Sequential Pattern Mining in Multi-Databases via Multiple Alignment

Hye-Chung (Monica) Kum[†] **Joong Hyuk Chang**[‡] **Wei Wang**[†]

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

[‡] Yonsei University, jhchang@amadeus.yonsei.ac.kr

Abstract

To efficiently find global patterns from a multi-database, information in each local database must first be mined and summarized at the local level. Then only the summarized information is forwarded to the global mining process. However, conventional sequential pattern mining methods based on support cannot summarize the local information and is ineffective for global pattern mining from multiple data sources. In this paper, we present an alternative local mining approach for finding sequential patterns in the local databases of a multi-database. We propose the theme of *approximate sequential pattern mining* roughly defined as *identifying patterns approximately shared by many sequences*. Approximate sequential patterns can effectively summerize and represent the local databases by identifying the underlying trends in the data. We present a novel algorithm, ApproxMAP, to mine approximate sequential patterns, called *consensus patterns*, from large sequence databases in two steps. First, sequences are clustered by similarity. Then, consensus patterns are mined directly from each cluster through multiple alignment. We conduct an extensive and systematic performance study over synthetic and real data. The results demonstrate that ApproxMAP is effective and scalable in mining large sequences databases with long patterns. Hence, ApproxMAP can efficiently summarize a local database and reduce the cost for global mining. Furthremore, we present an elegant and uniform model to identify both *high vote sequential patterns* and *exceptional sequential patterns* from the collection of these consensus patterns from each local databases.

1 Introduction

Many large organizations have multiple data sources residing in diverse locations. For example, multinational companies with branches around the world have local data in each branch. Each branch can mine its local data for local decision making using traditional mining technology. Naturally, these local patterns can then be gathered, analyzed, and synthesized at the central headquarter for decision making at the central level. Such post-processing of local patterns from multiple data sources is fundamentally different from traditional mono-database mining. Obviously, efficiency becomes a greater challenge that may only be tackled via distributed or parallel data mining. More important, global patterns mined via distributed data mining technology may offer additional insight than the local patterns. The problem formulation, difficulties, and framework for multi-database mining has been addressed in [26]. Recently mining global association rules from multiple data sources has been studied in [20, 25, 27]. However, to the best of our knowledge, no work has been done in mining global sequential patterns from multiple data sources.

The goal of sequential pattern mining is to detect patterns in a database comprised of sequences of sets. Conventionally, the sets are called itemsets. For example, retail stores often collect customer purchase records in sequence databases in which a sequential pattern would indicate a customer's buying habit. In such a database, each purchase would be represented as a set of items purchased, and a customer sequence would be a sequence of such itemsets. More formally, given a sequence database and a user-specified minimum support threshold, sequential pattern mining is defined as finding all frequent subsequences that meet the given minimum support threshold [2]. GSP [17], PrefixSpan [13], SPADE [24], and SPAM [1] are some well known algorithms to efficiently find such patterns. However, such problem formulation of sequential patterns has some inherent limitations that make it be inefficient for mining multi-databases as follows:

- *Most conventional methods mine the complete set of sequential patterns*. Therefore, the number of sequential patterns in a resulting set may be huge and many of them are trivial for users. Recently, methods for mining compact expressions for sequential patterns has been proposed [22]. However, in a general sequence database, the number of maximal or closed sequential patterns still can be huge, and many of them are useless to the users. We have found using the well known IBM synthetic data [2] that, given 1000 data sequences, there were over 250,000 sequential patterns returned when min_sup was 5%. The same data gave over 45,000 max sequential patterns for $min\_sup = 5\%$ [10].
- *Conventional methods mine sequential patterns with exact matching*. A sequence in the database supports a pattern if, and only if, the pattern is fully contained in the sequence. However, the exact match based paradigm is vulnerable to noise and variations in the data and may miss the general trends in the sequence database. Many customers may share similar buying habits, but few of them follow exactly the same buying patterns.
- *Support alone cannot distinguish between statistically significant patterns and random occurrences*. Both theoretical analysis and experimental results show that many short patterns can occur frequently simply by chance alone [10].

Table 1. Representing the underlying pattern

seq_1	⟨()	()	**(BC)**	(**D**E)⟩	
seq_2	⟨**(A)**	()	(**BC**X)	**(D)**⟩	
seq_3	⟨(**A**E)	(B)	**(BC)**	**(D)**⟩	
seq_4	⟨**(A)**	()	**(B)**	(**D**E)⟩	
$approx\_pat$	⟨(A)		(BC)	(D)⟩	

For efficient global pattern mining from multiple sources, information from local databases must first be mined and summarized at the local level. Then only the summarized information is forwarded to the global mining process. However, the support model cannot accurately summarize the local database due to the above mentioned inherent limitations. Rather, it generates many more short and trivial patterns than the number of sequences in the original database. Thus, the support based algorithms not only have limitations in the local data mining process, but also hinder the global mining process.

In this paper, we consider an alternative local pattern mining approach which can facilitate the global sequential pattern mining. In general, understanding the general trends in the sequence database for natural customer groups would be much more useful than finding all frequent subsequences in the database. Approximate sequential patterns can detect general trends in a group of similar sequences, and may be more useful in finding non-trivial and interesting long patterns. Based on this observation, we introduce the notion of approximate sequential patterns. *Approximate sequential patterns* are those patterns that are shared by many sequences in the database but not necessarily exactly contained in any one of them. Table 1 shows a group of sequences and a pattern that is approximately similar to them. In each sequence, the dark items are those that are shared with the approximate pattern. seq_1 has all items in $approx\_pat$, in the same order and grouping, except missing item A in the first itemset and having an additional item E in the last itemset. Similarly, seq_4 misses B in the third itemset and has an extra E in the last itemset. In comparison, seq_2 and seq_3 have all items in $approx\_pat$ but each has a couple of extra items. These evidences strongly indicate that $approx\_pat$ is the underlying pattern behind the data. Hence, $approx\_pat$ can effectively summarize all four sequences by representing the common underlying patterns in them. We adopt this new model for sequential pattern mining to identify local sequential patterns. It can help to summarize and represent concisely the sequential patterns in a local database.

Given properly mined approximate local patterns, we are interested in mining global sequential patterns from multiple data sources. In particular, we are interested in mining high vote sequential patterns and exceptional sequential patterns [26]. *High vote patterns* are those patterns common across many local

Table 2. Local Patterns

$lpat_{11}$	California	{diapers, crib, carseat}{mobile, diapers, shoes}{bibs, spoon, diapers}
$lpat_{21}$	Maine	{diapers, crib,carseat}{mobile, diapers}{bibs, spoon, diapers, shoes}
$lpat_{31}$	Alaska	{diapers, crib}{mobile, diapers}{bibs, spoon, diapers, shoes}
$lpat_{32}$	Alaska	{eskimo barbie, igloo}{sliegh, dogs}{boots, skies}
. . .		

databases. They reflect the common characteristics among the data sources. *Exceptional patterns* are those patterns that hold true in only a few local databases. These patterns depict the special characteristics of certain data sources. Such information is invaluable at the headquarter for developing customized policies for individual branches when needed.

To illustrate our problem let us consider a toy retail chain as an example. Each local branch of the toy retail chain collects customer purchase records in a sequence database. In the local mining phase, common buying patterns of major client groups are mined in each local branch as shown in Table 2. Subsequently, from the set of common buying patterns, high vote buying patterns and exceptional buying patterns can be found, and they can help to answer the following questions:

- What client groups have similar patterns across most branches and what are the similar patterns? *Answer: expecting parents are a major group in most braches. The common buying patterns for expecting parents are* $lpat_{11}$, $lpat_{21}$, *and* $lpat_{31}$.

- What are the differences in the similar patterns? *Answer: It seems in CA, parents buy their first baby shoes quicker than those in Maine.*

- What client groups only exist in a few branches? Which branches are they? *Answer: In AL, there is a group of girls who buy eskimo barbie followed by assecories for her.*

No current mining method can answer these questions. In this paper, we investigate how to mine such high vote sequential patterns and exceptional sequential patterns from multiple data sources. In short, this paper makes 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 and cover many short patterns. Approximate sequential patterns can effectively summarize and represent the local data for efficient global sequential pattern mining.

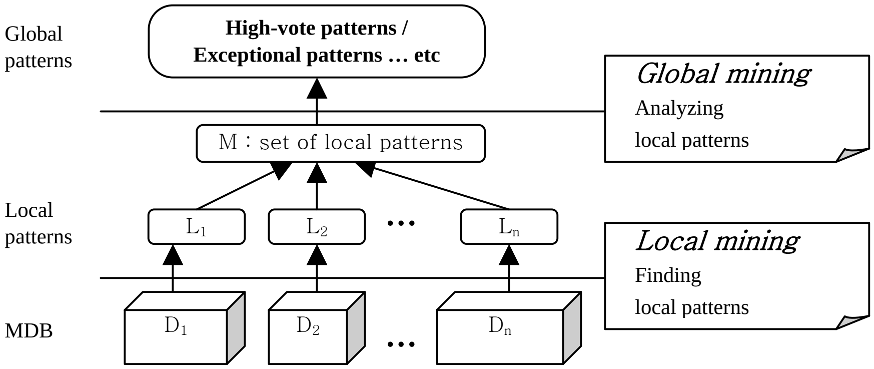


Figure 1. MDB process

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

- We formulate an elegant and uniform post-processing model to find both high vote sequential patterns and exceptional patterns from the set of locally mined consensus patterns.

The rest of the paper is organized as follows. Section 2 formulate the local approximate sequential pattern mining and the global sequential pattern mining problem. Section 3 details the distance function between sequences. Section 4 describes the ApproxMAP algorithm for local pattern mining. Evaluation of ApproxMAP is given in Section 5. Section 6 overviews the related work. Finally, Section 7 concludes with a discussion of future work.

2 Problem Formulation

In [26], Zhang et al. define multi-database mining as a two level process as shown in Figure 1. It can support two level decision making in large organizations: the branch decision (performed via local mining) and the central decision (performed via global mining). In local applications, a branch manager needs to analyze the local database to make local decisions. For global applications and for corporate profitability, top executives at the company headquarter are more interested in global patterns rather than the original raw data. The following definitions formally define local pattern mining and global pattern mining in sequence databases.

Definition 1 (Multi-Sequence Database) *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. Essentially, a sequence is an ordered list of sets. A* **local sequence database** *D_i is a set of such sequences. A* **multi-sequence database** *is a set of local sequence databases $D_1 \cdots D_n$.*

2.1 Local Sequential Pattern Mining

First, each local data has to be mined for common patterns of major groups. In our toy retail example, each branch must identify the major client groups, and then, mine the common buying pattern for each group. As shown in Table 1, a sequential pattern that is approximately similar to most sequences in the group can summarize and represent the sequences in a particular group effectively. With this insight, we can formulate the local sequential pattern mining problem as follows.

Definition 2 (Similarity Groups) *Let D_x be a local sequence database and $dist(seq_i, seq_j)$ be the distance measure for seq_i and seq_j between 0 and 1. Then D_x can be partitioned into* **similarity groups** *$G_{x_1} \ldots G_{x_n}$ such that $\sum_{i \neq j} dist(seq_{ia}, seq_{jb})$ is maximized and $\sum_{i=j} dist(seq_{ia}, seq_{jb})$ is minimized where $seq_{ia} \in G_{x_i}$ and $seq_{jb} \in G_{x_j}$.*

This is the classic clustering problem of maximizing the inter cluster distances and minimizing the intra cluster distances. Given a reasonable distance measure[1] $dist(seq_i, seq_j)$, similarity groups identified via clustering can detect all major groups in a local sequence database.

Definition 3 (Approximate Sequential Patterns) *Given the similarity groups $G_{x_1} \ldots G_{x_n}$ for the local sequence database D_x, an* **approximate sequential pattern** *for group G_{x_i}, denoted as $lpat_{x_i}$, is a sequence that minimizes $dist(lpat_{x_i}, seq_a)$ for all seq_a in similarity group G_{x_i}.*

$lpat_{x_i}$ may or may not be an actual sequence in D_x. Basically, $lpat_{x_i}$ is approximately similar to all sequences in G_{x_i}. Therefore, $lpat_{x_i}$ should effectively represent the common underlying pattern in all sequences in G_{x_i}. Moreover, the set of all approximate patterns from D_x, denoted as L_x, should accurately represent the local database D_x. Thus, only L_x is forwarded to the global mining process.

[1] We will investigate the distance measure in Section 3.

2.2 Analyzing Local Patterns

Let M be the collection of all local patterns, $L_x \cdots L_n$. Note that a local pattern is again an ordered list of sets. Hence, M will be in a very similar format to that of a local database D_x except that the sequences are labeled with the local database id. An important semantic difference is that in most cases M will be much smaller than D_x. In our toy retail example, D_x is the local database of all customers' sequences in a local branch. However M is the collection of all local patterns from all branches. Typically, the number of branches multiplied by the average number of major client groups is far less than the number of customers in any one branch. Furthermore M will have much less noise with local patterns forming natural distinct groups.

Mining the high vote pattern of expecting parents is important. In our toy retail example, we cannot expect the common buying patterns of expecting parents to be identical in all branches. Thus, when local patterns from one branch is matched with those from another branch we cannot use exact match. However, the common patterns should be highly similar. Hence, the local patterns representing the buying patterns of expecting parents from each branch will naturally form a tight cluster. After grouping such highly similar local patterns from each branch we can identify the high vote patterns by the large group size.

Similarly, exceptional patterns are those local patterns that are found in only a few branches. However, if exceptional patterns were found based on exact match, almost all local patterns would be exceptional patterns because most local patterns will not be identical to one another. Hence, as in the case of high vote patterns, we must consider approximate match to accurately identify exceptional patterns. A true exceptional pattern should be very distinct from most local patterns. Exactly how different the pattern needs to be is dependant on the application, and should be controlled by the user.

In short, the key to identifying the high vote sequential patterns and exceptional sequential patterns is in properly grouping the local patterns by the desired similarity level. Classification of local patterns by the desired similarity allows one simple model for identifying both the high vote patterns and the exceptional patterns. The notion of homogeneous set, defined below, provides an elegant and uniform model to specify the desired similarity among the local patterns.

Definition 4 (Homogeneous Set) *Given a set of local patterns M, its subset HS is a homogeneous set of range δ when the similarity between any two patterns p_i and p_j in HS is not less than δ,*
i.e., $p_i \in HS \wedge p_j \in HS \wedge sim(p_i, p_j) \geq \delta$, where $sim(p_i, p_j) = 1 - dist(p_i, p_j)$.

Table 3. High Vote Homogeneous Set and Pattern

$\frac{3}{3}$=100%	{diapers, crib} {mobile, diapers}{bibs, spoon, diapers}		
CA	{diapers, crib, carseat}	{mobile, diapers, shoes}	{bibs, spoon, diapers}
MA	{diapers, crib,carseat}	{mobile, diapers}	{bibs, spoon, diapers, shoes}
AL	{diapers, crib}	{mobile, diapers}	{bibs, spoon, diapers, shoes}

A homogenous set of δ range is essentially the group of local patterns that are approximately similar within range δ. The larger the δ, the more likely it is for the local patterns to be separated into different homogenous sets. In the extreme case when δ=100% (i.e. a HS(100%) is a subset of all local patterns with $dist(lpat_i, lpat_j) \leq 0 = 0$), almost all local patterns will belong to its own homogeneous set. On the other hand, too small a δ will lump different local patterns into one homogeneous set. The similarity between seq_4 and seq_2 in Table 4 is 1-0.278=72%. Thus, they will be grouped into one homogeneous set if $\delta \leq 72\%$, but separated if $\delta > 72\%$.

Definition 5 (Vote) *The $vote$ of a homogeneous set, HS, is defined as the size of the homegenous set.* $Vote(HS, \delta) = \|HS(\delta)\|$.

Definition 6 (High Vote Homogenous Set) *Given the desired similarity level δ and threshold Θ, a* **high vote homogenous set** *is a homogeneous set HS such that $vote(HS, \delta) \geq \Theta$.*

Definition 7 (Exceptional Homogenous Set) *Given the desired similarity level δ and threshold Ω, an* **exceptional homogenous set** *is a homogeneous set HS such that $vote(HS, \delta) \leq \Omega$.*

For example, a user might look for high vote homogeneous sets that hold true in at least 80% of the local databases with 90% similarity. An exceptional homogenous set of interest might be local patterns that are true in at most 10% of the local databases with 60% similarity.

Once the highly similar patterns have been identified as a homogeneous set, we have to consider what information will be of interest to the users as well as how to best present it. Given highly similar sequences, the longest common subsequence can effectively provide the shared pattern among the sequences in the group. That is the common pattern among all the expecting parent's buying pattern can be detected by constructing the longest common subsequence. This shared pattern can provide a reasonable representation of the high vote homogenous set. Thus, we have defined high vote sequential patterns as the longest common subsequence of all sequences in the high vote homogeneous set.

Definition 8 (High Vote Sequential Pattern) *Given a high vote homogenous set, the* **high vote sequential pattern** *is the longest common subsequence of all local patterns in the set.*

However, in most cases just presenting the high vote patterns will not be sufficient for the user. Users at the headquarter would be interested to know what the different variations are among the branches. Some variations might give hints on how to improve sales. For example, if we found that in warmer climate expecting parents buy the first walking shoes faster than those in colder climates, branches can introduce new shoes earlier in the warmer climates. The variations in sequences are most accessible to people when given in an aligned form as in Table 3. Table 3 depicts how a typical high vote pattern information should be presented. The longest common subsequence on top along with the percentage of branches in the homogeneous set, will give the user a quick summary of the homogenous set. Then, the actual sequences aligned and the branch identification gives the user access to more detailed information that could potentially be useful.

Unlike the high vote patterns, sequences grouped into one exceptional homogeneous set should not be combined into one longest common subsequence. By the nature of the problem formulation, users will look for exceptional patterns with a lower δ compared to the high vote patterns. When the patterns in a homogenous set are not highly similar, the longest common subsequence will be of little use. Rather, each of the local patterns in the exceptional homogenous set is an exceptional pattern. Nonetheless, the results will be more accessible to people when organized by homogenous sets.

Definition 9 (Exceptional Sequential Pattern) *Each local pattern in an exceptional homogenous set is an* **exceptional sequential pattern**. *Exceptional sequential patterns are made more accessible to users by organizing them by homogenous sets.*

Given the desired similarity level δ, the homogeneous set of local patterns with range δ can be easily identified by clustering the local patterns in M using the complete linkage algorithm. The merging stops when the link is greater than δ. Both high vote homogeneous set and exceptional homogeneous set can be easily identified by the same process. With the appropriate local pattern mining, detecting the global sequential patterns from the homogeneous sets is simple. Hence, in the remainder of this paper, we focus on efficiently detecting local approximate sequential patterns.

3 Distance Measure for Sequences of Sets

Both the local sequential pattern mining and the post-processing for global patterns rely on the distance function for sequences. The function $sim(seq_i, seq_j) = 1 - dist(seq_i, seq_j)$ should reasonably reflect the similarity between the given sequences. In this section, we define a reliable distance measure for sequences of sets.

In general, the *weighted edit distance* is often used as a distance measure for variable length sequences [6]. Also referred to as the Levenstein distance, the weighted edit distance is defined as the minimum cost of edit operations (i.e., insertions, deletions, and replacements) required to change one sequence to the other. An insertion operation on seq_1 to change it towards seq_2 is equivalent to a deletion operation on seq_2 towards seq_1. Thus, an insertion operation and a deletion operation have the same cost. $INDEL()$ is used to denote an insertion or deletion operation and $REPL()$ is used to denote a replacement operation. Often, for two sets X, Y the following inequality is assumed $REPL(X, Y) \leq INDEL(X) + INDEL(Y)$. Given two sequences $seq_1 = \langle X_1 \cdots X_n \rangle$ and $seq_2 = \langle Y_1 \cdots Y_m \rangle$, the weighted edit distance between seq_1 and seq_2 can be computed by dynamic programming using the following recurrence relation.

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

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

$$dist(seq_i, seq_j) = \frac{D(seq_i, seq_j)}{\max\{\|seq_i\|, \|seq_j\|\}} \tag{2}$$

3.1 Cost of Edit Operations for Sets: Normalized Set Difference

To extend the weighted edit distance to sequences of sets, we need to define the cost of edit operations (i.e., INDEL() and REPL() in Equation 1) for sets. The similarity of the two sets can be measured by how many elements are shared or not. To do so, here we adopt the *normalized set difference* as the cost of replacement of sets as follows. Given two sets, X and Y,

$$\begin{aligned}
&REPL(X,Y) = \frac{\|(X-Y)\cup(Y-X)\|}{\|X\|+\|Y\|} = \frac{\|X\|+\|Y\|-2\|X\cap Y\|}{\|X\|+\|Y\|} = 1 - \frac{2\|X\cap Y\|}{\|X-Y\|+\|Y-X\|+2\|X\cap Y\|} = D_S(X,Y) \\
&D_J(X,Y) = 1 - \frac{\|X\cap Y\|}{\|X\cup Y\|} = 1 - \frac{\|X\cap Y\|}{\|X-Y\|+\|Y-X\|+\|X\cap Y\|}
\end{aligned} \tag{3}$$

This measure is a metric and has a nice property that, $0 \leq REPL() \leq 1$. Following equation 3, the cost of an insertion/deletion is given in Equation 4 where X is a set.

$$INDEL(X) = REPL(X, ()) = REPL((), X) = 1, \tag{4}$$

Table 4. Examples of normalized edit distance between sequences

seq_9	⟨(I) (LM) ()⟩	seq_4	⟨(A) (B) (DE)⟩	seq_6	⟨(AY) (BD) (B) (EY)⟩
seq_{10}	⟨(V) (PW) (E)⟩	seq_2	⟨(A) (BCX) (D)⟩	seq_2	⟨(A) () (BCX) (D)⟩
$REPL()$	1 1 1	$REPL()$	0 $\frac{1}{2}$ $\frac{1}{3}$	$REPL()$	$\frac{1}{3}$ 1 $\frac{1}{2}$ 1
$dist()$	$3/3 = 1$	$dist()$	$(\frac{1}{2} + \frac{1}{3})/3 = 0.278$	$dist()$	$(2 + \frac{5}{6})/4 = 0.708$

Clearly, the normalized set difference, $REPL()$, is equivalent to the Sorensen coefficient, D_S, as shown in Equation 3. The Sorensen coefficient is an index similar to the more commonly used Jaccard coefficient, D_J, also defined in Equation 3 [12]. The difference is that $REPL()$ gives more weight to the common elements because in alignment what are shared by two sets is most important.

Note that $dist()$ inherits the properties of of the $REPL()$, i.e. $dist()$ satisfies the metric properties and is between 0 and 1. Table 4 illustrates some examples of the distance between sequences and itemsets. Given an itemset (A), the normalized set difference $REPL((\text{A}), (\text{A})) = 0$. Given itemsets that share no items $REPL((\text{LM}), (\text{PW})) = 1$. Given sets that share a certain number of items, $REPL()$ is a fraction between 0 and 1. Similarly, when two sequences do not share any items in common, e.g., $dist(seq_9, seq_{10}) = 1$ since each itemset distances is 1. The next example shows two sequences that share some items. When seq_4 and seq_2 are optimally aligned, three items, A, B, D, can be lined up resulting in $dist(seq_4, seq_2) = 0.278$. In comparison in the third example, when seq_2 and seq_6 are optimally aligned, only two items (A and B) are shared. There are many items that are not shared, which results in $dist(seq_6, seq_2) = 0.708$. Clearly, seq_4 is more similar to seq_2 than seq_6. That is, $dist(seq_4, seq_2) < dist(seq_6, seq_2)$.

4 Local Pattern Mining: ApproxMAP

In this section, we detail an efficient method, ApproxMAP (for APPROXimate Multiple Alignment Pattern mining), for multiple alignment sequential pattern mining. We will first demonstrate the method through an example and then discuss the details of each step in later sections.

Table 5 is a sequence database $\mathcal{D}$. Although the data is lexically sorted it is difficult to gather much information from the raw data even in this tiny example. The ability to view Table 5 is immensely improved by using the alignment model – grouping similar sequences then lining them up to construct the consensus patterns as in Tables 6 and 7. Note that the patterns ⟨(A)(B)(BC)(DE)⟩ and ⟨(IJ)(K)(LM)⟩ do not match any sequence exactly.

Given the input data shown in Table 5 ($N = 10$ sequences), ApproxMAP (1) calculates the $N * N$

Table 5. Sequence database $\mathcal{D}$ lexically sorted

ID	Sequences				
seq_4	⟨(A)	(B)	(DE)⟩		
seq_2	⟨(A)	(BCX)	(D)⟩		
seq_3	⟨(AE)	(B)	(BC)	(D)⟩	
seq_7	⟨(AJ)	(P)	(K)	(LM)⟩	
seq_5	⟨(AX)	(B)	(BC)	(Z)	(AE)⟩
seq_6	⟨(AY)	(BD)	(B)	(EY)⟩	
seq_1	⟨(BC)	(DE)⟩			
seq_9	⟨(I)	(LM)⟩			
seq_8	⟨(IJ)	(KQ)	(M)⟩		
seq_{10}	⟨(V)	(PW)	(E)⟩		

Table 6. Cluster 1 ($\theta = 40\% \wedge w \geq 3$)

seq_2	⟨(A)	()	(BCX)	()	(D)⟩	
seq_3	⟨(AE)	(B)	(BC)	()	(D)⟩	
seq_4	⟨(A)	()	(B)	()	(DE)⟩	
seq_1	⟨()	()	(BC)	()	(DE)⟩	
seq_5	⟨(AX)	(B)	(BC)	(Z)	(AE)⟩	
seq_6	⟨(AY)	(BD)	(B)	()	(EY)⟩	
seq_{10}	⟨(V)	()	()	(PW)	(E)⟩	
Weighted Seq	(A:5, E:1,V:1, X:1,Y:1):6	(B:3, D:1):3	(B:6, C:4,X:1):6	(P:1,W:1,Z:1):2	(A:1,D:4, E:5,Y:1):7	7
Consensus Pattern	⟨(A)	(B)	(BC)		(DE)⟩	

Table 7. Cluster 2 ($\theta = 40\% \wedge w \geq 2$)

seq_8	⟨(IJ)	()	(KQ)	(M)⟩	
seq_7	⟨(AJ)	(P)	(K)	(LM)⟩	
seq_9	⟨(I)	()	()	(LM)⟩	
Weighted Sequence	⟨(A:1,I:2,J:2):3	(P:1):1	(K:2,Q:1):2	(L:2,M:3):3	3
Consensus Pattern	⟨(IJ)		(K)	(LM)⟩	

sequence to sequence proximity matrix from the data, (2) partitions the data into two clusters ($k = 2$), (3) aligns the sequences in each cluster (Tables 6 and 7) – the alignment compresses all the sequences in each cluster into one weighted sequence per cluster, and (4) summarizes the weighted sequences into consensus patterns (Tables 6 and 7).

4.1 Clustering Sequences: Organize into Similarity Groups

The general objective of clustering methods that work on a distance function is to minimize the intra-cluster distances and maximize the inter-cluster distance [7]. By clustering the sequences in the local database D_i using the pairwise score between sequences (Equation 2), we can identify the similarity groups in D_i.

Density based methods, also called mode seeking methods, generate a single partition of the data in an attempt to recover the natural groups in the data. Clusters are identified by searching for regions of high density, called modes, in the data space. Then these clusters are grown to the valleys of the density function

[7]. These valleys can be considered as natural boundaries that separate the modes of the distribution [4]. In short, density based clustering methods have many benefits for clustering sequences:

- The basic paradigm of clustering around dense points fits the sequential data best because the goal is to form groups of arbitrary shape and size around similar sequences [3, 14].
- Density based k nearest neighbor clustering algorithms will automatically estimate the appropriate number of clusters from the data.
- Users can cluster at different resolutions by adjusting k.

We found that in general the density based k-nearest neighbor clustering methods worked well and was efficient for sequential pattern mining. In fact, in recent years many variations of density based clustering methods have been developed [3, 14]. Many use k-nearest neighbor or the Parzen window for the local density estimate [14]. Recently, others have also used the shared neighbor list [3] as a measure of density. Other clustering methods that can find clusters of arbitrary shape and size may work as well or better depending on the data. Any clustering method that works well for the data can be used in ApproxMAP. In the absence of a better choice based on actual data, a reasonably good clustering algorithm that finds clusters of arbitrary shape and size will suffice. For the purposes of demonstrating ApproxMAP, the choice of a density based clustering method was based on its overall good performance and simplicity. The following sections detail the clustering method used in ApproxMAP.

4.1.1 Uniform Kernel Density Based k Nearest Neighbor Clustering

ApproxMAP uses uniform kernel density based k nearest neighbor clustering. In this algorithm, the user-specified parameter k specifies not only the local region to use for the density estimate, but also the number of nearest neighbors that the algorithm will search for linkage. We adopt an algorithm from [15] based on [19] as given in Algorithm 1. The algorithm has complexity $O(k \cdot \|\mathcal{D}\|)$.

Algorithm 1 (Uniform kernel density based k-NN clustering)

```
Input: a set of sequences $\mathcal{D} = \{seq_i\}$, number of neighbor sequences $k$;
Output: a set of clusters $\{C_p\}$, where each cluster is a set of sequences;
Method: 1. Initialize every sequence as a cluster. For each sequence $seq_i$ in cluster $C_{seq_i}$, set
           $Density(C_{seq_i}) = Density(seq_i, k)$.
```

2. **Merge nearest neighbors based on the density of sequences**. *For each sequence seq_i, let $seq_{i_1}, \ldots, seq_{i_n}$ be the nearest neighbor of seq_i, where $n = n_k(seq_i)$ as defined in Equation 5. For each $seq_j \in \{seq_{i_1}, \ldots, seq_{i_n}\}$, merge cluster C_{seq_i} containing seq_i with a cluster C_{seq_j} containing seq_j, if $Density(seq_i, k) < Density(seq_j, k)$ and there exists no seq'_j having $dist(seq_i, seq'_j) < dist(seq_i, seq_j)$ and $Density(seq_i, k) < Density(seq'_j, k)$. Set the density of the new cluster to $\max\{Density(C_{seq_i}), Density(C_{seq_j})\}$.*
3. **Merge based on the density of clusters - merge local maxima regions**. *For all sequences seq_i such that seq_i has no nearest neighbor with density greater than that of seq_i, but has some nearest neighbor, seq_j, with density equal to that of seq_i, merge the two clusters C_{seq_j} and C_{seq_i} containing each sequence if $Density(C_{seq_j}) > Density(C_{seq_i})$.*

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. Formally, we measure the density of a sequence by a quotient of the number of similar sequences (nearest neighbors), n, against the space occupied by such similar sequences, d. In particular, for each sequence seq_i in a database $\mathcal{D}$, $p(seq_i) = \frac{n}{\|\mathcal{D}\| * d}$. Since $\|\mathcal{D}\|$ is constant across all sequences, for practical purposes it can be omitted.

Therefore, given k, which specifies the k-nearest neighbor region, ApproxMAP defines the density of a sequence seq_i in a database $\mathcal{D}$ as follows. Let $d_1, \ldots, d_k$ be the k smallest non-zero values of $dist(seq_i, seq_j)$ (defined in equation 2), where $seq_j \neq seq_i$, and seq_j is a sequence in $\mathcal{D}$. Then,

$$Density(seq_i, k) = \frac{n_k(seq_i)}{dist_k(seq_i)}, \tag{5}$$

where $dist_k(seq_i) = \max\{d_1, \ldots, d_k\}$ and $n_k(seq_i) = \|\{seq_j \in \mathcal{D} | dist(seq_i, seq_j) \leq dist_k(seq_i)\}\|$. $n_k(seq_i)$ is the number of sequences including all ties in the k-nearest neighbor space for sequence seq_i, and $dist_k(seq_i)$ is the size of the k-nearest neighbor region for sequence seq_i. $n_k(seq_i)$ is not always equal to k because of ties.

Theoretically, the algorithm is similar to the single linkage method. In fact, the normal single linkage method is a degenerate case of the algorithm with $k = 1$. The single linkage method builds a tree with each point linking to its closest neighbor. In the density based k nearest neighbor clustering, each point links to its closest neighbor, but (1) only with neighbors with greater density than itself, and (2) only up to k nearest neighbors. Thus, the algorithm essentially builds a forest of single linkage trees (each tree representing a natural cluster), with the proximity matrix defined as follows,

$$dist'(seq_i, seq_j) = \begin{cases} dist(seq_i, seq_j) & \text{if } dist(seq_i, seq_j) \leq dist_k(seq_i) \wedge Density(seq_j, k) < Density(seq_i, k) \\ MAX\_DIST & \text{if } dist(seq_i, seq_j) \leq dist_k(seq_i) \wedge Density(seq_j, k) = Density(seq_i, k) \\ \infty & \text{otherwise} \end{cases} \tag{6}$$

where $dist(seq_i, seq_j)$ is defined in Equation 2, $Density(seq_i, k)$ and $dist_k(seq_i)$ are defined in Equation 5, and $MAX\_DIST = \max\{dist(seq_i, seq_j)\} + 1$ for all i, j. Note that the proximity matrix is no longer symmetric. Step 2 in Algorithm 1 builds the single linkage trees with all distances smaller than MAX_DIST. Then in Step 3, the single linkage trees connected by MAX_DIST are linked if the density of one tree is greater than the density of the other to merge any local maximal regions. The density of a tree (cluster) is the maximum density over all sequence densities in the cluster. We use Algorithm 1 because it is more efficient than implementing the single linkage based algorithm.

The uniform kernel density based k-NN clustering has major improvements over the regular single linkage method. First, the use of k nearest neighbors in defining the density reduces the instability due to ties or outliers when $k > 1$ [3]. In density based k nearest neighbor clustering, the linkage is based on the local density estimate as well as the distance between points. That is, the linkage to the closest point is only made when the neighbor is more dense than itself. This still gives the algorithm the flexibility in the shape of the cluster as in single linkage methods, but reduces the instability due to outliers.

Second, use of the input parameter k as the local influential region provides a natural cut of the linkages made. An unsolved problem in the single linkage method is how to cut the one large linkage tree into clusters. In this density based method, by linking only up to the k nearest neighbors, the data is automatically separated at the valleys of the density estimate into several linkage trees.

Obviously, different k values will result in different clusters. However, this does not imply that the natural boundaries in the data change with k. Rather, different values of k determine the resolution when locating the valleys. That is, as k becomes larger, more smoothing occurs in the density estimates over a larger local area in the algorithm. This results in lower resolution of the data. It is similar to blurring a digital image where the boundaries are smoothed. Practically speaking, the final effect is that some of the local valleys are not considered as boundaries anymore. Therefore, as the value of k gets larger, similar clusters are merged together resulting in fewer number of clusters. The benefit of using a small k value is that the algorithm can detect more patterns. The tradeoff is that it may break up clusters representing strong patterns (patterns that occur in many sequences) to generate multiple similar patterns [3]. As shown in the performance study (section 5.3.1), in many applications, a value of k in the range from 3 to 9 works well.

Table 8. Alignment of seq_2 and seq_3

seq_2	$\langle$(A)	()	(BCX)	(D)$\rangle$
seq_3	$\langle$(AE)	(B)	(BC)	(D)$\rangle$
Edit distance	$REPL((\text{A}), (\text{AE}))$	$INDEL((\text{B}))$	$REPL((\text{BCX}), (\text{BC}))$	$REPL((\text{D}), (\text{D}))$

4.2 Multiple Alignment: Compress into Weighted Sequences

Once sequences are clustered, sequences within a cluster are similar to each other. Now, the problem becomes how to summarize the general pattern in each cluster and discover the trend. In this section, we describe how to compress each cluster into one weighted sequence through multiple alignment.

The global alignment of sequences is obtained by inserting empty itemsets (i.e., $()$) 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 [6].

As shown in Table 8, finding the optimal alignment between two sequences is mathematically equivalent to the edit distance problem. The edit distance between two sequences seq_a and seq_b can be calculated by comparing itemsets in the aligned sequences one by one. If seq_a and seq_b have X and Y as their i^{th} aligned itemsets respectively, where $(X \neq ())$ and $(Y \neq ())$, then a $REPL(X, Y)$ operation is required. Otherwise, (i.e., seq_a and seq_b have X and $()$ 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 the recurrence relation given in Equation 1.

Generally, for a cluster C with n sequences $seq_1, \ldots, seq_n$, finding the *optimal global multiple alignment* that minimizes $\sum_{j=1}^{n} \sum_{i=1}^{n} dist(seq_i, seq_j)$ is an NP-hard problem [6], and thus is impractical for mining large sequence databases with many sequences. Hence in practice, people have approximated the solution by aligning two sequences first and then incrementally adding a sequence to the current alignment of $p - 1$ sequences until all sequences have been aligned. At each iteration, the goal is to find the best alignment of the added sequence to the existing alignment of $p - 1$ sequences. Consequently, the solution might not be optimal because once p sequences have been aligned, this alignment is permanent even if the optimal alignment of $p + q$ sequences requires a different alignment of the p sequences. The various methods differ in the order in which the sequences are added to the alignment. When the ordering is fair, the results are reasonably good [6].

seq_2	⟨(A)	()	(BCX)	(D)⟩	
seq_3	⟨(AE)	(B)	(BC)	(D)⟩	
$wseq_1$	⟨(A:2, E:1):2	(B:1):1	(B:2,C:2,X:1):2	(D:2):2)⟩	2

Figure 2. seq_2 and seq_3 are aligned resulting in $wseq_{11}$

4.2.1 Representation of the Alignment : Weighted Sequence

To align the sequences incrementally, the alignment results need to be stored effectively. Ideally the result should be in a form such that the next sequence can be easily aligned 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 strengths of items in the pattern.

Here, we propose a new representation for aligned sequences. A weighted sequence, denoted as $wseq = \langle WX_1 : v_1, \ldots, WX_l : v_l\rangle : n$, 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 aligned in the i^{th} position. These itemset information is summarized into the weighted itemset WX_i, where $(1 \leq i \leq l)$;

3. a weighted itemset in the alignment is in the form of $WX_i = (x_{j_1} : w_{j_1}, \ldots, x_{j_m} : w_{j_m})$, 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 $(1 \leq k \leq m)$.

We illustrate how to use weighted sequences to do multiple alignment using the example given in Table 6. Cluster 1 has seven sequences. The density descending order of these sequences is seq_2-seq_3-seq_4-seq_1-seq_5-seq_6-seq_{10}. Therefore, the sequences are aligned as follows. First, sequences seq_2 and seq_3 are aligned as shown in Figure 2. Here, we use a *weighted sequence* $wseq_1$ to summarize and compress the information about the alignment. Since the first itemsets of seq_2 and seq_3, (A) and (AE), are aligned, the first itemset in the weighted sequence $wseq_1$ is (A:2,E:1):2. It means that the two sequences are aligned in this position, and A and E appear twice and once respectively. The second itemset in $wseq_1$, (B:1):1, means there is only one sequence with an itemset aligned in this position, and item B appears once.

$wseq_1$	⟨(A:2, E:1):2	(B:1):1	(B:2,C:2,X:1):2	(D:2):2)⟩	2
seq_4	⟨(A)	()	(B)	(DE)⟩	
$wseq_2$	⟨(A:3,E:1):3	(B:1):1	(B:3,C:2,X:1):3	(D:3,E:1):3)⟩	3

Figure 3. Weighted sequence $wseq_1$ and seq_4 are aligned

$wseq_2$	⟨(A:3,E:1):3	(B:1):1	(B:3,C:2,X:1):3		(D:3,E:1):3)⟩	3
seq_1	⟨()	()	(BC)		(DE)⟩	
$wseq_3$	⟨(A:3,E:1):3	(B:1):1	(B:4,C:3,X:1):4		(D:4,E:2):4)⟩	4
seq_5	⟨(AX)	(B)	(BC)	(Z)	(AE)⟩	
$wseq_4$	⟨(A:4,E:1,X:1):4	(B:2):2	(B:5,C:4,X:1):5	(Z:1):1	(A:1,D:4,E:3):5)⟩	5
seq_6	⟨(AY)	(BD)	(B)	()	(EY)⟩	
$wseq_5$	⟨(A:5,E:1,X:1,Y:1):5	(B:3,D:1):3	(B:6,C:4,X:1):6	(Z:1):1	(A:1,D:4,E:4,Y:1):6)⟩	6
seq_{10}	⟨(V)	()	()	(PW)	(E)⟩	
$wseq_6$	⟨(A:5,E:1,V:1,X:1,Y:1):6	(B:3,D:1):3	(B:6,C:4,X:1):6	(P:1, W:1, Z:1):2	(A:1,D:4,E:5,Y:1):7)⟩	7

Figure 4. The alignment of remaining sequences in cluster 1

4.2.2 Sequence to Weighted Sequence Alignment

After this first step, we need to iteratively align other sequences with the current weighted sequence. However, the weighted sequence does not explicitly keep information about the individual itemsets in the aligned sequences. Instead, this information is summarized into the various 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 $REPL()$ directly, we adopt a *weighted replace cost* as follows.

Let $WX = (x_1 : w_1, \ldots, x_m : w_m) : v$ be a weighted 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

$$REPL_W(WX, Y) = \frac{R' \cdot v + n - v}{n} \text{ where } R' = \frac{\sum_{i=1}^{m} w_i + \|Y\|v - 2\sum_{x_i \in Y} w_i}{\sum_{i=1}^{m} w_i + \|Y\|v} \tag{7}$$

Accordingly, we have

$$INDEL(WX) = REPL_W(WX, ()) = 1 \text{ and } INDEL(Y) = REPL_W(Y, ()) = 1 \tag{8}$$

We first illustrate this in our running example. The weighted sequence $wseq_1$ and the third sequence seq_4 are aligned as shown in Figure 3. Similarly, the remaining sequences in cluster 1 can be aligned as shown in Figure 4.

Now let us examine the example to better understand the notations and ideas. Table 9 depicts the situation after the first four sequences ($seq_2, seq_3, seq_4, seq_1$) in cluster 1 have been aligned into weighted sequence $wseq_3$, and the algorithm is computing the distance of aligning the first itemset of seq_5, s_{51} =(AX), to

Table 9. An example of $REPL_W()$

sequence ID	itemset ID	itemset	distance	
seq_5	s_{51}	(AX)	$R' = \frac{(4+2*3)-2*3}{(4+2*3)} = \frac{2}{5} = \frac{36}{90}$	
$wseq_3$	ws_{31}	(A:3,E:1):3 - n=4	$REPL_W = [(\frac{2}{5})*3+1]/4 = \frac{11}{20} = \frac{66}{120}$	
seq_2	s_{21}	(A)	$REPL((\text{A}),(\text{AX})) = \frac{1}{3}$	$Avg = \frac{7}{18} = \frac{35}{90}$
seq_3	s_{31}	(AE)	$REPL((\text{AE}),(\text{AX})) = \frac{1}{2}$	
seq_4	s_{41}	(A)	$REPL((\text{A}),(\text{AX})) = \frac{1}{3}$	
seq_1	$INDEL$	()	$INDEL((\text{AX})) = 1$	
Actual avg distance over the four sequences = $\frac{13}{24} = \frac{65}{120}$				

the first position (ws_{31}) in the current alignment. ws_{31}=(A:3,E:1):3 because there are three A's and one E aligned into the first position, and there are three non-null items in this position. Now, using Equation 7, $R' = \frac{2}{5} = \frac{36}{90}$ and $REPL_W = \frac{11}{20} = \frac{66}{120}$ as shown in the first two lines of the table. The next four lines calculate the distance of each individual itemset aligned in the first position of $wseq_3$ with s_{51} =(AX). The actual average over all non-null itemsets is $\frac{35}{90}$ and the actual average over all itemsets is $\frac{65}{120}$. In this example, R' and $REPL_W$ approximate the actual distances very well.

The rationale of $REPL_W()$ 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 position*. Then, R' is the estimated average replacement cost for all sequences that have a non-empty itemset aligned in this position. There are in total v such sequences. In Table 9, $R' = \frac{36}{90}$ which well approximates the actual average of $\frac{35}{90}$.

- *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 7 estimates the average of the cost in the two cases. Used in conjunction with $REPL_W(WX, Y)$, weighted sequences are an effective representation of the n aligned sequences and allow for efficient multiple alignment.

The distance measure $REPL_W(WX, Y)$ has the same useful properties of $REPL(X, Y)$– it is a metric and ranges from 0 to 1. Now we simply use the recurrence relation given in Equation 1 replacing $REPL(X, Y)$ with $REPL_W(WX, Y)$ to align all sequences in the cluster.

The result of the multiple alignment is a weighted sequence. A weighted sequence records the information of the alignment. The alignment result for all sequences in cluster 1 and 2 are summarized in the weighted sequences shown in Tables 6 and 7. Once a weighted sequence is derived, the sequences in the cluster will not be visited anymore in the remainder of the mining. Hence, after the alignment, the individual sequential data can be discarded and only the weighted sequences need to be stored for each cluster.

4.2.3 Order of the Alignment

In incremental multiple alignment, the ordering of the alignment should be considered. In a cluster, in comparison to other sequences, there may be some sequences that are more similar to many other sequences. In other words, such sequences may have many close neighbors with high similarity. These sequences are most likely to be closer to the underlying patterns than the other sequences. Hence, 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 5 measures the similarity between a sequence and its nearest neighbors. Thus, a sequence with 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 near optimal.*

The ordering works well because in a cluster, the densest sequence is the one that has the most similar sequences - in essence the sequence with the least noise. The alignment starts with this point, and then incrementally aligns the most similar sequence to the least similar. In doing so, the weighted sequence forms a center of mass around the underlying pattern to which sequences with more noise can easily attach itself. Consequently, ApproxMAP is very robust to the massive outliers in real data because it simply ignores those that cannot be aligned well with the other sequences in the cluster. The experimental results show that the sequences are aligned fairly well with this ordering.

But most importantly, although aligning the sequences in different order may result in slightly different weighted sequences, it does not change the underlying pattern in the cluster. To illustrate the effect, Table 10 shows the alignment result of cluster 1 using a random order(reverse id), seq_{10}-seq_6-seq_5-seq_4-seq_3-seq_2-seq_1. Interestingly, the two alignment results are quite similar, only some items shift positions slightly. Compared to Table 6, the first itemset and second itemset in seq_{10}, (V) and (PW), and the second itemset in seq_4, (B), each shifted one position. This causes the item weights to be reduced slightly. Yet the con-

Table 10. Aligning sequences in cluster 1 using a random order

seq_{10}	⟨()	(V)	(PW)	()	(E)⟩	
seq_6	⟨(AY)	(BD)	(B)	()	(EY)⟩	
seq_5	⟨(AX)	(B)	(BC)	(Z)	(AE)⟩	
seq_4	⟨(A)	(B)	()	()	(DE)⟩	
seq_3	⟨(AE)	(B)	(BC)	()	(D)⟩	
seq_2	⟨(A)	()	(BCX)	()	(D)⟩	
seq_1	⟨()	()	(BC)	()	(DE)⟩	
Weighted Seq	(A:5, E:1,X:1,Y:1):5	(B:4, D:1, V:1):5	(B:5, C:4,P:1,W:1,X:1):6	(Z:1):1	(A:1,D:4, E:5,Y:1):7	7
ConSeq ($w > 3$)	⟨(A)	(B)	(BC)		(DE)⟩	

sensus patterns from both orders, ⟨(A)(B)(BC)(DE)⟩, are identical. As verified by our extensive empirical evaluation, the alignment order has little effect on the underlying patterns.

4.3 Summarize into Consensus Patterns

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 $WS = \langle(x_{11} : w_{11}, \ldots, x_{1m_1} : w_{1m_1}) : v_1, \ldots, (x_{l1} : w_{l1}, \ldots, x_{lm_l} : w_{lm_l}) : v_l\rangle : n$, 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* $(0 \leq min\_strength \leq 1)$. 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.

In our running example $min\_strength = 40\%$. Thus, the consensus pattern in cluster 1 selected all items with weight greater than 3 sequences ($w \geq 40\% * 7 = 2.8$) while the consensus pattern in cluster 2 selected all items with weight greater than 1 sequence ($w \geq 40\% * 3 = 1.2$).

5 Empirical Evaluations

It is important to understand the approximating behavior of ApproxMAP. The accuracy of the approximation can be evaluated in terms of how well it finds the real underlying patterns in the data and whether or not it generates any spurious patterns. However, it is difficult to calculate analytically what patterns will be generated because of the complexity of the algorithm.

As an alternative, we have developed a general evaluation method that can objectively evaluate the quality of the results produced by any sequential pattern mining algorithm. Using this method, one can understand the behavior of an algorithm empirically by running extensive systematic experiments on synthetic data with

Table 11. Parameters and the default configuration for the data generator.

Notation	Original Notation	Meaning	Default value
$\|\|I\|\|$	N	# of items	1000
$\|\|\Lambda\|\|$	N_I	# of potentially frequent itemsets	5000
N_{seq}	$\|\|D\|\|$	# of data sequences	10000
N_{pat}	N_S	# of base pattern sequences	100
L_{seq}	$\|\|C\|\|$	Avg. # of itemsets per data sequence	20
L_{pat}	$\|\|S\|\|$	Avg. # of itemsets per base pattern	$0.7 \cdot L_{seq}$
I_{seq}	$\|\|T\|\|$	Avg. # of items per itemset in the database	2.5
I_{pat}	$\|\|I\|\|$	Avg. # of items per itemset in base patterns	$0.7 \cdot I_{seq}$

known base pattern [9].

5.1 Evaluation Method

Central to the evaluation method is a well designed synthetic dataset with known patterns. The well known IBM data generator takes several parameters and outputs a sequence database as well as a set of base patterns [2]. 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, base patterns are the underlying template behind the database. By matching the results of the sequential pattern mining methods back to these base patterns, we are able to determine how much of the base patterns have been found as well as how much spurious patterns have been generated. We summarize the parameters of the data generator and the default configuration in Table 11. The default configuration was used in the experiments unless otherwise specified.

In [9], we present a comprehensive set of evaluation criteria to use in conjunction with the IBM data generator to measure how well the results match the underlying base patterns. We define *pattern items* as those items in the result pattern that can be mapped back to the base pattern. Then the *extraneous items* are the remaining items in the results. The mix of pattern items and extraneous items in the result patterns determine the recoverability and precision of the results. Recoverability is a weighted measure that provides a good estimation of how much of the items in the base patterns were detected. Precision, adopted from ROC analysis, is a good measure of how many extraneous items are mixed in with the correct items in the result patterns. Both recoverability and precision are measured at the item level. On the other hand, the numbers of max patterns, spurious patterns, and redundant patterns along with the total number of patterns returned give an overview of the result at the sequence level. The evaluation criteria are summarized in Table 12. The details of each criteria can be found in [9] as well as the appendix. In summary, a good paradigm would

Table 12. Evaluation criteria

criteria	Meaning	Level	Unit
$\mathcal{R}$	Recoverability: the degree of the base patterns detected (See Appendix)	item	%
$\mathcal{P}$	Precision: 1-degree of extraneous items in the result patterns (See Appendix)	item	%
N_{max}	# of base patterns found	seq	# of patterns
N_{spur}	# of spurious patterns ($N_{extraI} > N_{patI}$)	seq	# of patterns
N_{redun}	# of redundant patterns	seq	# of patterns
N_{total}	Total # of result patterns returned $(= N_{max} + N_{spur} + N_{redun})$	seq	# of patterns

produce (1) high recoverability and precision, with (2) small number of spurious and redundant patterns, and (3) a manageable number of result patterns.

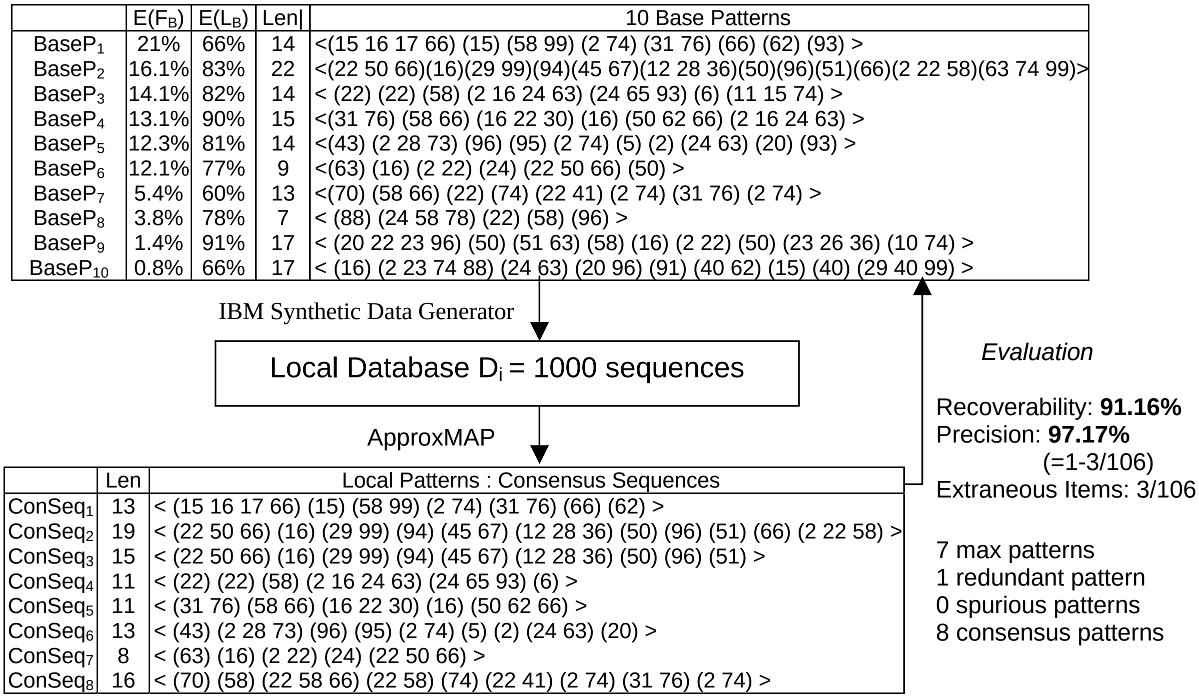
5.2 Effectiveness of ApproxMAP

In [10], we published a through comparison study of the support model and the alignment model for sequential pattern mining. In this section, we summarize the important results on the effectiveness of ApproxMAP under a variety of randomness and noise level.

Figure 5 demonstrates how 7 of the most frequent 10 base patterns were uncovered from 1000 sequences using ApproxMAP. Clearly, each of the 8 consensus patterns match a base pattern well. In general, the consensus patterns recover major parts of the base patterns with high expected frequency in the database. The recoverability is quite good at 91.16%.

Precision is excellent at $\mathcal{P} = 1 - \frac{3}{106} = 97.17\%$. Clearly all the consensus patterns are highly shared by sequences in the database. In all the consensus patterns, there is only three items (the items 58, 22, and 58 in the first part of $PatConSeq_8$) that do not appear on the corresponding position in the base pattern. These items are not random items injected by the algorithm, but rather repeated items, which clearly come from the base pattern $BaseP_7$. These items are still classified as extraneous items because the evaluation method uses the most conservative definition for pattern items.

There were no spurious patterns and only one redundant pattern. It is interesting to note that a base pattern may be recovered by multiple consensus patterns. For example, ApproxMAP forms two clusters whose consensus patterns approximate base pattern $BaseP_2$. This is because $BaseP_2$ is long (the actual length of the base pattern is 22 items and the expected length of the pattern in a data sequence is 18 items) and has a high expected frequency (16.1%). Therefore, many data sequences in the database are generated using $BaseP_2$ as a template. In the IBM data generator, sequences are generated by removing various parts of the base pattern and combining them with items from other base patterns. Thus, two sequences using the



	$E(F_B)$	$E(L_B)$	Len|	10 Base Patterns
$BaseP_1$	21%	66%	14	<(15 16 17 66) (15) (58 99) (2 74) (31 76) (66) (62) (93) >
$BaseP_2$	16.1%	83%	22	<(22 50 66)(16)(29 99)(94)(45 67)(12 28 36)(50)(96)(51)(66)(2 22 58)(63 74 99)>
$BaseP_3$	14.1%	82%	14	< (22) (22) (58) (2 16 24 63) (24 65 93) (6) (11 15 74) >
$BaseP_4$	13.1%	90%	15	<(31 76) (58 66) (16 22 30) (16) (50 62 66) (2 16 24 63) >
$BaseP_5$	12.3%	81%	14	<(43) (2 28 73) (96) (95) (2 74) (5) (2) (24 63) (20) (93) >
$BaseP_6$	12.1%	77%	9	<(63) (16) (2 22) (24) (22 50 66) (50) >
$BaseP_7$	5.4%	60%	13	<(70) (58 66) (22) (74) (22 41) (2 74) (31 76) (2 74) >
$BaseP_8$	3.8%	78%	7	< (88) (24 58 78) (22) (58) (96) >
$BaseP_9$	1.4%	91%	17	< (20 22 23 96) (50) (51 63) (58) (16) (2 22) (50) (23 26 36) (10 74) >
$BaseP_{10}$	0.8%	66%	17	< (16) (2 23 74 88) (24 63) (20 96) (91) (40 62) (15) (40) (29 40 99) >

IBM Synthetic Data Generator
Local Database D_i = 1000 sequences
ApproxMAP
Evaluation

Recoverability: **91.16%**
Precision: **97.17%**
(=1-3/106)
Extraneous Items: 3/106

7 max patterns
1 redundant pattern
0 spurious patterns
8 consensus patterns

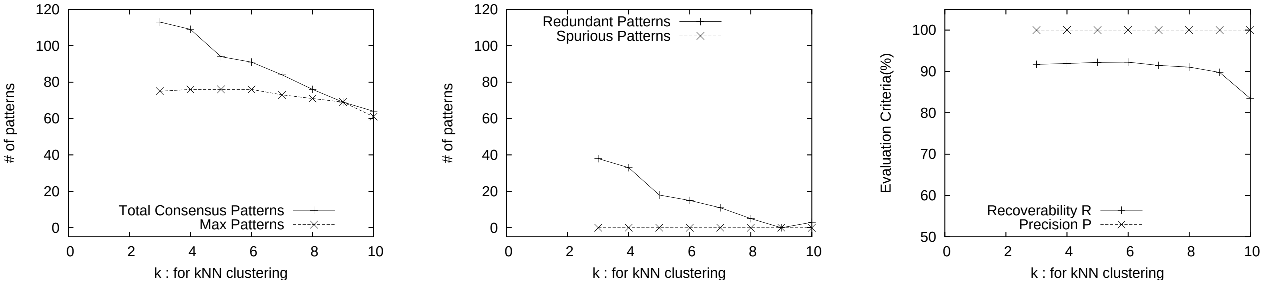
	Len	Local Patterns : Consensus Sequences
$ConSeq_1$	13	< (15 16 17 66) (15) (58 99) (2 74) (31 76) (66) (62) >
$ConSeq_2$	19	< (22 50 66) (16) (29 99) (94) (45 67) (12 28 36) (50) (96) (51) (66) (2 22 58) >
$ConSeq_3$	15	< (22 50 66) (16) (29 99) (94) (45 67) (12 28 36) (50) (96) (51) >
$ConSeq_4$	11	< (22) (22) (58) (2 16 24 63) (24 65 93) (6) >
$ConSeq_5$	11	< (31 76) (58 66) (16 22 30) (16) (50 62 66) >
$ConSeq_6$	13	< (43) (2 28 73) (96) (95) (2 74) (5) (2) (24 63) (20) >
$ConSeq_7$	8	< (63) (16) (2 22) (24) (22 50 66) >
$ConSeq_8$	16	< (70) (58) (22 58 66) (22 58) (74) (22 41) (2 74) (31 76) (2 74) >

Figure 5. Effectiveness of ApproxMAP

same long base pattern as the template are not necessarily similar to each other. As a result, the sequences generated from a long base pattern can be partitioned into multiple clusters by ApproxMAP. One cluster with sequences that have almost all of the 22 items from $BaseP_2$ ($PatConSeq_2$) and another cluster with sequences that are shorter ($PatConSeq_3$). The one which shares less with the base pattern, $PatConSeq_3$, is classified as a redundant pattern in the evaluation method.

In summary, the evaluation results reveal that ApproxMAP returns a succinct yet accurate summary of the base patterns with few redundant patterns and no spurious patterns. Further experiments demonstrate that ApproxMAP is also robust to both noise and outliers in the data [10].

In comparison, in the absence of noise in the data the support model can find the underlying patterns in the data. However, the real patterns are buried under the huge number of spurious and redundant sequences. In the presence of noise in the data, the recoverability degrades quickly in the support model [10].



(a) N_{total} & N_{max} w.r.t. k (b) N_{redun} & N_{spur} w.r.t. k (c) R & P w.r.t. k

Figure 6. Effects of k

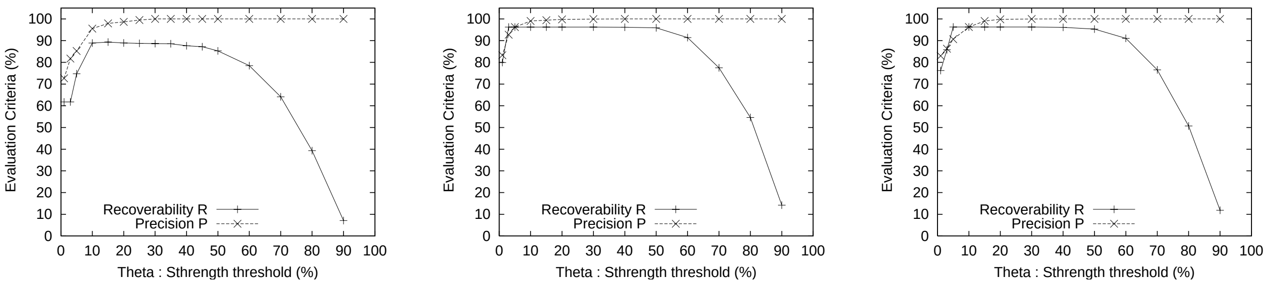
5.3 Parameters in ApproxMAP

5.3.1 k in k-Nearest Neighbor Clustering

Here, we study the influence and sensitivity of the user input parameter k. We fix other settings and vary the value of k from 3 to 10, where k is the nearest neighbor parameter in the clustering step. The results are shown in Figure 6. There were no extraneous items (i.e. Precision=100%) or spurious patterns. As expected, a larger value of k produces less number of clusters, which leads to less number of patterns. Hence, when k increases in Figure 6(a), the total number of consensus patterns decreases. However most of the reduction in the consensus patterns are redundant patterns for $k = 3..9$ as seen in Figure 6(b). That is the number of max patterns are fairly stable for $k = 3..9$ at around 75 patterns (Figure 6(a)). Thus, the reduction in the total number of consensus patterns returned does not have much effect on recoverability (Figure 6(c)). When k is too large though ($k = 10$), there is a noticeable reduction in the number of max patterns from 69 to 61 (Figure 6(a)). This causes loss of some weak base patterns and thus the recoverability decreases somewhat as shown in Figure 6(c). Figure 6(c) demonstrates that there is a wide range of k that give comparable results. In this experiment, the recoverability is sustained with no change in precision for a range of $k = 3..9$. In short, ApproxMAP is fairly robust to k. This is a typical property of density based clustering algorithms.

5.3.2 The Strength Cutoff Point $min\_strength$

In ApproxMAP, the strength cutoff point $min\_strength$ is used to filter out noise from weighted sequences. Here, we study the strength cutoff point to determine its properties empirically. We ran several experiments on different databases. The data was generated with the same parameters given in Table 11 except for L_{seq}.



(a) $L_{seq} = 10$ (b) $L_{seq} = 30$ (c) $L_{seq} = 50$

Figure 7. Effects of $min\_strength$

L_{seq} was varied from 10 to 50. We then studied the change in recoverability and precision as $min\_strength$ is changed for each database. Selected results are given in Figure 7. Without a doubt, the general trend is the same in all databases.

In all databases, as $min\_strength$ is decreased from 90%, recoverability increases quickly until it levels off at $\theta = 50\%$. Precision stays high at close to 100% until $min\_strength$ becomes quite small. Clearly, when $\theta = 50\%$, ApproxMAP is able to recover most of the items from the base pattern without picking up any extraneous items. That means that items with strength greater than 50% are all pattern items. Thus, as a conservative estimate, the default value for $min\_strength$ is set at 50%.

On the other side, when $min\_strength$ is too low precision starts to drop. Furthermore, in conjunction with the drop in precision, there is a point at which recoverability drops again. This is because, $min\_strength$ is too low the noise is not properly filtered out. As a result too many extraneous items are picked up. This in turn has two effects. By definition, precision is decreased. Even more damaging, the consensus patterns with more than half extraneous items now become spurious patterns and do not count toward recoverability. This results in the drop in recoverability. In the database with $L_{seq} = 10$ this occur at $\theta \leq 10\%$. When $L_{seq} > 10$, this occurs when $\theta \leq 5\%$. The drop in precision starts to occur when $\theta < 30\%$ for the database with $L_{seq} = 10$. In the databases of longer sequences, the drop in precision starts near $\theta = 20\%$. This indicates that items with strength greater than 30% are probably items in the base patterns. Moreover, in all databases, when $\theta \leq 10\%$, there is a steep drop in precision. This indicates, that many extraneous items are picked up when $\theta \leq 10\%$. These results indicate that most of the items with strength less than 10% are extraneous items, because recoverability is close to 100% when $\theta = 10\%$.

In summary, ApproxMAP is also robust to the strength cutoff point. This experiment indicates that 20%-50% is in fact a good range for the strength cutoff point for a wide range of databases.

Table 13. Results for different ordering

Order	Recoverability	N_{extraI}	N_{patI}	$N_{commonI}$	Precision	N_{spur}	N_{redun}	N_{total}
Descending Density	92.17%	0	2245	2107	100.00%	0	18	94
Ascending Density	91.78%	0	2207	2107	100.00%	0	18	94
Random (ID)	92.37%	0	2240	2107	100.00%	0	18	94
Random (Reverse ID)	92.35%	0	2230	2107	100.00%	0	18	94

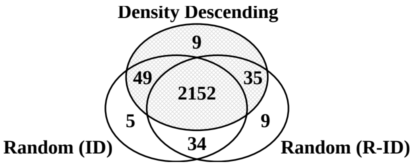


Figure 8. Comparison of pattern items found for different ordering

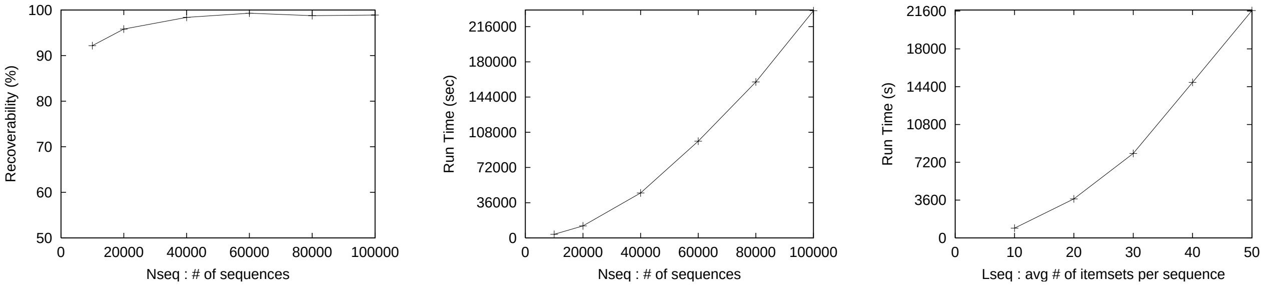
5.3.3 The Order in Multiple Alignment

Now, we study the sensitivity of the multiple alignment results on the order of sequences in the alignment. We compare the mining results using the density-descending order, density-ascending order, and two random orders (sequence-id ascending and descending order). As expected, although the exact alignment changes slightly depending on the orders, it has very limited effect on the consensus patterns. The results show that (Table 13), all four orders generated the exact same number of patterns that were very similar to each other. The number of pattern items detected that were identical in all four orders, column $N_{commonI}$, was 2107. In addition, each order found an additional 100 to 138 pattern items. Most of these additional items were found by more than one order. Therefore, the recoverability is basically identical at 92%.

While aligning patterns in density descending order tends to improve the alignment quality (the number of pattern items found, N_{patI}, is highest for density descending order at 2245 while lowest for density ascending order at 2207), ApproxMAP itself is robust with respect to alignment orders. In fact, the two random ordering tested gave comparable number of pattern items as the density descending order. Figure 8 gives a detailed comparison of the pattern items detected by the two random orders and the density descending order. Essentially, a random order detected about 98% ($2201/2245 = 98\% \simeq 2187/2245$) of the pattern items detected by the density descending order plus a few more pattern items (roughly $40 \simeq 34 + 5 \simeq 34 + 9$) not detected by the density descending order.

5.4 Scalability

Finally, the default configuration of the synthetic data was altered to test the effects of the parameters of the database. The details can be found in [9]. Limited by space, we briefly summarize the results.



(a) Recoverability w.r.t. N_{seq} (b) Running Time w.r.t. N_{seq} (c) Running Time w.r.t. L_{seq}

Figure 9. Select results

In general, we found that the larger the dataset, the better the effectiveness of ApproxMAP. For example, with respect to N_{seq}, the more the sequences in the database, the better the recoverability. Figure 9(a) is the recoverability with respect to N_{seq} given 100 base patterns. In large databases, there are more sequences approximating the patterns. For example, if there are 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. 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.

Moreover, we observe that ApproxMAP is scalable w.r.t. database size, as shown in Figures 9(b) and 9(c). ApproxMAP has total time complexity of $O(N_{seq}^2 \cdot L_{seq}^2 \cdot I_{seq} + k \cdot N_{seq}) = O(N_{seq}^2 \cdot L_{seq}^2 \cdot I_{seq})$ where N_{seq} is the total number of sequences, L_{seq} is the average length of the sequence, I_{seq} is the maximum number of items in an itemset, and k is the number of nearest neighbors considered. The complexity is dominated by the clustering step which has to calculate the proximity matrix ($O(N_{seq}^2 \cdot L_{seq}^2 \cdot I_{seq})$) and build the k nearest neighbor list ($O(k \cdot N_{seq})$).

Practically speaking, the running time is constant with respect to most other dimensions except the size of the database. That is, ApproxMAP scales well with respect to k and the length and number of patterns. The length and number of patterns in the data do not affect the running time because ApproxMAP finds all the consensus patterns directly from the data without having to build the patterns.

5.5 Case Study:Mining The Welfare Services DB

The above analysis strongly indicates that ApproxMAP can efficiently summarize a local database and reduce the cost for global mining. The next step is to evaluate ApproxMAP on real data. However, real sequential data is difficult to find. Most real data available in data repositories are too small for mining sequential data. Even the unusually large *Gazella* data set from KDD-CUP 2000 only has an average of less than 2 itemsets per sequence [8]. Hence, the dataset cannot be used to mine sequential patterns.

Nonetheless, we were able to access a private sequence data. Due to confidentiality issues, we breifly report the result on a real database 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.

As an example, 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 database, variations of this consensus pattern overwhelm the results, because roughly half of the sequences in this database 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 57 children who had 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 below.

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

where HM stands for Home Management Services and T stands for Transportation. There were 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 with min_sup set to 20%, there is more than $100,000$ sequential patterns and the users just cannot identify the needle from the straws.

6 Related Work

Multi-database mining has been widely researched in previous works [20, 21, 25, 26, 27, 28]. The overview of multi-database mining is introduced in [26] and [28]. They introduce the difference between multi-database mining and mono-database mining, and present novel significant patterns that are found in multi-database mining but not in mono-database mining. Wu et al. [21] proposes a database classification technique for multi-database mining. Local databases in the multi-database are classified into several groups based on their similarity between each other. This can reduce the search cost in the multi-database. To find more valuable information in a multi-database, techniques to synthesize local patterns to find exceptional patterns and high vote patterns are presented in [20, 25, 27]. These previous researches for mining multi-database have focused on finding global patterns from frequent itemsets or association rules in local databases. In this paper, we expand the research in multi-database mining to identify high vote sequential patterns and exceptional sequential patterns.

Many papers have proposed methods for finding frequent subsequences in a mono-database [1, 13, 17, 24]. There are three works in particular, that extend the support model to find more useful patterns. [16] extends the framework to find rules of the form ”if A then B” using the confidence framework. [23] presents a probabilistic model to handle noise in mining strings. However, it cannot be easily generalized to sequence of sets. [22] proposes a method for mining frequent closed subsequences using several efficient search space pruning methods. However, all these methods do not address the issue of generating huge number of patterns that share significant redundancy.

There is a rich body of literature on string analysis in computer science as well as computational biology that can be extended to this problem domain. In particular, multiple alignment has been studied extensively in computational biology [5, 6, 18] to find common patterns in a group of strings. In this paper, we have generalized string multiple alignment to find patterns in sequences of sets.

7 Future Work and Conclusion

7.1 Conclusion

Recently, global mining from multiple sources has received much attention. A key factor in efficient global mining is to summarize the local information in the local data mining process, and then to forward only the summary to the global mining process. However, conventional sequential pattern mining methods using the support model have inherent limitations that make it inefficient for mining multi-databases. In this paper, we have presented an alternative model, *approximate sequential pattern mining*, for accurately summarizing each local database, which can reduce the cost for global mining. An efficient algorithm, ApproxMAP, is proposed to find approximate sequential patterns, called *consensus patterns*, via multiple alignment. Extensive study on synthetic and real data demonstrate that ApproxMAP (1) can effectively summarize the local data (2) is robust to noise and outliers in the data, and (3) is robust to the input parameters k and $min\_strength$ as well as the order of alignment. In short, ApproxMAP is effective and scalable in mining large sequence databases with long patterns.

Furthermore, local consensus patterns mined using ApproxMAP can be collected and processed to find global patterns from multiple sources. We present an elegant and uniform model, *homogeneous sets*, to detect both the high vote sequential patterns and exceptional sequential patterns. By grouping local patterns by the desired similarity level, homogeneous sets can easily identify global patterns that are supported by most local databases as well as global patterns that are rare.

7.2 Future Work

One practical improvement to ApproxMAP would be to automatically detect the best strength threshold, θ, for each cluster of sequences. An interesting approach could be analyzing the distribution of the item weights dynamically. Initial investigation seems to suggest that the item weights may follow the Zipf distribution. Closer examination of the distribution might give hints for automatically detecting statistically

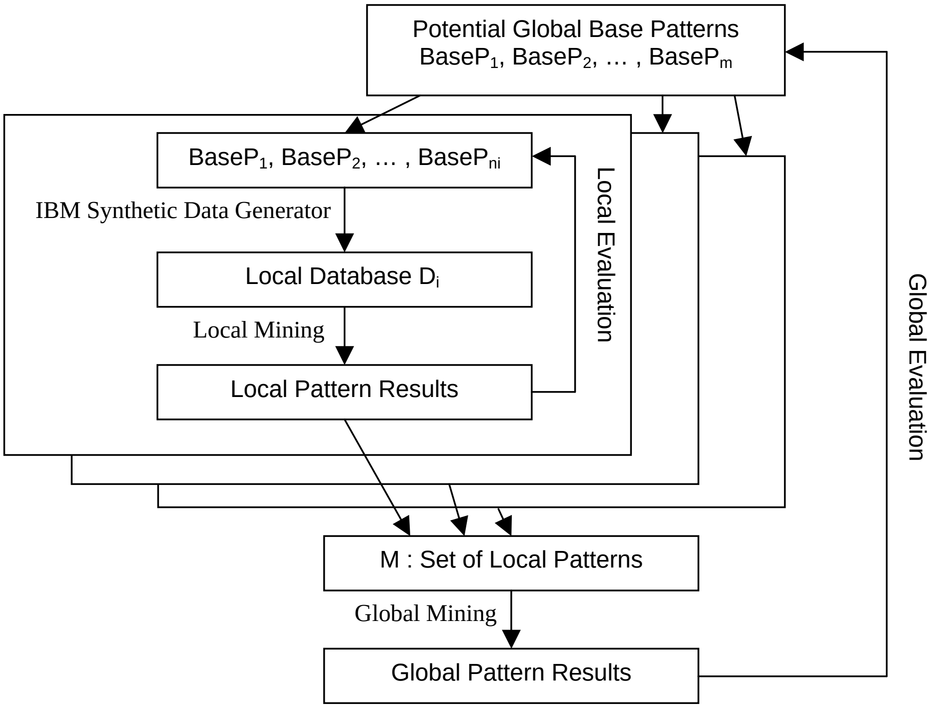

Potential Global Base Patterns
BaseP1, BaseP2, … , BasePm
BaseP1, BaseP2, … , BasePni
IBM Synthetic Data Generator
Local Database Di
Local Mining
Local Pattern Results
Local Evaluation
Global Evaluation
M : Set of Local Patterns
Global Mining
Global Pattern Results

Figure 10. Evaluation of Global Mining Algorithms

significant cutoff values customized for each cluster. When presenting an initial overview of the data, such approach could be quite practical.

A larger research question for future study would be the evaluation of global sequential pattern mining. Evaluation of global sequential pattern mining is still an open problem. Access to real sequential data that have useful global patterns is difficult with most data available in data repositories being too small even for local sequential pattern mining. In [9], we have proposed an effective evaluation method for assessing local sequential pattern mining methods using the IBM synthetic data generator. A similar approach can be taken for evaluating global pattern mining methods as shown in Figure 10. The additional layer of selecting local base patterns from the set of potential global base patterns can ensure that global patterns exit in the multi-database. As a preliminary study, we tested our algorithm on a multi-database with 10 local databases. We first generated 100 potential global base patterns. Then each of the 10 local databases randomly selected 10 unique base patterns from the 100 global base patterns to use as a template for its local database. To model the variation and noise in the real data, the selected base patterns were randomly perturbed so that each of the local databases did not have any base patterns that were identical. We then manually identified the high vote and exceptional patterns that were embedded into the mutli-database by analyzing the base patterns used in each of the 10 local databases. Finally, we could assess whether these embeded global patterns were found by reviewing the global result patterns returned from our algorithm. Indeed, we confirmed that all global patterns were properly detected via homogenous sets. For a more systematically evaluation of global mining algorithms, further research is needed on how to generate realistic multi-databases with known embedded global patterns, as well as good evaluation criteria to apply to the synthetic data.

References

[1] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the ACM International Conference on Knowledge discovery and data mining (SIGKDD)*, pages 429–435, July 2002.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of International Conference on Data Engineering (ICDE)*, pages 3–14, Taipei, Taiwan, March 1995.

[3] L. Ertoz, M. Steinbach, and V. Kumar. Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data. In *Third SIAM International Conference on Data Mining(SDM)*, pages 47–58 San Fransico. CA, 2003.

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

[5] Osamu Gotoh. Multiple sequence alignment: Algorithms and applications. In *Adv. Biophys.*, Vol. 36, pages 159–206. 1999.

[6] D. Gusfield. *Algorithms on strings, trees, and sequences: Computer Science and Computational Biology*. Cambridge Univ. Press, Cambridge, England. 1997.

[7] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. In *ACM Computing Surveys*, Vol. 31(3), pages 264–323, Sep 1999.

[8] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDD-CUP 2000 Organizers' Report: Peeling the Onion. In *Proc. SIGKDD Explorations*, vol 2, pages 86–98, 2000.

[9] H. C. Kum *Approximate Mining of Consensus Sequential Patterns.* Ph.D. Dissertation. University of North Carolina at Chapel Hill. 2004.

[10] H. C. Kum, S. Paulsen, and W. Wang Comparitive Study of Sequential Pattern Mining Models *IEEE ICDM Workshop on The Foundation of Data Mining and Discovery*. Maebashi, Japan, Dec 2002. To be published as Lecture Series in Computer Science.

[11] H. C. Kum, J. Pei, W. Wang, and D. Duncan. ApproxMAP : Approximate Mining of Consensus Sequential Patterns. In *Third SIAM International Conference on Data Mining(SDM)*, pages 311-315 San Fransico. CA, 2003.

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

[13] J. Pei, J. Han, et al. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. of International Conference on Data Engineering (ICDE)*, pages 215–224, April 2001.

[14] J. Sander, M. Ester, H. P. Kriegel, and X. Xu. Density based clustering in spatial databases: The algorithm gdbscan and its applications. In *Data Mining and Knowledge Discovery*, Vol 2(2), pages 169–194, 1998.

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

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

[17] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. 6th Intl. Conf Extending Database Technology (EDBT)*, pages 3–17. March 1996.

[18] J. Thompson, F. Plewniak, and O. Poch. A comprehensive comparison of multiple sequence alignment programs. In *Nucleic Acids Research.* Vol. 27(13), pages 2682–2690. Oxford Univ. Press. 1999.

[19] M. A. Wong and T. Lane. A kth Nearest Neighbor Clustering Procedure. In *Journal of the Royal Statistical Society*, Series B, 45, pages 362–368, 1983.

[20] X. Wu and S. Zhang. Synthesizing High-Frequency Rules from Different Data Sources. *IEEE Trans. Knowledge Data Engineering* 15(2): pages 353–367. 2003.

[21] X. Wu, C. Zhang, and S. Zhang. Database classification for multi-database mining. In *Information System* 30(1): pages 71–88. 2005.

[22] X. Yan, J. Han, and R. Afshar. CloSpan: Mining Closed Sequential Patterns in Larege Datasets. In *Third SIAM International Conference on Data Mining (SDM)*, pages 166–177, San Fransico. CA, 2003.

[23] J. Yang, P. S. Yu, W. Wang, and J. Han. Mining long sequential patterns in a noisy environment. In *Proc. of ACM Int'l Conf. On Management of Data (SIGMOD)*, pages 406–417, Madison, WI, 2002.

[24] M. J. Zaki. Efficient enumeration of frequent sequences. In *7th International Conference Information and Knowledge Management*, pages 68–75. Nov 1998.

[25] C. Zhang, M. Liu, W. Nie, and S. Zhang. Identifying Global Exceptional Patterns in Multi-database Mining. In *IEEE Computational Intelligence Bulletin* 3(1): pages 19–24. Feb 2004.

[26] S. Zhang, X. Wu, and C. Zhang. Multi-Database Mining. In *IEEE Computational Intelligence Bulletin* 2(1): pages 5–13. June 2003.

[27] S. Zhang, C. Zhang, and J. X. Yu. An efficient strategy for mining exceptions in multi-databases In *Information System* 165(1-2): pages 1–20. 2004.

[28] N. Zhong, Y. Yao, and S. Ohsuga. Peculiarity oriented multi-database mining. In *Proceedings of PKDD*, pages 136–146. 1999.