RC 22080 (99009) 1 June 2001 Computer Science

# IBM Research Report

## Mining Long Sequential Patterns in a Noisy Environment

**Jiong Yang, Wei Wang, Philip Yu**
IBM Research Division
Thomas J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598

**Jiawei Han**
Simon Fraser University

**Research Division**
**Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich**

# Mining Long Sequential Patterns in a Noisy Environment

Jiong Yang, Wei Wang, Philip S. Yu
IBM T. J. Watson Research Center
{jiyang, ww1, psyu}@us.ibm.com

Jiawei Han
Simon Fraser University
han@cs.sfu.ca

**Abstract**

Pattern discovery in long sequences is of great importance in many applications including computational biology study, consumer behavior analysis, system performance analysis, etc. In a noisy environment, an observed sequence may not accurately reflect the underlying behavior. For example, the amino acid N in human body is likely to mutate to D with little impact to the biological function of the protein. It would be desirable if the occurrence of D in the observation can be related to a possible mutation from N in an appropriate manner. Unfortunately, the support measure (i.e., the number of occurrences) of a pattern does not serve this purpose. In this paper, we introduce the concept of *compatibility matrix* as the means to provide a probabilistic connection from the observation to the underlying true value. A new metric *match* is also proposed to capture the "real support" of a pattern which would be expected if a noise-free environment is assumed. In addition, in the context we address, a pattern could be very long. The standard pruning technique developed for the market basket problem may not work efficiently. As a result, a novel algorithm that combines statistical sampling and a new technique (namely *border collapsing*) is devised to discover long patterns in a minimal number of scans of the sequence database with sufficiently high confidence. Empirical results demonstrate the robustness of the match model (with respect to the noise) and the efficiency of the probabilistic algorithm.

# 1 Introduction

Pattern discovery in long sequences is of great importance in many applications. As an important metric, the *support* [4, 11, 15, 21, 24] (or some derivation of support) is widely used to qualify significant patterns. Due to the presence of noise, a symbol may be misrepresented by some other symbols. This substitution may prevent an occurrence of a pattern from being recognized and in turn slashes the support of that pattern. As a result, a frequent pattern may be "concealed" by the noise. This phenomenon commonly exists in many applications.

- *Bio-Medical Study*. Mutation of amino acids is a common phenomenon studied in the context of biology. Some mutations are proved to occur with a non-negligible probability under normal circumstances and incur little change to its biological functionalities. For example, the amino acid N in human body is likely to mutate to D with little impact to the behavior [10]. In this sense, they should not be considered as totally independent individuals.
- *Performance Analysis*. Many system-monitoring applications involve collecting and analyzing attributes that take continuous numerical values. A common approach to process them is to quantize the domain into categories. If the true value of an attribute is close to the boundary of the quantization, there is a fair chance that the observed value may fall into the adjacent bin and be represented by a different label. It would be desirable if such kind of distortion can be taken into account during the data mining process.
- *Consumer Behavior*. It happens frequently that a customer may end up buying a slightly different merchant from what he (she) originally wanted due to various reasons, such as the desired one was out of stock or misplaced. Allowing obscurity in item matching may conduce to unveil the customer's real purchase intention.

This problem becomes critical when the pattern is long because a long pattern is much more vulnerable to distortion caused by noise. Our experiments in Section 5 show that, even with a moderate degree of noise, a frequent long pattern may have as much as $60\%$ chance to be labeled as an infrequent pattern. Let's take the gene sequence analysis as an example. The length of a gene expression can range up to a few hundreds if amino acids are taken as the granularity of the analysis. Figure 1(a) shows a fragment of gene expression that is found in campylobacter jejuni genome [12]. Some clinical studies show that, the amino acids N, K, and V are relatively more likely to mutate to amino acids D, R, and I, respectively. The corresponding gene expressions after the mutation are shown in Figure 1(b), (c), and (d) respectively. Even though all of these mutated gene expressions somewhat differ from the standard one in Figure 1(a), it is more equitable to treat them as possible (degraded) occurrences of the standard expression than to consider them as totally independent gene expressions.

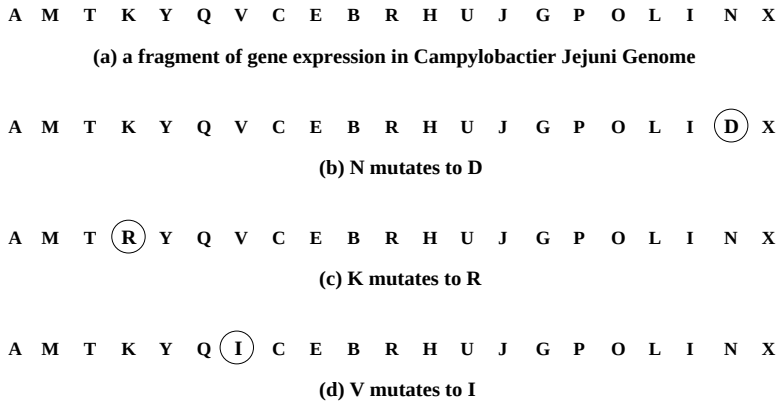



Figure 1: An example of gene expression

In order to accommodate the above circumstance, it is necessary to allow some flexibility in pattern matching. Unfortunately, most previously proposed models [4, 11, 15, 21, 24] for sequential patterns only take into account exact match of

the pattern in data[1]. In this paper, we present a more flexible model that allows obscurity in pattern matching. A so-called *compatibility matrix* is introduced to enable a clear representation of the likelihood of symbol substitution. Each entry in the matrix corresponds to a pair of symbols $(x, y)$ and specifies the conditional probability that $x$ is the true value given $y$ is observed. Figure 2 gives an example of compatibility matrix. The compatibility matrix essentially creates a natural bridge between the observation and the underlying substance. Each observed symbol is then interpreted as an occurrence of a set of symbols with various probabilities. For example, an observed $d_1$ corresponds to a true occurrence of $d_1$, $d_2$, and $d_3$ with probability 0.9, 0.05, and 0.05, respectively. Similarly, an observed symbol combination is treated as an occurrence of a set of patterns with various degrees. A new metric, namely *match*, is then proposed to quantify the significance of a pattern and is defined as the "aggregated amount of occurrences" of a pattern in the sequence database. The match of a pattern indeed represents the "real support" that were expected if no noise presents.

| true value \ observed value | d1 | d2 | d3 | d4 | d5 |
|---|---|---|---|---|---|
| **d1** | **0.9** | **0.1** | **0** | **0** | **0** |
| **d2** | **0.05** | **0.8** | **0.05** | **0.1** | **0** |
| **d3** | **0.05** | **0** | **0.7** | **0.15** | **0.1** |
| **d4** | **0** | **0.1** | **0.1** | **0.75** | **0.05** |
| **d5** | **0** | **0** | **0.15** | **0** | **0.85** |

Figure 2: An example of compatibility matrix

The well-known Apriori property also holds on the match measure, which states that *any superpattern*[2] *of an infrequent pattern is also infrequent and any subpattern of a frequent pattern is also frequent*. This guarantees that any previous algorithm designed for the support model [4, 11, 15, 21, 24] can be generalized to suit the match model, though it may not necessarily be efficient. Compared to the support model, a much larger number of patterns may possess some positive matches. In addition, the length of a pattern can be considerably long in the context we address, e.g., gene expression analysis. The combined effect of these two factors may force any direct generalization of existing algorithms (even including those designed for long patterns [1, 6]) to scan the entire sequence database many times. To tackle this problem, we propose a novel sampling-based algorithm that utilizes the Chernoff Bound [9, 16, 23, 25] to estimate the set of patterns whose matches in the sample are very close to the threshold so that there is no sufficient statistical confidence to tell whether the pattern would be frequent or not in the entire sequence database. These ambiguous patterns are then investigated against the entire sequence database to finalize the set of frequent patterns. Because the sample size is usually limited by the memory capacity and the distribution-independent nature of Chernoff Bound provides a very conservative estimation, the number of ambiguous patterns is usually very large. Consequently, significant amount of computation needs to be consumed in order to verify these ambiguous patterns in a level-wise manner. We observed that, for the protein sequence database, majority of the time would be spent in this verification step when the discovered pattern contains dozens of symbols. To expedite the process, we proposed a so called *border collapsing* technique to conduct the examination of these ambiguous patterns. While the super-pattern/sub-pattern relationship forms a lattice among all patterns, the set of ambiguous patterns "occupies" a contiguous portion of the lattice according to the Apriori property. Therefore, starting from the lower border and the upper border embracing these ambiguous patterns, the border of frequent patterns (in the entire sequence database) is located efficiently by successively collapsing the gap between these two borders until no ambiguous pattern exists. To maximize the extent of each gap collapsing operation, only the set of ambiguous patterns with the highest collapsing power are identified and probed. As a result, the expected number of scans

[1] Another approach to tackle the problem is to perform sub-space clustering on the sequence database and then derive patterns from each discovered clusters. However, since each sequence is relatively long (e.g., containing thousands of symbols), it is not clear how the clustering technique can be applied efficiently and accurately for such a high dimensional space.

[2] We will define shortly that a pattern $P$ is a superpattern of $P'$ if $P'$ can be obtained by replacing some position(s) in $P$ with the "don't care" symbol *. In such a case, $P'$ is also called a subpattern of $P$.

through the entire database is minimized.

There is a clear distinction between our algorithm and existing algorithms [23, 25] that also use sampling technique to mine frequent patterns. In the previous proposed approaches, the frequent patterns calculated from the sample is usually taken as the starting position of a level-wise search conducted in the entire sequence database until all frequent patterns have been identified. This strategy is efficient if the number of frequent patterns that fail to be recognized from the sample is small, which is typically the case under the assumption of a reasonably large sample size and a relatively short pattern length. However, in the problem we try to solve, the number of ambiguous patterns may be substantially larger, which makes a level-wise search an inefficient process. In contrast, our algorithm can successfully deal with such scenario by each time directly probing the set of ambiguous patterns that would lead to a collapse of the space of remaining ambiguous patterns to the largest extent, so that the number of necessary scans through the sequence database is minimized. This leads to substantially better performance than the existing sampling approach. We will investigate the effect of the border collapsing technique in more detail in a later section and will show that, in most cases, a couple of scans of the sequence database are sufficient even the pattern is very long when the border collapsing is employed.

In summary, the following contributions are claimed in this paper.

- The concept of *compatibility matrix* is introduced to define possible symbol substitutions caused by noise.
- A novel metric *match* is proposed to capture the significance of a pattern under the noisy environment.
- A sampling based algorithm is devised to efficiently mine long patterns that satisfy a match threshold.
  - The Chernoff Bound is used to estimate the set of ambiguous patterns with very high confidence.
  - Instead of using a level-wise search, a *border collapsing* technique is performed to locate the border of frequent patterns so that the expected number of passes through the sequence database is minimized.
- We conduct numerous experiments to demonstrate the robustness of the match model and the efficiency and effectiveness of the proposed algorithm.

The remainder of this paper is organized as follows. Section 2 gives a brief survey of related work. The model of obscure patterns is proposed in Section 3. Section 4 discusses the sampling based algorithm in detail. The experimental results are shown in Section 5. Finally, Section 6 draws the conclusion.

# 2 Related Work

## 2.1 Sequential Patterns

Much work has been done in the area of sequential pattern discovery [1, 4, 6, 20, 21, 24]. Ignoring other differences in the problem definition, a major common shortcoming among most of previous work is the lack of flexibility in pattern matching (i.e., only the exact match of a pattern in the input data is considered an occurrence of the pattern). In practice, given a pattern and an observed sequence, the disagreement of the pattern with some portion of the sequence may be a result of either truly behavior change or simply distortion of the appearance incurred by noise. An implicit assumption made by existing frameworks is that both cases are regarded as absence of the pattern, which is not necessarily the optimal approach to deal with noise in many applications. In contrast, we introduce a more flexible model that is able to successfully separate these two cases and restore the strength of the pattern diluted due to the distortion incurred by noise.

## 2.2 Algorithms on Mining Long Patterns

A widely used strategy to speed up the process of mining long patterns is to incorporate some look-ahead technique in the original Apriori-based scheme so that the set of maximum frequent itemsets[3] can be identified without traversal through every frequent itemset. Several algorithms [19, 6, 30] have been proposed along this direction, among which the Max-Miner [6] is the most noted advance. The Max-Miner offers simple and effective heuristics to generate candidates for long patterns throughout the mining process and is able to achieve a performance improvement of at least an order of magnitude compared to other look-ahead techniques. In this paper, we will use Max-Miner as the representative of this class of algorithms in the experimental study to compare the performance with that of our proposed approach.

More recently, extensive research [1, 14, 15, 31] has been carried out on mining patterns in a depth-first projection-based fashion as opposite to the traditional breadth-first Apriori-based traversal. As patterns are usually organized via a tree [1, 14, 15] (or lattice in [31]) structure, along with the discovery of frequent patterns when traversal through structure, data projection onto each newly identified frequent pattern is also taken to facilitate the subsequent examination of its super-patterns. In particular, Han et al. [14] 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 [14] is designed for mining frequent itemsets in general, the FreeSpan [15] and SPADE [31] are specifically tailored for mining sequential patterns. It is interesting to notice that, the depth-first approaches generally perform better than breadth-first ones if the data is memory-resident, and the advantage becomes more substantial when the pattern is long. However, in our model, we assume disk-resident data that is far beyond the memory capacity.

## 2.3 Sampling-based and Probabilistic Algorithms

Data mining on sample data has also been explored previously. Srikant et al. [23] and Toivonen et al. [25] are among the earliest to propose sampling-based algorithms to mine frequent itemsets. In this approach, a set of samples is first gathered. (The Chernoff bound can be used to determine the right sample size.) The frequent itemsets are computed based on the samples. Let $F$ be the set of frequent itemsets in the sample data and their immediate superpatterns. The supports of itemsets in $F$ are then computed based on the entire dataset and serve as the (advanced) starting position of a level-wise search that eventually identifies all frequent patterns. This approach is very efficient if the set of frequent patterns mined from the sample data is a good approximation of the exact result from the entire data. This is typically true when the sample size is large and the pattern is of moderate length. This observation is also confirmed in [29]. In our application domains, e.g., computational biology, the number of symbols in a pattern can be very large, e.g., up to a couple hundred. Thus, the number of candidate patterns examined in the last stage can be substantially large, which may require many scans of the entire database.

In addition, some research has been carried out on designing randomized algorithms to mine frequent patterns. Gunopulos et al. [13] is among the pioneers in this direction.

# 3 A Model of Obscure Patterns

In this paper, we are interested in finding patterns that may be concealed (to some extent) by noise in a sequence database. We first introduce some terminologies that will be used throughout this paper. Let $\Theta$ be a set of distinct symbols $\{d_1, d_2, \ldots, d_m\}$.

**Definition 3.1** *A* **sequence** *of* **length** $l$ *is an ordered list of* $l$ *symbols in* $\Theta$. *A* **sequence database** *is a set of tuple* $\langle Sid, S\rangle$ *where* $Sid$ *is the ID of the sequence* $S$.

[3] An itemset is *frequent* if its number of occurrences in a given database is above a certain threshold, and it is called a *maximum frequent itemset* if any of its superset is not frequent.

**Definition 3.2** *A* **pattern** *of* **length** $l$ *is represented as a list of* $l$ *symbols, each of which is either a symbol in* $\Theta$ *or an eternal symbol denoted by "*". The eternal symbol "*"*[4] *is used to indicate the "don't care" position in a pattern. A pattern is also referred to as a* $k$**-pattern** *if the pattern contains* $k$ *non-eternal symbols.*

Note that the formative difference between a sequence and a pattern is that a pattern is allowed to contain the eternal symbol *. Conventionally, we use *sequence* to refer to the raw data in the database which serves as the input to the data mining algorithm and use *pattern* to denote the output produced by the algorithm. Each eternal symbol * specified in the pattern can match a single symbol on the corresponding position (rather than a subsequence with (optional) length constraints) in the input sequence. In this sense, our model is a somewhat stricter than previous models [4, 15, 24, 31]. Nevertheless, we want to mention that the inclusion of the eternal symbol in the pattern specification enables the representation of fixed-length gap(s) between meaningful portions in a pattern. This is very important in many position-sensitive applications such as the analysis of DNA transcription factors. For example, Zinc Finger is a common transcription factor that has the signature $C**C**************H**H$ where $C$ and $H$ represent amino acid cysteine and amino acid histidine, respectively. In general, a $k$-pattern of length $l$ would contain $l-k$ eternal symbol * where $k \leq l$. To exclude trivial patterns from consideration, we also require that neither the first symbol nor the last symbol in a pattern can be the eternal symbol *.

**Definition 3.3** *Given two patterns* $P = d_1 d_2 \ldots d_l$ *and* $P' = d'_1 d'_2 \ldots d'_{l'}$ *where* $l \leq l'$, $P$ *is a* **subpattern** *of* $P'$ *if there exists an integer* $j$ *(*$1 \leq j \leq l' - l$*) such that, for each* $i$ $(1 \leq i \leq l)$*, either* $d_i = *$ *or* $d_i = d'_{i+j}$ *is true. In such a case,* $P'$ *is also referred to as a* **superpattern** *of* $P$.

Intuitively, $P$ is a subpattern of $P'$ if $P$ can be generated by (1) dropping a prefix and/or a suffix of $P'$; or (2) replacing some symbol $d'_i$ in $P'$ with the eternal symbol *; or (3) a combination of (1) and (2). For example, $d_1 * d_3$ and $d_1 * *d_4 d_5$ are subpatterns of $d_1 * d_3 d_4 d_5$ but $d_1 d_2$ is not. It is clear that the sub-/super-pattern relationship defines a lattice among all patterns. Figure 3 shows a fragment of the lattice.

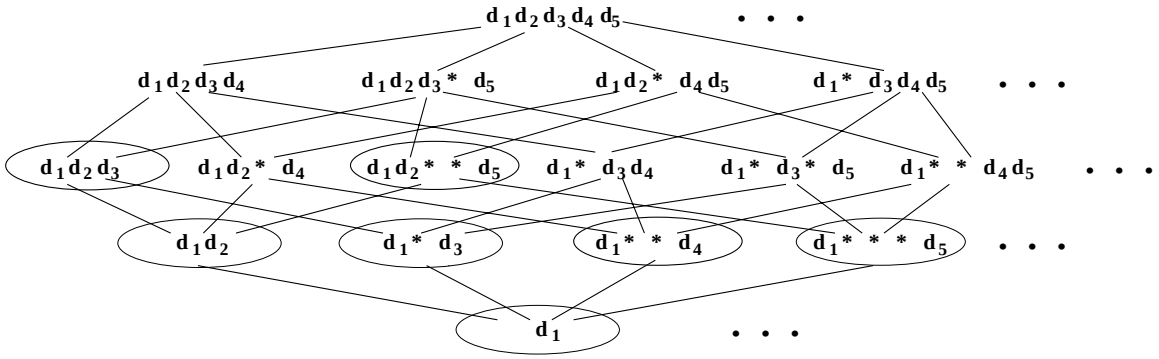



Figure 3: A fragment of lattice of sequential patterns

Our goal is to find the significant patterns in a sequence database in the presence of noise. In order to accommodate the noise, we propose a flexible model that allows obscurity in pattern matching. If the observed data does not match exactly but is somewhat "compatible" with a pattern, it can be regarded as a degraded occurrence of the pattern. To honor the "partial" occurrence of a pattern, we propose a new metric, namely *match*, to characterize the significance of the pattern in a symbol sequence. In particular, the conditional probability of the true value given an observed symbol is utilized to quantify "compatibility" between a pattern and an observed symbol sequence, and to assess the *match* of the pattern.

**Definition 3.4** *Let* $\Theta = \{d_1, d_2, \ldots, d_m\}$ *be a set of distinct symbols. An* $m \times m$ *matrix* $C$*, referred to as* **compatibility matrix***, can be used to represent the conditional probabilities for each pair of symbols. Given two symbols* $d_i$ *and* $d_j$*, the*

[4] It is equivalent to the symbol "." used in regular expression.

*entry $C(d_i, d_j) = Prob(true\_value = d_i \mid observed\_value = d_j)$ is the conditional probability that $d_i$ is the true value given that $d_j$ is observed.*

Figure 2 shows an example of the conditional probability matrix between 5 symbols $d_1$, $d_2$, $d_3$, $d_4$, and $d_5$. An entry $C(d_i, d_j) > 0$ indicates that $d_i$ might be (mis)represented as $d_j$ in the observation; while $C(d_i, d_j) = 0$ implies that the symbol $d_i$ cannot be represented as $d_j$ despite the presence of noise. For instance, $C(d_1, d_2) = 0.1$ and $C(d_1, d_3) = 0$ in Figure 2. This means that there is a chance that a $d_1$ flips to a $d_2$ in the observation but it is impossible that a $d_1$ may turn to a $d_3$. We also define $C(*, d_i) = 1$ for all $i (1 \leq i \leq m)$ to handle the don't care position(s) in the pattern. The intuition is that any (observed) symbol should be *fully compatible* with the don't care position and therefore should not incur any change to the match of the pattern. Note that the compatibility is not necessary a symmetric measurement in the sense that $C(d_i, d_j) \neq C(d_j, d_i)$ may be true in some occasion. In Figure 2, $C(d_1, d_2) = 0.1$ and $C(d_2, d_1) = 0.05$. We also want to point out that, in the case $C(d_i, d_i) < 1$, an observed symbol $d_i$ does not always imply that $d_i$ really occurs. $C(d_1, d_1) = 0.9$ implies that an observed $d_1$ truly represents itself with $90\%$ probability and is a misrepresentation of some other symbol with $10\%$ chance. (According to Figure 2, an observed $d_1$ has a $5\%$ chance to be a misrepresentation of $d_2$ and $d_3$, respectively.) It is conceivable that (1) the compatibility matrix provides a meaningful measure to reveal the substance given the observation, and (2) the assessment of each entry has great impact to the final result. In practice, this matrix can be either given by a domain expert or learned from a training data set. In gene sequence analysis, this matrix can be obtained through clinical study. Some practical examples can be found in [10] on the biology sequence analysis. In this paper, we assume that the compatibility matrix is given by some domain expert in advance and will not elaborate on how to obtain and justify the value of each entry in the matrix. We also demonstrate in Section 5 that, even with a certain degree of error contained in the compatibility matrix, our model can still produce results of reasonable quality.

Given a pattern $P = d_1 d_2 \ldots d_l$ and a segment[5] of $l$ observed symbols $s = d'_1 d'_2 \ldots d'_l$, the conditional probability $Prob(P \mid s)$ represents the probability that $s$ corresponds to an occurrence of $P$, and can be used as the indication of how much the pattern tallies with the observation. Therefore, we define the *match* of $P$ in $s$ to be the value of $Prob(P \mid s)$.

**Definition 3.5** *Given a pattern $P = d_1 d_2 \ldots d_l$ and a segment of $l$ observed symbols $s = d'_1 d'_2 \ldots d'_l$, the* **match** *of $P$ in $s$ (denoted by $M(P, s)$) is defined as the conditional probability $Prob(P \mid s)$.*

Assuming that each observed symbol is generated independently, we have $M(P, s) = Prob(P \mid s) = \Pi_{1 \leq i \leq l} C(d_i, d'_i)$. If $M(P, s) > 0$, then $s$ is regarded as a (degraded) occurrence of $P$ and $M(P, s)$ is viewed as the degree of which the pattern $P$ is retained/reflected in $s$. We also say that $P$ **matches** $s$ if $M(P, s) > 0$ and $P$ **does not match** $s$ otherwise. For example, the match of $P_1 = d_1 * d_2$ in a segment $s = d_1 d_2 d_2$ is $M(P_1, s) = C(d_1, d_1) \times C(*, d_2) \times C(d_2, d_2) = 0.9 \times 1 \times 0.8 = 0.72$. However, the pattern $P_2 = d_1 d_2 d_5$ does not match $s$ because $M(P_2, s) = C(d_1, d_1) \times C(*, d_2) \times C(d_5, d_2) = 0.9 \times 1 \times 0 = 0$.

**Definition 3.6** *For a symbol sequence $S$ of length $l_S$ and a pattern $P$ of length $l_P$ where $l_S \geq l_P$, the* **match** *of $P$ in $S$ is defined as the maximal match of $P$ in every distinct segment (of length $l_P$) in $S$. That is, $M(P, S) = \max_{s \in S} M(P, s)$ where $s$ is a segment of length $l_P$ in $S$.*

There are totally $l_S - l_P + 1$ distinct segments (of length $l_P$) in $S$. All distinct segments of length $l_P$ can be generated by maintaining a window of length $l_P$ and sliding it one position at a time through the sequence. For example, there are 5 distinct segments of length 2 in the sequence $d_1 d_2 d_2 d_3 d_4 d_1$. The match of $P = d_1 d_2$ in this sequence is equal to $\max\{M(P, d_1 d_2), M(P, d_2 d_2), M(P, d_2 d_3), M(P, d_3 d_4), M(P, d_4 d_1)\} = \max\{0.72, 0.08, 0.005, 0, 0\} = 0.72$. Informally, the match of a pattern $P$ in a sequence $S$ is equal to the match of $P$ in the segment (of $S$) which "best" aligns with $P$, and can be regarded as an indicator of the degree of which the pattern $P$ exhibits in the sequence $S$. We also say that $P$ **matches** $S$ if $M(P, S) > 0$ and $P$ **does not match** $S$ otherwise.

[5] A segment is defined as a contiguous portion of a sequence.

**Definition 3.7** *Given a pattern $P$ and a database $D$ of $N$ sequences, the* **match** *of $P$ in $D$ is the average match of $P$ in every sequence in $D$, i.e. $M(P, D) = \frac{\Sigma_{S \in D} M(P,S)}{N}$.*

Similar to the traditional support model, a user is asked to specify a minimum match threshold $min\_match$ to qualify significant patterns. All patterns that meet the $min\_match$ threshold are then referred to as **frequent** patterns. It is clear that the *match* model can accommodate misrepresentation of symbols due to noise in a seamless manner and provide a powerful means to properly separate the noise and change of behavior.

1. Given a pattern $P = d_1 d_2 \ldots d_l$ and a segment $s = d'_1 d'_2 \ldots d'_l$, if the symbol at a given position (e.g., $d'_i$) in $s$ cannot be a misrepresentation of the corresponding symbol (e.g., $d_i$) in $P$ (i.e., $C(d'_i, d_i) = 0$), then the match of the pattern $P$ in the segment $s$ is 0 and the $s$ would not be considered an occurrence of $P$.

2. For each "don't care" position in a pattern, the compatibility on this position is 1 (by definition), no matter what the corresponding observed symbol is. Hence, this position would not incur any loss to the match of the pattern.

3. The match model also provides a natural bridge towards the traditional support model that does not allow partial match between pattern and data. In a noise-free environment, the conditional probability matrix becomes an *identity matrix* (i.e., $C(d_i, d_j)$ is 1 if $i = j$ and is 0 otherwise). The occurrence of a pattern becomes binary: either 1 (present) or 0 (absent). The match of a pattern in the data would be identical to the support of the pattern. In general, the more noise the environment assumes, the less skew the conditional probability distribution. Consider an extreme case where the sequence database is dominated by noise and no dependency exists between the observation and the true value. Then, all entries in the compatibility matrix would have the same value $\frac{1}{m}$ where $m$ is the number of distinct symbols. As a result, all patterns would have exactly the same match value. This coincides with our intuition in the sense that, if the observed data is totally independent of the underlying system behavior, then no pattern should be considered more significant than others.

Figure 4(a) shows a database of 4 sequences. Figure 4(b) and (c) show the comparisons of supports and matches of each symbol and each pattern with two symbols, respectively. The number of patterns with positive match is usually much larger than that with positive support. In particular, as the pattern length increases, the match decreases at a much slower pace than the support. In the previous example (Figure 4(a)), consider patterns $d_3$, $d_3 d_2$, $d_3 d_2 d_2$, and $d_3 d_2 d_2 d_1$. Their supports are 0.5, 0, 0, 0, respectively; whereas their matches are 0.4, 0.07, 0.016, and 0.00522, respectively. This phenomenon is a direct consequence of the allowance of partial match between pattern and data segment. While each segment may increase the support of only one pattern with a full credit, its effect is dispersed among multiple patterns in terms of lifting their matches by various degrees. Figure 4(d) shows the amount of match that the segment $d_2 d_2$ may contribute to each pattern. There are totally 9 patterns that actually "benefit" from it. Note that the summation of these 9 numbers is still 1. It can be viewed as a "redistribution" of certain portion of the support in such a manner that the uncertainty introduced by noise is properly taken into account. For each pattern, the differential between the match and the support is the necessary rectification made towards the significance of the pattern. While the support can be viewed as the "face value" of a pattern, the match indeed represents the "expected value" (if no noise had presented).

The well-known Apriori property also holds on the match metric, which can be stated as in the following claims.

**Claim 3.1** *The match of a pattern $P$ in a symbol sequence $S$ is less than or equal to the match of any subpattern of $P$ in $S$.*

The proof of the above claim can be sketched as follows. Let $P = d_1 d_2 \ldots d_l$ and $P' = d'_1 d'_2 \ldots d'_l$ be two patterns and $P$ is a subpattern of $P'$. For any data segment $s = x_1 x_2 \ldots x_l$, the match of $P$ in $s$ is $M(P, s) = \Pi_{1 \le i \le l} C(d_i, x_i)$ and the match of $P'$ in $s$ is $M(P', s) = \Pi_{1 \le i \le l} C(d'_i, x_i)$. Since for each position $i$, either $d_i = d'_i$ or $d_i = *$ is true, we have either $C(d_i, x_i) = C(d'_i, x_i)$ or $C(d_i, x_i) = C(*, x_i) = 1 \ge C(d'_i, x_i)$. As a result, it must be true that $M(P, s) \ge M(P', s)$.

| ID | sequence |
|---|---|
| 1 | d1 d2 d3 d1 |
| 2 | d4 d2 d1 |
| 3 | d3 d4 d2 d1 |
| 4 | d2 d2 |

**(a) a sequence database**

| symbol | support | match |
|---|---|---|
| d1 | 0.75 | 0.538 |
| d2 | 1.00 | 0.800 |
| d3 | 0.50 | 0.400 |
| d4 | 0.50 | 0.425 |
| d5 | 0 | 0.075 |

**(b) support and match of each symbol**

| pattern | support | match | pattern | support | match |
|---|---|---|---|---|---|
| d1 d1 | 0 | 0.070 | d3 d4 | 0.25 | 0.136 |
| d1 d2 | 0.25 | 0.203 | d3 d5 | 0 | 0 |
| d1 d3 | 0 | 0.020 | d4 d1 | 0 | 0.070 |
| d1 d4 | 0 | 0.023 | d4 d2 | 0.50 | 0.321 |
| d1 d5 | 0.25 | 0.004 | d4 d3 | 0 | 0.023 |
| d2 d1 | 0.50 | 0.391 | d4 d4 | 0 | 0.053 |
| d2 d2 | 0.25 | 0.200 | d4 d5 | 0 | 0.004 |
| d2 d3 | 0.25 | 0.160 | d5 d1 | 0 | 0.033 |
| d2 d4 | 0 | 0.052 | d5 d2 | 0 | 0.006 |
| d2 d5 | 0 | 0.035 | d5 d3 | 0 | 0.008 |
| d3 d1 | 0.25 | 0.165 | d5 d4 | 0 | 0.028 |
| d3 d2 | 0 | 0.070 | d5 d5 | 0 | 0 |
| d3 d3 | 0 | 0.043 | | | |

**(c) support and match of patterns with two symbols**

| pattern | match | pattern | match | pattern | match | pattern | match | pattern | match |
|---|---|---|---|---|---|---|---|---|---|
| d1 d1 | 0.01 | d2 d1 | 0.08 | d3 d1 | 0 | d4 d1 | 0.01 | d5 d1 | 0 |
| d1 d2 | 0.08 | d2 d2 | 0.64 | d3 d2 | 0 | d4 d2 | 0.08 | d5 d2 | 0 |
| d1 d3 | 0 | d2 d3 | 0 | d3 d3 | 0 | d4 d3 | 0 | d5 d3 | 0 |
| d1 d4 | 0.01 | d2 d4 | 0.08 | d3 d4 | 0 | d4 d4 | 0.01 | d5 d4 | 0 |
| d1 d5 | 0 | d2 d5 | 0 | d3 d5 | 0 | d4 d5 | 0 | d5 d5 | 0 |

**(d) the match contributed to each pattern by an observation of "d2 d2"**

Figure 4: Comparison of *support* and *match*

By definition, the match of a pattern in a sequence is the maximal match of the pattern in every distinct segment of the sequence. It is very straightforward that, for any symbol sequence $S$, $M(P, S) \geq M(P', S)$ is also true. As a direct corollary of Claim 3.1, the follow claim also holds.

**Claim 3.2 (Apriori property)** *The match of a pattern $P$ in a sequence database $D$ is less than or equal to the match of any subpattern of $P$ in $D$.*

A direct implication of the Apriori property is that, given a $min\_match$ threshold, the set of frequent patterns occupy a "continuous portion" in the pattern lattice, and can be described using the notion of **border** [20]. Intuitively, the border demarcates the separation between the set of frequent patterns and the rest of the lattice, and can be represented by the set of frequent patterns whose immediate super-patterns are all infrequent. For example, if the patterns with solid circles are frequent in Figure 3, then the border should consist of three patterns: $d_1d_2d_3$, $d_1d_2 * *d_5$, and $d_1 * *d_4$. These three patterns are also referred to as **border elements**. We sometimes use the phrase "the border of $match'$" as the abbreviation of "the border of frequent patterns given $match'$ as the $min\_match$ threshold".

An interesting observation we will explore in a later section is that, given a reasonable threshold, the number of frequent patterns at each level (in the super-/sub-pattern lattice) using the match metric is usually larger than that using the support. This is because, as the pattern length increases, the match decreases at a much slower pace than the support. Even though any algorithm powered (sometimes implicitly) by the Apriori property can be adopted to mine frequent patterns according to the match metric, it will produce a less efficient solution. The weakness becomes more substantial in mining sequence data since the length of a pattern can easily range up to dozens and even hundreds in many applications, such as gene expression. Even a direct generalization of previously proposed approach for mining long patterns under the support model (e.g., Max-Miner [6]) still requires many scans of the sequence database if the database is disk-resident. In the next section, we design a novel

algorithm that can efficiently generate the border of frequent patterns in a few scans of the sequence database with very high confidence statistically.

# 4 A Probabilistic Approach

For a given sequence database, we want to find patterns whose match satisfies a user-specified threshold $min\_match$. As we mentioned before, any algorithm previously proposed for mining frequent patterns [1, 4, 6, 14, 15, 19, 24, 31] under the support framework can be generalized to mine patterns that satisfy the minimum match requirement. One factor that leads to algorithm inefficiency is that the mining of long patterns typically requires many scans of the data. Even the algorithms designed for mining long patterns [6] may still require many scans of the data when the pattern is substantially long. This problem becomes more evident in mining sequence data. For example, a gene expression typically contains at least a couple hundred amino acids. To reduce the number of necessary passes through the input sequences, we propose a fast mining algorithm that can discover the border of frequent patterns in a few scans of the sequence database. Sampling technique is used to obtain a quick estimation of the border and additional scan(s) of the sequence database can be performed to finalize the border.

In order to obtain an estimation of the border of frequent patterns without examining the entire sequence database, we use the additive Chernoff bound [9, 16] to estimate the range of the match of a pattern from a sample of the data with a high statistical confidence (e.g., $99.99\%$). Let $X$ be a random variable whose spread[6] is $R$. For example, in the context of the match model, the match can vary from 0 to 1 and therefore $R = 1$. Suppose that we have $n$ independent observations of $X$, and the mean is $\mu$. The Chernoff Bound states that with probability $1 - \delta$, the true mean of $X$ is at least $\mu - \epsilon$, where

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$$

For example, assume that the spread of a random variable is 1 and $\mu$ is the mean of 10000 samples of the random variable. Then we are able to say that the true value of the random variable is at least $\mu - 0.0215$ with $99.99\%$ confidence. Similarly, with probability $1 - \delta$, the expected value of variable $X$ is at most $\mu + \epsilon$. This provides the opportunity to estimate the range of the match for each pattern from a set of samples.

**Claim 4.1 (Chernoff bound estimation)** *Given a set of sample data and a threshold $min\_match$, a pattern is* **frequent** *with probability $1 - \delta$ if $\mu_{match} > min\_match + \epsilon$ and is* **infrequent** *with probability $1 - \delta$ if $\mu_{match} < min\_match - \epsilon$, where $\mu_{match}$ is the match of the pattern in the sample data[7]. Those patterns (referred to as* **ambiguous** *patterns) whose matches in the sample are between $min\_match - \epsilon$ and $min\_match + \epsilon$ remain undecided and need further examination.*

An attractive property of the Chernoff bound is that it is independent of the probability distribution that generates the observations, as far as such probability distribution remains static during the entire process. This distribution-free nature is very important because the underlying distribution that characterizes the match of a pattern is usually unknown. However, this generality comes with the price of a more conservative bound than a distribution-dependent estimation and would require a larger number of observations to reach the same bound. This weakness sometimes prevents us from obtaining a tight bound when the sample size $n$ is limited (e.g., due to memory size). Clearly, the number of ambiguous patterns highly depends on the value of $\epsilon$ which itself is a function of the sample size. A large number of ambiguous patterns may incur many scans of the entire sequence database. Therefore, the value of $\epsilon$ should be as small as possible. In order to further reduce $\epsilon$ under the constraint of memory capacity, instead of using $R = 1$, we employ an additional step to derive a more restricted spread $R$ for

[6] The spread of a random variable is defined as the difference between the maximum possible value and the minimum possible value of the random variable.

[7] By definition, the match of a pattern in the sample data is the average match of every sample.

the match of each pattern. According to the Apriori property (Claim 3.2), the match of a pattern is always less than or equal to the minimum match of each symbol in the pattern.

**Claim 4.2 (Restricted spread)** *The restricted spread $R$ for the match of a pattern $(d_1, d_2, \ldots, d_l)$ is $R = \min_{1 \le i \le l} match[d_i]$ where $match[d_i]$ is the match of the symbol $d_i$ in the entire sequence database.*

For example, the match of $(d_1, *, d_2)$ would not exceed the minimum match of $d_1$ and $d_2$. If the matches of $d_1$ and $d_2$ are 0.1 and 0.05 in the database respectively, then the match of $(d_1, *, d_2)$ has to be between 0 and 0.05 (instead of the original spread 1) in the database. Thus, we can use $R = 0.05$ when applying the Chernoff bound and reduce the value of $\epsilon$ by $95\%$. (Note that $\epsilon$ is linearly proportional to $R$.) Therefore, before we examine the in-memory sample, a scan of the entire sequence database is performed to compute the match of each individual symbol. Note that as a by-product of this step, a random sample of the data can be easily obtained and kept in memory without any extra overhead. This sample set can then be used directly to classify patterns using Chernoff bound.

Nevertheless, when the pattern is long (e.g., in the range of dozens to hundreds of symbols) and the tolerable error is very small, the number of ambiguous patterns can be still considerably large and may require significant amount of computation and many scans through the database. This problem is more severe when the match (rather than the support) is used as the metric. To address this issue, we propose a **border collapsing** technique to ensure a minimum number of scans through the sequence database. Hence, the following three-fold algorithm is developed for mining the obscure patterns of length $l$.

1. While scanning the sequence database, find the match of each individual symbol and take a random sample of sequences.
2. Identify the borders that embrace the set of ambiguous patterns (i.e., whose match is between $min\_match - \epsilon$ and $min\_match + \epsilon$) using Chernoff bound based on the sample taken at the previous step.
3. Locate the border of frequent patterns in the entire sequence database via border collapsing.

A question one may concern is that, since the Chernoff bound only provides a probabilistic bound (rather than an absolute one), there is a small chance (bounded by $\delta$) that a pattern $P$ is frequent (i.e., its actual match in the entire sequence database is at least $min\_match$) but $P$'s match in the sample data is below $min\_match - \epsilon$. Even though the measured error is much smaller than $\delta$ in practice, it is important to understand the characteristic of these misclassified patterns. According to the above algorithm, $P$ will be mislabeled as infrequent in the second phase. We now explore the impact of such mislabeled patterns to the quality of the result. Intuitively, it would be a less serious issue if the actual match of a mislabeled pattern is very close to $min\_match - \epsilon$ than the scenario where the actual match is far above $min\_match$. The rationale is that, in the former case, one can always lower the threshold slightly to include the originally mislabeled patterns in the result. Therefore, the match distribution of mislabeled patterns is very important. Let $dis(P)$ be the difference between the actual match of a mislabeled pattern $P$ and $min\_match$. It is easy to derive from the Chernoff bound that the probability $Prob(dis(P) > \rho)$ diminishes exponentially as $\rho$ grows. For example, $Prob(dis(P) > 2\rho) = Prob(dis(P) > \rho)^4$. This theoretically guarantees that the matches of most mislabeled patterns locate close to $min\_match - \epsilon$. This observation is also confirmed in the experimental results in Section 5.5. We now investigate each step in detail in the following subsections.

## 4.1 Phase 1: Finding Match of Individual Symbols and Sampling

In this phase, with one scan of the sequence database, we need to calculate the match of every symbol and obtain a sample set. Let's first look at the computation of the match of each symbol. The pseudo code of computing match of every symbol is shown in Algorithm 4.1. A counter $match[d]$ is initiated for each distinct symbol $d \in \Theta$ to store the match value of $d$ (Line 1-2). As we scan through each sequence $D_i$ in the database, $max\_match[d]$ stores the match of $d$ in $D_i$. For each symbol

encountered (e.g., $d'$), we update the value of $max\_match[d]$ if necessary (Line 5-9). Every time after examining a sequence, the match of each symbol $d$ in the database, $match[d]$, is increased by an amount of $\frac{max\_match[d]}{N}$ (Line 10-11). Figure 5(a) illustrates the value $max\_match$ of each symbol after examining each element in the first sequence in Figure 4(a) according to the compatibility matrix in Figure 2. When we examine the second symbol (i.e., $d_2$), $max\_match[d_2]$ and $max\_match[d_4]$ are updated to $C(d_2, d_2)$ and $C(d_4, d_2)$, respectively. The last column of Figure 4(a) shows the match of each symbol in this sequence. The value of $match$ of each symbol after examining each sequence in Figure 4(a) is shown in Figure 5(b). For instance, after processing the first sequence, the match of $d_2$ is increased by an amount of $\frac{max\_match[d_2]}{N} = \frac{0.8}{4} = 0.2$. Note that the match of multiple symbols may be updated when we examine each symbol in the sequence. After we examine the entire sequence database, $match[d]$ holds match of each symbol $d$ and $d$ is a frequent symbol if $match[d] \geq min\_match$.

| current_match | initial | d1 | d2 | d3 | d1 |
|---|---|---|---|---|---|
| d1 | 0 | 0.9 | 0.9 | 0.9 | 0.9 |
| d2 | 0 | 0.05 | 0.8 | 0.8 | 0.8 |
| d3 | 0 | 0.05 | 0.05 | 0.7 | 0.7 |
| d4 | 0 | 0 | 0.1 | 0.1 | 0.1 |
| d5 | 0 | 0 | 0 | 0.15 | 0.15 |

**(a) calculate max_match in "d1 d2 d3 d1"**

| match | initial | sequence | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 |
| d1 | 0 | 0.225 | 0.45 | 0.675 | 0.538 |
| d2 | 0 | 0.2 | 0.4 | 0.6 | 0.8 |
| d3 | 0 | 0.175 | 0.213 | 0.388 | 0.4 |
| d4 | 0 | 0.025 | 0.213 | 0.4 | 0.425 |
| d5 | 0 | 0.038 | 0.038 | 0.075 | 0.075 |

**(b) calculate match in Figure 4(a)**

Figure 5: Calculate match of each symbol

Obtaining the set of frequent symbols can be beneficial in two aspects.

- According to the Apriori property, only frequent symbols may participate in a frequent pattern. With the set of frequent symbols on hand, we can eliminate unnecessary counters to a large extent. This is even more important to the match model since an occurrence of a symbol combination may trigger updates to match counters of multiple patterns.
- The match of each (frequent) symbol in a (candidate) pattern can be used to provide a much restricted spread $R$ of the match for this pattern to produce a much tighter bound $\epsilon$. It will eliminate a large number of ambiguous patterns that need to be re-examined against the entire sequence database.

It is easy to see that the computational complexity of this procedure is $O(N \times \overline{l_S} \times m)$ where $\overline{l_S}$ and $m$ are the average sequence length and the number of distinct symbols, respectively. Note that, in the case where $\overline{l_S} \gg m$, it is easily to reduce the bound to $O(N \times (\overline{l_S} + m^2))$ by a simple optimization motivated by the following observation: for any given symbol $d$, its match in a sequence $D_i$ is the maximal compatibility of $d$ with any symbol $d'$ in the sequence, i.e., $max\_match[d] = \max_{d' \in D_i} C(d, d')$. Therefore, we only need to update the value of $max\_match$ for the first occurrence of each distinct symbol in the sequence[8]. For example, we can omit the calculation associated with the second occurrence of $d_1$ in Figure 5(a) (once we verified that $d_1$ has been encountered before in the sequence). In summary, the computational complexity is $O(N \times \min\{\overline{l_S} \times m, \overline{l_S} + m^2\})$.

During the scan of the sequence database, a sample of sequences is also taken and stored in memory. Let $n$ be the number of samples that can be held in memory. A very simple way [27][9] to guarantee a random sampling is to generate an independent uniform random variable for each sequence to determine whether that sequence should be chosen (Line 12). At the beginning, a sequence will be chosen with probability $\frac{n}{N}$. Subsequently, if $j$ sequences have been chosen from the first $i$ sequences, then the next sequence will be chosen with probability $\frac{n-j}{N-i}$ (Line 13-16). The computational complexity of the sampling procedure is $O(N + n \times \overline{l_S})$, which makes the total computational complexity still $O(N \times \min\{\overline{l_S} \times m, \overline{l_S} + m^2\})$.

**Algorithm 4.1** *Calculating Match of Each Symbol and Taking Samples*

[8] By maintaining a flag for each symbol, we can easily check whether a symbol has been encountered before in the current sequence.

[9] Interested readers please refer to [27] for a comprehensive discussion of various sampling methods.

SymbolMatch_Sampling($D$, $n$)
{ /* $D$ is the database of $N$ sequences
$n$ is the number of samples that need to be taken */
1: **for each** distinct symbol $d$ **do**
2: $match[d] \leftarrow 0$ /* Initialize the match array */
3: $j \leftarrow 0$ /* $j$ is the number of samples that have been chosen */
4: **for** $i \leftarrow 1$ **to** $N$ **do** /* exam each sequence sequentially */
5: **for each** distinct symbol $d$ **do**
6: $max\_match[d] \leftarrow 0$ /* $max\_match[d]$ records the match of $d$ in the current sequence ($D_i$) */
7: **for each** symbol $d' \in D_i$ **do** /* $D_i$ is the $i$th sequence in $D$ */
8: **for each** symbol $d$ where $C(d, d') > max\_match[d]$ **do**
9: $max\_match[d] \leftarrow C(d, d')$
10: **for each** distinct symbol $d$ **do**
11: $match[d] \leftarrow match[d] + \frac{max\_match[d]}{N}$
12: $p \leftarrow rand[0, 1]$ /* generating a random number between 0 and 1 */
13: **if** $p < \frac{n-j}{N-i}$ **do**
14: **then**
15: take $D_i$ as a sample
16: $j \leftarrow j + 1$
17: **return**($match$)
}

## 4.2 Phase 2: Ambiguous Pattern Discovery on Samples

Based on the samples taken in the previous phase, all patterns can be classified into three categories: *frequent* patterns, *infrequent* patterns, and *ambiguous* patterns, according to their observed matches in the sample data. In this phase, we want to find the two borders in the super-pattern/sub-pattern lattice, which separate these three categories. The border (denoted by FQT) between the frequent patterns and the ambiguous patterns is the set of frequent patterns whose immediate superpatterns are either ambiguous or infrequent, whereas the border (denoted by INFQT) between the ambiguous patterns and the infrequent patterns are the set of ambiguous patterns whose superpatterns are all infrequent. More specifically, these two borders correspond to the match thresholds $min\_match + \epsilon$ and $min\_match - \epsilon$ respectively (with respect to the sample data).

Since the Apriori property holds on the match metric, many (border discovery) algorithms presented for mining frequent patterns (with respect to a support threshold) [3, 6, 14, 20] can be adopted to solve this problem with one modification — the routine to update match(es) when examining each sample sequence. Let $P = d_1 d_2 \ldots d_l$ be a candidate pattern and $match[d_1, d_2, \ldots, d_l]$ denote the counter storing the match of the pattern $d_1 d_2 \ldots d_l$ in the sample data. By definition, the match of a pattern in the sample data is the average match of the pattern in every sample sequence. To compute the match of a pattern $P = d_1 d_2 \ldots d_l$ in a sequence $S$, a straightforward way is to maintain a sliding window of width $l$. As the window slides through the sequence by one position at a time, the match of $P$ with the segment covered by the window can be calculated, and the highest one is the match of $P$ in the whole sequence. The pseudo code is shown in Algorithm 4.2. This process would require $O(\mid S \mid \times l)$ computation theoretically where $\mid S \mid$ is the length of $S$. Nevertheless, since the compatibility matrix is usually a sparse matrix, we can easily obtain a much more efficient algorithm to compute the match in nearly $\Theta(\mid S \mid)$ time [5, 17, 18, 28]. Due to the space limitations, we will not elaborate on it in this paper.

With zero as the initial value, a straightforward way to compute the match of $P$ in the sample is to accumulate the value of $match[d_1, d_2, \ldots, d_l]$ by an amount of $\frac{M(P,S)}{n}$ for each sample sequence $S$ where $M(P, S)$ is the match of $P = d_1, d_2, \ldots, d_l$ in $S$. After we obtain $match[d_1, d_2, \ldots, d_l]$, $P$ is labeled as

- a *frequent* pattern if $match[d_1, d_2, \ldots, d_l] > min\_match + \epsilon$;
- an *ambiguous* pattern if $min\_match - \epsilon < match[d_1, d_2, \ldots, d_l] < min\_match + \epsilon$;
- an *infrequent* pattern otherwise;

where $\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$ and $R = \min_{1 \le i \le l} match[d_i]$ (Line 17-29). The borders FQT and INFQT are also updated accordingly (Line 22-23, 28-29). Because we aim at finding not only the border separating frequent patterns and ambiguous patterns but also the border separating ambiguous patterns from infrequent patterns, a pattern $P$ may be considered as a candidate pattern iff every sub-pattern of $P$ is either a frequent or ambiguous pattern by the Apriori property. Since the sample data is in memory, any pruning technique (such as breadth-first, depth-first, looking-ahead, etc., [1, 3, 6, 14, 20]) may be used to conduct the search and would produce a reasonable solution. Note that our objective is to identify the borders that separate frequent patterns, ambiguous patterns, and infrequent patterns. Therefore, we only need to keep track of the set of patterns that define these two borders. At the end of the search process, all patterns that have not been categorized are regarded as infrequent patterns.

**Algorithm 4.2** *Calculating Match and Labeling Patterns*

Calculate_Match($P$)
{ /* Calculate the match of pattern $P = d_1 d_2 \ldots d_l$ in sample */
1: **for each** sample sequence $S$ **do**
2: $max\_match \leftarrow 0$
/* record the maximal match of $P$ in $S$ */
3: $current\_match \leftarrow 0$
/* record the match of $P$ in a segment falls in */
/* the sliding window */
4: $i \leftarrow 1$
/* current starting position of the sliding window in $S$ */
5: $j \leftarrow 1$ /* current position in $P$ */
6: **while** $i \le |\ S\ | - l + 1$ **do** {
7: $current\_match \leftarrow C(d_j, S_{i+j-1})$
8: **while** $j < l$ **and** $current\_match > 0$ **do**
9: $j \leftarrow j + 1$
10: $current\_match \leftarrow current\_match \times C(d_j, S_i)$
11: **if** $current\_match > max\_match$
12: **then** $max\_match \leftarrow current\_match$
13: $i \leftarrow i + 1$ /* slide the window to right by one position */
14: $j \leftarrow 1$
/* reset $j$ so that we start to compute from */
/* the left most symbol of $P$ again */
} /* finish computing the match of $P$ in $S$ */
15: $match[P] \leftarrow match[P] + \frac{max\_match}{n}$
/* $n$ is the number of samples */
16: **return**($match$)
}

Label_Patterns($P$)
{ /* $P = d_1 d_2 \ldots d_l$ */
17: $R \leftarrow \min\{match(d_1), match(d_2), \ldots, match(d_l)\}$
18: $\epsilon \leftarrow \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$
19: **if** $match[P] > min\_match + \epsilon$
20: **then**
21: $label[P] \leftarrow `significant'$
22: $FQT \leftarrow FQT \cup \{P\}$
23: remove from $FQT$ any sub-pattern of $P$
24: **else if** $match[P] < min\_match - \epsilon$
25: **then** $label[P] \leftarrow `insignificant'$
26: **else**
27: $label[P] \leftarrow `ambiguous'$
28: $INFQT \leftarrow INFQT \cup \{P\}$
29: remove from $INFQT$ any sub-pattern of $P$
}

The optimal value of the confidence parameter $\delta$ used in the Chernoff bound is application dependent and can be adjusted by the user. Since the Chernoff bound is a very conservative bound, the actual error is usually much smaller than the theoretical

probability $\delta$. Empirically, when the pattern length is relatively short, a moderate value of $\delta$ (e.g., 0.001) is able to produce considerably high accuracy. This observation is also confirmed by our experiments discussed in the next section. However, as the pattern length grows, the number of patterns that need to be further verified against the entire database grows in an exponential pace. We will continue to investigate in this matter in the next section.

Assume the maximum length of any frequent pattern is $\widehat{l_P}$. There are up to $O((m+1)^{\widehat{l_P}})$ distinct patterns of length up to $\widehat{l_P}$, where $m$ is the number of symbols. (Note that there are $m+1$ choices for each position: $m$ specific symbols and *.) The computational complexity of this phase is $O((m+1)^{\widehat{l_P}} \times \mid S \mid \times \widehat{l_P} \times n)$ since it might take $O(\mid S \mid \times \widehat{l_P} \times n)$ computation to calculate the match of a pattern. Note that this only characterizes the theoretically worst scenario. In practice, much less computation is usually required and all computation can be done efficiently as all sample data are in memory.

### 4.3 Phase 3: Border Collapsing

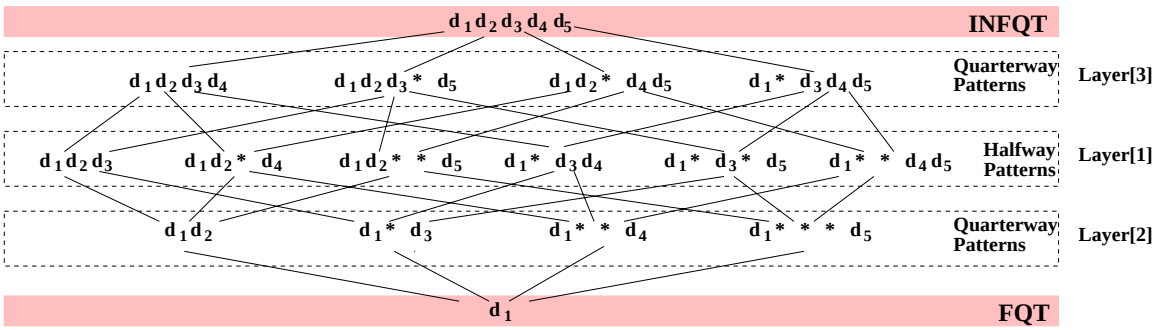
At this phase, we need to investigate ambiguous patterns further to determine the real border of frequent patterns. If the memory can hold the counters associated for all ambiguous patterns (i.e., all patterns between FQT and INFQT), a single scan of the entire sequence database would be able to calculate the exact match of each ambiguous pattern and the border of frequent patterns can be determined accordingly. However, we may experience with the scenario where a huge number of ambiguous patterns exist. This may occur when there are a large number of patterns whose matches happen to be very close to the threshold $min\_match$, which is typically the case when the pattern is long. In such a case, multiple scans of the sequence database become inevitable.

Our goal of this phase is to efficiently collapse the gap between the two borders embracing the ambiguous patterns into one single border. An iterative "probing-and-collapsing" procedure can be employed. In order to minimize the expected number of scans through the database, the ambiguous patterns that can provide high collapsing effect are always probed first. A greedy algorithm can be developed to repeatedly choose the pattern with the most collapsing power among the remaining ambiguous patterns until the memory is filled up. A scan of the database is then performed to compute the matches of this set of patterns and the result is used to collapse the space of the remaining ambiguous patterns. This iterative process continues until no ambiguous pattern exist.

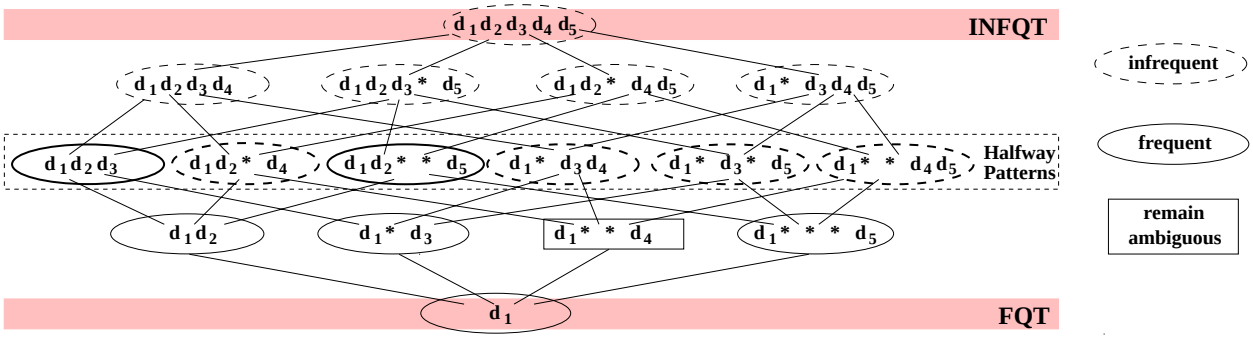
While the two borders embracing the ambiguous patterns act as the "floor" and the "ceiling" of the space of ambiguous patterns, an algorithm that is analogous to the binary search would serve our purpose. The algorithm is presented as Algorithm 4.3. The patterns on the halfway layer between the two borders can provide the most collapsing effect and in turn should be probed first. The patterns on the quarterway layers are the set of patterns that can produce the most collapsing effect among the remaining ambiguous patterns, and so on. Consider the set of ambiguous patterns $d_1$, $d_1d_2$, $d_1d_2d_3$, $d_1d_2d_3d_4$, and $d_1d_2d_3d_4d_5$ in Figure 6(a). It is easy to see that $d_1d_2d_3$ has the most collapsing power. If $d_1d_2d_3$ is frequent, then $d_1$ and $d_1d_2$ must be frequent by the Apriori property. Otherwise (i.e., $d_1d_2d_3$ is infrequent), $d_1d_2d_3d_4$ and $d_1d_2d_3d_4d_5$ should be infrequent as well. Therefore, no matter whether $d_1d_2d_3$ is frequent or not, two other patterns (among the five) can be properly labeled without any further investigation on them. Similarly, we can justify that $d_1d_2$ and $d_1d_2d_3d_4$ have more collapsing power than the remaining two. As depicted in Algorithm 4.3, the algorithm identifies the patterns on the halfway layer, quarterway layers, $\frac{1}{8}$ layers, ... successively until the memory is filled up by the corresponding counters. There are two quarterway layers (each of which is the halfway layer of a half), four $\frac{1}{8}$ layers (each $\frac{1}{8}$ layer is the halfway layer of a quarter), and in general $x/2$ $\frac{1}{x}$-layers (each one is the halfway of a $\frac{1}{x/2}$ region), where $x$ is a power of 2. The process is carried out by sequentially computing the halfway layer between two adjacent layers calculated previously in a recursive manner. For example, the halfway layer (Layer 1) is computed first, followed by generating two quarterway layers (Layers 2 and 3) as shown in Figure 6(a). Given two layers of patterns, the function $Halfway()$ in Algorithm 4.4 can be used to derive the halfway layer between them. Note that we do not physically store all ambiguous patterns. The set of ambiguous patterns that belongs to the desired layer is generated by the function $Halfway()$. Consider a pair of patterns $P_1$ and $P_2$ (one from each

border), where $P_1$ is a sub-pattern of $P_2$. The halfway patterns are the set of patterns that consist of $\lceil \frac{i_1+i_2}{2} \rceil$ non-* symbols and are super-patterns of $P_1$ and sub-patterns of $P_2$ (Line 4-6) , where $i_1$ and $i_2$ are the number of non-* symbols in $P_1$ and $P_2$ respectively.

To better understand the effect brought by the border collapsing, let's assume that only patterns on the halfway layer are held in memory. If a halfway pattern turns out to be frequent, then all of its sub-patterns are frequent. Otherwise (i.e., the pattern is infrequent), all of its super-patterns are infrequent as well. In either case, one of these two borders is collapsed to that halfway pattern. For example, if we know that $d_1d_2d_3d_4d_5$ is on the border separating the ambiguous patterns and infrequent patterns while $d_1$ is on the border between frequent patterns and ambiguous patterns as shown in Figure 6(b). Thus, the patterns $d_1d_2d_3$, $d_1d_2*d_4$, $d_1d_2**d_5$, $d_1*d_3d_4$, $d_1*d_3*d_5$, and $d_1**d_4d_5$ are ambiguous patterns on the halfway layer between two borders and will be examined first. It is obvious that one of the borders would collapse to the halfway layer if these halfway patterns have homogeneous label (i.e., either all are frequent or all are infrequent). In this case, the space of ambiguous patterns is reduced by half. A more interesting scenario is that the halfway patterns have mixed labels (i.e., some of them are frequent while the rest are not), which turns out to provide even more collapsing effect. Assume that $d_1d_2d_3$ and $d_1d_2**d_5$ are frequent (marked with solid circles on the halfway layer) while the remaining one (indicated by dashed circles on the halfway layer) are not. By applying the Apriori property, $d_1$, $d_1d_2$, $d_1*d_3$, and $d_1***d_5$ should also be frequent. Similarly, $d_1d_2d_3d_4$, $d_1d_2d_3*d_5$, $d_1d_2*d_4d_5$, $d_1*d_3d_4d_5$, and $d_1d_2d_3d_4d_5$ are all infrequent. Note that only $d_1**d_4$ still remains ambiguous as indicated by a solid rectangle in Figure 6(b). In general, if the memory can hold all patterns up to the "$\frac{1}{x}$ layer", the space of ambiguous patterns can be at least narrowed to $\frac{1}{x}$ of the original one where $x$ is a power of 2. As a result, **if it takes a level-wise search $y$ scans of the sequence database, only $O(\log_x y)$ scans are necessary when the border collapsing technique is employed**.




**(a) probing priority**




**(b) a scenario where the halfway patterns have mixed labels**

Figure 6: Border Collapsing of Ambiguous Patterns

**Algorithm 4.3** *Border Collapsing of Ambiguous Patterns*

BorderCollapsing(FQT, INFQT)
{ /* FQT is the set of patterns on the border separate the frequent and ambiguous patterns
while INFQT is the set of patterns on the border separate the infrequent and ambiguous patterns */
1: $j \leftarrow 1$
2: /* $j$ is an index that keeps track of the number of layers that have been put into memory */
3: **while** there exist some ambiguous patterns **do** {
4: **while** the memory still has space **do** {
/* Compute $Layer[j]$ (Line 5-16) */
/* First, determine the two layers of patterns of which the halfway should be taken. (Line 5-15) */
5: **if** $j = 1$
/* At the very beginning, the halfway patterns between two borders embracing
the ambiguous patterns are considered. */
6: **then**
7: $Layer[j].Lower\_parent \leftarrow FQT$
8: $Layer[j].Upper\_parent \leftarrow INFQT$
9: **else if** $j$ **mod** $2 = 0$
10: **then**
11: $Layer[j].Lower\_parent \leftarrow Layer[\lfloor j/2 \rfloor].Lower\_parent$
12: $Layer[j].Upper\_parent \leftarrow Layer[\lfloor j/2 \rfloor].patternset$
13: **else**
14: $Layer[j].Lower\_parent \leftarrow Layer[\lfloor j/2 \rfloor].patternset$
15: $Layer[j].Upper\_parent \leftarrow Layer[\lfloor j/2 \rfloor].Upper\_parent$
/* Then, compute patterns that belong to $Layer[j]$ */
16: $Layer[j].patternset \leftarrow Halfway(Layer[j].Lower\_parent, Layer[j].Upper\_parent)$
17: Initialize counter for patterns in $Layer[j]$
18: $j \leftarrow j + 1$
} /* The memory has been filled up */
19: scan the sequence database to compute the matches of patterns in memory
20: update $FQT$ and $INFQT$
21: $j \leftarrow 1$
} /* no ambiguous pattern exists */
22: **return** $FQT$
}

**Algorithm 4.4** *Finding Halfway Patterns*

Halfway(Layer1, Layer2)
{ /* Layer1 and Layer2 are two sets of ambiguous patterns */
1: $halfway \leftarrow \emptyset$
2: **for each** $P_1 \in Layer1$ and $P_2 \in Layer2$ **do**
3: **if** $P_2$ is a super-pattern of $P_1$ **do**
4: $i \leftarrow \lceil \frac{Num\_Non\_Eternal(P_1) + Num\_Non\_Eternal(P_2)}{2} \rceil$
/* $i$ is the number of non-eternal symbols of the halfway patterns for $P_1$ and $P_2$ */
5: **for each** $i$-pattern $P$ such that $P$ is a super-pattern of $P_1$ and subpattern of $P_2$ **do**
6: $halfway \leftarrow halfway \cup \{P\}$
7: **return** ($halfway$)
}

In summary, this approach can greatly reduce the number of scans through the sequence database by only examining a "carefully-chosen" small subset of all outstanding ambiguous patterns. While the traditional level-wise evaluation of ambiguous patterns push the border of frequent patterns forward across the pattern lattice in a gradual fashion; the border collapsing technique employs a globally optimal order to examine ambiguous patterns to minimize the overall computation and the number of scans through the database. When the pattern is relatively short, border collapsing achieves a comparable performance as the level-wise search. However, when the pattern is long (as in the applications we addressed earlier in this paper), the border collapsing technique can yield substantial improvement. Its efficiency is demonstrated in experimental studies presented in the next section. We also want to mention that the proposed algorithm can also be used to mine long patterns with the support model efficiently.

# 5 Experimental Results

## 5.1 Robustness of Match Model

Since misrepresentation of symbols may occur, some symbols may be substituted by others in the input sequence database. In this subsection, we compare the robustness of the support model and the match model with respect to varying degrees of noise. We use a protein database [12] that consists of 600K sequences of amino acids[10] as the *standard database* to generate *test databases* by embedding random noises. A probability $\alpha$ is introduced to control the degree of noise. $\alpha = 0$ means no misrepresentation and a higher value of $\alpha$ implies a greater degree of misrepresentation. For each sequence $S$ in the standard database, its counterpart $S_t$ in the test database is generated as follows: for each amino acid $d_i$ in the $S$, it will remain as $d_i$ with probability $1 - \alpha$ and will be substituted by another amino acid $d_j$ ($1 \leq j \leq m$ and $j \neq i$) with probability $\frac{\alpha}{m-1}$, where $m = 20$ is the number of distinct amino acids. $S$ and $S_t$ would have the same length. Each entry $C(d_i, d_j)$ in the corresponding compatibility matrix is $1 - \alpha$ if $i = j$ and is $\frac{\alpha}{m-1}$ otherwise. We also experienced with different noise distribution and, after a thorough study, we found that the degree of noise (rather than the distribution of the noise) plays a dominant role in the robustness of the model. Therefore, we only report the results under the assumption of uniform noise due to space limitations.

Let $R_M$ be the set of patterns discovered via match model and $R_S$ be the set of patterns discovered via support model on the standard sequence database with the same threshold $min\_match = min\_support = 0.001$. It is expected that $R_S$ equal to $R_M$ since the match model is equivalent to the support model if no noise is assumed. This set of patterns will be used as the standard to justify the quality of the results generated from test database. Given a test database, let $R'_M$ be the set of patterns discovered in the match model and $R'_S$ be the set of discovered patterns under the support model. Figure 7(a) and (b) show the accuracy and completeness of these two models with respect to various degree of noise $\alpha$, respectively. The accuracies of the match model and the support model are defined as $\frac{|R'_M \cap R_M|}{|R'_M|}$ and $\frac{|R'_S \cap R_S|}{|R'_S|}$ respectively. On the other hand, the completeness for the match and the support models are defined as $\frac{|R'_M \cap R_M|}{|R_M|}$ and $\frac{|R'_S \cap R_S|}{|R_S|}$, respectively. Intuitively, the accuracy describes how selective the model is while the completeness captures how well the model covers the expected results. For the match model, both the accuracy and the completeness are very high (i.e., more than $95\%$) due to the compensation of the compatibility matrices. This demonstrates that the match model is able to handle the noise in a proper manner. However, the support model appears vulnerable to the noise/misrepresentation in the data. When the misrepresentation factor $\alpha$ increases, the quality of the results by the support model degrades significantly. For example, when $\alpha = 0.6$, the accuracy and completeness of the support model are $61\%$ and $33\%$, respectively.

With a given degree of noise (e.g., $\alpha = 0.1$), the accuracy and completeness of the support and match models with different number of non-eternal symbols in a pattern are shown in Figure 7 (c) and (d), respectively. With more non-eternal

[10] Each sequence consists of dozens to thousands of amino acids.

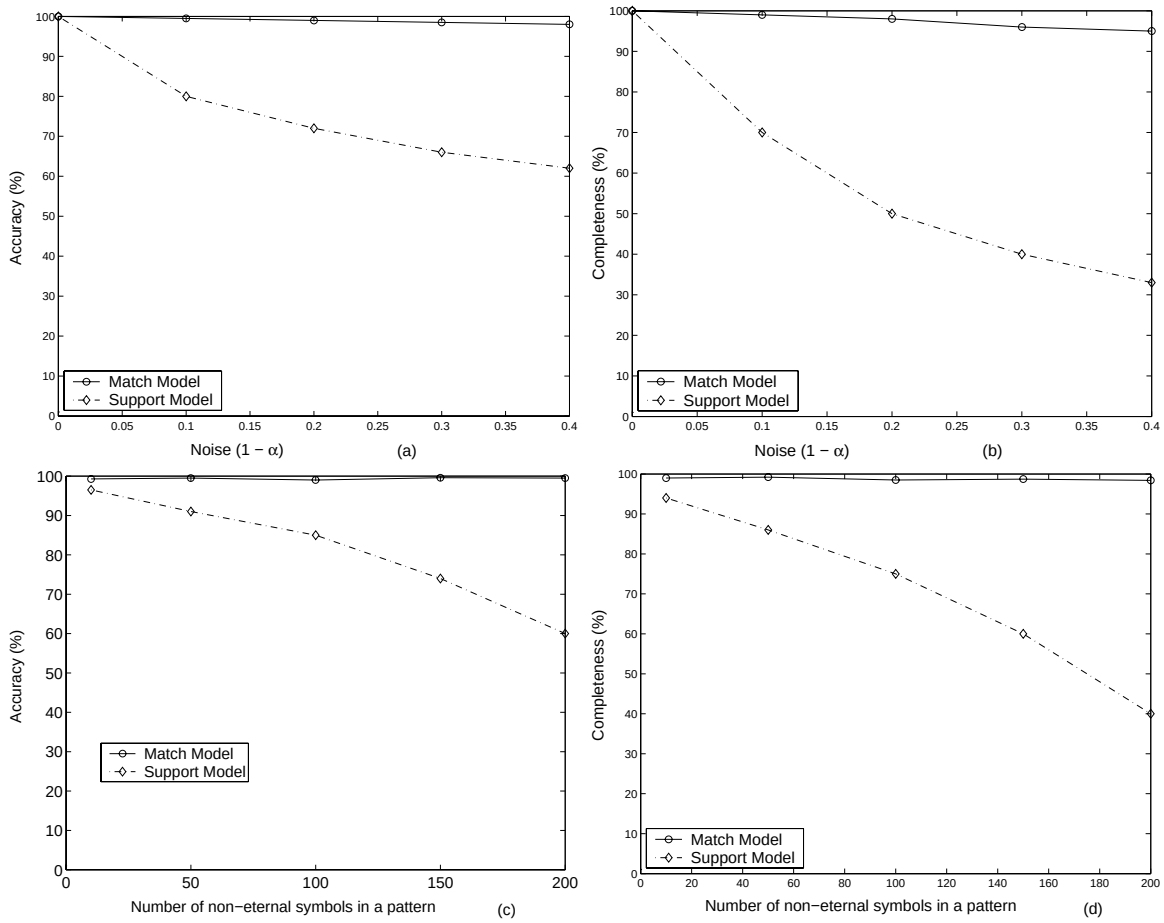



Figure 7: Accuracy and Completeness of the Two Models

symbols, the quality of the support degrades while the quality of the match model remains constant. This is due to the fact that for a pattern with a large number of non-eternal symbols, there is a higher probability that at least one position mutates.

We also experimented with the test database generated according to the BLOSUM50 matrix [10] which is widely used to characterize the likelihood of mutations between amino acids in the computational biology community. We then use both the support and the match model to mine patterns on the test database with the minimum threshold set to 0.001. Comparing to the patterns discovered on the standard database, we found that both the accuracy and the completeness of the match model are well over $99\%$ while the accuracy and the completeness of the support model are $70\%$ and $50\%$, respectively.

In the previous experiments, we assume that our knowledge of noise is "perfect", i.e., the compatibility matrix truly reflects the behavior of noise. However, in reality, the available compatibility matrix itself may contain some error and is indeed a (good) approximation of the real compatibility among symbols. This is typically the case when the compatibility matrix is generated from empirical studies. Thus, the quality of the compatibility matrix also plays a role in the performance of the match model. We also did some experiments to explore the robustness of the match model in this respect. Figure 8 shows the accuracy and completeness of the match model with respect to the amount of error contained in the compatibility matrix. In this experiment, we choose the test database generated from $\alpha = 0.2$. The error is incorporated into the compatibility matrix in the following manner. For each symbol $d_i$, the value of $C(d_i, d_i)$ is varied by $e\%$ (equally likely to be increased or decreased). The rest entries $C(d_j, d_i)$ $(j \neq i)$ in the same column are adjusted accordingly so that the summation $\Sigma_{1 \leq j \leq m} C(d_j, d_i)$ is still 1. Even though the completeness and accuracy degrades with the increase of error, the degradation is moderate even with high error rate. For example, with $10\%$ error, our match model still can achieve $88\%$ accuracy and $85\%$ completeness. Note

that the error in the compatibility matrix is usually very limited (i.e., $\ll 10\%$) in practice and hence the match model can perform very well.

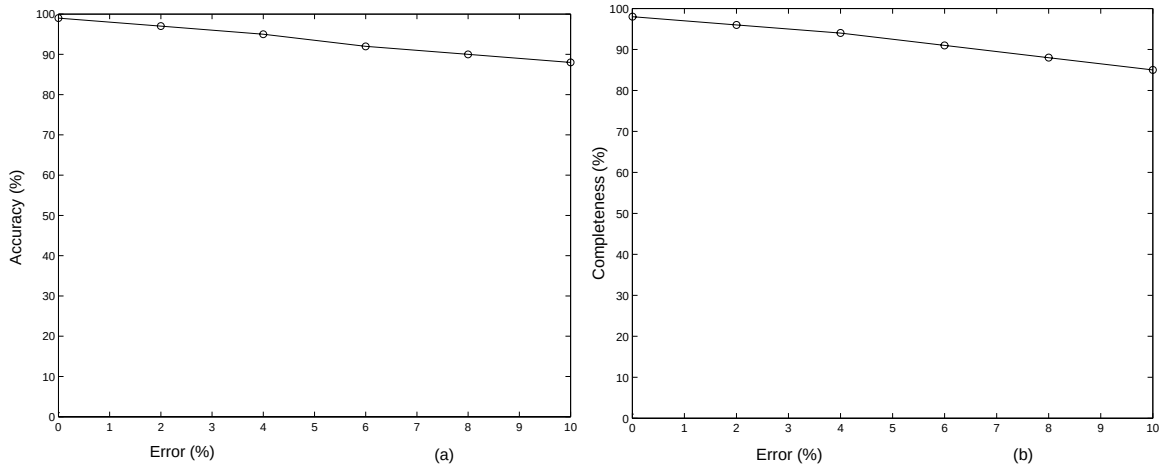



Figure 8: Robustness of Match Model

In addition, we also carried out the experiments on real trace logs of a multimedia search engine ScourNet [8] and obtained a similar result (i.e., accuracy/completeness with respect to both the degree of noise and the amount of error in the compatibility matrix). We will not present the details due to space limitations [28].

## 5.2 Number of Candidate Patterns

The number of candidate patterns[11] that need to be evaluated is a crucial factor towards the overall efficiency of any approach. In this subsection, we analyze the number of candidate patterns at each level of the super-/sub-pattern lattice for both the support and match models. The same test databases generated from the protein sequences in the previous subsection are used to compute the number of candidate patterns. Since they all have the similar behavior, we only report the number of candidate patterns for the test database with $\alpha = 0.2$ and for the threshold $min\_support = min\_match = 0.001$ in Figure 9. In this figure, the number of candidate patterns peak at the 10th 14th level, then begins to diminish. However, the number

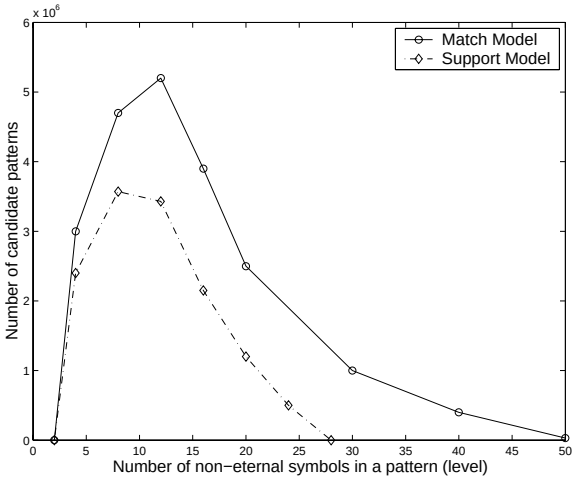



Figure 9: Number of candidate patterns at each level

[11] A candidate pattern is a pattern whose sub-patterns are all frequent.

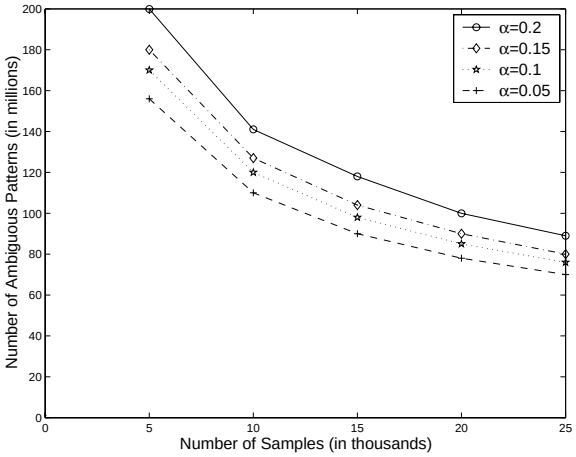



Figure 10: Ambiguous Patterns w.r.t. Sample Size

of candidate patterns under the match model diminishes at a much slower pace than that of the support model. Even at the fiftieth level, there are still dozens of thousands of candidate patterns under the match model. This phenomenon makes mining pattern under the match model a more challenging task. In general, the support model generates less candidate patterns at each level due to the fact that the presence of noise dilutes the strength of some qualified patterns and makes them disqualified. In contrast, the match model tends to restore the original strength of each diluted pattern and hence minimizes the impact of noise.

## 5.3 Sample size

As mentioned previously, the sample size could affect the number of ambiguous patterns significantly and in turn impact the overall efficiency of our approach greatly. Figure 10 shows the number of ambiguous patterns with respect to the number of samples. The number of ambiguous patterns decrease significantly as a function of the number of samples. Also with greater degree of noise (i.e., larger $\alpha$), the number of ambiguous patterns increases.

## 5.4 Spread of Match $R$

For any pattern, instead of applying the default value $R = 1$, a much constrained spread $R$ can be estimated from the match of each involved symbol in the pattern and used to provide a tighter Chernoff bound. This leads to a significantly reduced number of ambiguous patterns. The same test database generated in the previous subsection are used here. Figure 11(a) shows the average match spread $R$ of a candidate pattern with respect to the number of non-eternal symbols. $R$ of a candidate pattern is the minimum match of its involved symbols. Let $R(P)$ be the spread of the match of a pattern $P = d_1 d_2 \ldots d_l$, then $R(P) = \min\{match(d_1), match(d_2), \ldots, match(d_l)\}$. With more non-eternal symbols, the spread $R$ becomes tighter. With higher degree of noise (i.e., larger value of $\alpha$), the match spread reduces because the noise dilutes the strength of the true patterns. In Figure 11(b), we compute the ratio of the number of ambiguous patterns produced using the constrained $R$ over that with the default $R = 1$. It is evident that the number of ambiguous patterns can be reduced to less than $20\%$ (for pattern with more than 10 non-eternal symbols) when the constrained $R$ is applied. As a matter of fact, a five-folds pruning power is obtained.

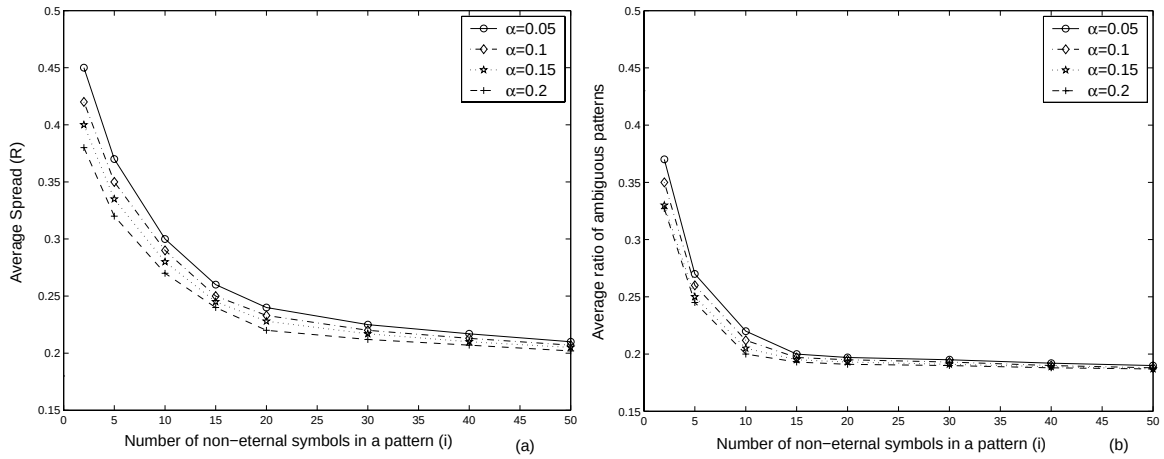



Figure 11: Effects of spread $R$

## 5.5 Effects of Confidence $1 - \delta$

In the previous experiments, we fix confidence $1 - \delta$ as 0.9999. In this subsection, we are examining the effects of different $\delta$. Figure 12 shows the effect of $1 - \delta$ on the number of ambiguous patterns and the accuracy of the results. We assume that 200,00 samples are used for this test. With smaller confidence, the number of ambiguous patterns decreases dramatically because the error bound $\epsilon$ decreases, which implies a much faster response time. On the other hand, the error rate of the algorithm could increase slightly with a smaller confidence as shown in Figure 12(b). The error rate is defined as the ratio of the number of mislabeled patterns over the number of frequent patterns. However, since the Chernoff Bound is a distribution independent estimation, the bound it provides is very conservative. The actual precision of the results is much higher than the specified confidence. For example, when confidence is 0.9, i.e., $\delta = 0.1$, the error rate is around $0.01$. When $1 - \delta = 0.9999$, the error rate can diminish to the order of $10^{-6}$.

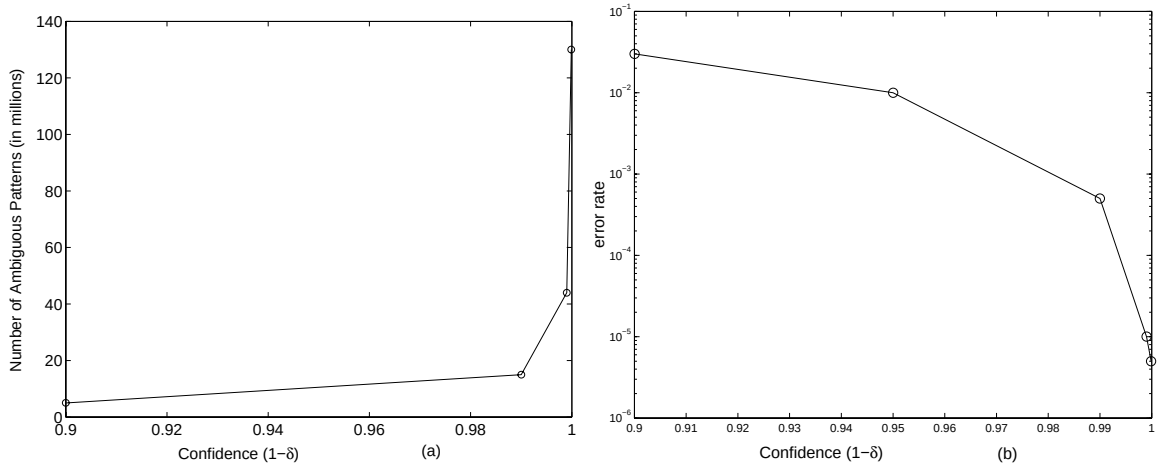



Figure 12: Effects of confidence $1 - \delta$

Figure 13 shows the distribution of the matches of mislabeled patterns in the above experiment. It is clear that over 90% of the missed patterns are those whose real match is within 5% over the threshold, while no pattern missing whose real match is 15% over the threshold. This means that most missing patterns are very close to the threshold. This observation coincides with the theoretical analysis in the previous section.

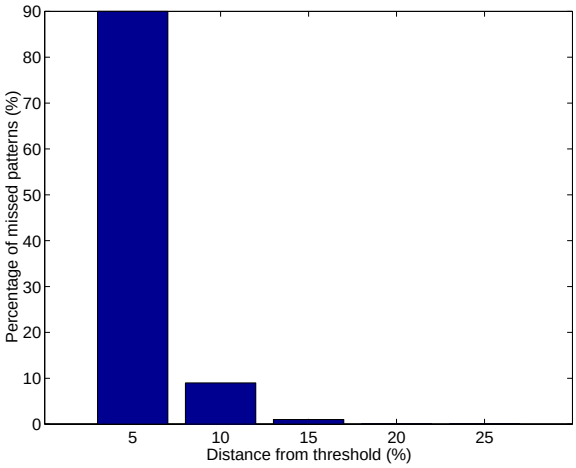



Figure 13: Missing patterns

## 5.6 Performance of Probabilistic Algorithm

The overall efficiency of our probabilistic algorithm is demonstrated in Figure 14. Max-Miner [6] is one of the fastest algorithm for mining frequent long patterns, which employs a look-ahead technique. We adopt the Max-Miner as the deterministic algorithm to compare the performance. (We also notice that the work in [1] yields fast computation time under the assumption that the entire data set can fit into memory. This is obviously not the scenario assumed in this paper. Thus, we do not furnish a comparison with it.) The only modification to the Max-Miner is the computation of match value of a pattern (instead of support value). Another algorithm we compared is the sampling based approach proposed by Toivonen [25]. In this approach, a level-wise search is used to finalize the border of frequent patterns after the sampling. We will refer to this approach as "sampling-based level-wise search" in the following discussion. The primary difference between this approach and our approach is that we employ a much more efficient method, namely border collapsing, to locate the border of frequent patterns. The confidence parameter of our algorithm is set to 0.9999. Figure 14(a) shows the CPU time of these three algorithms with respect to various match thresholds. Figure 14(b) shows the number of scans employed by these three algorithms.

It is evident that our algorithm can substantially reduce the CPU time and the number of scans through the database than both previous proposed schemes. This is due to the efficiency brought by the border collapsing technique. In our algorithm, the number of patterns that need to be examined against the entire database is much less than that in the other two algorithms. More specifically, when the match threshold is relatively high, our approach requires two scans of the sequence database while both Max-Miner and the sampling-based level-wise search requires at least five scans of data. As the match threshold decreases, the border collapsing algorithm requires three or four scans of the database while the other two approaches need 10 or more scans of the database. The significant reduction in number of database scans of our algorithm comes from the combined effect of sampling and border collapsing. We also would like to mention that the sampling-based level-wise approach spends majority of the time on finalizing the border of frequent patterns after estimating the border from the samples. We observed that there is a high likelihood that the final border is "far" from the estimated one and many scans of the data may be required before it is reached. This is because the match value usually changes very little from level to level in the pattern lattice especially when the pattern is long. This effect can clearly be observed from Figure 14(c).

## 5.7 Scalability with respect to the Number of Distinct Symbols $m$

In all above experiments, we utilize the gene expression data which consists of 20 symbols (i.e., amino acids). Now we analyze the performance of our algorithm with respect to the number of distinct symbols, $m$. In this experiment, we employ

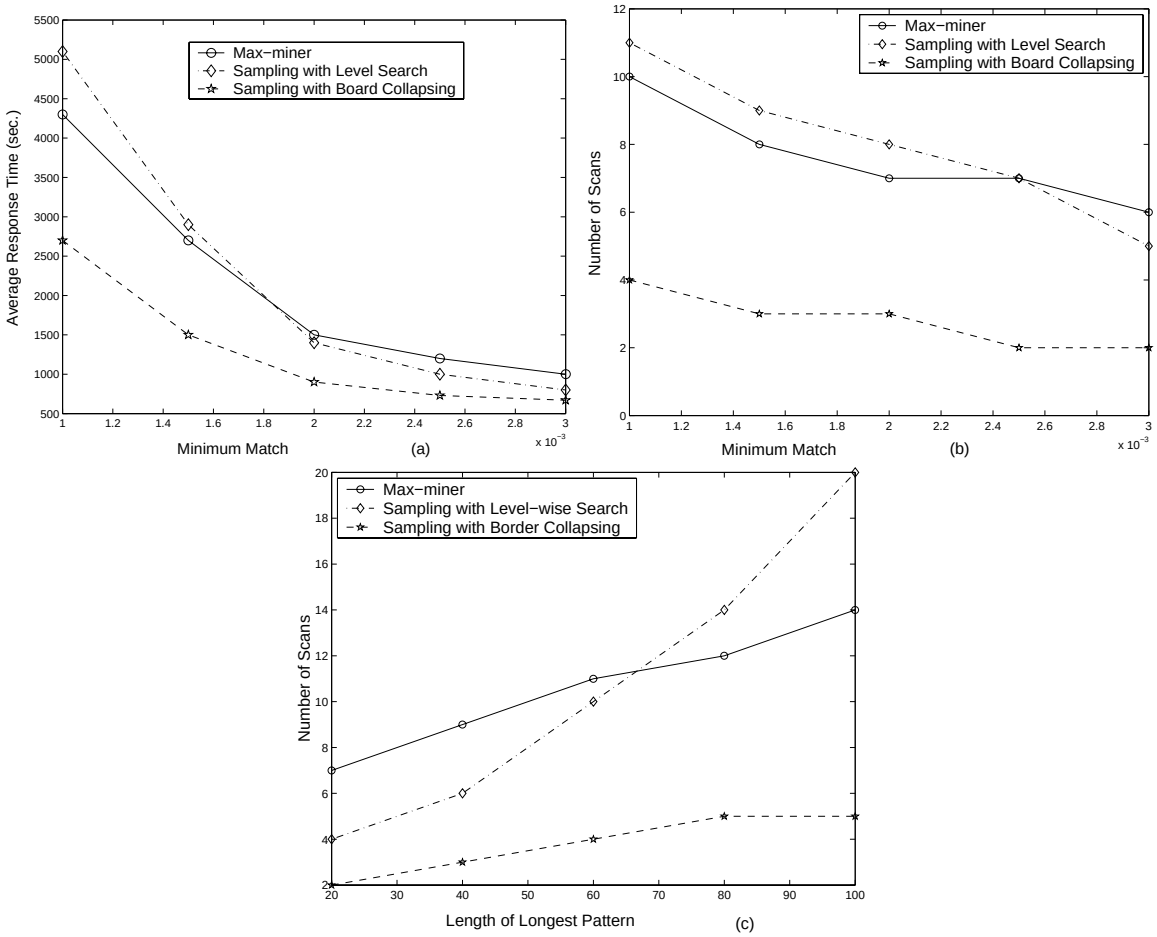



Figure 14: Performance of three algorithms

several synthetic data sets, each of which consists of 100K sequences and each sequence contains 1000 symbols on average. We vary the number of distinct symbols ($m$) in each data set. The minimum match threshold is set to 0.001. A compatibility matrix is constructed for each data set. In reality, most entries in a compatibility matrix is zero or near zero. Thus, the compatibility is generated in such a manner that a symbol is compatible to around 10% of other symbols with various degree. Figure 15(a)(b) shows the number of scans and response time of our algorithm, respectively. The number of scans decreases with the increase of $m$ because less patterns are qualified to be significant. However, this trend does not hold for the response time. The average response time decreases initially, but increases when $m$ gets large (e.g., greater than 10000). This is due to the fact that the size of the compatibility matrix is a quadratic function of $m$ and the computation cost for each scan increases significantly. For example, if $m = 10000$, then it requires about 40MB space to store the compatibility matrix if each non-zero entry occupies 4 Bytes. The performance of our algorithm degrades when $m$ is extremely large. Nevertheless, the algorithm performs very efficiently when the number of distinct symbols is within a reasonable range ($m \leq 10^4$).

# 6 Conclusion

In this paper, we are interested in discovering long sequential patterns in a noisy environment. In this environment, the observed symbol in a sequence may differ from the underlying true value. The concept of *compatibility matrix* is introduced to provide a probabilistic connection from the observation to the underlying true value. A new metric *match* is thus, proposed to capture the "real support" of a pattern which would be expected if a noise-free environment is assumed. Since the length

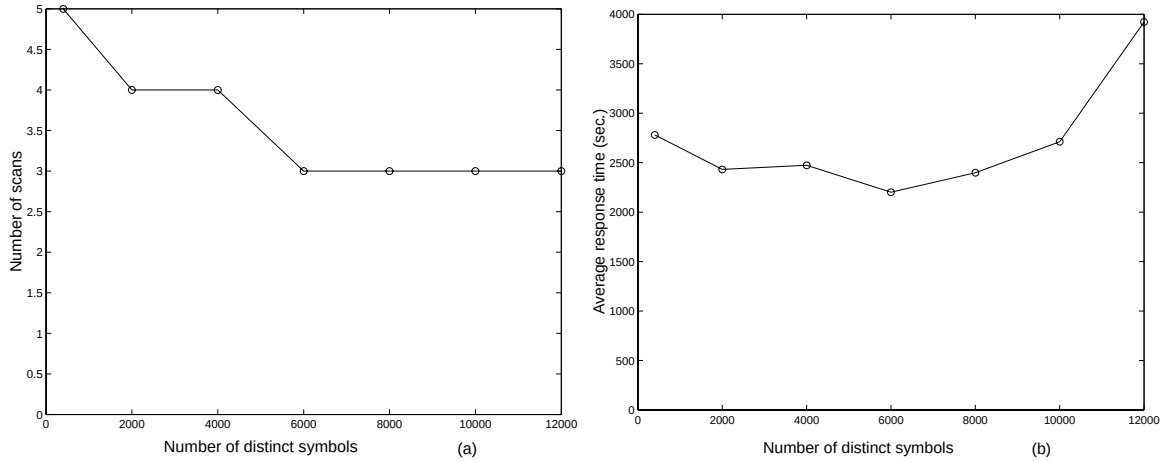



Figure 15: Scalability w.r.t. the number of distinct symbols

of a pattern could be very large, the standard pruning technique developed for the market basket problem may not work efficiently. As a result, a probabilistic algorithm is devised to discover long patterns in a minimal number of scans of the sequence (e.g., 2 to 4) with sufficiently high confidence. Empirical results demonstrate the robustness of the match model (w.r.t. the noise) and the efficiency of the probabilistic algorithm. As a future research direction, we are developing strategies that can further improve the performance of the algorithm for the applications where a huge number of distinct symbols exist (e.g., E-Commerce).

# 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, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *Proc. ACM SIGMOD Conf. on Management of Data*, 207-216, 1993.

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

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

[5] R. Baeza-Yates and G. Navarro. Faster approximate string matching. *Algorithmica*, 23(2) 127-158, 1999.

[6] R. J. Bayardo Jr. Efficiently mining long patterns from databases. *Proc. ACM SIGMOD Conf. on Management of Data*, 85-93, 1998.

[7] 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.

[8] Vincent Busam. Personal Communications, 2000.

[9] P. Domingos and G. Hulten. Mining high-speed data streams. *Proc. ACM SIGKDD*, 71-80, 2000.

[10] R. Durbin, S. Eddy, A. Krough, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, 1998.

[11] 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.

[12] National Center for Biotechnology Information. Available at “http://www.ncbi.nlm.nih.gov”.

[13] 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.

[14] 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.

[15] 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.

[16] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, vol 58, 13-30, 1963.

[17] H. V. Jagadish, R. T. Ng and D. Srivastava. Substring selectivity estimation. *Proc. ACM Symposium on Principles of Database Systems*, 249-260, 1999.

[18] H. V. Jagadish, N. Koudas, and D. Srivastava. On effective multi-dimensional indexing for strings. *Proc. ACM SIGMOD Conference on Management of Data*, 403-414, 2000.

[19] D. Lin and Z. Kedem. Pincer-search: a new algorithm for discovering the maximum frequent set. *Proc. 6th European Conf. on Extending Database Technology*, 1998.

[20] Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, vol. 1, no. 3, 241-258, 1997.

[21] 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.

[22] 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.

[23] R. Srikant and R. Agrawal. Mining generalized association rules. *Proc. 21st Int. Conf. on Very Large Data Bases*, 407-419, 1995.

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

[25] H. Toivonen. Sampling large databases for association rules. *Proc. 22nd Int. Conf. on Very Large Data Base*, 134-145, 1996.

[26] A. Tung, H. Lu, J. Han, and L. Feng. Breaking the barrier of transactions: mining inter-transaction association rules. *Proc. Int. Conf. on Knowledge Discovery and Data Mining*, 1999.

[27] J. Vitter. An efficient algorithm for sequential random sampling. *ACM Transactions on Mathematical Software*, 13(1), 58-67, 1987.

[28] J. Yang, W. Wang, and P. Yu. Mining long sequential patterns in a noisy environment. *IBM Research Report*, 2001.

[29] M. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of sampling for data mining of association rules. *Proc. 7th Int. Workshop on Research Issues in Data Engineering (RIDE–in conjunction with ICDE)*, 42-50, 1997.

[30] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. *Proc. 3rd Int. Conf. on Knowledge Discovery in Databases and Data Mining*, 283-286, 1997.

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