Intelligent Sequential Mining Via Alignment: Optimization Techniques for Very Large DB

Hye-Chung Kum[1], Joong Hyuk Chang[2], and Wei Wang[1]

[1] University of North Carolina at Chapel Hill, Chapel Hill, NC 27599, USA
[2] University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
{kum, weiwang}@cs.unc.edu, jhchang@uiuc.edu

Abstract. The shear volume of the results in traditional support based frequent sequential pattern mining methods has led to increasing interest in new intelligent mining methods to find more meaningful and compact results. One such approach is the *consensus sequential pattern mining* method based on sequence alignment, which has been successfully applied to various areas. However, the current approach to consensus sequential pattern mining has quadratic run time with respect to the database size limiting its application to very large databases. In this paper, we introduce two optimization techniques to reduce the running time significantly. First, we determine the theoretical bound for precision of the proximity matrix and reduce the time spent on calculating the full matrix. Second, we use a sample based iterative clustering method which allows us to use a much faster k-means clustering method with only a minor increase in memory consumption with negligible loss in accuracy.

1 Introduction

The goal of sequential pattern mining is to detect patterns in a database comprised of sequences of *itemsets*. For example, retail stores often collect customer purchase records in sequence databases in which a sequential pattern indicates a customer's buying habit. In such a database, each purchase is represented as a set of items, *itemsets*, purchased together, and a customer sequence would be a sequence of such itemsets.

Sequential pattern mining is commonly defined as finding the complete set of frequent subsequences [1]. Much research has been devoted to efficient discovery of such frequent sequential patterns [1] [7] [8]. However, such problem formulation of sequential patterns has some inherent limitations. First, the result set is huge and difficult to use without more post processing. Even the number of maximal or closed sequential patterns are huge, and many of the patterns are redundant and not useful. Second, the exact match based paradigm is vulnerable to noise and variations in the data. Many customers may share similar buying habits, but few follow exactly the same buying patterns. Finally, frequency alone cannot detect statistically significant patterns [4].

To overcome these limitations, recently there is an increasing interest in new intelligent mining methods to find more meaningful and compact results. The new methods abandon the traditional paradigm and take a fundamentally different

Z.-H. Zhou, H. Li, and Q. Yang (Eds.): PAKDD 2007, LNAI 4426, pp. 587–597, 2007.

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)**	(**DE**)⟩
consensus_pat	⟨(A)		(BC)	(D)⟩

approach. One such approach to intelligent sequential pattern mining is the *consensus sequential pattern mining* based on sequence alignment. Consensus 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. It can be used to detect general trends in the sequence database for natural customer groups, which is more useful than finding all frequent subsequences.

Formally, *consensus sequential patterns* are patterns 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 bold items are those that are shared with the consensus pattern. seq_1 has all items in *consensus_pat*, in the same order and grouping, except it is missing item A and has an additional item E. Similarly, seq_4 is missing C and has an extra E. In comparison, seq_2 and seq_3 have all items but each has a couple of extra items. These evidences strongly indicate that *consensus_pat* is the hidden underlying pattern behind the sequences. Such pattern mining of *consensus patterns* can effectively summarize the database into common customer groups and identify their buying patterns.

An effective algorithm for consensus sequential pattern mining has been proposed in [5]. The alignment based method, ApproxMAP(APPROXimate Multiple Alignment Pattern mining), has been applied to many areas such as multi-database mining [3], temporal streaming data mining [6], and policy analysis. Moreover, a detailed comparison study of the alignment based and support base methods has shown the effectiveness of ApproxMAP [4].

However, ApproxMAP has quadratic time complexity with respect to the size of the database limiting its application to very large databases. The time complexity is dominated by the clustering step which has to calculate the proximity matrix and build the clusters. In this paper, we introduce two effective optimization techniques. First, ApproxMAP can be optimized by calculating the proximity matrix to only the needed precision. In this paper, we introduce and prove the theoretical bound of the required precision reducing the running time considerably. Second, the clustering step can be improved by adapting the well known k-means method to ApproxMAP. Here, we introduce modifications to the typical algorithm that address the issues with calculating the mean, cluster initialization, and determining the number of clusters required. We further investigate the tradeoff between time and space empirically to determine the appropriate sample size and the utility of the optimization technique.

The remainder of the paper is organized as follows. Section 2 illustrates the basic ApproxMAP algorithm. The details and theoretical basis for the optimization are given in Section 3. Finally, Section 4 presents the experimental results.

Table 2. Sequence database $\mathcal{D}$ lexically sorted

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

Table 3. cluster 1 ($min\_strenth = 50\% \wedge w \geq 4$)

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 $wseq_1$	(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)		(BC)		(DE)⟩	

Table 4. cluster 2 ($min\_strength = 50\% \wedge w \geq 2$)

seq_8	⟨(IJ)	()	(KQ)	(M)⟩	
seq_7	⟨(AJ)	(P)	(K)	(LM)⟩	
seq_9	⟨(I)	()	()	(LM)⟩	
Weighted Sequence $wseq_2$	⟨(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)⟩	

2 Consensus Sequential Pattern Mining: ApproxMAP

We presented sequential pattern mining based on sequence alignment in [5]. Extending research on string analysis, we generalized string multiple alignment to find consensus sequential patterns in ordered lists of sets. The power of multiple alignment hinges on the following insight: the probability that any two long data sequences are the same purely by chance is very low. Thus, if several long sequences can be aligned with respect to particular frequent items, we will have implicitly found sequential patterns that are statistically significant.

ApproxMAP has three steps. First, k nearest neighbor clustering is used to partition the database. Second, for each partition, the optimal multiple alignment is approximated by the following greedy approach: in each partition, two sequences are aligned first, and then a sequence is added incrementally to the current alignment until all sequences have been aligned. At each step, the goal is to find the best alignment of the added sequence, p, to the existing alignment of $p-1$ sequences. A novel structure, *weighted sequence*, is used to summarize the alignment information in each cluster. In short, a weighted sequence is a sequence of itemsets with a weight associated with each item. The item weight represents the strength of the item where *strength* is defined as the percentage of sequences in the alignment that have the item present in the aligned position. Third, a consensus pattern is generated for each partition.

Tables 2 to 4 is an example. Given Table 2, ApproxMAP (1) calculates the proximity matrix and partitions the data into two clusters ($k = 2$), (2) aligns the sequences in each cluster – the alignment compresses all the sequences into one weighted sequence per cluster, and (3) summarizes the weighted sequences into consensus patterns using the cutoff point *min_strength*. Note that the consensus patterns ⟨(A)(BC)(DE)⟩ and ⟨(IJ)(K)(LM)⟩ do not match any sequence exactly.

3 Optimizations to ApproxMAP

ApproxMAP has total time complexity of $O(N_{seq}^2 \cdot L_{seq}^2 \cdot I_{seq} + k \cdot N_{seq})$ where N_{seq} is the total number of sequences, L_{seq} is the length of the longest sequence, I_{seq} is the maximum number of items in an itemset, and k is the number of nearest neighbors considered. The time complexity is dominated by the clustering step which requires the computation of the proximity matrix ($O(N_{seq}^2 \cdot L_{seq}^2 \cdot I_{seq})$) and builds clusters ($O(k \cdot N_{seq})$). The quadratic run time with respect to the database size may limit its applications to very large databases. There are two components constituting the running time for calculating the proximity matrix: (1) the per cell calculation, $O(L_{seq}^2 \cdot I_{seq})$, and (2) the total of N_{seq}^2 cell calculations needed for the proximity matrix. We discuss how to optimize both in this section.

3.1 *k*-Nearest Neighbor (*k*-NN) Clustering

ApproxMAP uses uniform kernel density based k-nearest neighbor (k-NN) clustering. We have found that such a density based method worked well due to its ability to build clusters of arbitrary size and shape around similar sequences. In this agglomerative method, each point links to its closest neighbor, but (1) only with neighbors that have greater density, 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,

$$dist'(seq_i, seq_j) =$$

$$\begin{cases} dist(seq_i, seq_j) & \text{if } dist(seq_i, seq_j) \le dist_k(seq_i) \text{ and } Density(seq_j, k) < Density(seq_i, k) \\ MAX\_DIST & \text{if } dist(seq_i, seq_j) \le dist_k(seq_i) \text{ and } Density(seq_j, k) = Density(seq_i, k) \\ \infty & \text{otherwise} \end{cases} \quad (1)$$

where $dist(seq_i, seq_j) = \frac{D(seq_i, seq_j)}{\max\{\|seq_i\|, \|seq_j\|\}}$ and $MAX\_DIST = \max\{dist(seq_i, seq_j)\} + 1$. $D(seq_i, seq_j)$ is the commonly used hierarchical edit distance defined via a recurrence relation. $dist_k(seq_i)$ is the k-NN region defined as the maximum distance over all k-NN and $Density(seq_i, k) = \frac{n_k(seq_i)}{dist_k(seq_i)}$ where $n_k(seq_i)$ is the number of sequences in the k-NN region. An effective implementation has three steps : (1) build the proximity matrix, (2) build the k-NN list using the matrix, and (3) merge the k-NN sequences when applicable. The details are in [3].

3.2 Optimizing the Proximity Matrix Calculation

Each cell in the proximity matrix is calculated using Equation 1. Thus, the time complexity is $O(L_{seq}^2 \cdot I_{seq})$ for solving the recurrence relation for $D(seq_i, seq_j)$

Table 5. Recurrence relation table

	seq_7	(AJ)		(P)		(K)		(LM)	
seq_8	**0**	1		2		3		4	
(IJ)	1	$\frac{1}{2}$	2	2	3	3	4	4	5
		2	$\searrow$ $\mathbf{\frac{1}{2}}$	$1\frac{1}{2}$	$\rightarrow$ $\mathbf{1\frac{1}{2}}$	$2\frac{1}{2}$	$\rightarrow$ $2\frac{1}{2}$	$3\frac{1}{2}$	$\rightarrow$ $3\frac{1}{2}$
(KQ)	2	2	$1\frac{1}{2}$	$1\frac{1}{2}$	$2\frac{1}{2}$	$1\frac{1}{2}+\frac{1}{3}=1\frac{5}{6}$	$3\frac{1}{2}$	$3\frac{1}{2}$	$4\frac{1}{2}$
		3	$\downarrow$ $1\frac{1}{2}$	$2\frac{1}{2}$	$\searrow$ $1\frac{1}{2}$	$2\frac{1}{2}$	$\searrow$ $\mathbf{1\frac{5}{6}}$	$2\frac{5}{6}$	$\rightarrow$ $2\frac{5}{6}$
(M)	3	3	$2\frac{1}{2}$	$2\frac{1}{2}$	$2\frac{1}{2}$	$2\frac{1}{2}$	$2\frac{5}{6}$	$1\frac{5}{6}+\frac{1}{3}=2\frac{1}{6}$	$3\frac{5}{6}$
		4	$\downarrow$ $2\frac{1}{2}$	$3\frac{1}{2}$	$\searrow$ $2\frac{1}{2}$	$3\frac{1}{2}$	$\searrow$ $2\frac{1}{2}$	$3\frac{1}{2}$	$\searrow$ $\mathbf{2\frac{1}{6}}$

through dynamic programming such as shown in Table 5. Often a straight forward dynamic programming algorithm can be improved by only calculating up to the needed precision. Here we discuss how to reduce the per cell calculation time by stopping the calculation of such a table midway whenever possible.

In Table 5, we show the intermediate calculation along with the final cell value. Each cell $RR(p,q)$ has four values in a 2x2 matrix. Let us assume we are converting seq_7 to seq_8. Then, the top left value shows the result of moving diagonally by replacing itemset p with itemset q. The top right value is the result of moving down by inserting q. The bottom left cell is the result of moving right by deleting p. The final value in the cell, shown in the bottom right position, is the minimum of the three. The arrows indicate the direction. The minimum path to the final answer, $RR(\|seq_i\|, \|seq_j\|) = D(seq_i, seq_j)$, is shown in bold.

For example, when calculating the value for $RR(3,2) = 1\frac{5}{6}$, you can either replace (KQ) with (K) (upper left: $RR(2,1)+REPL((\mathsf{KQ}),(\mathsf{K})) = 1\frac{1}{2}+\frac{1}{3} = 1\frac{5}{6}$), insert (KQ) (upper right: $RR(3,1) + INDEL = 2\frac{1}{2} + 1 = 3\frac{1}{2}$), or delete (K) (lower left: $RR(2,2) + INDEL = 1\frac{1}{2} + 1 = 2\frac{1}{2}$). Since $1\frac{5}{6}$ is the minimum, the replace operation is chosen (diagonal). The final distance $2\frac{1}{6}$ can be found by following the minimum path: diagonal (REPLACE), right(DELETE), diagonal, and diagonal. This path gives the pairwise alignment shown in Table 4.

In ApproxMAP, we note that we do no need to know $dist(seq_i, seq_j)$ for all i, j to full precision. In fact, the modified proximity matrix based on $dist'(seq_i, seq_j)$ has mostly values of ∞ because $k \ll N$. Thus, if a cell is clearly ∞ at any point, we can stop the calculation and return ∞. This will reduce the per cell calculation time significantly. $dist'(seq_i, seq_j)$ is clearly ∞ if seq_i is not a k-nearest neighbor of seq_j, and seq_j is not a k-nearest neighbor of seq_i. Remember that the modified proximity matrix is not symmetric. The following theorems prove that seq_i and seq_j are not k-nearest neighbor of each other when $\frac{min\_row(p)}{max\{\|seq_i\|,\|seq_j\|\}} > max\{dist_k(seq_i), dist_k(seq_j)\}$ for any row p. Here $dist_k(seq_i)$ is the radius of the k-nearest neighbor region for sequence seq_i, and $min\_row(p)$ is the smallest value of row p in the recurrence table. In the following theorems, we denote a cell in the recurrence table as $RR(p,q)$ with the initial cell as $RR(0,0) = 0$ and the final cell as $RR(\|seq_i\|, \|seq_j\|) = D(\|seq_i\|, \|seq_j\|)$.

Theorem 1. *There is a connected path from $RR(0,0)$ to any cell $RR(p,q)$ such that (1) cells along the path are monotonically increasing, (2) the two indices*

never decrease (i.e. the path always moves downward or to the right), and (3) there must be at least one cell from each row 0 *to* $p-1$*, in the connected path.*

Proof. The theorem comes directly from the definitions. First, the value of any cell $RR(p,q)$ is constructed from one of the three neighboring cells (up, left, or upper left) plus a non-negative number. Consequently, the values have to be monotonically increasing. Second, every cell must be constructed from three neighboring cells - namely up, left, or upper left. Hence, the path must move downward or to the right. Finally, since there has to be a connect path from $RR(0,0)$ to $RR(p,q)$, there must be at least one cell from each row 0 to $p-1$.

Theorem 2. $RR(\|seq_i\|, \|seq_j\|)$ *is greater than or equal to the minimum row value in any row. (i.e.* $RR(\|seq_i\|, \|seq_j\|) \geq min\_row(p)$ *for all* $0 \leq p \leq \|seq_i\|$*)*

Proof. Let us assume that there is a row, p, such that $RR(\|seq_i\|, \|seq_j\|) < min\_row(p)$. Let $min\_row(p) = RR(p,q)$. There are two possible cases. First, $RR(p,q)$ is in the connected path from $RR(0,0)$ to $RR(\|seq_i\|, \|seq_j\|)$. Since the connected path is monotonically increasing by Theorem 1, $RR(\|seq_i\|, \|seq_j\|)$ must be greater then equal to $RR(p,q)$. Thus, $RR(\|seq_i\|, \|seq_j\|) \geq RR(p,q) = min\_row(p)$. This is a contradiction. Second, $RR(p,q)$ is not in the connected path from $RR(0,0)$ to $RR(\|seq_i\|, \|seq_j\|)$. Now, let $RR(p,a)$ be a cell in the connected path. Then, $min\_row(p) = RR(p,q)$ and $RR(p,a) \geq RR(p,q)$. Thus, $RR(\|seq_i\|, \|seq_j\|) \geq RR(p,a) \geq RR(p,q) = min\_row(p)$. This is also a contradiction. Thus, by contradiction $RR(\|seq_i\|, \|seq_j\|) < min\_row(p)$ does not hold for any rows p. In other words, $RR(\|seq_i\|, \|seq_j\|) \geq min\_row(p)$ for all rows p.

Theorem 3. *If* $\frac{min\_row(p)}{max\{\|seq_i\|, \|seq_j\|\}} > max\{dist_k(seq_i), dist_k(seq_j)\}$ *for any row* p*, then* seq_i *is not a* k*-NN of* seq_j*, and* seq_j *is not a* k*-NN of* seq_i*.*

Proof. By Theorem 2, $RR(\|seq_i\|, \|seq_j\|) = D(seq_i, seq_j) \geq min\_row(p)$ for any row p. Thus, $dist(seq_i, seq_j) = \frac{D(seq_i, seq_j)}{max\{\|seq_i\|, \|seq_j\|\}} \geq \frac{min\_row(p)}{max\{\|seq_i\|, \|seq_j\|\}}$ $> max\{dist_k(seq_i), dist_k(seq_j)\}$ for any row p. By definition, when $dist(seq_i, seq_j) > max\{dist_k(seq_i), dist_k(seq_j)\}$, seq_i and seq_j are not k-NN of each other.

In summary by Theorem 3, as soon as the algorithm detects a row p in the recurrence table such that $\frac{min\_row(p)}{max\{\|seq_i\|, \|seq_j\|\}} > max\{dist_k(seq_i), dist_k(seq_j)\}$, it is clear that $dist'(seq_i, seq_j) = dist'(seq_j, seq_i) = \infty$. At this point, the recurrence table calculation can stop and simply return ∞. Checking for the condition $\frac{min\_row(p)}{max\{\|seq_i\|, \|seq_j\|\}} > max\{dist_k(seq_i), dist_k(seq_j)\}$ at the end of each row takes negligible time and space when $k \ll N$ and $k \ll L$.

3.3 Optimizing the Clustering Method

Now, we investigate how to reduce the N_{seq}^2 cell calculations by using an iterative clustering method similar to the well known k-mediods clustering methods. k-mediods clustering is exactly the same as the more popular k-means algorithm, except it works with the representative points in clusters rather than the means

Algorithm 1 (Sample Based Iterative Clustering)
Input: *a set of sequences* $\mathcal{D} = \{seq_i\}$*, the sampling percentage* α*, and the number of neighbor sequences* k' *for the sampled database;*
Output: *a set of clusters* $\{C_j\}$*, where each cluster is a set of sequences;*
Method:
1. **Randomly sample the database** $\mathcal{D}$ **into** $\mathcal{D}'$ **using** α*. The size of* $\mathcal{D}'$ *will be a trade off between time and accuracy. The experiments indicate that at a minimum* $\|\mathcal{D}'\|$ *should be 4000 sequences for the default* $k' = 3$*. Furthermore, roughly* 10% *of the data will give comparable results when* $N_{seq} \geq 40,000$*.*
2. **Run uniform kernel density based** k**-NN clustering [3] on** $\mathcal{D}'$ **with parameter** k'**.** *The output is a set of clusters* $\{C'_s\}$
3. **Center: Find the representative sequence for each cluster** C'_s*. The representative sequence,* seq_{sr}*, for a cluster,* C'_s*, is chosen such that* $\Sigma_j dist(seq_{sr}, seq_{sj})$ *for all sequences,* seq_{sj}*, in cluster* C'_s *is minimized (minimum intra-cluster distance).*
4. **Initialization: Initialize each cluster,** C_s**, with the representative sequence,** seq_{sr}*, found in the previous step.*
5. **Cluster: Assign all other sequences in the full database,** $\mathcal{D}$**, to the closest cluster.** *That is assign sequence* seq_i *such that* $dist(seq_i, seq_{sr})$ *is minimum over all representative sequences,* seq_{sr}*.*
6. **Recenter: Find the representative sequence for each cluster** C_s*. Repeat the centering step in 3 for all clusters* C_s *formed over the full database.*
7. **Iteratively repeat Initialization, Cluster, and Recenter.** *Steps 5 through 7 are repeated until no representative point change for any cluster or a certain iteration threshold,* $MAX\_LOOP = 100$*, is met.*

of clusters. There are two major difficulties in using the k-mediods clustering methods directly in ApproxMAP. First, without proper initialization, it is impossible to find the proper clusters. Thus, finding a good starting condition is crucial for k-mediods methods to give good results in terms of accuracy and speed [2]. Second, the general k-mediods method requires that the user input the number of clusters. However, the proper number of partitions is unknown in advance.

To overcome these problems, we introduce a *sample based iterative clustering* method. It involves two main steps. The first step finds the clusters and its representative sequences based on a small random sample of the data, $\mathcal{D}'$, using the density based k-NN method. Then in the second step, the number of clusters and the representative sequences are used as the starting condition to iteratively cluster and recenter the full database until the algorithm converges. The full algorithm is given above. When $\|\mathcal{D}'\| \ll \|\mathcal{D}\|$, the time complexity for clustering is obviously $O(t \cdot N_{seq})$ where t is the number of iterations needed to converge. The experimental results show that the algorithm converges very quickly. Figure 1(a) shows that in most experiments it takes from 3 to 6 iterations.

When using a small sample of the data, k (for k-NN algorithm) has to be smaller than what is used on the full database to achieve the clustering at the same resolution because the k-NN in the sampled data is most likely $(k+\alpha)$-NN in the full database. In ApproxMAP, the default value for k is 5. Hence, the default for k' in the sample based iterative clustering method is 3.

4 Evaluation

In our previous work, we have developed a benchmark that can quantitatively assess how well different sequential pattern mining methods can find the embedded

patterns in the data [4]. In this section, we apply the benchmark to conduct an extensive performance study on the two optimizations.

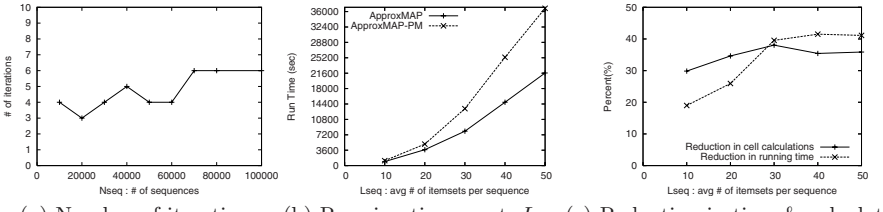
The benchmark uses the well known IBM data generator [1] which allows us to study the performance systematically. In addition, the IBM data generator embeds base patterns that represent the underlying trend in the data. By matching the results back to these embedded patterns, the benchmark can be used to measure the loss in accuracy due to the optimization. In particular, *recoverability* provides a good estimation of how well the items in the base patterns were detected. Recoverability is comparable to the commonly used recall except that it weights the results by the strength of the patterns in the database.

Most other criteria were not influenced by the optimizations. As expected in any alignment model, in all experiments there were no spurious patterns and negligible number of extraneous items resulting in excellent precision close to 100%. The amount of redundant patterns in the results remained similar to that of the basic ApproxMAP algorithm. The only criteria that was affected was the total number of patterns returned. Not surprisingly, recoverability is a good indicator for the number of total patterns returned increasing or decreasing accordingly. Thus, for simplicity we only report recoverability in our results.

4.1 Proximity Matrix Calculations

ApproxMAP can be optimized with respect to $O(L_{seq}^2)$ by calculating the proximity matrix used for clustering to only the needed precision. Here we study the speed up gained empirically. We only need to study the reduction in running time because this first optimization maintains the results of ApproxMAP. Figure 1(b) shows the speed up gained by the optimization with respect to L_{seq} in comparison to the basic algorithm. The figure indicates that such optimization can reduce the running time to almost linear with respect to L_{seq}.

To investigate the performance further, we examined the actual number of cell calculations reduced by the optimization. That is, with the optimization, the modified proximity matrix has mostly values of ∞ because $k \ll N$. For those $dist'(seq_i, seq_j) = \infty$, we investigated the dynamic programming calculation for $dist'(seq_i, seq_j)$ to see how many cells in the recurrence table were being skipped. To understand the savings in time, we report the following in Figure 1(c).



(a) Number of iterations (b) Running time w.r.t. L_{seq} (c) Reduction in time & calculation

Fig. 1. Results of optimizing the proximity matrix calculation (ApproxMAP-PM)

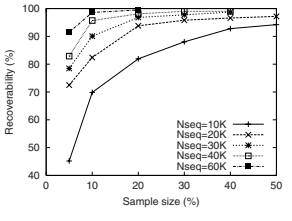
$$\frac{\sum \text{the number of cells in the recurrence table skipped}}{\sum \text{the total number of cells in the recurrence table}} \cdot 100\%$$

When $10 \leq L_{seq} \leq 30$, as L_{seq} increases more and more proportion of the recurrence table calculation can be skipped. Then at $L_{seq} = 30$, the proportion of savings levels off at around 35%-40%. This is directly reflected in the savings in running time in Figure 1(c). Figure 1(c) reports the reduction in calculations and running time due to the optimization as a proportion of the original algorithm. Clearly, the proportion of savings increase until $L_{seq} = 30$. At $L_{seq} = 30$ the running time levels off at around 40%. Thus, we expect that when $L_{seq} \geq 30$, the optimization will give a factor of 2.5 speed up in running time. This is a substantial improvement in speed without any loss in accuracy of the results.

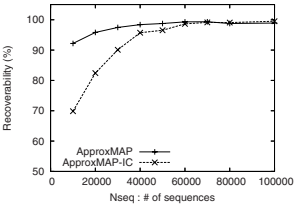
4.2 Sample Based Iterative Clustering

The sample based iterative clustering method can optimize the time complexity with respect to $O(N_{seq}^2)$ at the cost of some reduction in accuracy and larger memory requirement. The larger the sample size the better the accuracy with slightly longer running time. We investigate the tradeoff empirically.

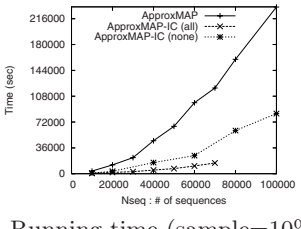
Figure 2(a) presents recoverability with respect to sample size ($k' = 3$). When $N_{seq} \geq 40,000$, recoverability levels off at 10% sample size with good recoverability at over 90%. When $N_{seq} < 40,000$, ApproxMAP requires a larger sample size of 20%-40%. In summary, the experiment suggests that the optimization should be used for databases when $N_{seq} \geq 40,000$ with sample size 10%. For databases with $N_{seq} < 40,000$ a larger sample size is required as 10% will result in significant loss in accuracy (Figure 2(b)). Essentially, the experiments indicate that the sample size be at least 4000 seqs to get comparable results when $k' = 3$.



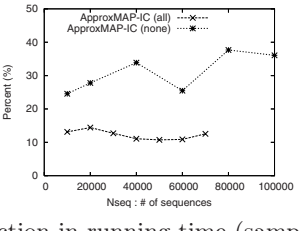
(a) Recoverability by sample size



(b) Recoverability by N_{seq} (sample=10%)



(c) Running time (sample=10%)



(d) Reduction in running time (sample=10%)

Fig. 2. Results for sample based iterative clustering (ApproxMAP-IC)

In the iterative clustering method more memory is required in order to fully realize the reduction in running time because the N_{seq}^2 proximity matrix needs to be stored in memory across iterations. In the basic method, although the full proximity matrix has to be calculated, the information can be processed one row at a time and there is no need to return to any values. That is, we only need to maintain the k-NN list without keeping the proximity matrix in memory. However in the iterative clustering method, it is faster to store the proximity matrix in memory over different iterations so as not to repeat the distance calculations. When N_{seq} is large, the proximity matrix is huge. Hence, there is a large memory requirement for the fastest optimized algorithm.

Nonetheless, the proximity matrix becomes very sparse when the number of clusters is much smaller than N_{seq}. Thus, much space can be saved by using a hash table instead of a matrix. Furthermore, a slightly more complicated scheme of storing only up to the possible number of values and recalculating the other distances when needed (much like a cache) will still reduce the running time compared to the basic method. Efficient hash tables are a research topic on its own and will be studied in the future. For now, the initial implementation of a simple hash table demonstrates the huge potential for reduction in time well. In order to fully understand the potential, we measured the running time assuming (1) memory was limitless and (2) no memory was available to store the proximity matrix. That is, distance values were never recalculated in the first experiment and always recalculated in the second experiment.

Figure 2(b) and (c) show the loss in recoverability and the gain in running time with respect to N_{seq} with the optimization (sample size=10%, $k' = 3$). Figure 2(d) depicts the relative running time with respect to N_{seq}. *optimized (all)* is a simple hash table implementation with all proximity values stored and *optimized (none)* is the implementation with none of the values stored. The implementation of a simple hash table was able to run up to $N_{seq} = 70,000$ with 2GB of memory (Figures 2(c) and 2(d) - *optimized (all)*). A more efficient hash table could easily improve the memory requirement.

A good implementation would give running times in between the *optimized (all)* and the *optimized (none)* line in Figure 2(c). The results clearly show that the optimization can speed up time significantly at the cost of negligible reduction in accuracy. Figure 2(d) show that the optimization can reduce running time to roughly 10%-40% depending on the size of available memory. Even in the worst case the running time is significantly faster by a factor of 2.5 to 4. In the best case, the running time is an order of magnitude faster.

5 Conclusions

Optimizing data mining methods is important in real applications which often have very large databases. In this paper, we proposed two optimization techniques for ApproxMAP that can reduce the running time significantly for consensus sequential pattern mining based on sequence alignment.

References

1. R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, pp. 3-14, 1995
2. A. Goswami, R. Jin, and G. Agrawal. Fast and Exact Out-of-Core K-Means Clustering. In *Proc. of the Int'l Conference on Data Mining (ICDM)*, pp. 83–90, 2004.
3. H.C. Kum, J.H. Chang, and W. Wang. Sequential pattern mining in multi-databases via multiple alignment. In *DMKD*, 12(2-3), pp. 151-180, 2006.
4. H.C. Kum, J.H. Chang, and W. Wang. Benchmarking the effectiveness of sequential pattern mining methods. In *Data and Knowledge Engineering*, 60, pp30-50, 2007.
5. H.C. Kum, J. Pei, W. Wang, and D. Duncan. ApproxMAP : Approximate mining of consensus sequential patterns. In *Proc. of SDM*, pp. 311-315, 2003.
6. A. Marascu and F. Masseglia. Mining data streams for frequent sequences extraction. In *Proc. of the IEEE Workshop on Mining Complex Data (MCD)*, 2005.
7. P. Tzvetkov, X. Yan, and J. Han. TSP: Mining top-k closed sequential patterns. In *Proc. of the Int'l Conference on Data Mining (ICDM)*, pp. 418–425, 2003.
8. X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large datasets. In *Proc. of the SIAM Int'l Conf on Data Mining*, pp. 166-177, 2003.