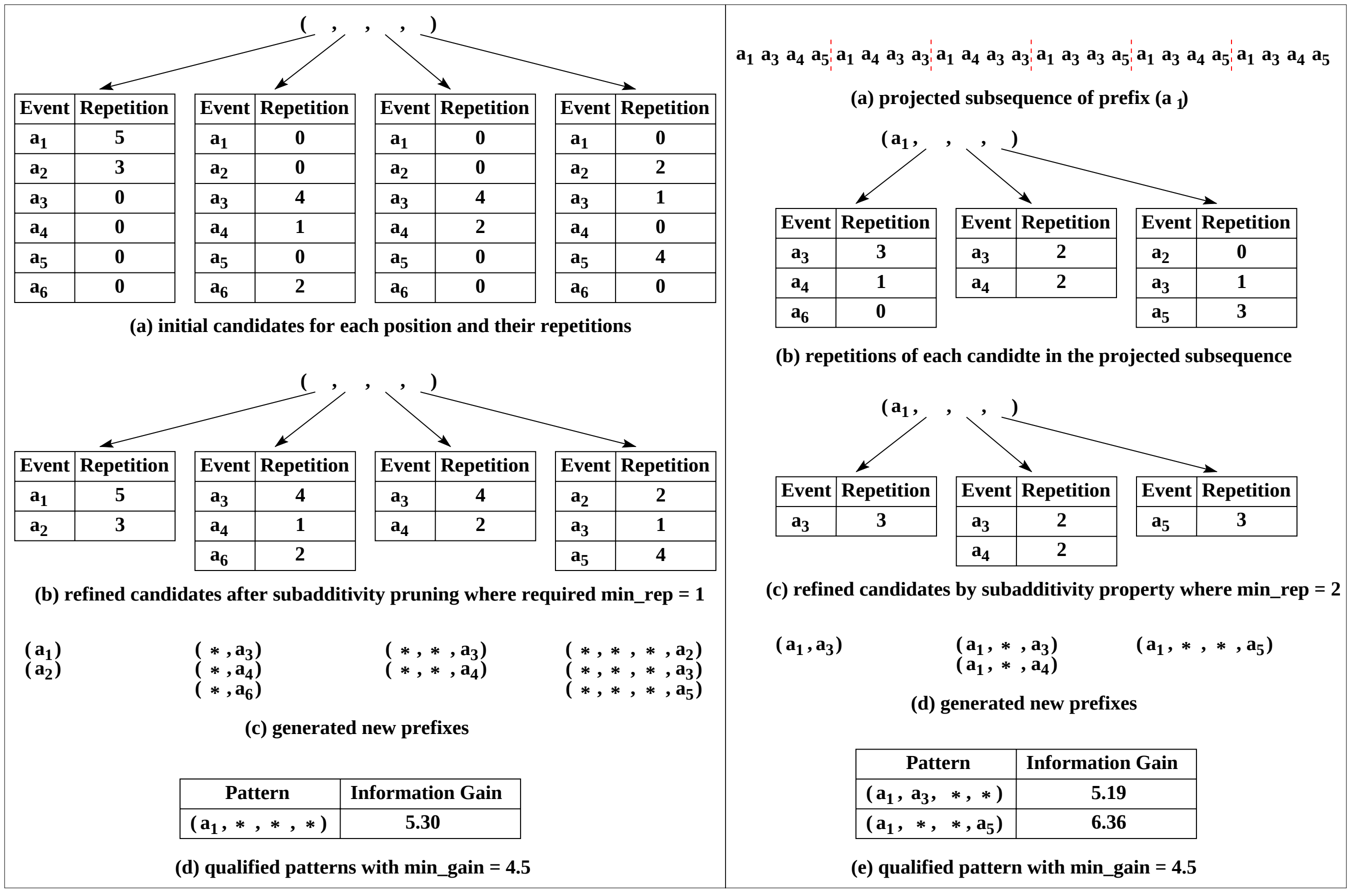
The bounded information gain pruning serves as the core of the entire algorithm in the sense that it plays a dominant role in the overall efficiency of the scheme. For each open position, only events that may generate extensible prefixes are retained. Even though the number of candidate events for each position is $O(|E|)$ theoretically, the bounded information gain pruning can substantially reduce the candidate scope in practice via successive candidate refinements. This can be easily seen through the above example and is further verified by the experimental results in Section 7.2. This unique pruning technique distinguishes our proposed method from any existing algorithm designed for mining frequent patterns.

4.4 Pattern Validation

This subroutine (Algorithm 4.4) is used to validate a candidate pattern P. The information of P can be easily obtained by aggregating the information of the event on each position. A decision is then made on whether the information gain (i.e., $repetition \times I(P)$) meets the requirement. In the case that P carries sufficient information gain, P is inserted into $Result$. In Figure 3(ii)(e), $(a_1, a_3, *, *)$ and $(a_1, *, *, a_5)$ are added into $Result$.

We note that, even if the pattern P generated from some $newprefix$ (Line 5-6) in Algorithm 4.1 does not carry sufficient information gain, the $newprefix$ may still be extended to a pattern that satisfies the information gain threshold (i.e., $newprefix$ is extensible). For example, the pattern $(a_1, *, a_4, *)$ (generated from prefix $(a_1, *, a_4)$ in Figure 3(ii)(d)) has information gain 4.44 (which is less than min_gain). However, the prefix $(a_1, *, a_4)$ can be extended to $(a_1, *, a_4, a_5)$ whose information gain is greater than min_gain. This illustrates the essential difference between the Apriori property and the bounded information gain property.

[9] A candidate pattern is a pattern that can be constructed by assigning each open position an event from the corresponding candidate list of that position.


(, , ,)
(a1, , ,)
$a_1\ a_3\ a_4\ a_5 \mid a_1\ a_4\ a_3\ a_3 \mid a_1\ a_4\ a_3\ a_3 \mid a_1\ a_3\ a_3\ a_5 \mid a_1\ a_3\ a_4\ a_5 \mid a_1\ a_3\ a_4\ a_5$

(i) The Initial Invocation of InfoMiner

Event	Repetition
a_1	5
a_2	3
a_3	0
a_4	0
a_5	0
a_6	0

Event	Repetition
a_1	0
a_2	0
a_3	4
a_4	1
a_5	0
a_6	2

Event	Repetition
a_1	0
a_2	0
a_3	4
a_4	2
a_5	0
a_6	0

Event	Repetition
a_1	0
a_2	2
a_3	1
a_4	0
a_5	4
a_6	0

(a) initial candidates for each position and their repetitions

Event	Repetition
a_1	5
a_2	3

Event	Repetition
a_3	4
a_4	1
a_6	2

Event	Repetition
a_3	4
a_4	2

Event	Repetition
a_2	2
a_3	1
a_5	4

(b) refined candidates after subadditivity pruning where required min_rep = 1

(a_1) (a_2) $(*, a_3)$ $(*, a_4)$ $(*, a_6)$ $(*, *, a_3)$ $(*, *, a_4)$ $(*, *, *, a_2)$ $(*, *, *, a_3)$ $(*, *, *, a_5)$

(c) generated new prefixes

Pattern	Information Gain
$(a_1, *, *, *)$	5.30

(d) qualified patterns with min_gain = 4.5

(ii) Level 2 Invocation of Info Miner for Prefix (a_1)

(a) projected subsequence of prefix (a_1)

Event	Repetition
a_3	3
a_4	1
a_6	0

Event	Repetition
a_3	2
a_4	2

Event	Repetition
a_2	0
a_3	1
a_5	3

(b) repetitions of each candidte in the projected subsequence

Event	Repetition
a_3	3

Event	Repetition
a_3	2
a_4	2

Event	Repetition
a_5	3

(c) refined candidates by subadditivity property where min_rep = 2

(a_1, a_3) $(a_1, *, a_3)$ $(a_1, *, a_4)$ $(a_1, *, *, a_5)$

(d) generated new prefixes

Pattern	Information Gain
$(a_1, a_3, *, *)$	5.19
$(a_1, *, *, a_5)$	6.36

(e) qualified pattern with min_gain = 4.5

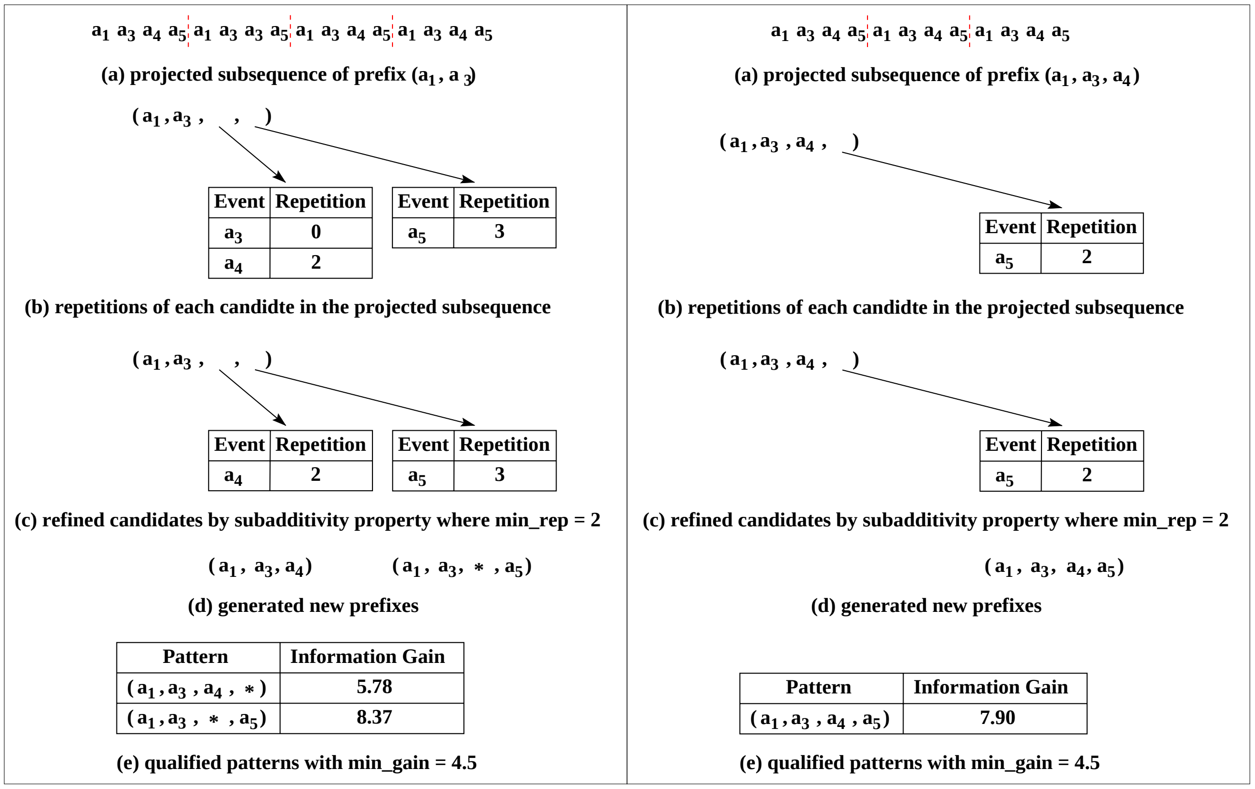
Figure 3: Initial Invocation of *InfoMiner* and Level 2 Invocation of *InfoMiner* for prefix (a_1)

4.5 Projected Subsequence Construction

This subroutine (Algorithm 4.5) is very straightforward. Given a prefix, the input sequence S is scanned and the segments are selected to form the projected subsequence. Note that all segments in this projected subsequence share a common prefix. For example, the projected subsequence for prefix (a_1) is given in Figure 3(ii)(a) and that for prefix (a_1, a_3) is given in Figure 4(i)(a).

5 Discussion: Finding the K Most Surprising Patterns

In the previous section, we presented an algorithm to find patterns whose information gain is greater than a threshold min_gain. In some applications, users may want to specify the number of most surprising patterns to be returned instead. In this section, we investigate the problem of mining the K most surprising patterns. In fact, this can be achieved with the following minor modification to the algorithm presented in the previous section. The set $Result$ is maintained throughout the entire process to hold the set of K most surprising patterns discovered so far, and is initialized to be null. The threshold min_gain is always equal to the minimal information gain carried by any pattern in $Result$ and is set to zero at the beginning. Each time after discovering a new pattern P whose information gain $G(P)$ is above the current threshold min_gain, P is added to $Result$ and, in the case where $Result$ already contains K patterns, the pattern with the smallest information gain in $Result$ will be replaced. The value of min_gain is then adjusted to the minimal information gain of


a1 a3 a4 a5 a1 a3 a3 a5 a1 a3 a4 a5 a1 a3 a4 a5
(a) projected subsequence of prefix (a1 , a 3)
(a1 , a3 , ,)
(b) repetitions of each candidte in the projected subsequence
(a1 , a3 , ,)
(c) refined candidates by subadditivity property where min_rep = 2
(a1 , a3, a4)
(a1 , a3, * , a5)
(d) generated new prefixes
(e) qualified patterns with min_gain = 4.5
a1 a3 a4 a5 a1 a3 a4 a5 a1 a3 a4 a5
(a) projected subsequence of prefix (a1 , a3 , a4)
(a1 , a3 , a4 ,)
(b) repetitions of each candidte in the projected subsequence
(a1 , a3 , a4 ,)
(c) refined candidates by subadditivity property where min_rep = 2
(a1 , a3, a4, a5)
(d) generated new prefixes
(e) qualified patterns with min_gain = 4.5

Event	Repetition
a_3	0
a_4	2

Event	Repetition
a_5	3

Event	Repetition
a_4	2

Event	Repetition
a_5	3

Pattern	Information Gain
$(a_1, a_3, a_4, *)$	5.78
$(a_1, a_3, *, a_5)$	8.37

Event	Repetition
a_5	2

Event	Repetition
a_5	2

Pattern	Information Gain
(a_1, a_3, a_4, a_5)	7.90

(i) Level 3 Invocation of InfoMiner for Prefix (a_1, a_3) **(ii) Level 4 Invocation of InfoMiner for Prefix (a_1, a_3, a_4)**

Figure 4: Level 3 and Level 4 Invocations of *InfoMiner* for prefixes (a_1, a_3) and (a_1, a_3, a_4) respectively

any pattern in $Result$. Clearly, with the algorithm proceeds, min_gain increases, thus, min_rep (used in the algorithm) also increases, and less candidates remain. At the end, the set $Result$ contains the K most surprising patterns (i.e., highest information gain).

In Section 7.2.3, we show the performance of finding the K most surprising patterns with comparison to finding these K patterns when the appropriate information gain threshold is given. When K is relatively small (e.g., 10), finding the K most surprising patterns may take twice as much time as finding patterns with given information gain threshold. On the other hand, when K is large (e.g., 1000), the disadvantage diminishes (e.g., less than 5%). We will discuss it in more detail in Section 7.2.3. Since the absolute response time of finding a small number of patterns that meet a very high information gain threshold is relatively short, the prolonged response time is still tolerable in most cases.

6 Optimization Issue

In this section, we explore some further optimization technique that may be used to improve the performance of *InfoMiner*. It is obvious that, as the algorithm proceeds from patterns with single filled position to more complex ones, we may examine open positions in different orders. Certainly, the imposed order would affect the performance of *InfoMiner* in the sense that, at a given level of the recursion, both the length of projected subsequence and the number of candidate events for an open position would be different. We now explore this issue in detail and propose some heuristic on how to choose an optimal order.

Given a pattern prefix p, let S_p be the projected subsequence of p. In order to generate qualified patterns with prefix p, *InfoMiner* scans S_p to collect the repetition of each candidate event at each open position and simultaneously compute the projected subsequences of some newly constructed prefixes. Therefore, the length of S_p is an important factor towards the performance of *InfoMiner*. In general, the length of a projected subsequence is a function of the length of the corresponding prefix, which decreases as the prefix includes more filled positions. This can be easily observed by examining the projected subsequences of (a_1), (a_1, a_3), and (a_1, a_3, a_4) in Figure 3(ii)(a), 4(i)(a), and 4(ii)(a), respectively. It would be desirable if we can minimize the length of the projected subsequence for any given prefix length. To achieve this goal, we propose a heuristic, namely *shortest projection first* (SPF), to determine the order to examine the open positions. The basic idea is that, the length of a projected subsequence that is going to be constructed for a newly generated prefix (in an invocation of *InfoMiner*) can be estimated from the repetition of the corresponding candidate event. For instance, the repetition of a_1 at the first position is 5 in Figure 3(i) and the length of the projected subsequence for (a_1) is going to contain $(5+1) \times 4 = 24$ events. Let $o_1, o_2, \ldots, o_l$ be the set of open positions and $length_i$ be the total length of projected subsequences for the prefix generated by filling position o_i with some candidate event. Then, all open positions can be ordered in ascending order of $length_i$. Let's consider the example in Figure 3(i). If the first position is specified, the total length of all projected subsequences is $(5+1) \times 4 + (3+1) \times 4 = 40$. On the other hand, if we consider the third position first, then the total length of all resulting projected subsequences is $(4+1) \times 4 + (2+1) \times 4 = 32$. Thus the third position should be examined before the first position according to this order. As a result, the total length of projected subsequences is minimized. This step can be performed after the $Bounded\_Information\_Pruning()$ to provide an order among all open positions. We will show in Section 7.2.2 that this optimization can produce as much as 40% reduction towards the length of projected subsequences.

7 Experimental Results

We implemented InfoMiner in C programming language on an IBM RS-6000 (300 MHz CPU) with 128MB running AIX operating system. To analyze the benefits of the information model and the performance of the InfoMiner algorithm, we employ two real traces and four synthetic traces.

7.1 IBM Intranet and Scour Traces

The IBM Intranet traces consist of 160 critical nodes, e.g., file servers, routers, etc., in the IBM T. J. Watson Intranet. Each node issues a message in response of certain situation, e.g., CPU saturation, router interface down, etc. There are total 20 types of messages. We treat a certain message from a particular node as a distinct event, thus there are total 500 distinct events in the trace because a certain type of node may only send out 4 or 5 types of messages. The IBM Intranet trace consists of 10,000 occurrences of the events. With the support threshold, the 100 most frequent patterns all contain the "I am alive" message which is a control message that a network node sends out periodically, e.g., every minute. Despite its high frequency, it is not particularly useful. On the other hand, by applying InfoMiner on this trace, we found some surprising patterns that are also interesting. For example, the pattern $(node_a\_fail, *, node_b\_saturated, *)$ has the eighth highest information gain. This pattern means that, shortly after a router ($node_a$) fails, the CPU on another node ($node_b$) is saturated. Under a thorough investigation, we found that $node_b$ is a file server and after $node_a$ fails, all requests to some files are sent to $node_b$, which cause the bottleneck. This particular pattern is not even ranked within the top 1000 most frequent patterns under the support model.

Scour is a web search engine that is specialized for multimedia contents. The web URL of the Scour search engine is "http://www.scour.net". The scour servers consist of a cluster of machines. Every five minutes, a distributed sensor records the CPU utilization of every node and the average is taken as the global CPU utilization level. We discretize the global CPU utilization level into events: event A stands for the utilization level between 0 and 0.05, B stands for the utilization level between 0.05 and 0.1, and so on. The scour trace consists of 28800 occurrences of 19 events (100 days). The event corresponding to the utilization between 0.95 and 1 does not occur in the trace.

Since the utilization level of a node is typically around 0.3, most of the frequent patterns consists of symbol F or G, such as $(F, G, *)$ or $(G, *, F)$, and so on. However, these patterns are not interesting nor useful. On the other hand, with the information gain measure, we are able to find some interesting and useful patterns, such as a pattern that S (utilization [0.9, 0.95]) followed by Q (utilization [0.8, 0.85]), which, in turn, is followed by P (utilization [0.75, 0.8]). This pattern means that if the nodes are saturated and the response time increases significantly, then many users may stop access the site shortly after, thus the workload decreases immediately. Although the support of this pattern is low, the information of this pattern is quite high (ranked as the 8th highest) due to the rare occurrences of S and Q. In the support model, this pattern is ranked beyond 10,000th. This means that we have to examine more than 10,000 other patterns before discovering this pattern in the support model. (The complete comparison of the number of patterns found in the information model and the support model for both IBM trace and Scour trace are shown in Table 1 in Section 1.)

Table 3 shows the number of discovered patterns and the response time under the information model and support model in the scour trace. In this table, the minimum support threshold (min_sup) is set based on the min_gain value so that the patterns found in the support model would include all patterns found in the information model. In addition to the set of patterns returned by the information model, the support model also discovers a huge number of non-surprising patterns. This not only creates an encumbrance to the user but also leads to a slower response time. It is easy to see that the response time of the support model is an order of magnitude slower than that of the information model. In addition, we find that the bounded information gain pruning is very effective in the sense that more than 95% of the patterns are pruned in this experiment.

Table 3: Number of Discovered Patterns and Response Time for the Scour Trace

Information Model			Support Model		
min_gain	Num. of Patterns	Response Time	min_sup	Num. of Patterns	Response Time
$-\log_{19} 0.00001 = 3.91$	8	1.5 min	0.000212	16,953	25.2 min
$-\log_{19} 0.00003 = 3.54$	21	2.7 min	0.000208	17,021	27.1 min
$-\log_{19} 0.0001 = 3.13$	53	4.2 min	0.000201	17,605	28.7 min

To further compare the surprising patterns and frequent patterns, we train two classifiers, one based on top 1000 frequent patterns and the other based on top 1000 surprising patterns, to predict whether the system operates normally at any given time. Each pattern is treated as a feature. The true status of the system (normal versus abnormal) is given by the administrator. The support vector machine (SVM) is adopted as the classification model. We found that SVM trained on frequent patterns yields 34.8% adjusted accuracy[10] whereas the classifier build upon surprising pattern achieve an adjusted accuracy of 77.3%. This further demonstrates the superiority of surprising patterns.

7.2 Synthetic Sequences

To analyze the performance of the InfoMiner Algorithm, four sequences are synthetically generated. Each sequence consists of 1024 distinct events and 20M occurrences of events. The synthetic sequence is generated as follows. First, at the beginning of the sequence, the period length l of the next pattern is determined, which is geometrical distributed with mean μ_l. The number of events in a pattern is randomly chosen between 1 and l. The number of repetitions m of this pattern is geometrical distributed with mean μ_m. The events in the pattern are chosen according to a normal distribution. (This means

[10]The adjusted accuracy is often used when the class distribution is very skewed. Given two classes $normal$ and $abnormal$, the adjusted accuracy is defined as $\frac{\frac{Pred(normal)\cap Act(normal)}{Act(normal)} + \frac{Pred(abnormal)\cap Act(abnormal)}{Act(abnormal)}}{2}$ where $Pred(X)$ is the percentage of time the classifier predict X and $Act(X)$ is the percentage of time the class is X.

that some events occurs more frequently than other.) However, the pattern may not perfectly repeat itself for m times. To simulate the imperfectness of the subsequence, we employ a parameter δ to control the noise. δ is uniformly distributed between 0.5 and 1. With probability δ, the next l events match the pattern. Otherwise, the next l events do not support the pattern. The replacement events are chosen from the event set with the same normal distribution. This subsequence ends when there are m matches, and a new subsequence for a new pattern starts. This process repeats until it reaches the end of the sequence. Four sequences are generated based on values of μ_l and μ_m in Table 4.

Table 4: Parameters of Synthetic Data Sets

Data Set	μ_l	μ_m	Distinct events	Total Events
$l3m20$	3	20	1024	20M
$l100m20$	100	20	1024	20M
$l3m1000$	3	1000	1024	20M
$l100m1000$	100	1000	1024	20M

7.2.1 Standard InfoMiner

In this subsection, we analyze the InfoMiner algorithm without the optimization technique presented in Section 6. The main pruning power of the InfoMiner algorithm is provided by the bounded information gain pruning technique. In our InfoMiner algorithm, for each prefix, we prune the candidate events on each remaining open position. Although the number of candidate events on each open position can be $|E|$ theoretically, in practice the average number of candidates in each open position decreases dramatically with the increase of the number of specified positions (i.e., the length of the prefix). This is due to the fact that the value of max_info is estimated from the candidate event with the highest information on each open position. Thus, with more positions specified, the max_info value decreases. In turn, the min_rep threshold ($min\_rep = \lceil \frac{min\_gain}{max\_info} \rceil$) increases and more candidate events are pruned. We conduct experiments with our InfoMiner algorithm on the four synthetic sequences. Figure 5(a) shows the average number of remaining candidate events on each open position as a function of the number of specified positions (i.e., the length of the prefix). The number of candidate events decreases dramatically with the number of specified positions. With data set $l3m20$ and $l3m1000$, since the average pattern length is 3, there is no candidate after 5 or 6 positions are specified. In addition, with all four data sets, when the number of specified positions is greater than 3, the average number of events on each open position is very small (i.e., less than 0.4). This leads to the overall efficiency of the InfoMiner algorithm.

The overall response time of the InfoMiner algorithm largely depends on the min_gain threshold. We ran several tests on the above data sets with different min_gain thresholds. Figure 5(b) shows the response time for each data set. Our bounded information gain pruning can reduce a large number of patterns. More than 99% of the patterns are pruned. When the min_gain threshold increases, the pruning effect becomes more dominant because more patterns can be eliminated by the bounded information gain pruning. Thus, the response time improves with increasing min_gain threshold on all four data sets. (Note that the Y-axis is in log scale in Figure 5(a) and (b).) To explore the scalability of the InfoMiner algorithm, we also experiment with event sequences of different lengths, from 1 million to 100 million. We found that the response time of InfoMiner is linear with respect to both the length of the event sequence and the period length.

7.2.2 Optimization Technique

In this subsection, we analyze the improvement rendered by the optimization techniques presented in Section 6. The shortest projection first (SPF) technique reduces the average length of a projected subsequence. Figure 6(a) shows the

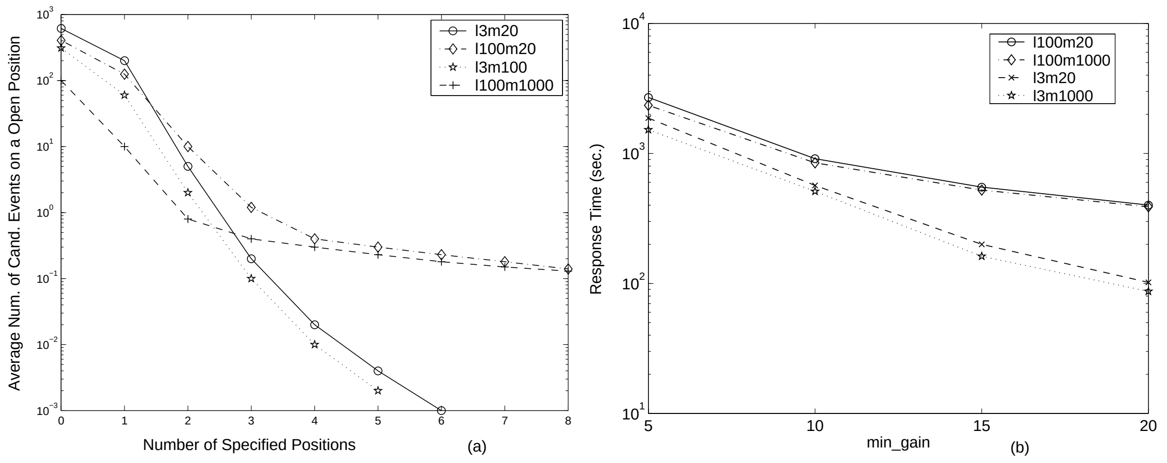


Figure 5: Pruning Effects and Response Time

improvement provided by the SPF technique on the synthetic sequences. Since all four synthetic sequences show a similar rate of improvement, we use the $l100m20$ sequence as the representative in Figure 6(a). The improvement is measured as $\frac{L-L_{spf}}{L}$ where L and L_{spf} are the average length of a projected subsequence with the standard InfoMiner and with the SPF technique, respectively. It is evident that the most (accumulative) improvement is achieved when the number of specified positions is 3. This is due to the fact that when the projected subsequence is substantially long and the candidate event list of each open position is comparatively large, there exists a relatively large room for further improvement, while such potential diminishes as the number of specified positions in the prefix increases. Figure 6(b) illustrates the response time of the InfoMiner algorithm with and without the SPF heuristic. It is obvious that the SPF technique can achieve 20% improvement on the overall response time of our InfoMiner algorithm.

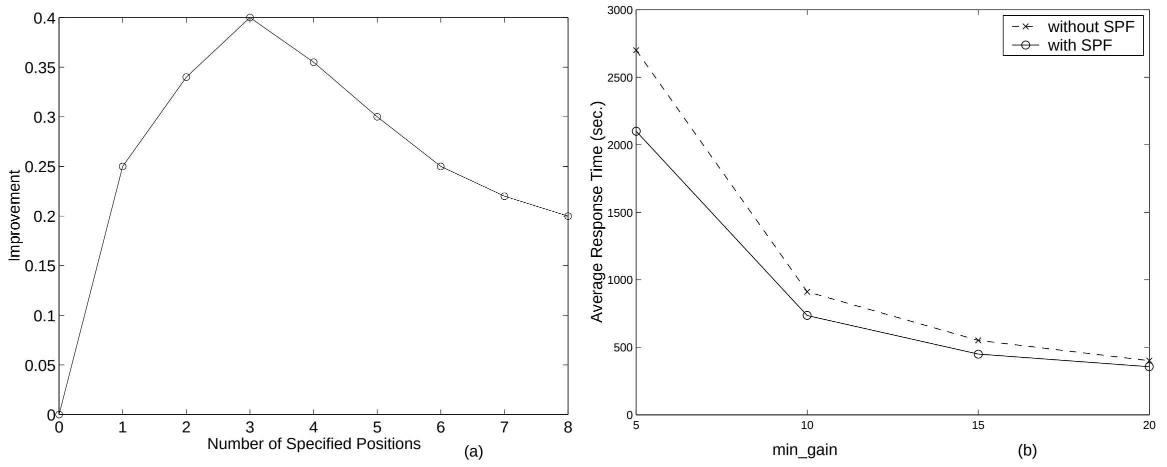


Figure 6: Improvements of the SPF heuristic

7.2.3 The K Most Surprising Patterns

Let $info\_gain(K)$ be the Kth highest information gain among all patterns. More specifically, $info\_gain(K)$ is a monotonic decreasing function of K. Figure 7(a) shows the ratio of the response time of finding the K most surprising patterns (without knowing the value of $info\_gain(K)$ in advance) and finding patterns above the given threshold $info\_gain(K)$. Obviously, these two settings would produce exactly the same result. The x-axis shows the number of desired patterns, K.

The higher the value of K, the less the overhead. (The overhead is less than 5% when $K = 1000$.) When K is small, finding the K most surprising patterns is about twice as long as that of find these K patterns when $info\_gain(K)$ is given. This is due to the fact that the threshold $info\_gain(K)$ is relatively high when K is small. Consequently, the overhead is higher since the mining process takes a relatively longer time to raise min_gain (from zero) to the proper threshold $info\_gain(K)$.

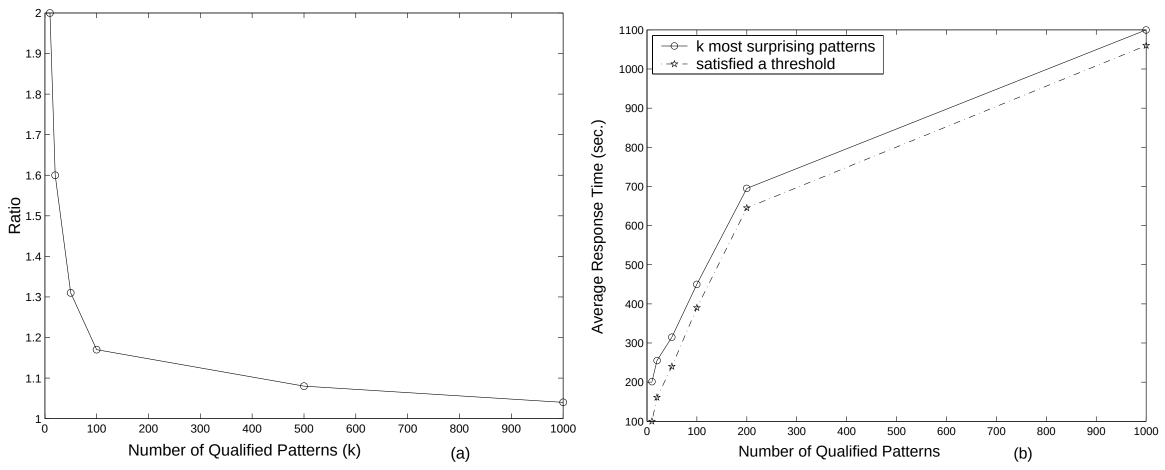


Figure 7: Performance of mining the K most surprising patterns

Figure 7 (b) illustrates the absolute response time of these two settings. The smaller the value of K, the shorter the overall response time. This is largely due to the effect of rapid adjustment of the threshold min_gain towards the value of $info\_gain(K)$ so that a large amount of patterns can be quickly pruned at early stage of the mining process.

8 Conclusions

In this paper, we study the problem of surprising periodic pattern discovery in a data sequence that consists of events with vastly different occurrence frequencies. Our goal is not to find the patterns that occur often, but rather to discover patterns that are surprising. Instead of using the support metric, we propose a new metric: information gain. Although the new metric does not possess the Apriori property as the support metric does, we identify the bounded information gain property in the information model. Based on the bounded information gain property, we devise a recursive algorithm for mining surprising periodic patterns by only exploring extensible prefixes. Furthermore, the user has a choice of either specifying a minimum information gain threshold or specifying the number of most surprising patterns wanted. Last but not least, the information model can be extended to define both surprising itemsets from transaction database and surprising sequential patterns from sequence database.

References

[1] R. Agarwal, C. Aggarwal, and V. Prasad. Depth first generation of long patterns. *Proc. 6th Intern. Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, 108-118, 2000.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *Proc. 20th Int. Conf. on Very Large Data Bases*, 487-499, 1994.

[3] R. Agrawal and R. Srikant. Mining Sequential Patterns. *Proc. Int. Conf. on Data Engineering (ICDE)*, Taipei, Taiwan, 3-14, March 1995.

[4] G. Berger and A. Tuzhilin. Discovering unexpected patterns in temporal data using temporal logic. *Temporal Databases - Research and Practice, Lecture Notes on Computer Sciences*, (1399) 281-309, 1998.

[5] D. Berndt and J. Clifford. Finding patterns in time series: a dynamic programming approach. *Advances in Knowledge Discovery and Data Mining*, 229-248, 1996.

[6] C. Bettini, X. S. Wang, S. Jajodia, and Jia-Ling Lin. Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Transaction on Knowledge and Data Engineering*, 10(2), 222-237, 1998.

[7] R. Blahut. *Principles and Practice of Information Theory*, Addison-Wesley Publishing Company, 1987.

[8] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic Itemset counting and implication rules for market basket data. *Proc. ACM SIGMOD Conf. on Management of Data*, 255-264, 1997.

[9] S. Brin, R. Motwani, C. Silverstein. Beyond market baskets: generalizing association rules to correlations. *Proc. ACM SIGMOD Conf. on Management of Data*, 265-276, 1997.

[10] A. Califano, G. Stolovitzky, and Y. Tu. Analysis of gene expression microarrays: a combinatorial multivariate approach, *IBM T. J. Watson Research Report*, 1999.

[11] S. Chakrabarti, S. Sarawagi, and B. Dom. Mining surprising patterns using temporal description length. *Proc. Int. Conf. on Very Large Data Bases*, 606-617, 1998.

[12] E. Cohen, M. Datar, S. Fuijiwara, A. Cionis, P. Indyk, R. Motwani, J. Ullman, and C. Yang. Finding interesting associations without support pruning. *Proc. 16th Int. Conf. on Data Engineering (ICDE)*, 489-499, 2000.

[13] G. Das, D. Gunopulos, and H. Mannila. Finding similar time series. *Proc. European Conf. on Principles of Data Mining and Knowledge Discovery*, 88-100, 1997.

[14] G. Das, K.-I. Lin, H. Mannila, Gopal Renganathan, and Padhraic Smyth. Rule discovery from time series. *Proc. Int. Conf. on Knowledge Discovery and Datamining*, 16-22, 1998.

[15] R. Feldman, Y. Aumann, A. Amir, and H. Mannila. Efficient algorithms for discovering frequent sets in incremental databases. *Proc. ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, 59-66, 1997.

[16] S. Fujiwara, J. Ullman, and R. Motwani. Dynamic miss-counting algorithms: finding implication and similarity rules with confidence pruning. *Proc. 16th Int. Conf. on Data Engineering (ICDE)*, 501-511, 2000.

[17] M. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: sequential pattern mining with regular expression constraints. *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 223-234, 1999.

[18] X. Ge and P. Smyth. Deformable Markov model templates for time-series pattern matching. *Proc. ACM SIGKDD*, 81-90, 2000.

[19] G. Gunopulos, H. Mannila, and S. Saluja. Discovering all most specific sentences by randomized algorithms. *Proc. 6th Int. Conf. on Database Theory*, 215-229, 1997.

[20] V. Guralnik, D. Wijesekera, and J. Srivastava. Pattern directed mining of sequence data. *Proc. ACM SIGKDD*, 51-57, 1998.

[21] V. Guralnik and J. Srivastava. Event detection from time series data. *Proc. ACM SIGKDD*, 33-42, 1999.

[22] J. Han, W. Gong, and Y. Yin. Mining segment-wise periodic patterns in time-related databases. *Proc. Int. Conf. on Knowledge Discovery and Data Mining*, 214-218, 1998.

[23] J. Han, G. Dong, and Y. Yin. Efficient mining partial periodic patterns in time series database. *Proc. Int. Conf. on Data Engineering*, 106-115, 1999.

[24] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD)*, 1-12, 2000.

[25] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu. FreeSpan: frequent pattern-projected sequential pattern mining. *Proc. Int. Conf. on Knowledge Discovery and Data Mining*, 2000.

[26] Y. Huhtala, J. Krkkinen, P. Porkka, and H. Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2) 100-111, 1999.

[27] E. J. Keogh and P. Smyth. A probabilistic approach to fast pattern matching in time series databases. *Proc. Int. Conf. on Knowledge Discovery and Datamining*, 24-30, 1997.

[28] M. Klemetinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. Verkamo. Finding interesting rules from large sets of discovered association rules. *Proc. CIKM*, 1994.

[29] B. Liu, W. Hsu, and Y. Ma. Mining association Rules with multiple minimum supports. *Proc. ACM SIGKDD*, 337-341, 1999.

[30] B. Liu, W. Hsu, and Y. Ma. Pruning and summarizing discovered associations. *Proc. ACM SIGKDD*, 125-134, 1999.

[31] B. Liu, M. Hu, and W. Hsu. Multi-level organization and summarization of the discovered rules. *Proc. ACM SIGKDD*, 208-217, 2000.

[32] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, vol. 1, no. 3, 259-289, 1997.

[33] H. Mannila, D. Pavlov, and P. Smyth. Prediction with local patterns using cross-entropy. *Proc. ACM SIGKDD*, 357-361, 1999.

[34] H. Mannila and C. Meek. Global partial orders from sequential data. *Proc. ACM SIGKDD*, 161-168, 2000.

[35] T. Oates. Identifying distinctive subsequences in multivariate time series by clustering. *Proc. ACM SIGKDD*, 322-326, 1999.

[36] T. Oates, M. D. Schmill, P. R. Cohen. Efficient mining of statistical dependencies. *Proc. 16th Int. Joint Conf. on Artificial Intelligence*, 794-799, 1999.

[37] B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. *Proc. 14th Int. Conf. on Data Engineering*, 412-421, 1998.

[38] B. Padmanabhan and A. Tuzhilin. Pattern discovery in temporal databases: a temporal logic approach. *Proc. ACM KDD*, 351-354, 1996.

[39] B. Padmanabhan and A. Tuzhilin. A belief-driven method for discovering unexpected patterns. *Proc. ACM KDD*, 94-100, 1998.

[40] B. Padmanabhan and A. Tuzhilin. Small is beautiful: discovering the minimal set of unexpected patterns. *Proc. ACM KDD*, 54-63, 2000.

[41] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. *Proc. Int. Conf. on Database Theory*, 398-416, 1999.

[42] J. Pei, J. Han, and R. Mao. CLOSET: an efficient algorithm for mining frequent closed itemsets. *Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 21-30, 2000.

[43] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. PrefixSpan: mining sequential patterns by prefix-projected growth. *Proc. IEEE Conf. Data Engineering*, 215-224, 2001.

[44] G. Piateski-Shapiro and C. Matheus. The interestingness of deviations. *Proc. AAAI Workshop Knowledge Discovery in Databases*, 25-36, 1994.

[45] Y. Qu, C. Wang, and X. S. Wang. Supporting fast search in time series for movement patterns in multiple scales. *Proc. 7th ACM Int. Conf. on Information and Knowledge Management*, 251-258, 1998.

[46] D. Rafiei. On similarity-based queries for time series data. *Proc. 15th Int. Conf. on Data Engineering*, 410-417, 1999.

[47] S. Ramaswamy, S. Mahajan, and A. Silberschatz. On the discovery of interesting patterns in association rules. *Proc. 24th Intl. Conf. on Very Large Data Bases (VLDB)*, 368-379, 1998.

[48] S. Sahar. Interestingness via what is not interesting. *Proc. 5th ACM Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 332-336, 1999.

[49] D. Shah, L. Lakshmanan, K. Ramamritham, and S. Sudarshan. Interestingness and pruning of mined patterns. *Proc. ACM SIGMOD Workshop on Research Issues in Datamining and Knowledge Discovery*, 1999.

[50] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discover systems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* vol. 8 no. 6, pp. 970-974, 1996.

[51] Myra Spiliopoulou. Managing interesting rules in sequence mining. *Proc. European Conf. on Principles and Practice of Knowledge Discovery in Databases*, 554-560, 1999.

[52] R. Srikant and R. Agrawal. Mining sequential patterns: generalizations and performance improvements. *Proc. 5th Int. Conf. on Extending Database Technology (EDBT)*, 3-17, 1996.

[53] S. Thomas and S. Sarawagi. Mining generalized association rules and sequential patterns using SQL queries. *Prof. of 4th Intl. Conf. on Knowledge Discovery and Data Mining (KDD98)*, pp. 344-348.

[54] J. Wang, G. Chirn, T. Marr, B. Shapiro, D. Shasha, and K. Zhang. Combinatorial pattern discovery for scientific data: some preliminary results. *Proc. ACM SIGMOD Conf. on Management of Data*, 115-125, 1994.

[55] K. Wang, Y. He, and J. Han. Mining frequent itemsets using support constraints. *Proc. Int. Conf. on on Very Large Data Bases*, 2000.

[56] J. Yang, W. Wang, and P. Yu. Mining asynchronous periodic patterns in time series data. *Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 275-279, 2000.

[57] B. Yi, H.V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. *Proc. Int. Conf. on Data Engineering*,201-208, 1998.

[58] M. Zaki. Sequence mining in categorical domains: incorporating constraints. *Proc. 9th Int. Conf. on Information and Knowledge Management*, 422-429, 2000.

[59] M. J. Zaki. Generating non-redundant association rules. *Proc. ACM SIGKDD*, 34-43, 2000.

[60] M. Zaki. SPADE: an efficiant algorithm for mining frequent sequences. *Machine Learning Journal, special issue on Unsupervised Learning*, 42(1/2), 31-60, 2001.