ApproxMAP: Approximate Mining of Consensus Sequential Patterns

Hye-Chung (Monica) Kum[†] **Jian Pei**[‡] **Wei Wang**[†] **Dean Duncan**[†]

[†] University of North Carolina at Chapel Hill, {kum, weiwang}@cs.unc.edu, dfduncan@email.unc.edu

[‡] State University of New York at Buffalo, jianpei@cse.buffalo.edu

Abstract

Sequential pattern mining is an important data mining task with broad applications. However, conventional methods may meet inherent difficulties in mining databases with long sequences and noise. They may generate a huge number of short and trivial patterns but fail to find interesting patterns approximately shared by many sequences. To attack these problems, in this paper, we propose the theme of *approximate sequential pattern mining* roughly defined as *identifying patterns approximately shared by many sequences.* We present an efficient and effective algorithm, ApproxMAP (for APPROXimate Multiple Alignment Pattern mining), to mine consensus patterns from large sequence databases. The method works in two steps. First, sequences are clustered by similarity. Then, consensus patterns are mined directly from each cluster through multiple alignment. A novel structure called weighted sequence is used to compress the alignment result. For each cluster, the longest consensus pattern best representing the cluster is generated from its weighted sequence. Our extensive experimental results on both synthetic and real data sets show that ApproxMAP is robust to noise and both effective and efficient in mining approximate sequential patterns from noisy sequence databases with lengthy sequences. In particular, we report a successful case of mining a real data set which triggered important investigations in welfare services.

1 Introduction

Given a sequence database, a sequential pattern is a subsequence that appears frequently in the database. Since it has been proposed in [2], mining sequential patterns in large databases has become an important data mining task and has broad applications, such as business analysis, web mining, security, and bio-sequences analysis.

For example, supermarkets often collect customer purchase records in sequence databases. Sequential patterns in customer purchase database may indicate customers' buying habits and can be used to design promotion campaigns.

Most of the previous researches on sequential pattern mining adopt an *exact matching approach.* A pattern is supported by sequences in the database only if the pattern *exactly repeats* in the sequences. A sequence is regarded as a sequential pattern if the number of its occurrences in the sequence database passes a user specified support threshold. In general, two kinds of algorithms have been developed to find the complete set of sequential patterns.

First, the *appriori-like breadth-first search methods*, such as GSP [21] and SPADE [26], conduct a level-by-level candidate-generation-and-test pruning following the *Apriori property: any super-pattern of an infrequent pattern cannot be frequent.* In the first scan, they find the length-1 sequential patterns, i.e., single items frequent in the database. Then, length-2 candidates are assembled using length-1 sequential patterns. The sequence database is scanned the second time and length-2 sequential patterns are found. At each level, only potentially frequent candidates are generated and tested.

Second, the *projection-based depth-first search methods*, such as PrefixSpan [16], FreeSpan [12], and SPAM [3], avoid the costly candidate-generation-and-test operations by growing long patterns from short ones. Once a sequential pattern is found, all sequences containing that pattern are collected as a projected database. Local frequent items are found in the projected databases and used to extend the current pattern to longer ones.

Although sequential pattern mining has been extensively studied and many methods have been proposed, there are two inherent obstacles within the conventional framework.

- *Most methods mine sequential patterns with exact matching.* That is, a pattern gets support from a sequence in the database if and only if the pattern is exactly contained in the sequence. However, the exact matching approach often may not find general long patterns in the database. For example, many customers may share similar buying habits, but few of them follow an exactly same buying pattern.

Thus, to find non-trivial interesting long patterns, we must consider mining *approximate* sequential patterns.

- *Most methods mine the complete set of sequential patterns.* When long patterns exist, mining the complete set of patterns is ineffective and inefficient. For example, if $\langle a_1 \cdots a_{20} \rangle$ is a sequential pattern, then each of its subsequence is also a sequential pattern. There are $(2^{20} - 1)$ patterns in total! On one hand, it is very hard for users to understand and manage a huge number of patterns. On the other hand, computing and storing a huge number of patterns is very expensive or even computationally prohibitive. In many situations, a user may just want the *long patterns* that cover many short ones.

  Recently, mining compact expressions for frequent patterns, such as max-patterns [4] and frequent closed patterns [15], has been proposed and studied in the context of frequent itemset mining. However, mining max-sequential patterns or closed sequential patterns is far from trivial. Furthermore, in a noisy sequence database, the number of max- or closed sequential patterns still can be huge, and many of them are trivial for users.

In this paper, we propose an effective and efficient framework for mining sequential patterns in databases of long sequences, and make the following contributions.

- We propose the theme of *approximate sequential pattern mining*. The general idea is that, instead of finding exact patterns, we identify patterns approximately shared by many sequences.

- Instead of mining a huge set of patterns, we propose to mine *consensus patterns* from databases of long sequences. Intuitively, a consensus pattern is shared by many sequences and covers many short patterns. Consensus patterns are more expressive and are more useful in many applications.

- We develop an efficient algorithm, ApproxMAP (for APPROXimate Multiple Alignment Pattern mining), to mine consensus sequential patterns from large databases. ApproxMAP finds the underlying consensus patterns directly from multiple alignment. It is effective and efficient for mining long sequences and is robust to noise. We conduct an extensive and systematic performance study over synthetic data sets. The results show that ApproxMAP is effective and scalable in mining large sequence databases with long patterns.

- We report a successful case study of ApproxMAP in a real application: mining the North Carolina State Welfare Services database. We illustrate some interesting patterns mined by ApproxMAP. Moreover, the mining result of ApproxMAP triggered some important investigations which help to improve the real business. The application results show that approximate sequential pattern mining can find interesting patterns that is highly promising in many applications.

The remainder of the paper is organized as follows. Section 2 defines the problem. Section 3 and 4 demonstrate the ApproxMAP method in detail. An extensive performance study is reported in Section 5. Particularly, in Section 6, we showcase a successful case of mining the North Carolina State Welfare Services database using ApproxMAP. In Section 7, we discuss related work. Section 8 concludes the paper.

2 Problem Definition

Let $I = \{i_1, \ldots, i_l\}$ be a set of *items*. An *itemset* $X = \{i_{j_1}, \ldots, i_{j_k}\}$ is a subset of I. Conventionally, itemset $X = \{i_{j_1}, \ldots, i_{j_k}\}$ is also written as $(x_{j_1} \cdots x_{jk})$. A *sequence* $S = \langle X_1 \ldots X_n \rangle$ is an ordered list of itemsets, where $X_1, \ldots, X_n$ are all itemsets. A *sequence database* SDB is a multi-set of sequences.

A sequence $S_1 = \langle X_1 \cdots X_n \rangle$ is called a *subsequence* of sequence $S_2 = \langle Y_1 \cdots Y_m \rangle$, and S_2 a *super-sequence* of S_1, if $n \leq m$ and there exist integers $1 \leq i_1 < \cdots < i_n \leq m$ such that $X_j \subseteq Y_{i_j}$ $(1 \leq j \leq n)$. Given a sequence database SDB, the *support* of a sequence P, denoted as $sup(P)$, is the number of sequences in SDB that are super-sequences of P. Conventionally, a sequence P is called a *sequential pattern* if $sup(P) \geq min\_sup$, where min_sup is a user-specified *minimum support threshold*.

In many applications, people prefer long sequential patterns shared by many sequences. However, due to noise, it is very difficult to find a long sequential pattern exactly shared by many sequences. Instead, many sequences may *approximately* share a long sequential pattern.

Motivated by the above observation, we introduce the notion of *mining approximate sequential patterns*. Let $dist$ be a normalized distance measure of two sequences with domain $[0, 1]$. For sequences S, S_1 and S_2, if $dist(S, S_1) < dist(S, S_2)$, then S_1 is said be *more similar* to S than S_2 is.

Naïvely, we can extend the conventional sequential pattern mining framework to get an approximate sequential pattern mining framework as follows. Given a *minimum distance threshold* min_dist, the *approximate support* of a sequence P in a sequence database SDB

is defined as $\widetilde{sup}(P) = \|\{S|(S \in SDB) \wedge (dist(S,P) \leq min\_dist)\}\|$. (Alternatively, the approximate support can be defined as $\widetilde{sup}(P) = \sum_{S \in SDB} dist(S,P)$. All the following discussion retains.) Given a minimum support threshold min_sup, all sequential patterns whose approximate supports passing the threshold can be mined.

Before we make any commitment, let us examine whether the above framework is good. Unfortunately, it may suffer from the following two problems.

First, *the mining may find many short and probably trivial patterns.* Short patterns tend to be easier to get similarity counts from the sequences than long patterns. Thus, short patterns may overwhelm the results.

Second, *the complete set of approximate sequential patterns may be larger than that of exact sequential patterns and thus difficult to understand.* By approximation, a user may want to get and understand the general trend and ignore the noise. However, a naïve output of the complete set of approximate patterns in the above framework may generate many (trivial) patterns and thus ruin the mining.

Based on the above analysis, we need to explore a more effective solution. We propose ApproxMAP, a *cluster and multiple alignment-based* approach, which works in two steps.

1. **Sequences in a database are clustered based on similarity**. Sequences in the same cluster may approximately follow some similar patterns.

2. **The longest approximate sequential pattern for each cluster is generated.** It is called the *consensus pattern.* To extract consensus patterns, a weighted sequence is derived for each cluster using multiple alignment to compress the sequential pattern information in the cluster. And then the longest consensus pattern best representing the cluster is generated from the weighted sequence.

Compared to the exact matching models and the naïve approximate pattern mining model, ApproxMAP has several distinct features.

1. ApproxMAP finds approximate sequential patterns from clusters based on multiple alignment. Even though a pattern is not exactly contained by many sequences, as long as it is shared by many sequences in the sense of multiple alignment, it will be identified by ApproxMAP.

2. In usual, each cluster has only one consensus pattern. Thus, many short and trivial patterns can be avoided. Since one cluster has multiple sequences, and one sequence only joins one cluster, the number of patterns mined by ApproxMAP is small.

3. ApproxMAP does not adopt a unified support threshold to prune patterns. Instead, it generates a pattern from each cluster regardless of its size. By doing so, ApproxMAP can find patterns strongly followed by a small number of sequences in the database. Such "rare" but "confident" patterns can be of great value in practice.

In the next two sections, we will discuss how to cluster sequences according to similarity and how to align sequences and generate patterns from clusters of sequences.

3 Clustering Sequences

In this section, we discuss *how to cluster sequences in a database.* First, let us develop a distance measure.

In general, the *hierarchical edit distance* is commonly used as a distance measure for sequences. It is defined as the minimum cost of editing operations (i.e., insertions, deletions, and replacements) required to change one sequence to the other. An insertion operation on S_1 to change it towards S_2 is equivalent to a deletion operation on S_2 towards S_1. Thus, an insertion operation and a deletion operation have the same cost. We use $INDEL()$ to denote an insertion or deletion operation, and $REPL()$ to denote a replacement operation. Often, the following inequality is assumed.

$$REPL(X,Y) \leq INDEL(X) + INDEL(Y)$$

Given two sequences $S_1 = \langle X_1 \cdots X_n \rangle$ and $S_2 = \langle Y_1 \cdots Y_m \rangle$, the hierarchical edit distance between X and Y can be computed by dynamic programming using the following recurrence relation.

$$\begin{aligned}
&D(0,0) = 0 \\
&D(i,0) = D(i-1,0) + INDEL(X_i) \text{ for } (1 \leq i \leq n) \\
&D(0,j) = D(0,j-1) + INDEL(Y_j) \text{ for } (1 \leq j \leq m) \\
&D(i,j) = \min \begin{cases} D(i-1,j) + INDEL(X_i) \\ D(i,j-1) + INDEL(Y_j) \\ D(i-1,j-1) + REPL(X_i,Y_j) \end{cases} \\
&\qquad \text{for } (1 \leq i \leq n) \text{ and } (1 \leq j \leq m)
\end{aligned} \tag{3.1}$$

To make the edit distances comparable between sequences with various lengths, we normalize the results by dividing the hierarchical edit distance by the length of the longer sequence in the pair, and call it the *normalized edit distance.* That is,

$$dist(S_1,S_2) = \frac{D(n,m)}{\max\{\|S_1\|,\|S_2\|\}} \tag{3.2}$$

To make the hierarchical edit distance applicable to sequences of sets, we need to define the cost of edit operations (i.e., INDEL() and REPL() in Equation 3.1) properly. Here, we adopt the *normalized set difference* as the cost of replacement of sets.

$$REPL(X,Y) = \frac{\|(X-Y)\cup(Y-X)\|}{\|X\|+\|Y\|} = \frac{\|X\|+\|Y\|-2\|X\cap Y\|}{\|X\|+\|Y\|} \tag{3.3}$$

This measure has a nice property as follows.

$$0 \le REPL() \le 1$$

Moreover, it is a metric [5]. Following Equation 3.3, the cost of an insertion/deletion is

$$INDEL(X) = REPL(X, \emptyset) = 1,$$

where X is an itemset. Table 1 shows some examples on the calculation of normalized edit distances.

X	Y	$REPL(X,Y)$	X	Y	$REPL(X,Y)$
(a)	(a)	0	(a)	(b)	1
(a)	(ab)	$\frac{1}{3}$	(ab)	(cd)	1
(ab)	(ac)	$\frac{1}{2}$	(a)	$()$	1

Table 1: Normalized edit distances between some itemsets.

Clearly, the normalized set difference is equivalent to the Sorensen coefficient.[1] It is also an index similar to the Jaccard coefficient[2] [14], except that it gives more weight to the common elements.

Using the hierarchical edit distance (Equation 3.2), we can apply a density-based clustering algorithm to cluster sequences. A density based clustering method groups the data according to the valley of the density function. The valleys can be considered as natural boundaries that separate the modes of the distribution [13, 7].

How can we define the density for a sequence? Intuitively, a sequence is "*dense*" if there are many sequences similar to it in the database. A sequence is "*sparse*", or "*isolated*", if it is not similar to any others, such as an outlier. Technically, the density of a sequence can be measured by a quotient of the number of similar sequences (nearest neighbors) against the space occupied by such similar sequences.

In particular, for each sequence S_i in a database $\mathcal{S}$, let $d_1, \ldots, d_k$ be the k smallest non-zero values of $dist(S_i, S_j)$, where $S_j \neq S_i$, is a sequence in $\mathcal{S}$. Then,

$$\begin{array}{rl} & Density(S_i) = \frac{n}{\|\mathcal{S}\| d^3} \\ \text{where} & d = \max\{d_1, \ldots, d_k\} \\ \text{and} & n = \|\{S_j \in \mathcal{S} | dist(S_i, S_j) \le d\}\|. \end{array} \tag{3.4}$$

In Equation 3.4, n is the number of sequences in the k-nearest neighbor space (including all ties). Here k is a user-specified parameter.

We adopt an algorithm from [19] as follows.

Algorithm 3.1. (Uniform kernel k-NN clustering)

Input: a set of sequences $\mathcal{S} = \{S_i\}$, number of neighbor sequences k;

Output: a set of clusters $\{C_j\}$, where each cluster is a set of sequences;

Method:

1. **Initialize every sequence as a cluster**.
   For each sequence S_i in cluster C_{S_i}, set $Density(C_{S_i}) = Density(S_i)$.
2. **Merge nearest neighbors based on the density of sequences**.
   For each sequence S_i, let $S_{i_1}, \ldots, S_{i_n}$ be the nearest neighbor of S_i, where n is defined in Equation 3.4. For each $S_j \in \{S_{i_1}, \ldots, S_{i_n}\}$, merge cluster C_{S_i} containing S_i with a cluster C_{S_j} containing S_j, if $Density(S_i) < Density(S_j)$ and there exists no S'_j having $dist(S_i, S'_j) < dist(S_i, S_j)$ and $Density(S_i) < Density(S'_j)$. Set the density of the new cluster to $\max\{Density(C_{S_i}), Density(C_{S_j})\}$.
3. **Merge based on the density of clusters**.
   For all sequences S_i such that S_i has no nearest neighbor with density greater than that of S_i, but has some nearest neighbor, S_j, with density equal to that of S_i, merge the two clusters C_{S_j} and C_{S_i} containing each sequence if $Density(C_{S_j}) > Density(C_{S_i})$. This step is to merge "*plateau neighbor regions*". ■

It is easy to show that the above algorithm has complexity $O(kN_{seq})$.

Intuitively, in Algorithm 3.1, two clusters are merged if they are similar (in the sense of normalized edit distance). Thus, sequences in a resulting cluster are similar to each other.

The key parameter for the clustering process in Algorithm 3.1 is k, the number of nearest neighbors that the algorithm will search. A larger k value tends

[1]The Sorensen coefficient is defined as follows.

$$\begin{aligned} D_s(A,B) &= 1 - \frac{2\|A\cap B\|}{\|A-B\|+\|B-A\|+2\|A\cap B\|} \\ &= \frac{\|A\|+\|B\|-2\|A\cap B\|}{\|A\|+\|B\|} \\ &= REPL(A,B) \end{aligned}$$

[2]The Jaccard coefficient in dissimilarity notation is defined as follows.

$$\begin{aligned} D_J(A,B) &= 1 - \frac{\|A\cap B\|}{\|A\cup B\|} \\ &= 1 - \frac{\|A\cap B\|}{\|A-B\|+\|B-A\|+\|A\cap B\|} \end{aligned}$$

to merge more sequences, and results in a smaller number of large clusters, while a smaller k value tends to break up clusters. The benefit of using a small k value is that the algorithm can detect less frequent patterns. The tradeoff is that it may break up clusters representing strong patterns to generate multiple similar patterns. As shown in our performance study, in many applications, a value of k in the range from 3 to 10 works well.

4 Multiple Alignment and Pattern Generation

Once sequences are clustered by similarity, sequences within a cluster are similar to each other. Now, the problem becomes *how to summarize the general patterns in clusters and discover the trend.* In this section, we develop a method using multiple alignment. First, we discuss how to align sequences in a cluster, then we explore how to summarize sequences and generate patterns.

4.1 Multiple Alignment of Sequences

The global alignment of sequences is obtained by inserting empty itemsets (i.e., $\emptyset$) into sequences such that all the sequences have the same number of itemsets. The empty itemsets can be inserted into the front or the end of the sequences, or between any two consecutive itemsets [11].

The edit distance between two sequences S_1 and S_2 can be calculated by comparing itemsets in the aligned sequences one by one. If S_1 and S_2 have X and Y as their i^{th} aligned itemsets, respectively, where $(X \neq \emptyset)$ and $(Y \neq \emptyset)$, then a $REPL(X, Y)$ operation is required.

Otherwise, (i.e., S_1 and S_2 have X and $\emptyset$ as their i^{th} aligned itemsets, respectively) an $INDEL(X)$ operation is needed. The optimal alignment is the one in which the edit distance between the two sequences is minimized. Clearly, the optimal alignment between two sequences can be calculated by dynamic programming using Recurrence Relation 3.1.

In general, for a cluster C with n sequences $S_1, \ldots, S_n$, finding the *optimal global alignment* that minimizes

$$\sum_{j=1}^{n}\sum_{i=1}^{n} dist(S_i, S_j)$$

is an NP-hard problem [11], and thus is impractical for mining large sequence databases with many sequences. We need to look for some heuristics.

In a cluster, some sequences may be similar to many other sequences in the cluster. In other words, such sequences may have many close neighbors in terms of similarity. These sequences are most likely to be closer to the underlying patterns than the other sequences. It is more likely to get an alignment close to the optimal one, if we start the alignment with such "*seed*" sequences.

Intuitively, the *density* defined in Equation 3.4 measures the similarity between a sequence and its nearest neighbors. Thus, a sequence with a high density means that it has some neighbors very similar to it, and it is a good candidate for a "*seed*" sequence in the alignment. Based on the above observation, in ApproxMAP, we use the following heuristic to apply multiple alignment to sequences in a cluster.

HEURISTIC 1. *If sequences in a cluster C are aligned in the density-descending order, the alignment result tends to be good.*

As the first step in the clustering (see Algorithm 3.1), the density for each sequence is calculated. We only need to sort all sequences within a cluster in the density descending order.

How can we store the alignment results effectively? Ideally the result should be in a form such that we can easily align the next sequence to the current alignment. This will allow us to build a summary of the alignment step by step until all sequences in the cluster have been aligned. Furthermore, various parts of a general pattern may be shared with different strengths, i.e., some items are shared by more sequences and some by less sequences. The result should reflect the strengthes of items in the pattern.

Here, we propose a notion of weighted sequence as follows. A weighted sequence WS=$\langle X_1 : v_1, \ldots, X_l : v_l\rangle : n$ carries the following information:

1. the current alignment has n sequences, and n is called the *global weight* of the weighted sequence;

2. in the current alignment, v_i sequences have a non-empty itemset X_i aligned in the i^{th} itemset, where $(1 \leq i \leq l)$;

3. an itemset in the alignment is in the form of $X_i = (x_{j_1} : w_{j_1}, \ldots, x_{j_m} : w_{j_m})$, which means, in the current alignment, there are w_{j_k} sequences that have item x_{j_k} in the i^{th} position of the alignment, where $(1 \leq i \leq l)$ and $(1 \leq k \leq m)$.

The notations and the ideas are demonstrated in the following example.

EXAMPLE 1. (MULTIPLE ALIGNMENT) Suppose that, in a cluster C, there are 5 sequences as shown in the left two columns in Table 2. The density descending order of these sequences is S_3-S_2-S_4-S_5-S_1. The sequences are aligned as follows.

Seq-id	Sequence	Alignment				
S_1	$\langle(ag)(f)(bc)(ae)(h)\rangle$	$\langle(ag)$	(f)	(bc)	(ae)	$(h)\rangle$
S_2	$\langle(ae)(h)(b)(d)\rangle$	$\langle(ae)$	(h)	(b)	(d)	$\rangle$
S_3	$\langle(a)(b)(de)\rangle$	$\langle(a)$		(b)	(de)	$\rangle$
S_4	$\langle(a)(bcg)(d)\rangle$	$\langle(a)$		(bcg)	(d)	$\rangle$
S_5	$\langle(bci)(de)\rangle$	$\langle$		(bci)	(de)	$\rangle$
Weighted sequence		$\langle(a:4,e:1,g:1):4$	$(f:1,h:1):2$	$(b:5,c:3,g:1,i:1):5$	$(a:1,d:4,e:3):5$	$(h:1):1\rangle:5$

Table 2: Sequences in a cluster and the complete alignment.

S_3	$\langle(a)$		(b)	$(de)\rangle$	
S_2	$\langle(ae)$	(h)	(b)	$(d)\rangle$	
WS_1	$\langle(a:2,e:1):2$	$(h:1):1$	$(b:2):2$	$(d:2,e:1):2\rangle$	$:2$

Figure 1: S_3 and S_2 are aligned resulting in WS_1.

First, sequences S_3 and S_2 are aligned as shown in Figure 1.

Here, we use a *weighted sequence* WS_1 to summarize and compress the information about the alignment. Since the first itemsets of S_3 and S_2, (a) and (ae), are aligned, the first itemset in the weighted sequence WS_1 is $(a:2,e:1):2$. It means that the two sequences are aligned in this itemset, and a and e appear twice and once, respectively. The second itemset in WS_1, $(h:1):1$, means there is only one sequence with an itemset aligned in this itemset, and item h appears once. ■

After the first step, we need to iteratively align other sequences with the current weighted sequence. The weighted sequence does not explicitly keep the information about various itemsets in the sequences. Instead, this information is summarized into the item weights in the weighted sequence. These weights need to be taken into account when aligning a sequence to a weighted sequence.

Thus, instead of using Equation 3.3 directly to calculate the distance between a weighted sequence and a sequence in the cluster, we adopt a *weighted replace cost* as follows.

Let $X=(x_1:w_1,\ldots,x_m:w_m):v$ be an itemset in a weighted sequence, while $Y=(y_1\cdots y_l)$ is an itemset in a sequence in the database. Let n be the global weight of the weighted sequence. The replace cost is defined as

$$REPL(X,Y)=\frac{e_R\cdot v+n-v}{n}$$

$$\text{where}\quad e_R=\frac{\sum_{i=1}^{m}w_i+\|Y\|v-2\sum_{x_i\in Y}w_i}{\sum_{i=1}^{m}w_i+\|Y\|v} \tag{4.5}$$

Accordingly, we have $INDEL(X)=REPL(X,\emptyset)=1$ and $INDEL(Y)=REPL(Y,\emptyset)=1$.

The rationale of the Equation 4.5 is as follows. After aligning a sequence, its alignment information is incorporated into the weighted sequence. There are two cases.

- *A sequence may have a non-empty itemset aligned in this itemset.* Then, e_R is the estimated average replacement cost for all sequences that have a non-empty itemset aligned in this itemset. There are in total v such sequences.
- *A sequence may have an empty itemset aligned in this itemset.* Then, we need an $INDEL()$ operation (whose cost is 1) to change the sequence to the one currently being aligned. There are in total $(n-v)$ such sequences.

Equation 4.5 estimates the average of the cost in the two cases.

Example 2. (Multiple alignment (con'd)) In the next step, the weighted sequence WS_1 and the third sequence S_4 are aligned as shown in Figure 2. Similarly, we can align the remaining sequences. The results are shown in Figure 3.

The alignment result for all sequences are summarized in the weighted sequence WS_4 shown in Figure 3. After the alignment, we only need to store WS_4. All the sequences in the cluster are not needed any more in the remainder of the mining. ■

Aligning the sequences in different order may result in slightly different weighted sequences. To illustrate the effect, Table 3 shows the alignment result using the id order, S_1-S_2-S_3-S_4-S_5.

Interestingly, the two alignment results are quite similar, only some items shift positions slightly. This causes the item weights to be reduced slightly. As verified by our extensive empirical evaluation, the alignment order has little effect on the underlying patterns.

As shown in the above example, for a cluster of n sequences, the complexity of the multiple alignment of all sequences is $O(n\cdot t)$, where t is the maximal cost of aligning two sequences. The result of the multiple alignment is a weighted sequence. A weighted sequence records the information of the alignment. Once a

WS_1	$\langle(a:2,e:1):2$	$(h:1):1$	$(b:2):2$	$(d:2,e:1):2\rangle$	$:2$
S_4	$\langle(a)$		(bcg)	$(d)\rangle$	
WS_2	$\langle(a:3,e:1):3$	$(h:1):1$	$(b:3,c:1,g:1):3$	$(d:3,e:1):3\rangle$	$:3$

Figure 2: Sequences WS_1 and S_4 are aligned.

WS_2	$\langle(a:3,e:1):3$	$(h:1):1$	$(b:3,c:1,g:1):3$	$(d:3,e:1):3\rangle$		$:3$
S_5	$\langle$		(bci)	$(de)\rangle$		
WS_3	$\langle(a:3,e:1):3$	$(h:1):1$	$(b:4,c:2,g:1,i:1):4$	$(d:4,e:2):4\rangle$		$:4$
S_1	$\langle(ag)$	(f)	(bc)	(ae)	$(h)\rangle$	
WS_4	$\langle(a:4,e:1,g:1):4$	$(f:1,h:1):2$	$(b:5,c:3,g:1,i:1):5$	$(a:1,d:4,e:3):5$	$(h:1):1\rangle$	$:5$

Figure 3: The alignment of remaining sequences.

weighted sequence is derived, the sequences in the cluster will not be visited anymore.

Now, the remaining problem is *how to generate patterns from weighted sequences.*

4.2 Generation of Consensus Patterns

As shown in Section 4.1, a weighted sequence records the statistics of the alignment of the sequences in a cluster. Intuitively, a pattern can be generated by picking up parts of a weighted sequence shared by most sequences in the cluster.

For a weighted sequence $WS = \langle(x_{11} : w_{11}, \dots, x_{1m_1} : w_{1m_1}) : v_1, \dots, (x_{l1} : w_{l1}, \dots, x_{lm_l} : w_{lm_l}) : v_l\rangle : n$, the *strength* of item $x_{ij} : w_{ij}$ in the i^{th} itemset is defined as $\frac{w_{ij}}{n} \cdot 100\%$. Clearly, an item with a larger strength value indicates that the item is shared by more sequences in the cluster.

Motivated by the above observation, a user can specify a *strength threshold min_strength* ($0 \leq min\_strength \leq 1$). A *consensus pattern* P can be extracted from a weighted sequence by removing items in the sequence whose strength values are lower than the threshold.

Example 3. (Consensus pattern generation) Suppose a user specifies a strength threshold $min\_strength = 30\%$. The consensus pattern extracted from weighted sequence WS_4 is $\langle(a)(bc)(de)\rangle$.

Interestingly, if we compare the sequences in the sequence database (Table 2) and the consensus pattern mined from the database, the pattern is shared by the sequences, but it is not exactly contained in any one of them. In particular, every sequence except S_2 approximately contains the pattern by one insertion. These evidences strongly indicate that the consensus pattern is the general template behind the data. ■

5 Empirical Evaluations

In this section, we report an extensive set of empirical evaluations on ApproxMAP. We use both synthetic and real data sets (Section 6) to test the method. All experiments were run on a 4-processor Pentium III 700 MHz Linux machine with 2GB of main memory. Our program only uses one CPU in all experiements.

5.1 Synthetic Data Generator

To gain insight on how ApproxMAP behaves under various settings, we use the IBM synthetic data generator [2] to generate various synthetic data sets. The IBM data generator takes several parameters and outputs a sequence database as well as a set of base patterns. The sequences in the database are generated in two steps. First, *base patterns* are generated randomly according to the user's specification. Then, these base patterns are corrupted (drop random items) and merged to generate the sequences in the database. Thus, these base patterns are approximately shared by many sequences. The base patterns are the underlying template behind the database.

We summarize the parameters of the data generator and the mining in Table 4.

5.2 Evaluation Criteria

Although many people have used this synthetic data generator to generate benchmark data sets for sequential pattern mining, to the best of our knowledge, no previous study examines whether the sequential pattern mining can discover the base patterns properly. It is difficult for conventional sequential pattern mining methods to uncover the long base patterns. Furthermore, the conventional methods usually generate much more than just the base patterns. To test the effectiveness of ApproxMAP, we examine whether ApproxMAP can find the base patterns without generating many trivial or irrelevant patterns.

Specifically, for a base pattern B and a consensus pattern P, we denote $B \otimes P$ as the longest common subsequence S of both B and P. Then, we define the *recoverability* as follows to measure the quality of the

S-id	Sequence	Alignment				
S_1	$\langle(ag)(f)(bc)(ae)(h)\rangle$	$\langle(ag)$	(f)	(bc)	(ae)	$(h)\rangle$
S_2	$\langle(ae)(h)(b)(d)\rangle$	$\langle(ae)$	(h)	(b)		$(d)\rangle$
S_3	$\langle(a)(b)(de)\rangle$	$\langle(a)$		(b)		$(de)\rangle$
S_4	$\langle(a)(bcg)(d)\rangle$	$\langle(a)$		(bcg)		$(d)\rangle$
S_5	$\langle(bci)(de)\rangle$	$\langle$		(bci)		$(de)\rangle$
Weighted sequence		$\langle(a:4,e:1,g:1):4(f:1,h:1):2(b:5,c:3,g:1,i:1):5(a:1,e:1):1(d:4,e:2,h:1):5\rangle:5$				

Table 3: Aligning Sequences in another order.

Notation	Meaning	Default value
$\|I\|$	# of items	1000
N_{seq}	# of data sequences	10000
N_{pat}	# of base pattern sequences	100
L_{seq}	Avg. # of itemsets per data sequence	10
L_{pat}	Avg. # of itemsets per base pattern	$0.7 \cdot L_{seq}$
I_{seq}	Avg. # of items per itemset in the database	2.5
I_{pat}	Avg. # of items per itemset in base patterns	$0.7 \cdot I_{seq}$
k	# of neighbor sequences	5
$min\_strength$	The strength threshold consensus patterns	50%

Table 4: Parameters and default values for the data generator and the mining.

mining.

$$R = \sum_{\text{base pat } B} E(F_B) \cdot \min \begin{cases} 1 \\ \frac{\max_{\text{con pat } P}\{\|B \otimes P\|\}}{E(L_B)} \end{cases} \tag{5.6}$$

where $E(F_B)$ and $E(L_B)$ are expected frequency and expected length of base pattern B. They are given by the data generator. Since $E(L_B)$ is an expected value, sometimes the actual observed value, $\max\{\|B \otimes P\|\}$ is greater than $E(L_B)$. In such cases, we cutoff the value of $\frac{\max\{\|B \otimes P\|\}}{E(L_B)}$ to be 1 so that recoverability stays between 0 and 1.

Intuitively, if the recoverability of the mining is high, major parts of the base patterns will be found.

5.3 Effectiveness of ApproxMAP

We ran many experiments with various synthetic data sets. The trend is clear and consistent. Limited by space, we report only the results on some selected data sets here.

First, let us take a close look at the mining result from a small data set with $1,000$ sequences. The data generator uses 10 base patterns to generate the data. ApproxMAP mines the data set using the following parameters: the number of nearest neighbors $k = 6$ (for clustering), and strength threshold $min\_strength = 30\%$. Under such settings, ApproxMAP also finds 10 consensus patterns. The patterns are shown in Table 5.

As shown, each of the 10 consensus patterns match some base pattern. The consensus patterns do not cover the last two base patterns. The recoverability is 92.46%. In general, the consensus patterns recover major parts of the base patterns. The consensus patterns cannot recover the complete base patterns because, during the data generation, only parts of base patterns are embedded into a sequence. Hence, some items in the base patterns may have much lower frequency than the others. We also checked the worksheet of ApproxMAP. The less frequent items in the base patterns are in the weighted sequence. However, they are not included in the consensus patterns because their item strengths are weak.

It is interesting to note that a base pattern may be recovered by multiple consensus patterns. For example, ApproxMAP forms three clusters whose consensus patterns approximate base pattern $BasePat_2$. This is because $BasePat_2$ is long (the acutal length of the base pattern is 22 items and the expected length of the pattern in the data is 18 items) and has a high expected frequency (16.1%). Therefore, many data sequences in the database are generated using $BasePat_2$ as a template. As discussed above, sequences are generated by removing various parts of the base pattern and inserting noise items. Thus, two sequences using the same long base pattern as the template are not necessarily similar to each other. As a consequence, the sequences generated from a long base pattern can be partitioned into multiple clusters by ApproxMAP.

In all the consensus patterns, there is only one item (in the first itemset of $ConPat_2$) that does not appear on the corresponding position in the base pattern. This fact indicates that the consensus patterns are highly

Exp freq	Exp len	Observed pat. len	Type	Patterns
		13	$ConPat_1$	⟨(15,16,17,66)(15)(58,99)(2,74)(31,76)(66)(62)⟩
0.21	0.66	14	$BasePat_1$	*⟨(15,16,17,66)(15)(58,99)(2,74)(31,76)(66)(62)(93)⟩*
		11	$ConPat_2$	⟨(16,22)(29,99)(94)(45,67)(50)(96)(51)(66)⟩
		13	$ConPat_3$	⟨(22,50,66)(16)(29,99)(94)(45,67)(12,28,36)(50)⟩
		19	$ConPat_4$	⟨(22,50,66)(16)(29,99)(94)(45,67)(12,28,36)(50)(96)(51)(66)(2,22,58)⟩
0.161	0.83	22	$BasePat_2$	*⟨(22,50,66)(16)(29,99)(94)(45,67)(12,28,36)(50)(96)(51)(66)(2,22,58)(63,74,99)⟩*
		11	$ConPat_5$	⟨(22)(22)(58)(2,16,24,63)(24,65,93)(6)⟩
0.141	0.82	14	$BasePat_3$	*⟨(22)(22)(58)(2,16,24,63)(24,65,93)(6)(11,15,74)⟩*
		11	$ConPat_6$	⟨(31,76)(58,66)(16,22,30)(16)(50,62,66)⟩
0.131	0.9	15	$BasePat_4$	*⟨(31,76)(58,66)(16,22,30)(16)(50,62,66)(2,16,24,63)⟩*
		13	$ConPat_7$	⟨(43)(2,28,73)(96)(95)(2,74)(5)(2)(24,63)(20)⟩
0.123	0.81	14	$BasePat_5$	*⟨(43)(2,28,73)(96)(95)(2,74)(5)(2)(24,63)(20)(93)⟩*
		8	$ConPat_8$	⟨(63)(16)(2,22)(24)(22,50,66)⟩
0.121	0.77	9	$BasePat_6$	*⟨(63)(16)(2,22)(24)(22,50,66)(50)⟩*
		11	$ConPat_9$	⟨(70)(58,66)(22)(74)(22,41)(2,74)(31,76)⟩
0.0539	0.6	13	$BasePat_7$	*⟨(70)(58,66)(22)(74)(22,41)(2,74)(31,76)(2,74)⟩*
		15	$ConPat_{10}$	⟨(20,22,23,96)(50)(51,63)(58)(16)(2,22)(50)(23,26,36)⟩
0.0135	0.91	17	$BasePat_8$	*⟨(20,22,23,96)(50)(51,63)(58)(16)(2,22)(50)(23,26,36)(10,74)⟩*
0.0382	0.78	7	$BasePat_9$	*⟨(88)(24,58,78)(22)(58)(96)⟩*
0.00809	0.66	17	$BasePat_{10}$	*⟨(16)(2,23,74,88)(24,63)(20,96)(91)(40,62)(15)(40)(29,40,99)⟩*

Table 5: Consensus patterns and the base patterns in a small data set.

shared by sequences in the database.

Based on the above analysis, we can see that ApproxMAP summarizes $1,000$ sequences in this small data set into 10 consensus patterns accurately. The 10 consensus patterns resemble the base patterns that generates the sequences very well (recoverability=92.46%). No trivial nor irrelevant pattern is generated.

5.4 Effects of Parameters And Scalability

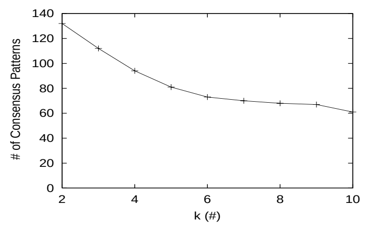
Now, let us examine the effects of various parameters on the recoverability. The default configuration of the data sets used in the remaining experiments is given in Table 4. The expected frequencies of the 100 base patterns range from 7.63% to 0.005%. We test the recoverability against 5 factors, namely the nearest neighbor parameter k, the number of items in the data set $\|I\|$, the data set size in terms of number of sequences N_{seq}, the average number of itemsets in a sequence L_{seq}, and the average number of items per itemset I_{seq}.

First, we fix other settings and vary the value of k from 2 to 10, where k is the nearest neighbor parameter in the clustering. The results are shown in Figure 4. As analyzed before, a larger value of k produces less number of clusters, which leads to less number of patterns. As expected, as k increases in Figure 4(a), the number of consensus patterns decreases. This causes loss of some weak base patterns and thus the recoverability decreases slightly, as shown in Figure 4(b). In addition, like most density based clustering algorithms, the recoverability is sustained for a range of k (5 to 9) (Figure 4(b)). Figure 4(c) indicates that the performance of ApproxMAP is not very sensitive to parameter k. It is stable.

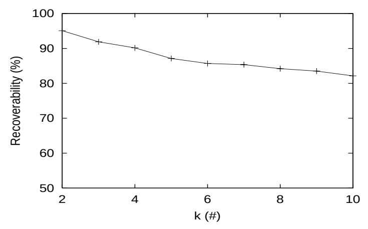
Second, we studied the effect of the number of items in the set I, $\|I\|$. A smaller value of $\|I\|$ results in a denser database (i.e., patterns are with higher frequencies) because the items come from a smaller set of literals. In multiple alignment, the positions of the items have strong effect on the results. Thus, even when the density of the database changes, the alignment statistics does not change substantially. We observe that the performance of ApproxMAP in terms of number of consensus patterns, recoverability and runtime remains stable. Limited by space, we omit the details here.

Third, we test the effect of data set size in terms of number of sequences in the data set. The results are shown in Figure 5. As the data set size increases, the number of clusters also goes up. That increases the number of consensus patterns, as shown in Figure 5(a). However, it is interesting to note that the recoverability also increases as the data set size goes up, as shown in 5(b). It can be explained as follows.

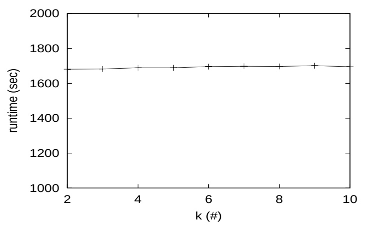
With multiple alignment, the more the sequences in the data set, the easier the recovery of the base patterns. In large data sets, there are more sequences approximating the patterns. For example, if there are



(a) Number of consensus patterns w.r.t. k

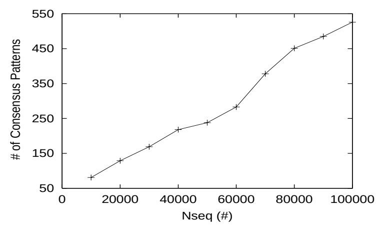


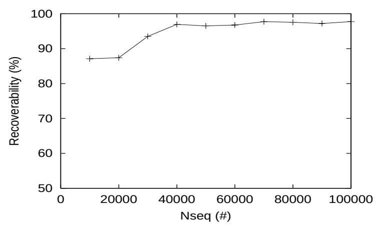
(b) Recoverability w.r.t. k

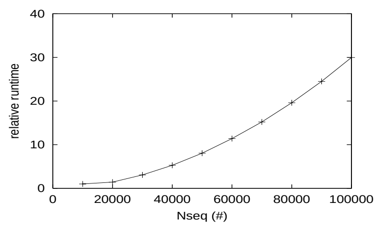


(c) Runtime w.r.t. k

Figure 4: Effect of k, the nearest neighbor parameter.







(a) # of consensus patterns w.r.t. data set size (b) Recoverability w.r.t. data set size (c) Runtime w.r.t. data set size

Figure 5: Effect of N_{seq}, the number of sequences in the data set.

only 1000 sequences, a base pattern that occurs in 1% of the sequences will only have 10 sequences approximately similar to it. However, if there are $100,000$ sequences, then there would be $1,000$ sequences similar to the the base pattern. It would be much easier for ApproxMAP to detect the general trend from $1,000$ sequences than from 10 sequences. Moreover, we observe that ApproxMAP is scalable w.r.t. data set size, as shown in Figure 5(c).

We observe similar effects from factors L_{seq}, the average number of itemsets in a sequence, and I_{seq}, the average number of items per itemset in the sequences. Limited by space, we omit the details here.

The above analysis strongly indicates that ApproxMAP is effective and scalable in mining large databases with long sequences.

5.5 The Strength Threshold

In ApproxMAP, we use a strength threshold $min\_strength$ to filter out noise from weighted sequences. Here, we test the effect of the strength threshold on the mining results.

In Figure 6, the percentage of noise items, the percentage of consensus patterns having any noise item, and the percentage of items in base patterns appearing in consensus patterns w.r.t. the strength threshold is plotted, respectively. As $min\_strength$ is decreased from 100%, both the percentage of noise items in Figure 6(a) and the number of patterns containing any noise item in Figure 6(b) start to increase at $min\_strength =$ 30% and increase drastically at 20%. This indicates that items with $strength \geq 30\%$ are probably items in the base patterns. On the other hand, the average percentage of items from the base pattern recovered by the consensus pattern is reasonably stable when $min\_strength \leq 50\%$, but goes down quickly when $min\_strength > 50\%$. This observation is also verified by experiments on other data sets. Our experience with data sets generated by the IBM data generator indicates that 20%-50% is in fact a good range for $min\_strength$.

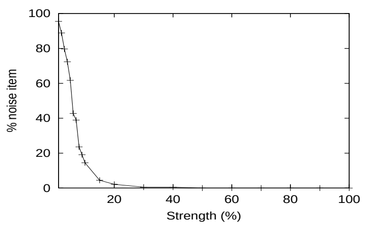
In all our experiments, the default value of $min\_strength$ is 50%.

5.6 Spurious Patterns

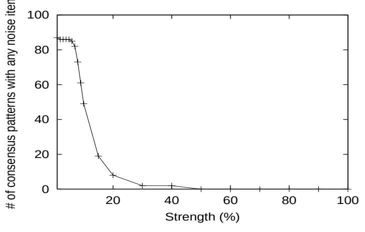
Spurious patterns are defined as consensus patterns that are not similar to any base pattern. These are in essence consensus patterns with almost all noise items. Our experimental results show that ApproxMAP is able to uncover the base patterns without generating many *spurious patterns*, i.e., the number of consensus patterns that are very different from all of the base patterns is usually very small. Almost all consensus patterns are very close to the base patterns. Only a very small number of noise items appear in the consensus patterns.

5.7 The Orders in Multiple Alignment

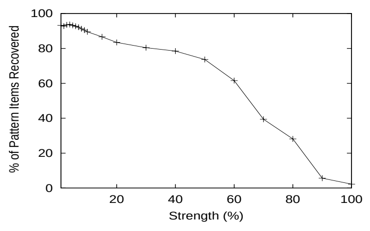
Now, we studied the sensitivity of the multiple



(a) Percentage of noise items



(b) Percentage of consensus patterns having noise items



(c) Percentage of items in base patterns appearing in consensus patterns

Figure 6: Effects of strength thresholds

alignment results to the order of sequences in the alignment. We compare the mining results using the density-descending order, density-ascending order, and the sequence-id ascending order. As expected, although the exact alignment changes slightly depending on the orders, it has very limited effect on the consensus patterns. All three orders generated the exact same number of patterns, which were very similar to each other. Therefore, the recoverability is basically identical.

This result shows that, while aligning patterns in density descending order tends to improve the alignment quality, ApproxMAP itself is robust w.r.t. alignment orders.

6 Case Study: Mining The Welfare Services Database

We also tested ApproxMAP on a real data set of welfare services accumulated over a few years in North Carolina State. The services have been recorded monthly for children who had a substantiated report of abuse and neglect, and were placed in foster care. There were 992 such sequences. In summary we found 15 interpretable and useful patterns.

In total 419 sequences were grouped together into one cluster which had the following consensus pattern.

$$\langle (RPT)(INV, FC)\overbrace{(FC)\cdots(FC)}^{11}\rangle$$

In the pattern, RPT stands for a report, INV stands for an investigation, and FC stands for a foster care service. The pattern indicates that many children who are in the foster care system after getting a substantiated report of abuse and neglect have very similar service patterns. Within one month of the report, there is an investigation and the child is put into foster care. Once children are in the foster care system, they stay there for a long time. This is consistent with the policy that all reports of abuse and neglect must be investigated within 30 days. It is also consistent with our analysis on the length of stay in foster care.

Interestingly, when a conventional sequential algorithm is applied to this data set, variations of this consensus pattern overwhelm the results, because roughly half of the sequences in this data set followed the typical behavior approximately.

The rest of the sequences in this data set split into clusters of various sizes. One cluster formed around the few children (57 sequences) who have short spells in foster care. The consensus pattern was

$$\langle (RPT)(INV, FC)(FC)(FC)\rangle.$$

There were several consensus patterns from very small clusters with about 1% of the sequences. One such pattern of interest is shown in Figure 7.

There are 39 sequences in the cluster. Our clients were interested in this pattern because foster care services and home management services were expected to be given as an "either/or" service, but not together to one child at the same time. Thus, this led us to go back to the original data to see if indeed many of the children received both services in the same month. Our investigation found that this was true, and lead our client to investigate this further in real practice. Was this a systematic data entry error or was there some components to Home Management Services (originally designed for those staying at home with their guardian) that were used in conjunction with Foster Care Services on a regular basis? Which counties were giving these services in this manner? Such an important investigation would not have been triggered without our analysis because no one ever suspected there was such a pattern. It is difficult to achieve the same results using the conventional sequential analysis methods because when the support threshold is set to $min\_support = 20\%$, there is more than $100,000$ sequential patterns and the users just cannot identify the needle from the straws.

$$\langle (RPT)(INV,FC,T)(FC,T)\overbrace{(FC,HM)\cdots(FC,HM)}^{8}(FC)(FC,HM)\rangle$$

Figure 7: An interesting pattern mined from a real data set, where T stands for transportation and HM stands for Home Management Services.

7 Related Work

Basically, there are three categories of studies highly related to our exploration in this paper, namely *sequential pattern mining*, *multiple alignments* and *approximate frequent pattern mining*. We survey them briefly in the following subsections.

7.1 Sequential Pattern Mining

Since it was first introduced in [2], sequential pattern mining has been studied extensively. Conventional sequential pattern mining finds frequent subsequences in the database based on exact match. There are two classes of algorithms. On one hand, the breadth-first search methods (e.g., GSP [21]) are based on the Apriori principle [1] and conduct level-by-level candidate-generation-and-tests. On the other hand, the depth-first search methods (e.g., PrefixSpan [16] and SPAM [3]) grow long patterns from short ones by constructing projected databases. Some variations of the depth-first search methods mine sequential patterns with vertical format [25]. Instead of recording sequences of items explicitly, they record item-lists, i.e., each item has a list of sequence-ids and positions where the item appears.

There are several interesting extensions to sequential pattern mining. For example, various constraints have been explored to reduce the number of patterns and make the sequential pattern mining more effective [9, 21, 27, 17]. Moreover, some methods (e.g., [20]) mine confident rules in the form of "$A \rightarrow B$". Such rules can be generated by a post-processing step after the sequential patterns are found.

Recently, Guralnik and Karypis used sequential patterns as features to cluster sequential data [10]. They project the sequences onto a feature vector comprised of the sequential patterns, then uses a k-means like clustering method on the vector to cluster the sequential data. Interestingly, their work concurs with this study that the similarity measure based on edit distance is a valid measure in distance based clustering methods for sequences. However, we use clustering to group similar sequences here in order to detect approximate sequential patterns. Their feature-based clustering method would be inappropriate for this purpose because the features are based on exact match.

7.2 Multiple Alignments

In parallel to the sequential pattern mining techniques, another category of string/sequence analysis techniques, *multiple alignments*, have been studied extensively in the last two decades in computational biology ([11], [8], [22]). In general, multiple alignment is used to find common patterns in a group of strings.

In this paper, we generalized string multiple alignment to find patterns in ordered lists of sets. We select an appropriate measure for distance between sequences of itemsets. We also propose a new representation, weighted sequences, to maintain the alignment information. Users can use the strength threshold to control the level of detail included in the consensus patterns.

7.3 Approximate Frequent Pattern Mining

Approximate frequent itemset mining is studied in [18, 23]. Although the two methods are quite different in techniques, they both explored approximate matching among itemsets.

Recently, Yang et al. presents a probabilistic model [24] to handle noise in mining strings. A compatibility matrix is introduced to represent the probabilistic connection from observed items to the underlying true items. Consequently, partial occurrence of an item is allowed and a new measure, *match*, is used to replace the commonly used support measure to represent the accumulated amount of occurrences. However, it cannot be easily generalized to apply to the sequential data targeted in this paper, and it does not address the issue of generating huge number of patterns that share significant redundancy. By lining up similar sequences and detecting the general trend, the multiple alignment framework in this paper effectively finds consensus patterns that are approximately similar to many sequences and dramatically reduce the redundancy among the derived patterns.

In [6], Chudova and Smyth used a Bayes error rate framework under a Markov assumption to analyze different factors that influence string pattern mining in computational biology. Extending the theoretical framework to mining sequences of sets could shed more light to the future research direction.

As analyzed above, to the best of our knowledge, though there are some related work, this is the first study on mining consensus patterns from sequence databases. It distinguishes itself from the previous studies in the following two aspects.

- It proposes the theme of approximate sequential pattern mining, which reduces number of patterns substantially and provides much more accurate and informative insights into sequential data.

- It generalizes the multiple alignment techniques to handle sequences of itemsets. Mining sequences of itemsets extends the application domain substantially. The method is applicable to many interesting problems, such as business analysis, security, and complex bio-sequences analysis.

8 Conclusions

In this paper, we introduce ApproxMAP, a new approach to approximate sequential pattern mining. Its goal is to organize and summarize sequence of sets to uncover the underlying consensus patterns in the data. ApproxMAP uses clustering as a preprocessing step to group similar sequences, and then mines the underlying consensus patterns in each cluster directly through multiple alignment. A novel structure, weighted sequences, is proposed to summarize and compress the alignment information. Our empirical evaluation on synthetic data sets and real case study verify that ApproxMAP is capable of handling long sequential data and is robust to noisy data. Our study illustrates that approximate pattern mining does find general, useful, concise and understandable knowledge and thus is an interesting and promising direction.

9 Acknowledgements

We would like to thank Jan Prins, Jack Snoeyink, and Susan Paulsen for their constructive feedback on the first draft of this report. We are also grateful to James Coggins and Stephen Aylward for their encouragement in the initial stages of this research. Finally, this report would not have been possible without the numerous discussions we had with Jisung Kim and Sang-Uok Kum. This research was supported in part by the North Carolina Department of Health and Human Services.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, Sept. 1994.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering (ICDE'95)*, pages 3–14, Taipei, Taiwan, Mar. 1995.

[3] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *Proc. 2002 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'02)*, pages 429–435, Edmonton, Alberta, Canada, July 2002.

[4] R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 85–93, Seattle, WA, June 1998.

[5] J. Coggins. Dissimilarity measures for clustering strings. In *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison.* D. Snakoff, & J. Kruskal, (Eds.), pp 253-310. Addison-Wesley Pub. Co. MA. 1983.

[6] D. Chudova and P. Smyth. Pattern discovery in sequences under a markov assumption. In *Proc. 2002 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'02)*, Edmonton, Alberta, Canada, July 2002.

[7] K. K. Fukunaga and P.M. Narendra. A branch and bound algorithm for computing k-nearest neighbours. In *IEEE Transactions on Computers*, 24:750-753, 1975.

[8] Osamu Gotoh. Multiple sequence alignment: Algorithms and applications. In *Adv. Biophys.*, Vol. 36, pp159-206. 1999.

[9] M. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *Proc. 1999 Int. Conf. Very Large Data Bases (VLDB'99)*, pages 223–234, Edinburgh, UK, Sept. 1999.

[10] V. Guralnik and G. Karypis. A scalable algorithm for clustering sequential data. In *Proc. 2001 Int. Conf. Data Mining (ICDM'01)*, pages 179–186, San Jose, CA, Nov. 2001.

[11] Dan Gusfield. Algorithms on strings, trees, and sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge, England. 1997.

[12] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. FreeSpan: Frequent pattern-projected sequential pattern mining. In *Proc. 2000 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'00)*, pages 355–359, Boston, MA, Aug. 2000.

[13] W. L. G. Koontz, Patrenahalli M. Narendra, and K. Fukunaga. A Branch and Bound Clustering Algorithm. In em IEEE Transactions on Computers. 24(9): 908-915 (1975).

[14] G. R. McPherson and S. DeStefano. Applied Ecology and Natural Resource Management. Cambridge University Press, Cambridge, England. 2002.

[15] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. 7th Int. Conf. Database Theory (ICDT'99)*, pages 398–416, Jerusalem, Israel, Jan. 1999.

[16] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen,

U. Dayal, and M.-C. Hsu. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)*, pages 215–224, Heidelberg, Germany, April 2001.

[17] J. Pei, J. Han, and W. Wang. Constraint-based sequential pattern mining in large databases. In *submitted to publication*, March 2002.

[18] J. Pei, A. K. H. Tung, and J. Han. Fault-tolerant frequent pattern mining: Problems and challenges. In *Proc. 2001 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD'01)*, pages 7–12, Santa Barbara, CA, May 2001.

[19] Sas Institute. Proc Modeclust. In SAS/STAT User Guide. Sas online Document. 2000

[20] Myra Spiliopoulou. Managing interesting rules in sequence mining. In *Proc. European Conf. on Principles and Practice of Knowledge Discovery in Databases*, pp554-560, 1999.

[21] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. 5th Int. Conf. Extending Database Technology (EDBT'96)*, pages 3–17, Avignon, France, Mar. 1996.

[22] J. Thompson, F. Plewniak, and O. Poch. A comprehensive comparison of multiple sequence alignment programs. In *Nucleic Acids Research.* Vol. 27. No. 13: pp 2682-2690. Oxford University Press. 1999.

[23] C. Yang, U. Fayyad, and P.S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 194–203. ACM Press, 2001.

[24] J. Yang, P. S. Yu, W. Wang, and J. Han. Mining long sequential patterns in a noisy environment. In *Proc. 2002 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'02)*, Madison, WI, June 2002.

[25] M. J. Zaki. Efficient enumeration of frequent sequences. In *Proc. 7th Int. Conf. Information and Knowledge Management (CIKM'98)*, pages 68–75, Washington DC, Nov. 1998.

[26] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. Parallel algorithm for discovery of association rules. *Data Mining and Knowledge Discovery*, 1:343–374, 1997.

[27] M.J. Zaki. Sequence mining in categorical domains: incorporating constraints. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 422–429. ACM Press, 2000.