Approximate Mining of Consensus Sequential Patterns

by

Hye-Chung (Monica) Kum

(Draft of March 15, 2004 at 17:12)

ABSTRACT

In any particular data mining problem, the most important task is to define patterns operationally. The model (definition of patterns) should ultimately lead to useful understandable patterns. Designing a good model and evaluating what patterns are generated from a particular model is difficult.

In this dissertation, we examine closely the problem of mining sequential patterns. Sequential pattern mining finds interesting patterns in sequences of sets. Since it has been proposed in 1995, mining sequential patterns in large databases has become an important data mining task with broad applications.

Sequential pattern mining is commonly defined as finding the complete set of frequent subsequences. However, such conventional model may meet inherent difficulties in mining databases with long sequences and noises. They may generate a huge number of short trivial patterns but fail to find the underlying interesting patterns.

Motivated by these observations, we examine an entirely different model for analyzing sequential data. In this dissertation,

- We propose a novel model, *multiple alignment sequential pattern mining*. By lining up similar sequences and detecting the general trend, the multiple alignment model effectively finds consensus patterns that are approximately shared by many sequences.
- We demonstrate that ApproxMAP (for APPROXimate Multiple Alignment Pattern mining) is an efficient algorithm to find such consensus patterns.
- We design a general evaluation method for sequential pattern mining based on how well the underlying (embedded) patterns are recovered and how much confounding information is included in the results.
- We conduct an extensive performance study. The trend is clear and consistent. Together the consensus patterns form a succinct but comprehensive and accurate model of the sequential data. Furthermore, ApproxMAP is robust to its input parameters, robust to noise and outliers in the data, and scalable with respect to the size of the database. In comparison to the conventional model, ApproxMAP can better recover the underlying patterns with little confounding information under most circumstances.
- In addition, we report a successful case study on social welfare service patterns which illustrates that multiple alignment sequential pattern mining can find general, useful, concise and understandable knowledge and thus is an interesting and promising direction.

TABLE OF CONTENTS

Chapter 1

Introduction

Knowledge discovery and data mining (KDD) has been defined as "the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns from data" [20]. Clearly this definition serves more as a guideline than an operational definition. "Novel," "useful," and "understandable," are not quantitative criteria that can be implemented computationally. In any particular data mining problem, the first and most important task is to define patterns operationally. The algorithms are only as good as the definition (model). Designing a good model and evaluating what patterns are generated from a particular model is difficult. The model should ultimately lead to useful and understandable patterns.

In this dissertation, we examine closely the problem of mining sequential patterns. Sequential pattern mining finds interesting patterns in sequences of sets. Conventionally the sets are called *itemsets*. Since it has been proposed in [2], mining sequential patterns in large databases has become an important data mining task and has broad applications, such as social science research, policy analysis, business analysis, career analysis, web mining, and security.

Sequential pattern mining is commonly defined as finding the complete set of frequent subsequences in a set of sequences. However, conventional sequential pattern mining methods based on this *support model* may meet inherent difficulties in mining databases with long sequences and noises. They may generate a huge number of short and trivial patterns but fail to find the underlying patterns (interesting patterns approximately shared by many sequences) in the data.

Motivated by these observations, we examined an entirely different model for analyzing sequential data. In this dissertation,

- We propose a novel model for sequential pattern mining, *multiple alignment sequential pattern mining*, in databases with long sequences. By lining up similar sequences and detecting the general trend, the multiple alignment model effectively finds consensus patterns that are approximately shared by many sequences.

- We develop an efficient and effective algorithm, ApproxMAP (for APPROXimate Multiple Alignment Pattern mining), to mine consensus sequential patterns from large databases. ApproxMAP finds the underlying consensus patterns directly through multiple alignment.

- We design a general evaluation method for assessing the quality of any sequential pattern mining algorithm empirically. We then employ the method to conduct a detailed study of ApproxMAP as well as a through comparison study with the conventional model. We complete the evaluation with a real case study that illustrates the effectiveness of ApproxMAP.

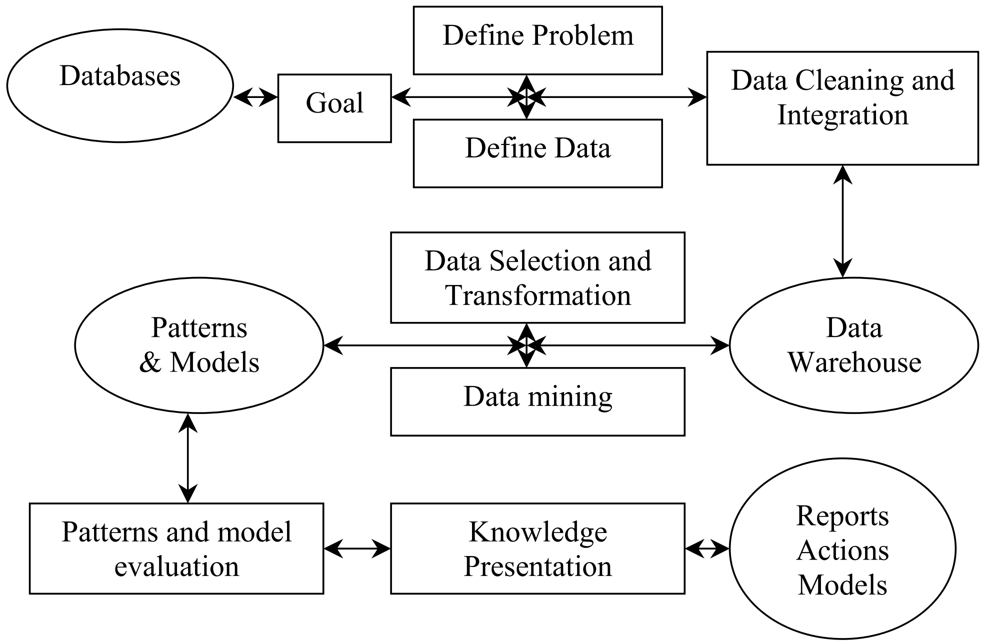
1.1 Knowledge Discovery and Data Mining (KDD)

Our society is accumulating massive amounts of data, much of which resides in large database management systems (DBMS). With the explosion of the Internet the rate of accumulation is increasing exponentially. Methods to explore such data would stimulate research in many fields. Knowledge discovery and data mining (KDD) is the area of computer science that tries to generate an integrated approach to extracting valuable information from such data by combining ideas drawn from databases, machine learning, artificial intelligence, knowledge-based systems, information retrieval, statistics, pattern recognition, visualization, and parallel and distributed computing [18, 20, 31, 56]. It has been defined as "The nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data" [19]. The goal is to discover and present knowledge in a form, which is easily comprehensible to humans [19].

A key characteristic particular to KDD is that it uses "observational data, as opposed to experimental data" [33]. That is, the objective of the data collection is something other than KDD. Usually, it is operational data collected in the process of operation, such as payroll data. Operational data is sometimes called administrative data when it is collected for administration purposes in government agencies. This means that often times the data is a huge convenience sample. Thus, in KDD attention to the large size of the data is required and care must be given when making generalization. This is an important difference between KDD and statistics. In statistics, such analysis is called secondary data analysis [33].

In KDD, useful information extracted are often in the form of (1) previously unknown relationships between observations and/or variables or (2) summarization or compression of the huge data in novel ways allowing humans to "see" the large bodies of data. The relationships and summaries must be new, understandable, and potentially useful [18, 33]. "These relationships and summaries are often referred to as models or patterns" [33]. Roughly, models are global summaries of a data set while patterns are local descriptions of a subset of the data [33]. As done in this dissertation, sometimes a group of local patterns can be a

Figure 1.1: The complete KDD process



global model that summaries a data set.

This dissertation deals with summarizations and compressions. The summarizations and compressions must be done "in such a way that the result is more comprehensible, without any notion of generalization" [33]. The key to successful summarization is how you view the data and the problem. It requires flexibility and creativity in finding the proper definitions for both. As with much in life, one innovative perspective can give you a simple answer to a complicated problem. The difficulty is in realizing the right point of view for the given question and data. As such, it is essential to interpret the summaries in the context of the defined problem and understanding of the data.

An important aspect of KDD is that it is an ongoing iterative process. Following are the steps in the iterative KDD process along with an example from the real world [33]:

1. **Data Acquisition**: The raw data is collected usually as a by product of operation of the business.

   - As various welfare programs are administered, many raw data have been collected on who has received what welfare programs in the past 5 years. One might be a database on all the TANF [1] welfare checks issued and pulled.

2. **Choose a Goal**: Choose a question to ask the database.

[1]TANF (Temporary Assistance of Needy Families) is the monthly cash assistance welfare program since Welfare Reform.

- There are data on various welfare program participation. From the database, one could be interested in the following questions: What are the common patterns of participation in these welfare programs? What are the main variations?

3. **Define the Problem**: Define the problem statement so that the data can give the answer.

   - What are the underlying patterns in the sequences of sets? (See chapter 3 for details)

4. **Define the Data**: Define/view the data to answer the problem statement?

   - Organize the welfare data into sequences of sets of welfare programs received during a particular month per person. See Table 1.1 in section 1.2.1 for an example.

5. **Data Cleaning and Integration**: Remove noise and errors in the data and combine multiple data sources to build the data warehouse as defined in the "Define the Data" step.

   - Cleaning: Clean all bad data (such as those with invalid data points)
   - Integration: Merge the appropriate database on different welfare programs by recipients.

6. **Data Selection and Transformation**: Select the appropriate data and transform them as necessary for analysis.

   - Selection: Separate out adults and children for separate analysis.
   - Transform: Define participation for each program and transform the data accordingly. For example, if someone received at least one TANF welfare check then code as T for that month.
   - Transform: Build the transformed welfare program participation data into actual sequences.

7. **Data Mining**: The essential step where intelligent methods are applied to extract the information.

   - Apply ApproxMAP to the sequences of monthly welfare programs received. (See chapter 4 for details)

8. **Patterns and Model Evaluation**: Identify interesting and useful patterns.

- View the consensus sequences generated by ApproxMAP for any interesting and previously unknown patterns. Also, view the aligned clusters of interest in more detail and use pattern search methods to confirm your findings.

9. **Knowledge Presentation**: Present the patterns and model of interest as appropriate to the audience.

   - Write up a report on previously unknown and useful welfare patterns of participation in welfare programs for policy makers.

Figure 1.1 shows the diagram of the complete KDD process [18, 31]. Although the process is depicted in a linear fashion keep in mind that the process is iterative. The earlier steps are frequently revisited and revised as needed while future steps have to be taken into account when completing earlier steps. For example, when defining the problem, along with the databases and the goal one should consider what established methods of data mining might work best in the application. This dissertation presents a novel data mining technique along with the appropriate problem and data definitions.

Data analysts have different objectives for utilizing KDD. Some of them are exploratory data analysis, descriptive modeling, predictive modeling, discovering patterns and rules, and retrieval of similar patterns when given a pattern of interest [33]. The objective of this dissertation is to assist the data analyst in exploratory data analysis by descriptive modeling. The models are built through lose compression and data reduction. "The goal of descriptive modeling is to describe all the data" [33] through the model. These models built in KDD are empirical by nature. Such models are "simply a description of the observed data" [33] and should not be viewed outside the context of the data. "The fundamental objective [of the model] is to produce insight and understanding about the structure of the data, and see its important features" [33].

1.2 Sequential Pattern Mining

Classical exploratory data analysis methods used in statistics and many of the earlier KDD methods tend to focus only on basic data types, such as interval or categorical data, as the unit of analysis. However, some information cannot be interpreted unless the data is treated as a unit, leading to complex data types. For example, the research in DNA sequences involves interpreting huge databases of amino acid sequences. The interpretation could not be obtained if the DNA sequences were analyzed as multiple categorical variables. The interpretation requires a view of the data at the sequence-level. DNA sequence research has developed many methods to interpret long sequences of alphabets [25, 30, 57].

In fact, sequence data is very common and "constitute a large portion of data stored in computers" [5]. For example, data collected over time is best analyzed as sequence data.

Table 1.1: Different examples of monthly welfare services given to clients in sequential form

clientID	Sequential Data
A	{AFDC (A), Medicaid (M), Food Stamp (FS)} {A, M, FS } {M, FS} {M, FS} {M, FS} {FS}
B	{Report (R)} {Investigation (I), Foster Care (FC)} {FC, Transportation (Tr)} {FC} {FC, Tr}
C	{Medicaid (M)} {AFDC (A), M} {A, M} {A,M} {M, Foster Care (FC)} {FC} {FC}

Table 1.2: A segment of the frequency of combinations of the first four events table

Pattern ID	Sequential Pattern	Frequency	Percentage
1	Medicaid Only	95,434	55.2%
2	Medicaid Only ⇒ AFDC	13,544	7.8%
3	Medicaid Only ⇒ AFDC ⇒ Foster Care	115	0.1%

Analysis of these sequences would reveal information about patterns and variation of variables over time. Furthermore, if the unit of analysis is a sequence of complex data, one can investigate the patterns of multiple variables over time. When appropriate, viewing the database as sequences of sets can reveal useful information that could not be extracted in any other way.

Analyzing suitable databases from such a perspective can assist many social scientists and data analysts in their work. For example, all states have administrative data about who has received various welfare programs and services. Some even have the various databases linked for reporting purposes [26]. Table 1.1 shows examples of monthly welfare services given to clients in sequential form. Using the linked data it is possible to analyze the pattern of participation in these programs. This can be quite useful for policy analysis: What are some commonly occurring patterns? What is the variability of such patterns? How do these patterns change over time? How does certain policy changes effect these patterns?

1.2.1 Sequence Analysis in Social Welfare Data

However, the methods to deal with such data (sequences of sets) are very limited. In addition, existing methods that analyze basic interval or categorical data yield poor results on these data due to exploding dimensionality [32].

Conventional methods used in policy analysis cannot answer the broad policy questions such as finding common patterns of practice. Thus, analysts have been forced to substitute their questions. Until recently, simple demographic information was the predominant method used (66% of those receiving Food Stamp also received AFDC[2] benefits in June). Survival analysis is gaining more popularity but still only allows for analyzing the rate of occurrence of some particular events (50% of participants on AFDC leave within 6 months). In [26], they studied specific transitions of interest (15% of children who entered AFDC in Jan 95, were in foster care before entering AFDC). These methods are very limited in their ability to describe the whole body of data.

[2]AFDC (Aid to Families with Dependent Children) was the cash assistance welfare program that was the precursor TANF.

Thus, some have tried enumeration [26, 58] - frequency counts of participants by program participation. Table 1.2 gives a small section of a table included in the technical report to Department of Health and Human Services (HHS) [26]. It reports the frequency of combinations of the first four events. For example the third line states that 0.1% of the children (115 sequences) experienced "Medicaid only" followed by "AFDC" followed by "foster care". Client C in Table 1.1 would be encoded as having such a pattern.

There are numerous problems with enumeration. First, program participation patterns do not line up into a single sequence easily. Most people receive more than one service in a particular month. For example, most clients receiving AFDC also receive Medicaid. As a work around, analysts carefully define a small set of alphabets to represent programs of most interest and its combinations. In [26], only three programs, AFDC, Medicaid, and foster care, were looked at. In order to build single sequences, they defined the following five events as most interesting.

- Medicaid Only
- AFDC: probably receiving Medicaid as well but no distinction was made as to whether they did or not
- Foster care: could be receiving Medicaid in conjunction with foster care, but no distinction was made
- No services
- Some other services

Second, even with this reduction of the question, many times the combinatoric nature of simple enumeration does not give you much useful information. In the above example, looking at only the first four events, the number of possible patterns would be $5^4 = 625$. Not surprisingly, there are only a few simple patterns such as, "AFDC $\Rightarrow$ Some other service", that are frequent. The more complex patterns of interest do not get compressed enough to be comprehensible. The problem is that almost all the frequent patterns are already known and the rest of the information in the result is not easily understandable by people. There were a total of 179 patterns reported in the analysis [26].

A much better method developed in sociology, *optimal matching*, has not yet gained much popularity in social sciences [1]. Optimal matching is a direct application of pattern matching algorithms developed in DNA sequencing to social science data [25, 30]. It simply applies the hierarchical edit distance to pairs of simple sequences of categorical data and runs the similarity matrix through standard clustering algorithms. The researcher looks at the clusters and tries to manually organize and interpret the clusters. This is possible because up to now researchers have only used it for fairly small data sets that were collected and coded manually. Optimal matching could be applied to the data discussed in the previous paragraph. It should give more useful results than simple enumeration. Then the analysis would not be limited to the most important programs or the first n elements.

There are two problems with optimal matching. First, you are limited to strings. Thus, one could not handle multiple services received in one month very well. In real datasets used in social science, sequences of sets are much more common than sequences of letters. Second, once the clustering and alignment is done there is no mechanism to summarize the cluster information automatically. Finding consensus strings from DNA sequences have not been applied to social science data yet. Thus, the applicability needs to be investigated. Without some automatic processing to produce cluster descriptors social scientists would be limited to very small data sets.

1.2.2 Conventional Sequential Pattern Mining

Traditionally in data mining, sequential patterns have been defined as a subsequence that appears frequently in a sequence database [2]. Such definition leads to a *support* model based sequential pattern mining methods. Currently as far as we know, the only methods available for analyzing sequences of sets is based on such a support model.

Although the support model based sequential pattern mining has been extensively studied and many methods have been proposed [4, 34, 48, 55, 66], there are some inherent obstacles within the conventional model. These methods meet inherent difficulties in mining databases with long sequences and noise. These limitations are discussed in detail in chapter 7.

More importantly, finding frequent subsequences will not answer the policy questions about service patterns in social welfare. In order to find common service patterns and the main variations, the mining algorithm must find the general underlying trend in the sequence data through summarization. However, the conventional methods have no mechanism for summarizing the data. In fact, often times many more patterns are output from the method than the number of sequences input into the mining process. These methods tends to generate a huge number of short and trivial patterns but fail to find interesting patterns approximately shared by many sequences.

1.3 Multiple Alignment Sequential Pattern Mining

In this dissertation, we present a new approach to sequential analysis, *multiple alignment sequential pattern mining*, that can detect common patterns and their variations in sequences of sets. Multiple alignment sequential pattern mining partitions the database into similar sequences, and then summarizes the underlying pattern in each partition through multiple alignment.

The exact solution to multiple alignment sequential pattern mining is too expensive. Here, we design an effective and efficient approximate solution, ApproxMAP, to mine consensus patterns from large sequence databases. Our goal is to assist the data analyst in exploratory data analysis through organization, compression, and summarization. ApproxMAP has three steps. First, similar sequences are grouped together using k-nearest neighbor clustering. Then we organize and compress sequences within each cluster into *weighted sequences* using multiple alignment. In the last step, the weighted sequences for each cluster is summarized into the longest consensus pattern best fitting each cluster via user specified *strength* cutoff. We use color to visualize item strength. *Item strength*, color in the consensus sequence, indicates how many sequences in the cluster contain the item in that position.

1.4 Evaluation

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 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, in this dissertation we have developed a general evaluation method that can objectively evaluate the quality of the results produced by any sequential pattern mining method. It uses the well known synthetic data generator built by [2]. The evaluation is based on how well the base patterns are recovered and how much confounding information (in the form of spurious patterns, redundant patterns, or extraneous items) is in the results. The base patterns, which are output along with the database by the data generator in [2], are the patterns used to generate the data.

We conduct an extensive and systematic performance study of ApproxMAP using the evaluation method. The trend is clear and consistent. The results show that ApproxMAP (1) is robust to the input parameters, (2) returns a succinct but accurate summary of the base patterns with few redundant or spurious patterns, (3) is robust to both noise and outliers in the data, and (4) is effective and scalable in mining large sequence databases with long patterns.

We further employ the evaluation method to conduct a comparative study of the conventional support model and the multiple alignment model. To the best of our knowledge, no research has examined in detail what patterns are actually generated from the popular support model for sequential pattern mining. The results clearly demonstrate that too much confounding information is generated. With so much redundant and spurious patterns there was no way to pick out the primary underlying patterns in the results. In addition, our theoretically analysis of the expected support of random sequences under the null hypothesis demonstrate that support along cannot distinguish between statistically significant patterns and random occurrences. Furthermore, the support model is vulnerable to noise in the data because it is based on exact match. We demonstrate that in comparison, the sequence alignment model is able to better recover the underlying patterns with little confounding information under most circumstances including those where the support model fails.

We complete the evaluation by reporting on a successful case study of ApproxMAP in a real application: mining the North Carolina State welfare services database. We illustrate some interesting patterns mined by ApproxMAP. Moreover, the mining result of ApproxMAP triggered some important investigations by our clients. The application results show that approximate sequential pattern mining can find interesting patterns that are highly promising in many applications.

1.5 Thesis Statement

The author of this dissertation asserts that multiple alignment is an effective model to uncover the underlying trend in sequences of sets. We will show that ApproxMAP, a novel method to apply multiple alignment techniques to sequences of sets, will effectively extract the underlying trend in the data by organizing the large database into clusters as well as give good descriptors (weighted sequences and consensus sequences) for the clusters using multiple alignment. Furthermore, we will show that ApproxMAP is robust to its input parameters, robust to noise and outliers in the data, scalable with respect to the size of the database, and in comparison to the conventional support model ApproxMAP can better recover the underlying patterns with little confounding information under most circumstances. In addition, we will demonstrate the usefulness of ApproxMAP using real world data.

1.6 Contributions

This dissertation makes the following contributions:

- defines a new model for sequential analysis based on multiple alignment, *Multiple Alignment Sequential Pattern Mining*,

- describes a novel solution ApproxMAP (for <u>APPROX</u>imate <u>M</u>ultiple <u>A</u>lignment <u>P</u>attern mining) that (1) introduces a new metric for itemsets, (2) a new representation of alignment information for sequences of sets (weighted sequences), and (2) the effective use of strength cutoffs to control the level of detail included in the consensus patterns,

- designs a general evaluation method to assess the quality of results from any sequential pattern mining algorithm,

- employs the evaluation method to conduct an extensive set of empirical evaluations of ApproxMAP on synthetic data,

- employs the evaluation method to compare the effectiveness of ApproxMAP to the conventional methods based on the support model,

- derives the expected support of random sequences under the null hypothesis of no patterns in the database to better understand the behaviour of the support model,
- and demonstrates the usefulness of ApproxMAP using real world data.

1.7 Synopsis

The rest of the dissertation is organized as follows. Chapter 2 reviews related works. Chapter 3 defines the model, *Multiple Alignment Sequential Pattern Mining*. Chapter 4 details the method, ApproxMAP. Chapter 5 introduces the evaluation method and chapter 6 reports the evaluation results. Chapter 7 reviews the conventional support model, details its limitations, and reports the results of the comparison study. Chapter 8 presents a case study on real data. Finally, in chapter 9, we conclude with a summary of our research and discuss areas for future work.

Chapter 2

Related Work

There are three areas of research highly related to our exploration in this dissertation, namely sequential pattern mining, multiple alignment, and approximate frequent pattern mining. We survey each in the following sections.

2.1 Support Model Based Sequential Pattern Mining

Since it was first introduced in [2], sequential pattern mining has been studied extensively. Conventional sequential pattern mining finds frequent subsequences in the database based on exact match. Much research has been done to efficiently find patterns, seq_p, such that $sup(seq_p) \geq min\_sup$ [1].

There are two classes of algorithms. The bottleneck in applying the support model occurs when counting the support of all possible frequent subsequences in database $\mathcal{D}$. Thus the two classes of algorithms differ in how to efficiently count support of potential patterns.

The apriori based breadth-first algorithms such as GSP and SPADE [2, 45, 55] pursue level-by-level candidate-generation-and-test pruning following the Apriori property: any super-pattern of an infrequent pattern cannot be frequent. In the first scan, they find the length-1 sequential patterns, i.e., single items frequent in the database. Then, length-2 candidates are assembled using length-1 sequential patterns. The sequence database is scanned the second time and length-2 sequential patterns are found. These methods scan the database multiple times until the longest patterns have been found. At each level, only potentially frequent candidates are generated and tested. This pruning saves much computational time compared to naively counting support of all possible subsequences in the database. Nonetheless, the candidate-generation-and-test is still very costly.

In contrast, the projection based depth-first algorithms such as PrefixSpan, FreeSpan, and SPAM [4, 34, 48] avoid costly candidate-generation-and-test by growing long patterns

[1] the *support* of a sequence seq_p, denoted as $sup(seq_p)$, is the number of sequences in $\mathcal{D}$ that are supersequences of seq_p. The exact definition is given in chapter 3

from short ones. Once a sequential pattern is found, all sequences containing that pattern are collected as a projected database. And then local frequent items are found in the projected databases and used to extend the current pattern to longer ones. Another variant of the projection-based methods mine sequential patterns with vertical format [66]. Instead of recording sequences of items explicitly, they record item-lists, i.e., each item has a list of sequence-ids and positions where the item appears.

It is interesting to note that the depth first methods generally do better than the breadth first methods when the data can fit in memory. The advantage becomes more evident when the patterns are long [64].

There are several interesting extensions to sequential pattern mining. For example, various constraints have been explored to reduce the number of patterns and make the sequential pattern mining more effective [28, 49, 55, 65]. Last year, Yan published the first method for mining frequent closed subsequences using several efficient search space prunning methods [63]. In addition, some methods (e.g., [54]) mine confident rules in the form of "$A \rightarrow B$". Such rules can be generated by a post-processing step after the sequential patterns are found.

Recently, Guralnik and Karypis used sequential patterns as features to cluster sequential data [29]. They project the sequences onto a feature vector comprised of the sequential patterns, then use a k-means like clustering method on the vector to cluster the sequential data. Interestingly, their work concurs with this study that the similarity measure based on edit distance is a valid measure in distance based clustering methods for sequences. However, we use clustering to group similar sequences here in order to detect approximate sequential patterns. Their feature-based clustering method would be inappropriate for this purpose because the features are based on exact match.

2.2 Multiple Alignment

The support model based sequential pattern mining algorithms extend the work done in association rule mining. Association rule mining ignores the sequence dimension and simply finds the frequent patterns in the itemsets. Approaching sequential analysis from association rule mining might be natural. Association rules mine intra-itemset patterns in sets while sequential mining finds inter-itemset patterns in sequences.

However, there is one important difference that can make sequential data mining easier. Unlike sets, sequences have extra information that can come in very handy - the ordering. In fact there is a rich body of literature on string (simple ordered lists) analysis in computer science as well as computational biology.

The most current research in computational biology employs the multiple alignment technique to detect common underlying patterns (called consensus strings or motifs) in a group of simple sequences (strings) [30, 25, 57]. In computational biology, multiple alignment has

Table 2.1: Multiple alignment of the word *pattern*

seq1	P	A	T	T	T	E	R	N
seq2	P	A	{}	{}	T	E	R	M
seq3	P	{}	{}	T	T	{}	R	N
seq4	O	A	{}	T	T	E	R	B
seq5	P	{}	S	Y	Y	R	T	N
Underlying pattern	P	A	{}	T	T	E	R	N

Table 2.2: The profile for the multiple alignment of the word *pattern* given in Table 2.1

position	1	2	3	4	5	6	7	8
A		3						
B								1
E						3		
M								1
N								3
O	1							
P	4							
R						1	4	
S			1					
T			1	3	4		1	
Y				1	1			
{}		2	3	1		1		
Consensus string	P	A	{}	T	T	E	R	N

been studied extensively in the last two decades to find consensus strings in DNA and RNA sequences.

In the simple edit distance problem, one is trying to find an alignment of two sequences such that the edit distance is minimum. In multiple alignment, the purpose is to find an alignment over N strings such that the total pairwise edit distance for all N strings is minimal. Aligned strings are defined to be the original string with null, {}, inserted where appropriate so that the lengths of all strings are equivalent. A good alignment is one in which similar characters are lined up in the same column. In such an alignment, the concatenation of the most common characters in each column would represent the underlying pattern in the group of strings.

For example, if we collect the various typos for a word *pattern*, and then align them, we should be able to uncover the correct spelling of the word. This is shown in Table 2.1. Even though non of the strings have the proper spelling we are able to find the correct spelling by multiple alignment.

Although, this is a straight extension of the well known edit distance problem, and can be solved with dynamic programming, the combinatory nature of the problem makes the solution impractical for even small number of sequences [30]. In practice, people have employed the greedy approximation solution. That is, the solution is approximated 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. 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 [30].

For strings, a sequence of simple characters, the alignment of p sequences is usually represented using a profile. A profile is a matrix whose rows represent the characters, whose columns represent the position in the string, and whose entries represent the frequency or percentage of each character in each column. Table 2.2 is the profile for the multiple alignment of the word *pattern* given in Table 2.1. During multiple alignment at each iteration we are then aligning a sequence to a profile. This can be done nicely using dynamic programming as in the case of aligning two sequences. Only the recurrence is changed to use the sum of all pairwise characters in the column with a character in the inserted string. Although we cannot reconstruct the p sequences from the profile, we have all the information we need to calculate the distance between two characters for all p sequences. Therefore, profiles are an effective representation of the p aligned strings and allow for efficient multiple alignment.

In this dissertation, we generalized string multiple alignment to find patterns in ordered lists of sets. Although much of the concept could be borrowed, most of the details had to be reworked. First, we select an appropriate measure for distance between sequences of itemsets. Second, we propose a new representation, *weighted sequences*, to maintain the alignment information. The issue of representing p sequences is more difficult in this problem domain because the elements of the sequences are itemsets instead of simple characters. There is no simple depiction of p itemsets to use as a dimension in a matrix representation. Thus, effectively compressing p aligned sequences in this problem domain demands a different representation form. In this dissertation, we propose to use weighted sequences to compress a group of aligned sequences into one sequence. And last, unlike strings the selection of the proper items in each column is not obvious. Recall that for strings, we simply take the most common character in each column. For sequence of itemsets, users can use the *strength* cutoffs to control the level of detail included in the consensus patterns. These ideas are discussed in detail in chapter 4.

In [12], Chudova and Smyth used a Bayes error rate model under a Markov assumption to analyze different factors that influence string pattern mining in computational biology. Extending the theoretical model to mining sequences of sets could shed more light to the future research direction.

2.3 Approximate Frequent Pattern Mining

A fundamental problem of the conventional methods is that the exact match on patterns does not take into account the noise in the data. This causes two potential problems. In real data, long patterns tend to be noisy and may not meet the support level with exact matching [50]. Even with moderate noise, a frequent long pattern can be mislabeled as an infrequent pattern [64]. Not much work has been done on approximate pattern mining.

[62] was the first to do approximate matching on frequent itemsets. Although the methodology is quite different, in spirit ApproxMAP is most similar to [62] in that both try to uncover structure in large sparse databases by clustering based on similarity in the data and exploiting the high dimensionality of the data. The similarity measure is quite different because [62] works with itemsets while ApproxMAP works on sequences of itemsets.

[50] also presents an apriori-based algorithm to incorporate noise for frequent itemsets, but is not efficient enough. Although the two methods introduced in [62] and [50] are quite different in techniques, they both explored approximate matching among itemsets. Neither address approximate matching in sequences.

Recently, Yang et al. presented a probabilistic model to handle noise in mining strings [64]. A compatibility matrix is introduced to represent the probabilistic connection from observed items to the underlying true items. Consequently, partial occurrence of an item is allowed and a new measure, *match*, is used to replace the commonly used support measure to represent the accumulated amount of occurrences. However, it cannot be easily generalized to apply to sequences of itemsets targeted in this research, and it does not address the issue of generating huge number of patterns that share significant redundancy.

By lining up similar sequences and detecting the general trend, the multiple alignment model in this paper effectively finds consensus patterns that are approximately similar to many sequences and dramatically reduce the redundancy among the derived patterns.

2.4 Unique Aspects of Our Research

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

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

- It generalizes the multiple alignment techniques to handle sequences of itemsets. Mining sequences of itemsets extends the application domain substantially.

Chapter 3

Problem Definition

In this dissertation, we examine closely the problem of mining sequential patterns. We start by introducing the concept of approximate sequential pattern mining. We then propose a novel model for approximate sequential pattern mining, *multiple alignment sequential pattern mining*, in databases with long sequences. By lining up similar sequences and detecting the general trend, the multiple alignment model effectively finds consensus patterns that are approximately shared by many sequences.

3.1 Sequential Pattern Mining

Let $\mathcal{I} = \{I_1, \ldots, I_p\}$ be a set of *items* $(1 \leq p)$. An *itemset* $s = \{x_1, \ldots, x_k\}$ is a subset of $\mathcal{I}$. A *sequence* $seq_i = \langle s_{i1} \ldots s_{in} \rangle$ is an ordered list of itemsets, where $s_{i1}, \ldots, s_{in}$ are itemsets. For clarity, sometimes the subscript i for the sequence is omitted and seq_i is denoted as $\langle X_1 \cdots X_n \rangle$. Conventionally, the itemset $s_{ij} = \{x_1, \ldots, x_k\}$ in sequence seq_i is also written as $s_{ij} = (x_{j1} \cdots x_{jk})$. The subscript i refers to the i^{th} sequence, the subscript j refers to the j^{th} itemset in seq_i, and the subscript k refers to the k^{th} item in the j^{th} itemset of seq_i. A *sequence database* $\mathcal{D}$ is a multi-set of sequences. That is multiple sequences that are exactly the same are allowed in the database.

Table 3.1 demonstrates the notations. Table 3.2 gives an example of a fragment of a sequence database $\mathcal{D}$. All sequences in $\mathcal{D}$ are built from the set of items $\mathcal{I}$ given in Table 3.1.

A sequence $seq_i = \langle X_1 \cdots X_n \rangle$ is called a *subsequence* of another sequence $seq_j = \langle Y_1 \cdots Y_m \rangle$, and seq_j a *supersequence* of seq_i, if $n \leq m$ and there exist integers $1 \leq a_1 < \cdots < a_n \leq m$ such that $X_b \subseteq Y_{a_b}$ $(1 \leq b \leq n)$. That is, seq_j is a supersequence of seq_i and seq_i is a subsequence of seq_j, if and only if seq_i is derived by deleting some items or whole itemsets from seq_j.

Given a sequence database $\mathcal{D}$, the *support* of a sequence seq_p, denoted as $sup(seq_p)$, is the number of sequences in $\mathcal{D}$ that are supersequences of seq_p. Conventionally, a sequence seq_p is called a *sequential pattern* if $sup(seq_p) \geq min\_sup$, where min_sup is a user-specified *minimum support threshold*.

Table 3.1: A Sequence, an itemset, and the set $\mathcal{I}$ of all items from Table 3.2

Items $\mathcal{I}$	Itemset s_{22}	Sequence seq_2
{ A, B, C, $\cdots$, X, Y, Z }	(BCX)	$\langle$(A)(BCX)(D)$\rangle$

Table 3.2: A Fragment of a Sequence Database $\mathcal{D}$

ID	Sequences
seq_1	$\langle$(BC) (DE)$\rangle$
seq_2	$\langle$(A) (BCX) (D)$\rangle$
seq_3	$\langle$(AE) (B) (BC) (D)$\rangle$
seq_4	$\langle$(A) (B) (DE)$\rangle$

Table 3.3: Multiple Alignment of Sequences in Table 3.2 ($\theta = 75\%, \delta = 50\%$)

seq_1	$\langle$()	()	(BC)	(DE)$\rangle$	
seq_2	$\langle$(A)	()	(BCX)	(D)$\rangle$	
seq_3	$\langle$(AE)	(B)	(BC)	(D)$\rangle$	
seq_4	$\langle$(A)	()	(B)	(DE)$\rangle$	
Weighted Sequence $wseq_3$	(A:3, E:1):3	(B:1):1	(B:4, C:3, X:1):4	(D:4, E:2):4	4
Pattern Con Seq ($w \geq 3$)	$\langle$(A)		(BC)	(D)$\rangle$	
Wgt Pat Con Seq ($w \geq 3$)	$\langle$(A:3)		(B:4, C:3)	(D:4)$\rangle$	4
Variation Con Seq ($w \geq 2$)	$\langle$(A)		(BC)	(DE)$\rangle$	
Wgt Var Con Seq ($w \geq 2$)	$\langle$(A:3)		(B:4, C:3)	(D:4, E:2)$\rangle$	4

In the example given in Table 3.2, seq_2 and seq_3 are supersequences of $seq_p = \langle$(A)(BC)$\rangle$. Whereas seq_1 is not a supersequence of seq_p because it does not have item A in the first itemset. Similarly, seq_4 is not a supersequence of seq_p because it does not have item C in the second itemset. Thus, $sup(seq_p = \langle$(A)(BC)$\rangle) = 2$ in this example.

3.2 Approximate Sequential Pattern Mining

In many applications, people prefer long sequential patterns shared by many sequences. However, due to noise, it is very difficult to find a long sequential pattern exactly shared by many sequences. Instead, many sequences may *approximately* share a long sequential pattern. For example, although expecting parents would have similar buying patterns, very few will have exactly the same pattern.

Motivated by the above observation, we introduce the notion of *mining approximate sequential patterns*. Let *dist* be a normalized distance measure of two sequences with range $[0, 1]$. For sequences seq_i, seq_1, and seq_2, if $dist(seq_i, seq_1) < dist(seq_i, seq_2)$, then seq_1 is said be *more similar* to seq_i than seq_2 is.

Naïvely, we can extend the conventional sequential pattern mining model to get an approximate sequential pattern mining model as follows. Given a *minimum distance threshold* *min_dist*, the *approximate support* of a sequence seq_p in a sequence database $\mathcal{D}$ is defined as $\widetilde{sup}(seq_p) = \|\{seq_i|(seq_i \in \mathcal{D}) \wedge (dist(seq_i, seq_p) \leq min\_dist)\}\|$. (Alternatively, the approximate support can be defined as $\widetilde{sup}(seq_p) = \sum_{seq_i \in \mathcal{D}} dist(seq_i, seq_p)$. All the following discussion retains.) Given a minimum support threshold min_sup, all sequential patterns whose approximate supports passing the threshold can be mined.

Unfortunately, the above model may suffer from the following two problems. First, the mining may find many short and probably trivial patterns. Short patterns tend to be easier to get similarity counts from the sequences than long patterns. Thus, short patterns may overwhelm the results.

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

Based on the above analysis, we abandon the threshold-based model in favor of a multiple alignment model. By grouping similar sequences, and then lining them up, the multiple alignment model can effectively uncover the hidden trend in the data. It effect, we are able to summarize the data into few long patterns that are approximately shared by many sequences in the data.

In the following section we present the formal problem statement along with definitions used in the multiple alignment model for sequence of sets. The definitions have been expanded from those for strings in computational biology [30].

3.3 Definitions

The global multiple alignment of a set of sequences is obtained by inserting a null itemset, (), either into or at the ends of the sequences such that each itemset in a sequence is lined up against a unique itemset or () in all other sequences. In the rest of the paper, alignment will always refer to global multiple alignment.

Definition 1. (Optimal Multiple Alignment) Given two aligned sequences and a distance function for itemsets, the **pairwise score(PS)** between the two sequences is the sum over all positions of the distance between an itemset in one sequence and the corresponding itemset in the other sequence. Given a multiple alignment of N sequences, the **multiple alignment score(MS)** is the sum of all pairwise scores. Then the **optimum multiple alignment** is one in which the multiple alignment score is minimal.

$$\begin{array}{rcll} PS(seq_i, seq_j) & = & \Sigma distance(s_{ik_i}(x), s_{jk_j}(x)) & \text{(for all aligned positions } x\text{)} \\ MS(N) & = & \Sigma PS(seq_i, seq_j) & \text{(over all } 1 \leq i \leq N \wedge 1 \leq j \leq N\text{)} \end{array} \quad (3.1)$$

where $s_{ik_i}(x)$ refers to the k_i^{th} itemset for sequence i which has been aligned to position x.

The first four rows in Table 3.3 show the optimal multiple alignment of the fragment of sequences in database $\mathcal{D}$ given in Table 3.2. Clearly, itemsets with shared items are lined up into the same positions as much as possible. The next row in Table 3.3 demonstrates how the alignment of the four sequences can be represented in the form of one weighted sequence. Weighted sequences are an effective method to compress a set of aligned sequences into one sequence.

Definition 2. (Weighted Sequence) A **weighted itemset**, denoted in equation 3.2, is defined as an itemset that has a weight associated with each item in the itemset as well as the itemset itself. Then a **weighted sequence**, denoted in equation 3.2, is a sequence of weighted itemsets paired with a separate weight for the whole sequence. The weighted sequence carries the following three information. (1) The weight associated with the weighted sequence, n, is the total number of sequences in the current alignment. (2) The weight associated with the itemset ws_j, v_j, represents how many sequences have a nonempty itemset aglined in position j. And (3) the weight associated with each item i_{jk} in itemset ws_j, w_{jk}, represents the total number of the item i_{jk} present in all itemsets in the aligned position j.

$$\begin{aligned} \text{weighted itemset } ws_j &= (i_{j1}: w_{j1}, \cdots, i_{jm}: w_{jm}): v_j \\ \text{weighted seq } wseq_i &= \langle (i_{11}: w_{11}, \cdots, i_{1s}: w_{1s}): v_1 \cdots (i_{l1}: w_{l1}, \cdots, i_{lt}: w_{lt}): v_l \rangle : n \end{aligned} \tag{3.2}$$

In this dissertation, the weighted sequences are numbered as $wseq_i$ where i is the i^{th} alignment in the partition. Hence, $wseq_i$ denotes that $i+1$ sequences have been aligned and compressed into the weighted sequence.

In the example given in Table 3.3, the weighted sequence $wseq_3$ indicates that 4 sequences (seq_1 to seq_4) have been aligned. And there are three A's and one E in the first column of the alignment. This information is summarized in the weights $w_{11} = 3$ and $w_{12} = 1$ associated with the items A and E respective in the first weighted itemset (A:3, E:1):3. The weight $v_1 = 3$ associated with the whole itemset indicates that one sequence out of four sequences ($n - v_j = 4 - 3 = 1$) has a null itemset aligned to this position. seq_1 has a null itemset in the first position of the alignment. The total number of sequences, $n = 4$ shown in the last column, is specified as the weight for the full weighted sequence.

Definition 3. (Strength) The **strength** of an item, i_{jk}, in an alignment is defined as the percentage of sequences in the alignment that have item i_{jk} present in the aligned position j.

$$strength(i_{jk}) = \frac{w_{jk}}{n} * 100\% \tag{3.3}$$

Clearly, larger strength value indicates that more sequences share the item in the same aligned position. In Table 3.3, the strength of item A in the first itemset is $\frac{w_{11}}{n} * 100\% = \frac{3}{4} * 100\% = 75\%$.

Definition 4. (Consensus Sequence) Given a cutoff point, ψ, and a multiple alignment of N sequences, the **consensus itemset** for position j in the alignment is an itemset of all items that occur in at least ψ sequences in position j. Then a **consensus sequence** is simply a concatenation of the consensus itemsets for all positions excluding any null consensus itemsets. When item weights are included it is called a **weighted consensus sequence**. Thus, a weighted consensus sequence is the weighted sequence minus all items that do not meet the strength cutoff point. By definition, the consensus sequence will always be a subsequence of the weighted sequence.

$$\begin{aligned} \text{Consensus itemset } (j) &= \{x_{jk} | \forall x_{jk} \in ws_j \wedge strength(x_{jk}) \geq \psi\} \\ \text{Consensus Sequence} &= \langle \Upsilon_{1 \geq j \geq l} \{x_{jk} | \forall x_{jk} \in ws_j \wedge strength(x_{jk}) \geq \psi\} \rangle \end{aligned} \tag{3.4}$$

where $\Upsilon_{1 \geq j \geq l}\ s_j$ is defined as the concatenation of the itemsets $s_j \neq NULL$ and l is the length of the weighted sequence.

Based on item strengths, items in an alignment are divided into three groups: rare items, common items, and frequent items. The rare items may represent noise and are in most cases not of any interest to the user. The frequent items occur in enough sequences to constitute the underlying pattern in the group. The common items do not occur frequently enough to be part of the underlying pattern but occur in enough sequences to be of interest. The common items generally constitute variations to the primary pattern. That is, they are the items most likely to occur regularly in a subgroup of the sequences.

Using this categorization we make the final results more understandable by defining two types of consensus sequences corresponding to two cutoff points: (1) The *pattern consensus sequence*, which is composed solely of frequent items and (2) the *variation consensus sequence*, an expansion of the pattern consensus sequence to include common items. This method presents both the frequent underlying patterns and their variations while ignoring the noise. It is an effective method to summarize the alignment because the user can clearly understand what information is being dropped. Furthermore, the user can control the level of summarization by defining the two cutoff points for frequent and rare items as desired.

Definition 5. (Pattern Consensus Sequence and Variation Consensus Sequence) Given two cutoff points, θ for frequent items and δ for rare items, (1) the **Pattern Consensus Sequence** is the consensus sequence composed of all items with strength greater than equal to θ and (2) the **Variation Consensus Sequence** is the consensus sequence composed of all items with strength greater than equal to δ where $0 \leq \delta \leq \theta \leq 1$. Sometimes the pattern consensus sequence is also called the *consensus pattern*.

$$
\begin{aligned}
\text{Pattern Consensus Sequence} &= \langle \Upsilon_{1 \geq j \geq l} \{x_{jk} | \forall x_{jk} \in ws_j \wedge strength(x_{jk}) \geq \theta\} \rangle \\
\text{Variation Consensus Sequence} &= \langle \Upsilon_{1 \geq j \geq l} \{x_{jk} | \forall x_{jk} \in ws_j \wedge strength(x_{jk}) \geq \delta\} \rangle \\
& \quad \text{where } 0 \leq \delta \leq \theta \leq 1
\end{aligned} \tag{3.5}
$$

The last four rows in Table 3.3 show the various consensus sequences for the example given in Table 3.2. The cutoff points for frequent and rare items are defined as follows : $\theta = 75\%$ and $\delta = 50\%$. The four frequent items - A from the first position in the alignment, B and C from the third position, and D from the fourth position - with weights greater than or equal to 3 ($w \geq \theta * n = 75\% * 4 = 3$ sequences) are concatenated to construct the pattern concensus sequence $\langle(\mathsf{A})(\mathsf{BC})(\mathsf{D})\rangle$. In addition to these frequent items, the variation consensus sequence,$\langle(\mathsf{A})(\mathsf{BC})(\mathsf{DE})\rangle$, also includes the common items with weights greater than or equal to 2 ($w \geq \delta * n = 50\% * 4 = 2$ sequences) - E from the fourth position in the alignment.

3.4 Multiple Alignment Sequential Pattern Mining

Given (1) N sequences, (2) a distance function for itemsets, and (3) strength cutoff points for frequent and rare items (users can specify different cutoff points for each partition), the problem of multiple alignment sequential pattern mining is (1) to partition the N sequences into K sets of sequences such that the sum of the K multiple alignment scores is minimum, (2) to find the optimal multiple alignment for each partition, and (3) to find the pattern consensus sequence and the variation consensus sequence for each partition.

3.5 Why Multiple Alignment Sequential Pattern Mining ?

Multiple alignment pattern mining is an effective approach to analyzing sequential data for practical use. It is a versatile exploratory data analysis tool for sequential data. (1) The partition and alignment organize the sequences, (2) the weighted sequences compress them, (3) the weighted consensus sequences summarize them at the user specified level, and (4) given reasonable strength levels, consensus sequences are patterns that are approximately similar to many sequences in the database. Multiple alignment pattern mining organizes and compresses the huge high dimensional database into something that is viewable by people. Data analysts can look at the weighted consensus sequences to find the partitions for interest. They can then explore the partitions of interest in more detail by looking at the aligned sequences and the weighted sequences to find potential underlying patterns in the database. Moreover, once the data is partitioned and aligned, it is trivial to view different levels of summarization (consensus sequences) in real time. Since "humans are better able to recognize something than generate a description of it" [41], the data analyst can then use the more efficient pattern search methods to do confirmatory data analysis.

In addition, multiple alignment pattern mining is an effective clustering method for grouping similar sequences in the sequence database. Intuitively, minimizing the sum of the K multiple alignment scores will partition the data into similar sequences. Grouping similar sequences together has many benefits. First, we can do multiple alignment on a group of similar sequences to find the underlying pattern (consensus sequence) in the group. Second, once sequences are grouped the user can specify strength as a percentage of the group instead of the whole database. Then, frequent uninteresting patterns (grouped into large partitions) will have a higher cutoff number than infrequent interesting patterns (grouped into small partitions). Thus, frequent uninteresting patterns will not flood the results as they do in the conventional algorithms. The grouping has the effect of using multiple cutoff points without the complications of specifying the multilevel cutoff points [42, 43, 60].

Chapter 4

Method : ApproxMAP

In this chapter 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.

4.1 Example

Table 4.1 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 4.1 is immensely improved by using the alignment model - grouping similar sequences then lining them up and coloring the consensus sequences as in Tables 4.2 through 4.4. Note that the patterns $\langle(\mathsf{A})(\mathsf{BC})(\mathsf{DE})\rangle$ and $\langle(\mathsf{IJ})(\mathsf{K})(\mathsf{LM})\rangle$ do not match any sequence exactly.

Given the input data shown in Table 4.1 ($N = 10$ sequences), ApproxMAP (1) calculates the $N * N$ 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 4.2 and 4.3) - the alignment compresses all the sequences in each cluster into one weighted sequence per cluster, and (4) summarizes the weighted sequences (Tables 4.2 and 4.3) into weighted consensus sequences (Table 4.4).

4.2 Partition: Organize into similar sequences

The first task is to partition the N sequences into K sets of sequences such that the sum of the K multiple alignment scores is minimum. The exact solution would be too expensive to be practical. Thus, in ApproxMAP we approximate the optimal partition by clustering the sequence database according to the similarity of the sequences. In the remainder of this

Table 4.1: 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 4.2: cluster 1 $(\theta = 50\% \wedge w \geq 4)$

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	(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
PatternConSeq	⟨(A)		(BC)		(DE)⟩	
WgtPatConSeq	⟨(A:5)		(B:6, C:4)		(D:4, E:5)⟩	7

Table 4.3: cluster 2 $(\theta = 50\% \wedge w \geq 2)$

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
Pattern Consensus Seq	⟨(IJ)		(K)	(LM)⟩	
Weighted Pattern Consensus Seq	⟨(I:2, J:2):3		(K:2):2	(L:2, M:3):3⟩	3

Table 4.4: Consensus Sequences $(\theta = 50\%, \delta = 20\%)$

100%: 85%: 70%: 50%: 35%: 20%		
Pattern Consensus Seq$_1$	strength = 50% = 3.5 > 3 seqs	(A) (BC) (DE)
Variation Consensus Seq$_1$	strength = 20% = 1.4 > 1 seqs	(A) (B) (BC) (DE)
Pattern Consensus Seq$_2$	strength = 50% = 1.5 > 1 seqs	(IJ) (K) (LM)
Variation Consensus Seq$_2$	Not appropriate in this small set	

dissertation, we use partition, cluster, and group interchangeably. The exact choice of the word depends on the flow of the context.

4.2.1 Distance Measure

The minimum pairwise score between two sequences is a good distance function to measure its similarity. In fact, the minimum pairwise score is equivalent to the weighted edit distance, often used as a distance measure for variable length sequences [30]. 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. We use $INDEL()$ to denote an insertion or deletion operation and $REPL()$ 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.

$$\begin{aligned}
D(0,0) &= 0 \\
D(i,0) &= D(i-1,0) + INDEL(X_i) && \text{for } (1 \leq i \leq n) \\
D(0,j) &= D(0,j-1) + INDEL(Y_j) && \text{for } (1 \leq j \leq m) \\
D(i,j) &= \min \begin{cases} D(i-1,j) + INDEL(X_i) \\ D(i,j-1) + INDEL(Y_j) \\ D(i-1,j-1) + REPL(X_i, Y_j) \end{cases} && \text{for } (1 \leq i \leq n) \text{ and } (1 \leq j \leq m)
\end{aligned} \tag{4.1}$$

Cost of edit operations for sets

We now have to define the cost of edit operations (i.e., INDEL() and REPL() in Equation 4.1) 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,

$$\begin{aligned}
REPL(X, Y) &= \frac{\|(X-Y) \cup (Y-X)\|}{\|X\| + \|Y\|} \\
&= \frac{\|X\| + \|Y\| - 2\|X \cap Y\|}{\|X\| + \|Y\|}
\end{aligned} \tag{4.2}$$

This measure has a nice property that,

$$0 \leq REPL() \leq 1 \tag{4.3}$$

Moreover, it satisfies the following metric properties [13].

1. nonnegative property: $distance(a, b) \geq 0$ for all a and b
2. zero property: $distance(a, b) = 0$ if and only if $a = b$

3. symmetry property: $distance(a, b) = distance(b, a)$ for all a and b
4. triangle inequality: $distance(a, b) + distance(b, c) \geq distance(a, c)$ for all a, b, c

Following equation 4.2, the cost of an insertion/deletion is

$$INDEL(X) = REPL(X, ()) = 1, \tag{4.4}$$

where X is a set. Table 4.5 shows some examples on the calculation of normalized set difference.

Table 4.5: Examples of normalized set difference between itemsets

X	Y	$REPL(X,Y)$	X	Y	$REPL(X,Y)$
(A)	(A)	0	(A)	(B)	1
(A)	(AB)	$\frac{1}{3}$	(AB)	(CD)	1
(AB)	(AC)	$\frac{1}{2}$	(A)	()	1

Clearly, the normalized set difference, $REPL()$, is equivalent to the Sorensen coefficient, D_S, as shown in equation 4.5. The Sorensen coefficient is an index similar to the more commonly used Jaccard coefficient [44]. The Jaccard coefficient, D_J, in dissimilarity notation is defined in equation 4.6. The difference is that $REPL()$ gives more weight to the common elements because what are shared by two sets is more important than what are not.

$$\begin{aligned} D_S(X,Y) &= 1 - \frac{2\|X \cap Y\|}{\|X-Y\|+\|Y-X\|+2\|X \cap Y\|} \\ &= \frac{\|X\|+\|Y\|-2\|X \cap Y\|}{\|X\|+\|Y\|} \\ &= REPL(X,Y) \end{aligned} \tag{4.5}$$

$$\begin{aligned} D_J(X,Y) &= 1 - \frac{\|X \cap Y\|}{\|X \cup Y\|} \\ &= 1 - \frac{\|X \cap Y\|}{\|X-Y\|+\|Y-X\|+\|X \cap Y\|} \end{aligned} \tag{4.6}$$

Pairwise Score between Sequences: Normalized Edit Distance

Since the $N * N$ sequence to sequence distances will have to be compared with each other and the length of the sequences vary, 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,

$$dist(seq_i, seq_j) = \frac{D(seq_i, seq_j)}{\max\{\|seq_i\|, \|seq_j\|\}} \tag{4.7}$$

Clearly, $dist()$ satisfies the metric properties and $dist()$ is between 0 and 1. This follows directly from the properties of the itemset distance, $REPL()$. Table 4.6 illustrates some examples. As seen in the first example, when two sequences seq_9 and seq_{10} do not share any items in common, $dist(seq_9, seq_{10}) = 1$ since each itemset distances is 1. The second example

Table 4.6: 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
$dist()$	$3/3 = 1$	$dist()$	$(\frac{1}{2} + \frac{1}{3})/3 = 0.278$	$dist()$	$(2 + \frac{5}{6})/4 = 0.708$

shows the two sequences that are most similar among the 10 sequences in the example given in section 4.1. 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, B are shared and there are many that are not shared ($dist(seq_6, seq_2) = 0.708$). Clearly, seq_4 is more similar to seq_2 than seq_6. That is, $dist(seq_4, seq_2) < dist(seq_6, seq_2)$.

4.2.2 Clustering

Now we use the distance function $dist(seq_i, seq_j)$ to cluster sequences. Clustering methods identify homogeneous groups of objects based on whatever data is available. Data is often given in the form of a pattern matrix with features measured on a ratio scale, or as a proximity matrix [6, 36]. As in ApproxMAP, the proximity matrix is sometimes given as a distance function between data points.

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 [15, 35]. By using the pairwise score between sequences (described in section 4.2.1) as the distance function for sequences of sets, clustering can approximate a partition that minimizes the sum of the K multiple alignment scores.

In particular, 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 [36]. These valleys can be considered as natural boundaries that separate the modes of the distribution [38, 22].

Building clusters from dense regions that may vary in shape and size [51, 17, 11] results in clusters of arbitrary shapes. In addition, growing the clusters to the valleys of the density function can identify the number of natural groups in the data automatically. Thus, with a good density function these methods can best approximate a partition that minimizes the sum of the K multiple alignment scores.

Definition of Density for Sequences

What is a good definition of density for a sequence? Intuitively, a sequence is "*dense*" if there are many sequences similar to it in the database. A sequence is "*sparse*," or "*isolated*," if it is not similar to any others, such as an outlier.

Technically, the nonparametric probability density estimate at a point x, $p(x)$, is proportional to the number of points, n, falling in a local region, d, around x. Formally,

$$p(x) = \frac{n}{N * d}, \tag{4.8}$$

where N is the total number of points. For a given dense point x, its local region d will have many points, resulting in a large n. This in turn will return a large density estimate. In contrast, a point x that has few points in d, will result in a small n and thus a small density estimate [36, 15].

The local region, d, can be specified by either a Parzen window approach or a k nearest neighbor approach. In the Parzen window approach the size of the local region is fixed. The user specifies the radius used as the local window size when estimating the local density at a point [15, 36]. On the other hand, in the k nearest neighbor approach the size of the local region is variable. The user specifies the number of neighbors, k, which will influence the local density estimate at a point. The local region d, is then defined by the furthest neighbor of the k nearest neighbors.

Parzen window is inappropriate for ApproxMAP because the unit of the distance measure it not Euclidean. The users would have no basis on which to specify the fixed window size. In contrast, k nearest neighbor works well because the users can intuitively specify the number of neighbors, k, that should have local influence on the density estimate of a data point. That is, how many nearest points should be considered as a neighbor.

Adopting such a nonparametric probability density estimate for sequences, 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 a 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 4.7), where $seq_j \neq seq_i$, and seq_j is a sequence in $\mathcal{D}$. Then,

$$Density(seq_i, k) = \frac{n_k(seq_i)}{dist_k(seq_i)}, \tag{4.9}$$

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.

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 4.9. 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 $Density(C_{seq_j}) > Density(C_{seq_i})$.

Density Based k Nearest Neighbor Clustering

ApproxMAP uses uniform kernel density based k nearest neighbor clustering. In this algorithm, the user-specified parameter k not only specifies 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 [53] based on [61] as given in Algorithm 1. The algorithm has complexity $O(k \cdot \|\mathcal{D}\|)$.

Theoretically, the algorithm is similar to the single linkage method. The single linkage method builds a tree with each point linking to its closest neighbor [15]. 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 a proximity matrix defined as follows,

$$dist'(seq_i, seq_j) = \begin{cases} dist(seq_i, seq_j) & \text{if } dist(seq_i, seq_j) \leq dist_k(seq_i) \\ & \text{and } Density(seq_j, k) < Density(seq_i, k) \\ MAX\_DIST & \text{if } dist(seq_i, seq_j) \leq dist_k(seq_i) \\ & \text{and } Density(seq_j, k) = Density(seq_i, k) \\ \infty & \text{otherwise} \end{cases} \tag{4.10}$$

where $dist(seq_i, seq_j)$ is defined in Equation 4.7, $Density(seq_i, k)$ and $dist_k(seq_i)$ are defined in Equation 4.9, 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 then the density of the other to merge any local maxima 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.

This results in 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 $k > 1$ [17]. 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 like 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.

In short, the key parameter k determines the resolution of clustering. Since k defines the neighbor space, a larger k value tends to merge more sequences and results in a smaller number of large clusters, while a smaller k value tends to break up clusters. Thus, k controls the level of granularity at which the data is partitioned. The benefit of using a small k value is that the algorithm can detect less frequent patterns. The tradeoff is that it may break up clusters representing strong patterns (patterns that occur in many sequences) to generate multiple similar patterns [17]. As shown in our performance study (section 6.1.1), in many applications, a value of k in the range from 3 to 9 works well.

Note that in practice, the normal single linkage method is a degenerate case of the algorithm with $k = 1$. When $k = 1$, no smoothing occurs and not enough local information is being used to estimate the local density. Thus, the algorithm becomes instable because the outliers can easily influence the density estimates. Without the reasonable density estimates, it becomes difficult to locate the valleys and the proper boundaries. In fact, the use of this density base method with $k > 1$ provides an automatic adjustment for differences in the density [17].

Choosing the right clustering method

In summary, density based clustering methods have many benefits for clustering sequences:

1. The basic model of clustering around dense points fits the sequential data best because our goal is to form groups of arbitrary shape and size around similar sequences.

2. Density based k nearest neighbor clustering algorithms will automatically estimate the appropriate number of clusters from the data.

3. Users can cluster at different resolutions by adjusting k.

Nonetheless, if $\|\mathcal{D}\|$ is huge and calculating the k nearest neighbor list becomes prohibitive, ApproxMAP can be extended to use the sample based iterative partitioning method. This is discussed further in section 4.6.2.

Note that there are numerous clustering methods out there and still many more are being developed. Jain, a notable expert in clustering methods, has said [35]:

> There is no clustering technique that is universally applicable in uncovering the variety of structures present in multidimensional data sets...This explains the large number of clustering algorithms which continue to appear in the literature; each new clustering algorithm performs slightly better than the existing ones on a specific distribution of patterns...It is essential for the user of a clustering algorithm to not only have a thorough understanding of the particular technique being utilized, but also to know the details of the data gathering process and to have some domain expertise; the more information the user has about the data at hand, the more likely the user would be able to succeed in assessing its true class structure [36].

In essence, choosing the right clustering method is difficult and depends heavily on the application. Having a good understanding of the data and the clustering method is important in making the right choice.

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 [11, 17, 51]. Many use k-nearest neighbor or the Parzen window for the local density estimate. Recently, others have also used the shared neighbor list [17] 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. But for the purposes of demonstrating ApproxMAP, the choice of a density based clustering method was based on its overall good performance and simplicity.

Furthermore, it should be understood that, as discussed in section 4.6.2, the steps after the clustering step make up for most of the inaccuracy introduced in it. In fact, as seen in our study of sample based iterative clustering method (section 6.1.5), the multiple alignment step can compensate for much of the differences between the different clustering algorithms to give comparable results. Thus, 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.

4.2.3 Example

Table 4.7: The $N*N$ proximity matrix for database $\mathcal{D}$ given in Table 4.1

ID	seq_1	seq_2	seq_3	seq_4	seq_5	seq_6	seq_{10}	seq_7	seq_8	seq_9
seq_1	0									
seq_2	0.511	0								
seq_3	0.583	0.383	0							
seq_4	0.444	0.278	0.417	0						
seq_5	0.700	0.707	0.500	0.567	0					
seq_6	0.708	0.708	0.542	0.458	0.533	0				
seq_{10}	0.778	1	1	0.778	0.867	0.833	0			
seq_7	1	0.833	0.875	0.833	0.900	0.875	0.833	0		
seq_8	1	1	1	1	1	1	1	0.542	0	
seq_9	1	1	1	1	1	1	1	0.750	0.556	0

Table 4.7 is the proximity matrix for the example given in section 4.1. It has been arrange so that the two triangle areas are the intra cluster distances for the two clusters, and the rectangular area is the inter cluster distances. Clearly, the clustering method has been able to partition the sequences such that the intra cluster distance is much smaller than the inter cluster distances. Except for the one outlier, seq_{10}, all intra cluster distances are less than

0.8 while the inter cluster distances are greater than that. The average distance in the intra cluster distance is 0.633 and 0.616 for cluster 1 and 2 respectively. In comparison, the average distance for the inter cluster distance is 0.960.

Table 4.8: k nearest neighbor list and density for each sequence in $\mathcal{D}$

ID	Sequences	Density	1st NN(ID:dist)	2nd NN
seq_1	⟨(BC) (DE)⟩	3.913	$seq_4 : 0.444$	$seq_2 : 0.511$
seq_2	⟨(A) (BCX) (D)⟩	5.218	$seq_4 : 0.278$	$seq_3 : 0.383$
seq_3	⟨(AE) (B) (BC) (D)⟩	4.801	$seq_2 : 0.383$	$seq_4 : 0.417$
seq_4	⟨(A) (B) (DE)⟩	4.801	$seq_2 : 0.278$	$seq_3 : 0.417$
seq_5	⟨(AX) (B) (BC) (Z) (AE)⟩	3.750	$seq_3 : 0.500$	$seq_6 : 0.533$
seq_6	⟨(AY) (BD) (B) (EY)⟩	3.750	$seq_4 : 0.458$	$seq_5 : 0.533$
seq_7	⟨(AJ) (P) (K) (LM)⟩	2.667	$seq_8 : 0.542$	$seq_9 : 0.750$
seq_8	⟨(IJ) (KQ) (M)⟩	3.600	$seq_7 : 0.542$	$seq_9 : 0.556$
seq_9	⟨(I) (LM)⟩	2.667	$seq_8 : 0.556$	$seq_7 : 0.750$
seq_{10}	⟨(V) (PW) (E)⟩	2.572	$seq_1 : 0.778$	$seq_4 : 0.778$

Table 4.9: How sequences from $\mathcal{D}$ are merged into clusters

	ID	Sequences	Step 2
1	seq_1	⟨(BC) (DE)⟩	$seq_4 \Rightarrow seq_2$
2	seq_2	⟨(A) (BCX) (D)⟩	
3	seq_3	⟨(AE) (B) (BC) (D)⟩	seq_2
4	seq_4	⟨(A) (B) (DE)⟩	seq_2
5	seq_5	⟨(AX) (B) (BC) (Z) (AE)⟩	$seq_3 \Rightarrow seq_2$
6	seq_6	⟨(AY) (BD) (B) (EY)⟩	$seq_4 \Rightarrow seq_2$
7	seq_{10}	⟨(V) (PW) (E)⟩	$seq_1 \Rightarrow seq_4 \Rightarrow seq_2$
8	seq_7	⟨(AJ) (P) (K) (LM)⟩	seq_8
9	seq_8	⟨(IJ) (KQ) (M)⟩	
10	seq_9	⟨(I) (LM)⟩	seq_8

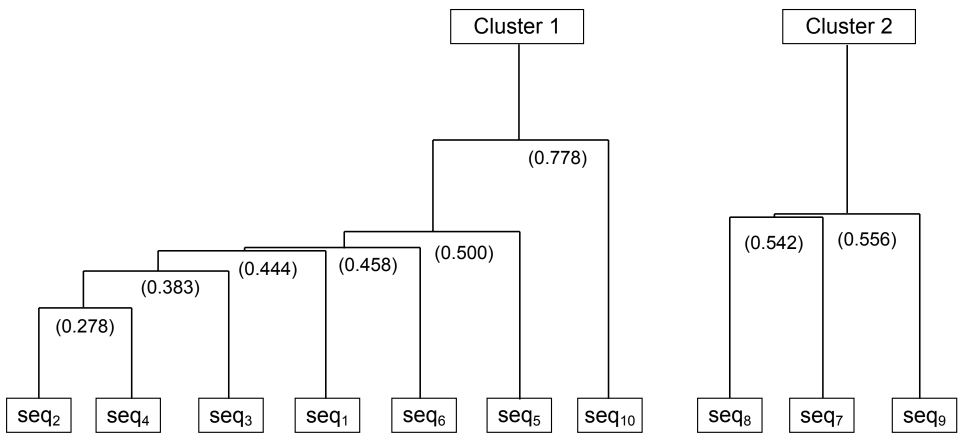
Using the proximity matrix given in Table 4.7, the clustering algorithm first calculates the k-nearest neighbor list and the density as given in Table 4.8. Table 4.9 shows how the clusters are formed in the following steps. For each sequence, column 'Step 2' first shows the nearest denser neighbor sequence with which the sequence merged with. If that denser neighbor merged with another sequence in the same step, the indirect linkage is shown with an arrow. The first row shows that seq_1 merged with seq_4 which in turn merged with seq_2 (as shown in the fourth row). The seventh row shows that seq_{10} merged with seq_1 which in turn merged with seq_4 then seq_2 (as shown in the first row). Such notation shows the densest sequence of the cluster, to which the sequence belongs to, as the last sequence in column 'Step 2'. The densest sequence of each cluster has the column blank. Thus, Table 4.9 shows that, in step 2 of the clustering algorithm, $seq_1 - seq_6$ and seq_{10} get merged together into cluster 1 and $seq_7 - seq_9$ get merged into cluster 2. The most dense point in cluster 1 and 2 are seq_2 and seq_8 respectively.

Table 4.10 shows the proximity matrix that could be used to construct the same clusters

Table 4.10: The modified proximity matrix for Table 4.7

ID	seq_1	seq_2	seq_3	seq_4	seq_5	seq_6	seq_{10}	seq_7	seq_8	seq_9
seq_1	0	0.511	∞	0.444	∞	∞	∞	∞	∞	∞
seq_2	∞	0	∞	∞	∞	∞	∞	∞	∞	∞
seq_3	∞	0.383	0	M_D	∞	∞	∞	∞	∞	∞
seq_4	∞	0.278	M_D	0	∞	∞	∞	∞	∞	∞
seq_5	∞	∞	0.500	∞	0	M_D	∞	∞	∞	∞
seq_6	∞	∞	∞	0.458	M_D	0	∞	∞	∞	∞
seq_{10}	0.778	∞	∞	0.778	∞	∞	0	∞	∞	∞
seq_7	∞	∞	∞	∞	∞	∞	∞	0	0.542	M_D
seq_8	∞	∞	∞	∞	∞	∞	∞	∞	0	∞
seq_9	∞	∞	∞	∞	∞	∞	∞	M_D	0.556	0

Figure 4.1: Dendrogram for sequences in $\mathcal{D}$ (Table 4.1)



through the single linkage method for this example. Using this proximity matrix, the single linkage method would generate the dendrogram shown in Figure 4.1 resulting in the same two clusters as shown in Table 4.9. Each internal node in the dendrogram is labeled with the actual distance of the link.

To demonstrate how the local maxima region is merged in step 3 of the clustering algorithm, we give a second example. The first two columns in Table 4.12 give the ten sequences for the sequence database $\mathcal{D}_2$. Table 4.11 is the proximity matrix used to calculate the k-nearest neighbor list and the density given in Table 4.12. Again the sequences have been arrange so that the two triangle areas are the intra cluster distances for the two clusters, and the rectangular area is the inter cluster distances.

Then, Table 4.13 shows how the clusters are merged in steps 2 and 3. At the end of step 2, there are three clusters. seq_1, seq_2, and seq_4 merge into one cluster with seq_1 being the densest sequence, $seq_5, seq_6, seq_7, seq_9$, and seq_{10} merge into a second cluster with seq_9 as the densest sequence, and seq_3 and seq_8 merge into the third cluster with seq_3 being the densest

Table 4.11: The $N * N$ proximity matrix for database $\mathcal{D}_2$ given in Table 4.12

ID	seq_1	seq_2	seq_4	seq_3	seq_5	seq_6	seq_7	seq_8	seq_9	seq_{10}
seq_1	0									
seq_2	0.640	0								
seq_4	0.556	0.767	0							
seq_3	0.833	0.867	0.875	0						
seq_5	0.875	0.867	1	0.458	0					
seq_6	1	1	1	0.625	0.583	0				
seq_7	1	1	0.867	0.633	0.467	0.667	0			
seq_8	1	1	1	0.458	0.542	0.625	0.467	0		
seq_9	1	1	1	0.500	0.417	0.444	0.533	0.500	0	
seq_{10}	1	0.800	1	0.833	0.833	1	0.867	0.833	0.778	0

Table 4.12: k nearest neighbor list and density for each sequence in $\mathcal{D}_2$

ID	Sequences	Density	1st NN(ID:dist)	2nd NN
seq_1	⟨(A) (BCY) (D)⟩	3.125	$seq_4 : 0.556$	$seq_2 : 0.640$
seq_2	⟨(A) (X) (BC) (AE) (Z)⟩	2.609	$seq_1 : 0.640$	$seq_4 : 0.767$
seq_3	⟨(AI) (Z) (K) (LM)⟩	4.364	$seq_8 : 0.458$	$seq_5 : 0.458$
seq_4	⟨(AL) (DE)⟩	2.609	$seq_1 : 0.556$	$seq_2 : 0.767$
seq_5	⟨(IJ) (B) (K) (L)⟩	4.364	$seq_9 : 0.417$	$seq_3 : 0.458$
seq_6	⟨(IJ) (LM)⟩	3.429	$seq_9 : 0.444$	$seq_5 : 0.583$
seq_7	⟨(IJ) (K) (JK) (L) (M)⟩	4.286	$seq_8 : 0.467$	$seq_5 : 0.467$
seq_8	⟨(IM) (K) (KM) (LM)⟩	4.286	$seq_3 : 0.458$	$seq_7 : 0.467$
seq_9	⟨(IJ) (LM)⟩	4.500	$seq_5 : 0.417$	$seq_6 : 0.444$
seq_{10}	⟨(V) (K W) (Z)⟩	2.500	$seq_9 : 0.778$	$seq_2 : 0.800$

Table 4.13: How sequences from Table 4.12 are merged into clusters in Steps 2 and 3

	ID	Sequences	Step 2	Step 3
1	seq_1	⟨(A) (BCY) (D)⟩		
2	seq_2	⟨(A) (X) (BC) (AE) (Z)⟩	seq_1	
3	seq_4	⟨(AL) (DE)⟩	seq_1	
6	seq_5	⟨(IJ) (B) (K) (L)⟩	seq_9	
5	seq_6	⟨(IJ) (LM)⟩	seq_9	
7	seq_7	⟨(IJ) (K) (JK) (L) (M)⟩	$seq_5 \Rightarrow seq_9$	
4	seq_9	⟨(J) (K) (L M)⟩		
8	seq_{10}	⟨(V) (K W) (Z)⟩	seq_9	
9	seq_3	⟨(AI) (Z) (K) (LM)⟩		seq_9
10	seq_8	⟨(IM) (K) (KM) (LM)⟩	seq_3	seq_9

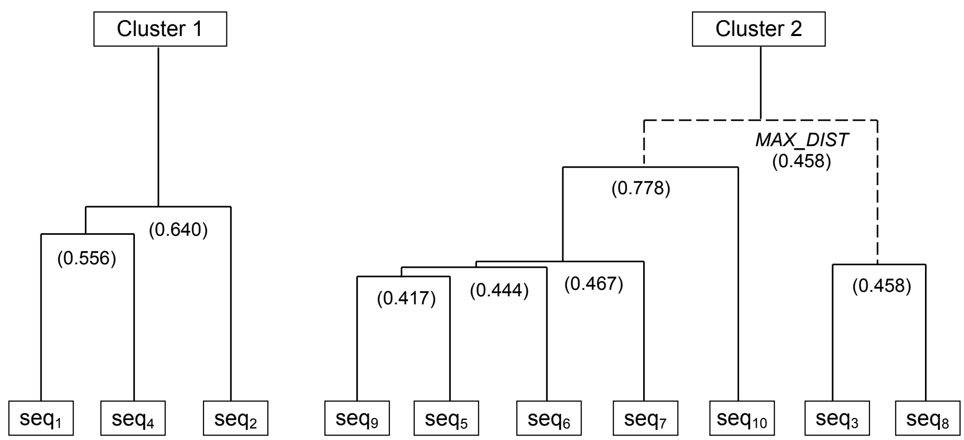
sequence.

seq_3 does not have any sequence denser than itself in its neighbor list (seq_8 and seq_5) but seq_5 has the same density as seq_3. That is seq_3 is a local maxima which has a neighboring region with the same or higher density than itself. If the neighboring region has higher density than itself, the cluster containing seq_3 as its maximum is merged into the neighboring cluster in step 3. Thus in step 3, the cluster density of the two clusters containing seq_3 and seq_5 are

Table 4.14: The modified proximity matrix for Table 4.11

ID	seq_1	seq_2	seq_4	seq_3	seq_5	seq_6	seq_7	seq_8	seq_9	seq_{10}
seq_1	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
seq_2	0.640	0	M_D	∞	∞	∞	∞	∞	∞	∞
seq_4	0.556	M_D	0	∞	∞	∞	∞	∞	∞	∞
seq_3	∞	∞	∞	0	M_D	∞	∞	∞	∞	∞
seq_5	∞	∞	∞	M_D	0	∞	∞	∞	0.417	∞
seq_6	∞	∞	∞	∞	0.583	0	∞	∞	0.444	∞
seq_7	∞	∞	∞	∞	0.467	∞	0	M_D	∞	∞
seq_8	∞	∞	∞	0.458	∞	∞	M_D	0	∞	∞
seq_9	∞	∞	∞	∞	∞	∞	∞	∞	0	∞
seq_{10}	∞	0.800	∞	∞	∞	∞	∞	∞	0.778	0

Figure 4.2: Dendrogram for sequences in $\mathcal{D}_2$ (Table 4.12)



compared. $Density(C_{seq_3}) = Density(seq_3) = 4.364$ and $Density(C_{seq_5}) = Density(seq_9) = 4.500$ where C_{seq_i} denotes the cluster containing seq_i. Hence, cluster C_{seq_3} is merged into cluster C_{seq_5} resulting in the final two clusters. seq_1 and seq_9 are the densest sequence of each cluster.

Table 4.14 and Figure 4.2 demonstrate this example through the single linkage method. In the dendrogram, all the linkages made in step 2 are shown in solid lines. After step 2 three trees are formed - rooted at nodes labeled 0.640, 0.778, and 0.458. Then, since the two trees rooted at nodes labeled 0.778 and 0.458 are linked by MAX_DIST (in the proximity matrix seq_3 and seq_5 are linked by M_D) they are merged in step 3 shown in a dotted line, giving two clusters. The actual linkage value of seq_3 and seq_5 is 0.458 as shown in parenthesis.

4.3 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 [30].

As shown in Table 4.15, 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 Recurrence Relation 4.1.

Table 4.15: Alignment of seq_2 and seq_3

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

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 [30], 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 [30].

4.3.1 Representation of the alignment : Weighted Sequence

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

Here, we propose a notion of weighted sequence as defined in section 3.3. Recall that a weighted sequence $wseq = \langle WX_1 : v_1, \ldots, WX_l : v_l \rangle : n$ 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 summerized 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 $(1 \leq k \leq m)$.

One potential problem with the weighted sequence is that it 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.

4.3.2 Sequence to weighted sequence alignment

Thus, instead of using $REPL()$ (Equation 4.2) directly to calculate the distance between a weighted sequence and a sequence in the cluster, 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

$$
\begin{aligned}
REPL_W(WX, Y) &= \frac{R' \cdot v + n - v}{n} \\
\text{where} \quad R' &= \frac{\sum_{i=1}^{m} w_i + \|Y\| v - 2 \sum_{x_i \in Y} w_i}{\sum_{i=1}^{m} w_i + \|Y\| v}
\end{aligned}
\tag{4.11}
$$

Accordingly, we have

$$\begin{aligned} INDEL(WX) &= REPL_W(WX, ()) = 1 \\ INDEL(Y) &= REPL_W(Y, ()) = 1 \end{aligned} \tag{4.12}$$

Let us first look at an example to better understand the notations and ideas. In the example given in section 4.1, Table 4.16 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 $wseq_3$ (Table 3.3 in section 3 shows the alignment of the first four sequences and the full weighted sequence $wseq_3$), and the algorithm is computing the distance of aligning the first itemset of seq_5, s_{51} =(AX), to 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 4.12, $R' = \frac{2}{5} = \frac{36}{90}$ 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 real average over all itemsets is $\frac{65}{120}$. In this example, R' and $REPL_W$ approximate the actual distances quite well.

Table 4.16: 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	$s_{31}(1)$	(A)	$REPL((\text{A}), (\text{AX})) = \frac{1}{3}$	$Avg = \frac{7}{18} = \frac{35}{90}$
seq_3	$s_{21}(1)$	(AE)	$REPL((\text{AE}), (\text{AX})) = \frac{1}{2}$	
seq_4	$s_{11}(1)$	(A)	$REPL((\text{A}), (\text{AX})) = \frac{1}{3}$	
seq_1	$s_{40}(1)$	()	$INDEL((\text{AX})) = 1$	
Actual avg distance over the four sequences $= \frac{13}{24} = \frac{65}{120}$				

The rationale of $REPL_W()$ (Equation 4.12) 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 4.16 $R' = \frac{36}{90}$ compares very well to 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 4.12 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 Recurrence Relation 4.1 given in section 3.3 replacing $REPL(X, Y)$ with $REPL_W(WX, Y)$ to align all sequences in the cluster.

4.3.3 Order of the alignment

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

Intuitively, the *density* defined in Equation 4.9 measures the similarity between a sequence and its nearest neighbors. Thus, a sequence with a high density means that it has some neighbors very similar to it, and it is a good candidate for a "*seed*" sequence in the alignment. Based on the above observation, in ApproxMAP, we use the following heuristic to apply multiple alignment to sequences in a cluster.

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

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. Our experimental results show that the sequences are aligned fairly well with this ordering.

As the first step in the clustering (see Algorithm 1), the density for each sequence is calculated. We only need to sort all sequences within a cluster in the density descending order in the alignment step.

4.3.4 Example

In the example given in Table 4.1, 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}. The sequences are aligned as follows.

First, sequences seq_2 and seq_3 are aligned as shown in Figure 4.3.

Figure 4.3: seq_2 and seq_3 are aligned resulting in $wseq_1$

seq_2	⟨(A)	()	(BCX)	(D)⟩	
seq_3	⟨(AE)	(B)	(BC)	(D)⟩	
$wseq_{11}$	⟨(A:2, E:1):2	(B:1):1	(B:2,C:2,X:1):2	(D:2):2)⟩	2

Figure 4.4: Sequences $wseq_{11}$ and seq_4 are aligned

$wseq_{11}$	⟨(A:2, E:1):2	(B:1):1	(B:2,C:2,X:1):2	(D:2):2)⟩	2
seq_4	⟨(A)	()	(B)	(DE)⟩	
$wseq_{12}$	⟨(A:3,E:1):3	(B:1):1	(B:3,C:2,X:1):3	(D:3,E:1):3)⟩	3

Figure 4.5: The alignment of remaining sequences in cluster 1

$wseq_{12}$	⟨(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)⟩	
$wseq_{13}$	⟨(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)⟩	
$wseq_{14}$	⟨(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)⟩	
$wseq_{15}$	⟨(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)⟩	
$wseq_{16}$	⟨(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 4.6: The alignment of sequences in cluster 2.

seq_8	⟨(IJ)	()	(KQ)	(M)⟩	
seq_7	⟨(AJ)	(P)	(K)	(L M)⟩	
$wseq_{21}$	⟨(A:1,I:1,J:2):2	(P:1):1	(K:2,Q:1):2	(L:1,M:2):2⟩	2
seq_9	⟨(I)	()	()	(LM)⟩	
$wseq_{22}$	⟨(A:1,I:2,J:2):3	(P:1):1	(K:2,Q:1):2	(L:2,M:3):3⟩	3

Here, we use a *weighted sequence* $wseq_{11}$[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 $wseq_{11}$ 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 $wseq_{11}$, (B:1):1, means there is only one sequence with an itemset aligned in this position, and item B appears once.

After this first step, we need to iteratively align other sequences with the current weighted sequence. Thus, in the next step, the weighted sequence $wseq_{11}$ and the third sequence seq_4 are aligned as shown in Figure 4.4. Similarly, we can align the remaining sequences in cluster 1 as shown in Figure 4.5. The results of aligning sequences in Cluster 2 are shown in Figure 4.6.

The alignment result for all sequences in cluster 1 are summarized in the weighted sequence $wseq_{16}$ shown in Figure 4.5. For cluster 2 the summarized information is in the weighted

[1] The weighted sequences are denoted as $wseq_{ci}$ to indicate the i^{th} weighted sequence for cluster c. Thus, the first 1 indicates that the alignment is for cluster 1 and the second 1 indicates that there are two sequences being aligned. For brevity, sometimes the cluster number is obmited when there is no confusion.

Table 4.17: Aligning Sequences in another 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)⟩	

sequence $wseq_{22}$ shown in Figure 4.6. After the alignment, we only need to store $wseq_{16}$ and $wseq_{22}$.

As shown in the above example, for a cluster of n sequences, the complexity of the multiple alignment of all sequences is $O(n \cdot l_{seq}^2 \cdot i_{seq})$, where $l_{seq}^2 \cdot i_{seq}$ is the maximal cost of aligning two sequences. l_{seq} denotes the length of the longest sequence in the cluster and i_{seq} denotes the maximum number of items in an itemset. The result of the multiple alignment is a weighted sequence. A weighted sequence records the information of the alignment. Once a weighted sequence is derived, the sequences in the cluster will not be visited anymore in the remainder of the mining.

Aligning the sequences in different order may result in slightly different weighted sequences but does not change the underlying pattern in the cluster. To illustrate the effect, Table 4.17 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 4.2, 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 consensus sequence is the same as the variation consensus sequence given in Table 4.4.

As verified by our extensive empirical evaluation, the alignment order has little effect on the underlying patterns.

4.4 Summarize and Present Results

The remaining problem is how to uncover the final underlying patterns from the weighted sequences. It is practically impossible for a computer to automatically generate definitive patterns accurately with no knowledge of the problem domain. Indeed, the best data mining algorithms allow for proper interaction with the domain expert in order to incorporate appropriate domain knowledge where needed. In ApproxMAP, the domain expert is brought into the data mining process in this summarization step. The goal in this step is not to automatically generate definitive patterns from the weighted sequences, but rather to assist the domain experts in doing so themselves.

In this section, we discuss a simple but effective presentation scheme that allows the user to explore the weighted sequences by summarizing the more important components. The level of summarization is easily controlled by a few simple knobs. Such presentation of the weighted sequences can assist a domain expert to extract the patterns from the K weighted sequences representing each of the aligned clusters. It does so by providing an effective scheme for domain experts to interact with the weighted sequences.

4.4.1 Properties of the Weighted Sequence

To design such a scheme we first make some observations about the weighted sequence. Table 4.18 shows a real example from one of our experiments in section 6. The example is from a cluster of 162 sequences. Thus, the full weighted sequence compresses 162 sequences from an aligned cluster. As seen in the Table, although the weighted sequences effectively compress the n aligned sequences into one sequence, the full weighted sequence is still quite large. In fact, the weighted sequence is too much data for a person to view and understand.

The weighted sequence is normally very long because it includes information on all items from all sequences in the group. The domain expert needs a way to focus on the items that are shared by most sequences in the cluster while ignoring those that occur rarely.

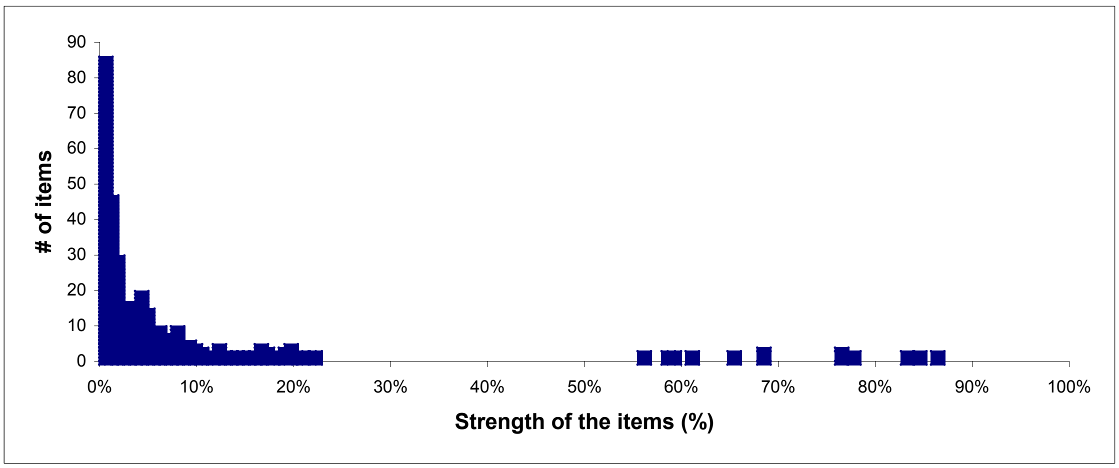
Remember that in the weighted sequence each item has a weight representing its presence in the particular aligned itemset. The strength of an item is calculated as a percentage of sequences in the cluster that have the item in the aligned position. That is $strength(i_{jk}) = \frac{w_{jk}}{n} \cdot 100\%$, where n is the number of sequences in the cluster (i.e. the weight associated with the full weighted sequence). In Table 4.18, B, in the third itemset second line, has $strength = \frac{99}{162} = 61\%$. That is 61% of the sequences in this partition (99 sequences) has the item B in this position. Clearly, an item with larger strength value indicates that the item is shared by more sequences in the partition. Consequently, item strength is a good measure of the importance of an item.

Figure 4.7 is a histogram of all the item strengths for the weighted sequence given in Table 4.18. The weighted sequence has a total of 303 items: 84 of them occur in only 1 sequence, 45 occur in only 2 sequences, and 24 occur in only 3 sequences. In fact, 83% of the items occur in less than 10% of the sequences. On the other hand, only 13 of the items occur in more than 50% of the sequences. Although the clear jump (between 22% and 56%) in the strength of items is not always present, it is typical for weighted sequences to have the bulk of items occur infrequently. In fact, if a cluster had no underlying pattern (i.e., the sequences could not be aligned well), most of the item weights will be infrequent. Most users are not interested in these rare items.

Table 4.18: An Example of a Full Weighted Sequence

Full Weighted Sequence	⟨(E:1, L:1, R:1, T:1, V:1, d:1) (A:1, B:9, C:8, D:8, E:12, F:1, L:4, P:1, S:1, T:8, V:5, X:1, a:1, d:10, e:2, f:1, g:1, p:1) (**B:99**, C:96, D:91, E:24, F:2, G:1, L:15, P:7, R:2, S:8, T:95, V:15, X:2, Y:1, a:2, d:26, e:3, g:6, l:1, m:1) (A:5, B:16, C:5, D:3, E:13, F:1, H:2, L:7, P:1, R:2, S:7, T:6, V:7, Y:3, d:3, g:1) (A:13, B:126, C:27, D:1, E:32, G:5, H:3, J:1, L:1, R:1, S:32, T:21, V:1, W:3, X:2, Y:8, d:13, e:1, f:8, i:2, p:7, l:3, g:1) (A:12, B:6, C:28, D:1, E:28, G:5, H:2, J:6, L:2, S:137, T:10, V:2, W:6, X:8, Y:124, a:1, d:6, g:2, i:1, l:1, m:2) (A:135, B:2, C:23, E:36, G:12, H:124, K:1, L:4, O:2, R:2, S:27, T:6, V:6, W:10, X:3, Y:8, Z:2, a:1, d:6, g:1, h:2, j:1, k:5, l:3, m:7, n:1) (A:11, B:1, C:5, E:12, G:3, H:10, L:7, O:4, S:5, T:1, V:7, W:3, X:2, Y:3, a:1, m:2) (A:31, C:15, E:10, G:15, H:25, K:1, L:7, M:1, O:1, R:4, S:12, T:10, V:6, W:3, Y:3, Z:3, d:7, h:3, j:2, l:1, n:1, p:1, q:1) (A:3, C:5, E:4, G:7, H:1, K:1, R:1, T:1, W:2, Z:2, a:1, d:1, h:1, n:1) (A:20, C:27, E:13, G:35, H:7, K:7, L:111, N:2, O:1, Q:3, R:11, S:10, T:20, V:111, W:2, X:2, Y:3, Z:8, a:1, b:1, d:13, h:9, j:1, n:1, o:2) (A:17, B:2, C:14, E:17, F:1, G:31, H:8, K:13, L:2, M:2, N:1, R:22, S:2, T:140, U:1, V:2, W:2, X:1, Z:13, a:1, b:8, d:6, h:14, n:6, p:1, q:1) (A:12, B:7, C:5, E:13, G:16, H:5, K:106, L:8, N:2, O:1, R:32, S:3, T:29, V:9, X:2, Z:9, b:16, c:5, d:5, h:7, l:1) (A:7, B:1, C:9, E:5, G:7, H:3, K:7, R:8, S:1, T:10, X:1, Z:3, a:2, b:3, c:1, d:5, h:3) (A:1, B:1, H:1, R:1, T:1, b:2, c:1) (A:3, B:2, C:2, E:6, F:2, G:4, H:2, K:20, M:2, N:3, R:19, S:3, T:11, U:2, X:4, Z:34, a:3, b:11, c:2, d:4) (H:1, Y:1, a:1, d:1)⟩	:162

Figure 4.7: Histogram of strength (weights) of items



Not surprising, our experimental results further show that in practice:

1. The small number of items that have strength greater than 50% are items from the embedded patterns [2].

2. The bulk of the items that have strength less than 10% are all random noise (i.e., items that were not part of the embedded patterns).

[2] In the experiment, we randomly embedded sequential patterns into the database then tried to recover these embeded patterns.

3. Items with strength between 10% and 50% were either items from the embedded patterns or random noise, with higher strength items being more likely to be items from the embedded patterns.

In most applications the weights of the items will probably be similar to Zipf distribution. That is, there are many rare items and very few frequent items [9]. These observations support our hypothesis that the items in a weighted sequence can be divided into three groups based on their strengths as follows:

1. The frequent items constitute the underlying pattern in the group. These are the few frequent items shared by many of the sequences in the partition.

2. The rare items are random occurrences (noise). Most items in a weighted sequences are rare items.

3. The common items, which do not occur frequently enough to be part of the underlying pattern but occur in enough sequences to be of some interest, are the variations to the underlying pattern. These items most likely occur regularly in a subgroup of the sequences.

4.4.2 Summarizing the Weighted Sequence

Utilizing such insight into the properties of the items in a weighted sequence, we summarize the final results by defining two types of consensus sequences corresponding to two cutoff points, θ and δ, as follows:

1. The frequent items ($w \geq \theta \cdot n$) are summarized into the pattern consensus sequence.

2. The common items ($\delta \cdot n \leq w < \theta \cdot n$) are added onto the pattern consensus sequence to form the variation consensus sequence. When the variation consensus sequence is presented together with the pattern consensus sequence, it clearly depicts the common items as the variations to the underlying pattern. In Table 4.19, looking at the pattern consensus sequence and the variations consensus sequence (lines 2 and 3), it is clear that E in the fourth, G in the fifth, and Z in the last itemsets are the possible variations to the main pattern.

3. The rare items ($w < \delta \cdot n$) , i.e., random occurrences, are not presented to the user.

See section 3.3 for a formal definition. Table 4.19 shows how the weighted sequence given in Table 4.18 is summarized into the two consensus sequences with the cutoff points $\theta = 50\%$ for pattern consensus sequence and $\delta = 20\%$ for variation consensus sequence.

This summarization method presents both the frequent underlying pattern and their variations while ignoring the noise. Furthermore, the user can control the level of summarization by defining the two cutoff points for frequent(θ) and rare items(δ) as desired (Equation 3.5).

Table 4.19: A Weighted Consensus Sequence

Full Weighted Sequence	⟨(E:1, L:1, R:1, T:1, V:1, d:1) (A:1, B:9, C:8, D:8, E:12, F:1, L:4, P:1, S:1, T:8, V:5, X:1, a:1, d:10, e:2, f:1, g:1, p:1) (**B:99, C:96, D:91**, E:24, F:2, G:1, L:15, P:7, R:2, S:8, **T:95**, V:15, X:2, Y:1, a:2, d:26, e:3, g:6, l:1, m:1) (A:5, B:16, C:5, D:3, E:13, F:1, H:2, L:7, P:1, R:2, S:7, T:6, V:7, Y:3, d:3, g:1) (A:13, **B:126**, C:27, D:1, E:32, G:5, H:3, J:1, L:1, R:1, S:32, T:21, V:1, W:3, X:2, Y:8, d:13, e:1, f:8, i:2, p:7, l:3, g:1) (A:12, B:6, C:28, D:1, E:28, G:5, H:2, J:6, L:2, **S:137**, T:10, V:2, W:6, X:8, **Y:124**, a:1, d:6, g:2, i:1, l:1, m:2) (**A:135**, B:2, C:23, **E:36**, G:12, **H:124**, K:1, L:4, O:2, R:2, S:27, T:6, V:6, W:10, X:3, Y:8, Z:2, a:1, d:6, g:1, h:2, j:1, k:5, l:3, m:7, n:1) (A:11, B:1, C:5, E:12, G:3, H:10, L:7, O:4, S:5, T:1, V:7, W:3, X:2, Y:3, a:1, m:2) (A:31, C:15, E:10, G:15, H:25, K:1, L:7, M:1, O:1, R:4, S:12, T:10, V:6, W:3, Y:3, Z:3, d:7, h:3, j:2, l:1, n:1, p:1, q:1) (A:3, C:5, E:4, G:7, H:1, K:1, R:1, T:1, W:2, Z:2, a:1, d:1, h:1, n:1) (A:20, C:27, E:13, **G:35**, H:7, K:7, **L:111**, N:2, O:1, Q:3, R:11, S:10, T:20, **V:111**, W:2, X:2, Y:3, Z:8, a:1, b:1, d:13, h:9, j:1, n:1, o:2) (A:17, B:2, C:14, E:17, F:1, G:31, H:8, K:13, L:2, M:2, N:1, R:22, S:2, **T:140**, U:1, V:2, W:2, X:1, Z:13, a:1, b:8, d:6, h:14, n:6, p:1, q:1) (A:12, B:7, C:5, E:13, G:16, H:5, **K:106**, L:8, N:2, O:1, R:32, S:3, T:29, V:9, X:2, Z:9, b:16, c:5, d:5, h:7, l:1) (A:7, B:1, C:9, E:5, G:7, H:3, K:7, R:8, S:1, T:10, X:1, Z:3, a:2, b:3, c:1, d:5, h:3) (A:1, B:1, H:1, R:1, T:1, b:2, c:1) (A:3, B:2, C:2, E:6, F:2, G:4, H:2, K:20, M:2, N:3, R:19, S:3, T:11, U:2, X:4, **Z:34**, a:3, b:11, c:2, d:4) (H:1, Y:1, a:1, d:1)⟩	:162
Pattern ConSeq	⟨(B,C,D,T) (B) (S,Y) (A,H) (L,V) (T) (K) ⟩	
Variation ConSeq	⟨(B,C,D,T) (B) (S,Y) (A,E,H) (G,L,V) (T) (K) (Z)⟩	
Weighted Pattern Consensus Seq	⟨(B:99, C:96, D:91, T:95) (B:126) (S:137, Y:124) (A:135, H:124) (L:111,V:111) (T:140) (K:106)⟩	:162
Weighted Variation Consensus Seq	⟨(B:99, C:96, D:91, T:95) (B:126) (S:137, Y:124) (A:135, E:36, H:124) (G:35, L:111,V:111) (T:140) (K:106) (Z:34)⟩	:162

These two settings can be used uniformly across all partitions to present two consensus sequences per partition to the user: the estimated underlying pattern and the estimated variation of it. Our experiments (section 6) show that such uniform estimation of the patterns and variations is reasonable for most partitions. These estimated consensus sequences provide a comprehensive summary of the database and is a good starting point for the domain experts to determine the underlying trend in the data.

Another key to our simple method is specifying the uniform cutoff points as a percentage of the partition size. This allows ApproxMAP to customize the cutoff values appropriately for each partition according to the size of the partition. This simple scheme allows users to specify what items are frequent in relation to similar sequences. Such a multilevel threshold approach has been proposed for association rule mining to overcome problems with using the uniform support threshold. However unlike the methods developed for association rules [42, 43, 60], our method is simple for users to specify. The users are only required to specify two uniform cutoff points relative to the number of similar sequences..

In practice, there is no easy way to know the correct settings for the two cutoffs θ and δ. Indeed, the theoretical properties of the strength threshold is too complex to be studied

Table 4.20: User Input Parameters for ApproxMAP and their Default Values

Parameter	Default	Description
k	5	for k-nearest neighbor clustering. Defines the neighbor region and controls the resolution of the partitions.
θ	50% of cluster	the cutoff point for pattern consensus sequences specified as a percentage of cluster size
δ	20% of cluster	the cutoff point for variation consensus sequences specified as a percentage of cluster size

mathematically. However, we have studied the parameters empirically using various data sets in section 6.1.2. The experiments suggests that ApproxMAP is in fact quite robust to the cutoff value with a wide range giving reasonable results. We found that 20%-50% is a good range for the strength threshold for a wide range of datasets.

In ApproxMAP, the default values are set to $\theta = 50\%$ for pattern consensus sequences and $\delta = 20\%$ for variation consensus sequences. These setting are based on the empirical study of the strength threshold in section 6.1.2. On the one hand, with the conservative estimate of $\theta = 50\%$, ApproxMAP is able to recover most of the items from the embedded pattern without picking up any extraneous items in the pattern consensus sequence. On the other hand, with a modest estimate of $\delta = 20\%$, ApproxMAP can detect almost all of the items from the embedded pattern while picking up only a small number of extraneous items in the variation consensus sequence.

Table 4.20 summarizes the user input parameters for ApproxMAP and their default values. The default values in ApproxMAP are set so that a quick scan over all the two consensus sequences (the pattern consensus sequence and the variation consensus sequence) for each partition would give as best an overview of the data as possible without manually setting any of the parameters. Our experimental results show that ApproxMAP is robust to all three input parameters. In fact, the default values worked well for all the large datasets used in our experiments. In practice, the default values provide a more than adequate starting point for most applications.

4.4.3 Visualization

To further reduce the data presented to the user we have designed a simple color-coding scheme to represent the cluster strength for each item in the pattern. As demonstrated in the last two lines of Table 4.19, even when noise is filtered out, looking at a sequence of sets of items together with their weights is too much data for the human eye. The color scheme reduces a dimension in the weighted sequence by using color instead of numbers to depict the item weights. Taking into account human vision, we used the standard gray scale display technique [21] on the color scheme as shown in Table 4.21.

The coloring scheme is simple but effective. In fact, color is better for visualizing the

Table 4.21: Presentation of consensus sequences

100%: 85%: 70%: 50%: 35%: 20%									
	(B:61%, C:59%, D:56%, T:59%)	(B: 78%)	(S:85%, Y:77%)	(A:83%, E:22%, H:77%)	(G:22%, L:69%, V:69%)	(T: 86%)	(K: 65%)	(B: 21%)	: 162
Pattern	(B C D T)	(B)	(S Y)	(A H)	(L V)	(T)	(K)		:162
Variation	(B C D T)	(B)	(S Y)	(A E H)	(G L V)	(T)	(K)	(Z)	:162

Table 4.22: Optional Parameters : the default values need not change

Parameter	Default	Description
max_DB_strength	10% of $\|\mathcal{D}\|$	the cutoff point for variation consensus sequences specified as the percentage of the full database
min_DB_strength	10 sequences	the cutoff point for pattern consensus sequences specified as the minimum number of required sequences

variations in the item strength of the cluster. In Table 4.21, we can easily see that A in the fourth itemset is present in almost all sequences while E in the same itemset is present in only a small proportion of the sequences. We omit the itemset weights from the weighted sequences because it has little value once alignment is finished. The sequence weight is given to indicate how many sequences have been summarized in the consensus sequence.

4.4.4 Optional Parameters for Advanced Users

Finally, we provide the user with one more optional scheme to control what they view. Although uniform cutoff points work well for most clusters, it can be misleading when the partitions are tiny or huge.

For example, when a few outlier sequences form a tiny cluster (say 5 sequences), then the 50% cutoff point for generating the underlying pattern could generate patterns from items that occur in only a very small number of sequences ($w \geq \theta \cdot n = 0.5 * 5 = 2.5 > 2$ sequences). Although it should be clear to the user that the underlying pattern and the dark items are relative to the partition size, it can still be misleading. Furthermore, it is more likely that these small partitions do not have an underlying pattern.

Thus, the users have the option to specify an absolute minimum database strength for pattern consensus sequences. The default in ApproxMAP is set to 10 sequences. That means, regardless of the size of the cluster, an item has to occur in a particular location in at least 10 sequences for it to be considered as a definitive item in the pattern consensus sequence. This default should not be changed unless specifically required by the application. This screens out the tiny partitions and prevents these partitions from generating misleading patterns.

Note that this cutoff point does not affect the variation consensus sequence. Thus, if there are patterns of interest that do not meet this cutoff point it could still be spotted in the variation consensus sequence. In essence, that means ApproxMAP considers these items as ambiguous items which might or might not be part of the underlying pattern.

In a similar way, in huge clusters (say half of the 1000 sequences in the database $\mathcal{D}$ clustered into one partition) the current scheme could eliminate frequent items as noise ($w < \delta \cdot n = 0.2 * 1000 * 0.5 = 100$ sequences). Although the user should be aware of this, again this could be misleading.

Here the user has the option to specify an absolute maximum database strength for variation consensus sequences. This value does not affect the pattern consensus sequence. The default in ApproxMAP is set to 10% of the database. That is, if an item occurs regularly in a certain location in 10% of the full database, we assume that the item is not a random occurrence and include it in the variation consensus sequence. The user is than able to note and determine what it might be. Again, this default should not be changed unless specifically required by the application.

4.4.5 Algorithm

The final consensus sequences are generated by picking only those items that meet the four strength levels as given in Algorithm 2.

In summary, our presentation scheme is simple but powerful for summarizing and visualizing the weighted sequences. The simplicity of the scheme is its strength. It is simple for people to fully control dynamically. That is, once the weighted sequence for all clusters have been found, generating appropriate consensus patterns for differing cutoff values is a trivial process. Hence, understanding the process to control it is easy for users. In addition, computing new results is very quick allowing users to interactively try many different cutoff points.

In addition, the summarized consensus sequences are succinct yet expressive. We say it is expressive because the user can easily include or exclude as much information as they want by setting the cutoff points as desired. Furthermore, the user is fully aware of what is being omitted from the summary. That is the users know the cutoff point used to drop items from the weighted sequence. Note that advanced users can also set the cutoff point for frequent and rare items individually for each partition after careful investigation.

We say it is succinct because, ApproxMAP is able to reduce the hug high dimensional sequential database into a handful of consensus sequences. Yet the consensus sequences offer comprehensive information on how many sequences are in each partition and how frequent items occur in the partition via color.

We emphasize that the goal in this step is not to automatically generate the definitive patterns in each group, but rather provide the users with the proper tools to find the under-

ALGORITHM 2. (GENERATE CONSENSUS SEQUENCES)

Input:

1. a weighted sequence per cluster
2. cutoff point θ and $min\_DB\_strength$ for pattern consensus sequences (frequent items)
3. cutoff point δ and $max\_DB\_strength$ for variation consensus sequences (rare items)
4. the size of the full database, $\|\mathcal{D}\|$

θ and δ are specified as a percentage of the cluster, $min\_DB\_strength$ is specified as the absolute number of sequences, and $max\_DB\_strength$ is specified as a percentage of all sequences.

Output: two consensus sequences per weighted sequence:

1. pattern consensus sequences
2. variation consensus sequences

Method: For each weighted sequence, $wseq = \langle WX_1 : v_1, \ldots, WX_l : v_l \rangle : n$

1. **pattern consensus sequences**
   $Npat\_cutoff = n * \theta$;
   if ($Npat\_cutoff \leq min\_DB\_strength$) then
   select all items with weight at least $min\_DB\_strength$;
   else
   select all items with weight at least $Npat\_cutoff$;
2. **variation consensus sequences**
   $Nvar\_cutoff = n * \delta$;
   $Nmax\_DB = \|\mathcal{D}\| * max\_DB\_strength$
   if ($Nvar\_cutoff \leq Nmax\_DB$) then
   select all items with weight at least $Nvar\_cutoff$;
   else
   select all items with weight at least $Nmax\_DB$;

lying patterns themselves. An interactive program that enables the user to explore (1) the weighted sequences via different cutoff points, which will generate various consensus sequences dynamically, and (2) the aligned clusters themselves would be the best interface for finding the sequential patterns.

4.4.6 Example

In the example given in section 4.1, the default cutoff points for pattern consensus sequence ($\theta = 50\%$) and variation consensus sequence ($\delta = 20\%$) are used. However for such a tiny database the minimum and maximum database cutoff points do not have much meaning. For completeness of the example, we specified the minimum database strength as 1 sequence, and the maximum database strength as 100%=10 sequences.

Thus, the pattern consensus sequence in cluster 1 selected all items with weight greater than 3 sequences ($w \geq 50\% * 7 = 3.5 > 3$) while the pattern consensus sequence in cluster 2 selected all items with weight greater than 1 sequence ($w \geq 50\% * 3 = 1.5 > 1$). Cluster 2 is too small to have any meaningful variations. Since the pattern cutoff point was defined at 2 sequences, the variation cutoff point would have to be set at 1 sequence which would give all items in the weighted sequence. However, cluster 1 generated the variation consensus sequences with weight greater than 1 sequence ($w \geq 20\% * 7 = 1.4 > 1$). ApproxMAP detects a variation to the underlying pattern of $\langle$(A)(BC)(DE)$\rangle$ between the first and second itemset. It indicates that in about 20% of the sequences in the partition, there is a repeat of the itemset (B) between the first and second itemset resulting in a variation pattern of $\langle$(A)(B)(BC)(DE)$\rangle$ (Table 4.4).

The small example in section 4.1 is not illustrative of the effectiveness of our presentation scheme. However, the experimental results indicate that such presentation of the weighted sequence is succinct but comprehensive. Section 6.2.2 demonstrates this point well with a small data set. Leaving the details for later, we summarize the main points here. Table 6.14 gives the 16 consensus sequences (labeled $PatConSeq_i$ and $VarConSeq_i$), which summarize 1000 sequences, presented to the user along with the 10 base patterns (labeled $BaseP_i$) used to generated the dataset [3]. Manual inspection indicates how well the consensus sequences match the base patterns used to generate the data. Each consensus pattern found is a subsequence of considerable length of a base pattern. Clearly, the 16 consensus sequences provide a succinct but comprehensive overview of the 1000 data sequences.

4.5 Time Complexity

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 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 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, the length and number of patterns, and the four strength cutoff points $\theta, \delta, max\_DB\_strength$, and $min\_DB\_strength$. The length and number of patterns in the data do not affect the running time because ApproxMAP finds all the consensus sequences directly from the data without having to build the patterns. The value of the cutoff parameters obviously do not effect the

[3] As discussed in section 5, each patterned dataset is generated by embedded base patterns. These base patterns are the underlying trend in the data which need to be uncovered.

running time.

ApproxMAP scales reasonably well with respect to the size of the database. It is quadratic with respect to the data size and has reasonable execution times on common CPU technology and memory size. In our experiments, as expected, the execution time increases quadratically with respect to the number of sequences, the average number of itemsets per sequence, and linearly with respect to the average number of items in the sequences. As a reference, the example given for 1000 sequences with $L_{seq} = 10$ and $I_{seq} = 2.5$ took roughly 10 seconds on an 2GHZ Intel Xeon processor while the example of 10,000 sequences with $L_{seq} = 10$ and $I_{seq} = 2.5$ took roughly 15 minutes.

To combat the $O(L_{seq}^2)$ and $O(N_{seq}^2)$ time complexity, for larger datasets we introduce improvements to the basic algorithm. First, ApproxMAP can speedup $O(L_{seq}^2)$ by calculating the distance matrix to only the needed precision because it only needs to know the exact distance of the k nearest neighbors. All other sequences have distance=∞ so the accurate distance does not need to be calculated. Second, ApproxMAP can be extended to use a sample based iterative clustering approach of sampling then iteratively recentering and reassigning clusters to speedup $O(N_{seq}^2)$. The next section discusses these improvements to the basic algorithm.

4.6 Improvements

The bottle neck in the running time is the clustering step which has to calculate the $N_{seq} * N_{seq}$ proximity matrix and build the k-nearest neighbor list. Here we discuss how to speedup this process. There are two components to the running time for calculating the proximity matrix: (1) the per cell calculation in the proximity matrix and (2) the total of N_{seq}^2 cell calculations needed for the proximity matrix. Both of these components can be sped up.

4.6.1 Reduced Precision of the Proximity Matrix

Each cell in the proximity matrix is calculated using Equation 4.7. Thus, the time complexity is $O(L_{seq}^2 * I_{seq})$ for solving the recurrence relation for $D(seq_i, seq_j)$ (Equation 4.1). A recurrence relation such as Equation 4.1 can be solved through dynamic programming by building a table that stores the optimal subproblem as shown in Table 4.23.

In Table 4.23, we show the intermediate calculation along with the final cell value. Each cell $RR(p, q)$ has four values in a 2*2 matrix. 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 deleting itemset q. The bottom left cell is the result of moving right by inserting itemset p. The final value in the cell, shown in the bottom right position in large font, is the minimum of these three values. The arrow next to the final value, indicates the direction

Table 4.23: Recurrence Relation Table

	seq_6	(AY)		(BD)		(B)		(EY)	
seq_2	**0**	1		2		3		4	
(A)	1	$\frac{1}{3}$	2	2	3	3	4	4	5
		2	$\searrow \mathbf{\frac{1}{3}}$	$1\frac{1}{3}$	$\rightarrow \mathbf{1\frac{1}{3}}$	$2\frac{1}{3}$	$\rightarrow 2\frac{1}{3}$	$3\frac{1}{3}$	$\rightarrow 3\frac{1}{3}$
(BCX)	2	2	$1\frac{1}{3}$	$\frac{1}{3}+\frac{3}{5}=\frac{14}{15}$	$2\frac{1}{3}$	$1\frac{1}{3}+\frac{1}{2}=1\frac{5}{6}$	$3\frac{1}{3}$	$3\frac{1}{3}$	$4\frac{1}{3}$
		3	$\downarrow 1\frac{1}{3}$	$2\frac{1}{3}$	$\searrow \frac{14}{15}$	$1\frac{14}{15}$	$\searrow \mathbf{1\frac{5}{6}}$	$2\frac{5}{6}$	$\rightarrow 2\frac{5}{6}$
(D)	3	3	$2\frac{1}{3}$	$1\frac{2}{3}$	$1\frac{14}{15}$	$1\frac{14}{15}$	$2\frac{5}{6}$	$2\frac{5}{6}$	$3\frac{5}{6}$
		4	$\downarrow 2\frac{1}{3}$	$3\frac{1}{3}$	$\searrow 1\frac{2}{3}$	$2\frac{2}{3}$	$\searrow 1\frac{14}{15}$	$2\frac{14}{15}$	$\searrow \mathbf{2\frac{5}{6}}$

taken in the cell. 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 (B) with (BCX) (shown in the upper left $RR(3-1,2-1) + REPL((\mathsf{B}),(\mathsf{BCX})) = 1\frac{1}{3} + \frac{1}{2} = 1\frac{5}{6}$), insert (B) (shown in the upper right $RR(3,2-1) + INDEL = 2\frac{1}{3} + 1 = 3\frac{1}{3}$), or delete (BCX) (shown in the lower left $RR(3-1,2) + INDEL = \frac{14}{15} + 1 = 1\frac{14}{15}$). Since $1\frac{5}{6}$ is the minimum of the three, the replace operation is chosen showing the diagonal movement. The final distance between seq_2 and seq_6 is $2\frac{5}{6}$, which can be found by following the minimum path: diagonal (REPLACE), right(INSERT), diagonal, and diagonal. This path gives the pairwise alignment shown in Table 4.6.

Often times a straight forward dynamic programming algorithm for solving such recurrence relation can be improved by only calculating up to the needed precision [14]. In ApproxMAP, we note that we do no need to know $dist(seq_i, seq_j)$ for all i, j to full precision. The modified proximity matrix (Tables 4.10 and 4.14) includes all information needed to cluster the sequences in the database. Furthermore, the modified proximity matrix has mostly values of ∞ because $k << N$. Thus, we only need to calculate to the precision of the modified proximity matrix. That is, if a cell is clearly ∞ at any point in the dynamic programming algorithm, we can stop the calculation and return ∞.

$dist(seq_i, seq_j)$ 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 (Equation 4.9), 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 $RR(\|seq_i\|, \|seq_j\|) = D(\|seq_i\|, \|seq_j\|)$.

$$min\_row(p) \quad = \quad min\{RR(p,q)\} \quad \text{for all } 0 \le q \le \|seq_i\| \tag{4.13}$$

THEOREM 1. There is a connected path from $RR(0,0) = 0$ to any cell $RR(p,q)$ such that (1) the values of the cells in the path are monotonically increasing, (2) the two indices never decrease (i.e. the path is always moving downward or to the right), and (3) there must be at least one cell from each rows 0 to $p-1$, in the connected path.

Proof: The theorem comes directly from the definitions. First, the value of any cell $RR(p,q)$ are 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, the value of all cells are constructed from only three neighboring cells - namely up, left, or upper left - so the path can only move downward or to the right. And last, since there has to be a connect path from $RR(0,0)$ to cell $RR(p,q)$, there must be at least one cell from each rows 0 to $p-1$.

THEOREM 2. $RR(\|seq_i\|, \|seq_j\|)$ is greater than or equal to the minimum row value in any row.

$$RR(\|seq_i\|, \|seq_j\|) \quad \ge \quad min\_row(p) \quad \text{for all } 0 \le p \le \|seq_j\| \tag{4.14}$$

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 a monotonically increasing according to Theorem 2, $RR(\|seq_i\|, \|seq_j\|)$ must be greater then equal to $RR(p,q)$. Thus, $RR(\|seq_i\|, \|seq_j\|) \ge RR(p,q) = min\_row(p)$. This is a contradiction. Second, $RR(p,q)$ is not in the connected path from $RR(0,0)$ to $RR(\|seq_i\|, \|seq_j\|)$. Then, let $RR(p,a)$ be a cell in the connected path. Since $min\_row(p) = RR(p,q)$, $RR(p,a) \ge RR(p,q)$. Thus, $RR(\|seq_i\|, \|seq_j\|) \ge RR(p,a) \ge RR(p,q) = min\_row(p)$. 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, $RR(\|seq_i\|, \|seq_j\|) \ge min\_row(p)$ 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-nearest neighbor of seq_j, and seq_j is not a k-nearest neighbor of seq_i.

Proof: By Theorem 2, $RR(\|seq_i\|, \|seq_j\|) = D(seq_i, seq_j) \ge min\_row(p)$ for any row p. Thus, $dist(seq_i, seq_j) = \frac{D(seq_i, seq_j)}{max\{\|seq_i\|, \|seq_j\|\}} \ge \frac{min\_row(p)}{max\{\|seq_i\|, \|seq_j\|\}} > max\{dist_k(seq_i), dist_k(seq_j)\}$ 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-nearest neighbor 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 $k << N$ and $k << L$.

Hueristically, setting up the recurrence table such that the shorter of the two sequences goes across the recurrence table and the longer sequence goes down can save more time. This is because the algorithm has to finish calculating a full row before checking for the stop condition. Thus intuitively, recurrence tables with shorters rows are likely to find the stopping condition faster. The experimental results in section 6.1.4 show that in practice, such optimization can speedup time by a factor of upto 40%. The running time becomes almost linear with respect to L_{seq} (Figure 6.4).

4.6.2 Sample Based Iterative Clustering

The $O(N_{seq}^2)$ for the clustering step can be improved by using an iterative partitioning method similar to the well known kmediods[4] clustering methods. However, there are two major difficulties with using the kmediods clustering methods. First, without proper initialization, it is impossible to find the right clusters. Thus, finding a reasonably good starting condition is crucial for kmediods methods to give good results. Second, the general kmediods method requires that the user input the number of clusters in the data. However, the proper number of partitions is not known in our application.

To overcome these problems but still benefit from the efficiency in time, 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-nearest neighbor method discussed in section 4.2.2. 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 in Algorithm 3.

When $\|\mathcal{D}'\| << \|\mathcal{D}\|$, the time complexity for the clustering method is obviously $O(t \cdot N_{seq})$ where t is the number of iterations needed to converge. Our experiments show that the algorithm converges very quickly. Figure 4.8 shows that for most datasets it take from 3 to 6 iterations. The experimental results in section 6.1.5 show that in practice, this optimization can speedup time considerably at the cost of a slight reduction in accuracy (Figure 6.6(b)) and larger memory requirements. Given enough memory the running time is reduced to about 10%-15% and becomes almost linear with respect to N_{seq} (Figures 6.6(c) and 6.6(d)

[4]kmediods clustering is exactly the same as the more popular kmeans algorithm, but it works with the representative points in clusters rather than the means of clusters.

Algorithm 3. (Sample Based Iterative Clustering)

Input: a set of sequences $\mathcal{D} = \{seq_i\}$, sampling percentage α, and number of neighbor sequences k' for the sampled dataset;

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

Method:

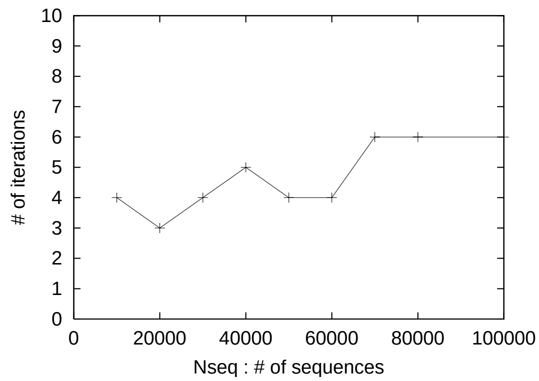
1. **Randomly sample the database $\mathcal{D}$ into $\mathcal{D}'$ using α.** The size of $\mathcal{D}'$ will be a trade off between time and accuracy. In our experiments, we found that at a minimum $\|\mathcal{D}'\|$ should be 4000 sequences for the default $k' = 3$. We also found that roughly 10% of the data will give you comparable results when $N_{seq} \geq 40,000$.
2. **Run uniform kernel density based k-NN clustering (Algorithm 1) 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 for each cluster is the one which minimizes the sum of the intra-sequence cluster distance. That is, 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.
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 changes for any cluster or a certain iteration threshold, $MAX\_LOOP = 100$, is met.

- *optimized (all)*[5]). Even when there is not enough memory, in the worst case the running time is still reduced to about 25%-40% (Figures 6.6(c) and 6.6(d) - *optimized (none)*).

There is a few implementation details to note. First, when using a small sample of the data, k for k-nearest neighbor algorithm has to be smaller than what is used on the full database to achieve the clustering at the same resolution because the k-nearest neighbor in the sampled data is most likely $(k + \alpha)$-nearest neighbor in the full database. Again this

[5] *optimized (all)* is a simple hash table implementation with all proximity values stored and *optimized (none)* is the implementation with none of the proximity values stored.

Figure 4.8: Number of iterations



might be best understood if you think of a digital image. When less pixels are sampled, blurring has automatically occured from the sampling and less manual blurring (averaging of the sampled points) need to happen for a certain resolution of the picture. In ApproxMAP, since the default value for k is 5, the default for k' in the sample based iterative clustering method is 3.

Second, in the iterative clustering method much more memory is required in order to fully realize the reduction in running time because the $N_{seq} * N_{seq}$ proximity matrix needs to be stored in memory. In the normal method, although we need to calculate the full $N_{seq} * N_{seq}$ proximity matrix, the information can be processed one row at a time and there is no need to return to any value. That is, we only need to maintain the k nearest neighbor 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 that the distance calculation does not have to be repeated. When N_{seq} is large, the proximity matrix is quite large. Hence, there is a large memory requirement for the fastest optimized algorithm.

Nonetheless, we should note that the proximity matrix becomes very sparse when the number of clusters is much smaller than N_{seq} ($\|C_s\| << N_{seq}$). Thus, much space can be saved by using a hash table instead of a matrix. Our initial implementation of a simple hash table ran upto $N_{seq} = 70,000$ with 2GB of memory. A more efficient hash table could easily improve the memory requirement. Furthermore, a slightly more complicated scheme of storing only upto the possible number of values and recalculating the other distances when needed (much like a cache) will reduce the running time compared to the normal method. Efficient hash tables are a research topic on its own. Our initial implementation of the hash table demonstrates the huge potential for reduction in time well.

In order to study the full potential we also measured the running time when no memory was used to store the proximity matrix. That is, distances values were always recalculated whenever needed. We found that even in this worst case the running time was reduced significantly to about 25%-40% (Figures 6.6(c) and 6.6(d) - *optimized (none)*). Thus, a good implementation would give running times in between the *optimized (all)* and the *optimized (none)* line in Figure 6.6(c). More efficient hash table implementations will be studied in future research.

The loss of accuracy is from the difficulty kmediods clustering algorithms have with outliers and clusters of different sizes and non-globular shapes [17]. As with most iterative clustering methods, this sampled based algorithm tries to optimize a criterion function. That is the sequences are partitioned so that the intra-cluster distance is minimized. This tends to build clusters of globular shape around the representative sequences regardless of the natural shape of the clusters. Thus, the clusters are not as accurate as when they were built around similar sequences following the arbitrary shape and size of the cluster as done in the pure density based k-nearest neighbor algorithm.

Moreover, it is not robust to outlier sequences. Although kmediods methods based on representative points are more robust to outliers than kmeans methods, the outliers will still influence the choice of the representative points. This in turn, will change how the data is partitioned arbitrarily.

Fortunately, the steps following the clustering step makeup for most of the inaccuracy introduced in it. That is, multiple alignment of each cluster into weighted sequences and then summarization of the weighted sequences into consensus sequences is very robust to outliers in the cluster. In fact by the time we get to the final results, the consensus sequences, only the general trend in each cluster remain. In practice, the core sequences that form the main trend do not change regardless of the different clustering methods because these sequences will be located close to each other near the center. Thus, the clustering step only needs to be good enough to group these core sequences together in order to get comparable consensus sequences.

Chapter 5

Evaluation Method

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 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 method. Using this method, we are able to understand the behavior of an algorithm empirically by running extensive systematic experiments on synthetic data.

The evaluation method is a matrix of four experiments - (1) random data, (2) patterned data, and patterned data with (3) varying degree of noise, and (4) varying number of outliers - assessed on five criteria: (1) recoverability, (2) precision, (3) the total number of result patterns returned, (4) the number of spurious patterns, and (5) the number of redundant patterns. Recoverability, defined in section 5.2, provides a good estimation of how well the underlying trends in the data are detected. Precision, adopted from ROC analysis [46], is a good measure of how many incorrect 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 spurious and redundant patterns along with the total number of patterns returned give an overview of the result at the sequence level. In summary, a good model would produce (1) high recoverability and precision, with (2) small number of spurious and redundant patterns, and (3) a manageable number of result patterns.

This evaluation method will enable researchers not only to use synthetic data to benchmark performance in terms of speed, but also to quantify the quality of the results. Such benchmarking will become increasingly important as more data mining methods focus on approximate solutions.

Table 5.1: Parameters for the random data generator

Notation	Meaning
$\|\mathcal{I}\|$	# of items
N_{seq}	# of data sequences
L_{seq}	Average # of itemsets per data sequence
I_{seq}	Average # of items per itemset in the database

Table 5.2: Parameters for the IBM patterned data generator

Notation	Meaning
$\|\mathcal{I}\|$	# of items
$\|\Lambda\|$	# of potentially frequent itemsets
N_{seq}	# of data sequences
N_{pat}	# of base patterns (potentially frequent sequential patterns)
L_{seq}	Average # of itemsets per data sequence
L_{pat}	Average # of itemsets per base pattern
I_{seq}	Average # of items per itemset in the database
I_{pat}	Average # of items per itemset in base patterns

5.1 Synthetic Data

In this section, we describe the four class of synthetic datasets used for each of the four experiments : (1) random data, (2) patterned data, and patterned data with (3) varying degree of noise, and (4) varying number of outliers.

5.1.1 Random Data

Random data is generated by assuming independence between items both within and across itemsets. The probability of an item occurring is uniformly distributed. The number of distinct items and the number of sequences generated are determined by user set parmeters $\|\mathcal{I}\|$ and N_{seq} respective. The number of itemsets in a sequence and the number of items in an itemset follow a Poisson distribution with mean L_{seq} and I_{seq} respectively. The full user parameters are listed in Table 5.1

5.1.2 Patterned Data

For patterned data, we use the well known IBM synthetic data generator first introduced in [2]. Given several parameters (Table 5.2), the IBM data generator produces a patterned database and reports the base patterns used to generate it. Since it was first published in 1995, the IBM data generator has been used extensively as a performance benchmark in association rule mining and sequential pattern mining. However, to the best of our knowledge, no previous study has measured how well the various methods recover the known base patterns. In this dissertation, we develop some evaluation criteria to use in conjunction with the IBM data generator to measure the quality of the results.

The data is generated in two phases. First, it generates N_{pat} potentially frequent sequential patterns, called *base patterns*, according to user parameters L_{pat} and I_{pat}. Secondly, each sequence in the database is built by combining these base patterns until the size specified by user parameters L_{seq} and I_{seq} are met. Along with each base pattern, the data generator reports the expected frequency, $E(F_B)$, and the expected length (total number of items), $E(L_B)$, in the database for each base pattern. The $E(F_B)$ is given as a percentage of the size of the database and the $E(L_B)$ is given as a percentage of the number of items in the base pattern.

There are two steps involved in building the base patterns. First, the set of potentially frequent itemsets, Λ, are built by randomly selecting items from the distinct set of items in $\mathcal{I}$. The probability of an item occurring is exponentially distributed. The size of each itemset is randomly determined using a Poisson distribution with mean I_{pat}. The number of distinct items and the number of potentially frequent itemsets are determined by user set parameters $\|\mathcal{I}\|$ and $\|\Lambda\|$ respective.

Second, the base patterns are then built by selecting, corrupting, and concatenating itemsets selected from the set of potentially frequent itemsets. The selection and corruption is based on the $P(select)$ and $P(corrupt)$ randomly assign to each potentially frequent itemset. The selection probability is exponentially distributed then normalized to sum to 1. The corruption probability is normally distributed. Corrupting means randomly deleting items from the selected potentially frequent itemset. N_{pat} determines how many base patterns to construct, and L_{pat} determines the average number of itemsets in the base patterns. More precisely, the number of itemsets in a base pattern is randomly assigned from a Poisson distribution with mean L_{pat}. Base patterns built in this manner then become the potentially frequent sequential patterns.

The database is then built in a similar manner by selecting, corrupting, and combining the base patterns. As with potentially frequent itemsets, each base pattern is also assigned a separate $P(select)$ and $P(corrupt)$. Just as with potentially frequent itemsets, the $P(select)$ is exponentially distributed then normalized to sum to 1 and $P(corrupt)$ is normally distributed. The $P(select)$ is the likelihood a base pattern will appear in the database. Thus, it is equal to the expected frequency of a base pattern, $E(F_B)$, in the database. The $P(corrupt)$ is the likelihood of a selected base pattern to be corrupted before it is used to construct a database sequence. Corrupting base patterns is defined as randomly deleting items from the selected base pattern. Hence, $1 - P(corrupt)$ is roughly the expected length (total number of items), $E(L_B)$, of the base pattern in a sequence in the database.

$$\begin{aligned} E(F_B) &= P(select) \\ E(L_B) &\simeq 1 - P(corrupt) \end{aligned} \tag{5.1}$$

Each sequence is built by combining enough base patterns until the size required, deter-

Table 5.3: A DB sequence built from 3 base patterns

		(A, J)	**(B)**		**(A)**	**(E, H)**	**(C)**	**(L)**	
Base Patterns	**(D)**							**(F)**	~~**(M)**~~
	(G)	~~**(L)**~~		**(K)**	**(F, I)**			**(D)**	
DB Sequence	**(D, G)**	**(A, J)**	**(B)**	**(K)**	**(A, F, I)**	**(E, H)**	**(C)**	**(D, F, L)**	

mined by L_{seq} and I_{seq} for the database, is met. Hence, many sequences are generated using more than one base pattern. Base patterns are combined by interleaving them so that the order of the itemsets are maintained.

Table 5.3 demonstrates how 3 base patterns are combined to build a database sequence. The parameters for the database were: L_{seq}=10, I_{seq}=2.5 and the parameters of the base patterns were: $L_{pat} = 7$, I_{pat}=2. The itemsets that are crossed out were deleted in the corruption process.

In essence, sequential pattern mining is difficult because the data has confounding noise rather than random noise. The noise is introduced into each data sequence in the form of tiny bits of another base pattern. In section 5.1.3 we discuss how to add controlled level of random noise in addition to the confounding noise in the patterned data in order to test for the effects of noise.

Similarly, outliers may exist in the data in the form of very weak base patterns. The expected frequency of the base patterns have exponential distribution. Thus, the weakest base patterns can have expected frequency be so small that the base pattern occurs in only a handful of the database sequences. These are in practice outliers that occur rarely in the database. For example, when there are 100 base patterns, 11 base patterns have expected frequency less than 0.1% of which 1 base pattern has expected frequency less than 0.01%. Thus, even when N_{seq}=10000, the weakest base pattern would occur in less than 1 sequence (10,000*0.01%=1 seq). In section 5.1.4 we discuss how to add controlled level of strictly random sequences in addition to outliers in the form of very weak base patterns in the patterned data to test for effects of outliers.

Example

To better understand the properties of the synthetic data generated we look closely at a particular synthetic dataset. A common configuration of the synthetic dataset used in the experiments is given in Table 5.4. The configuration can be understood as follows.

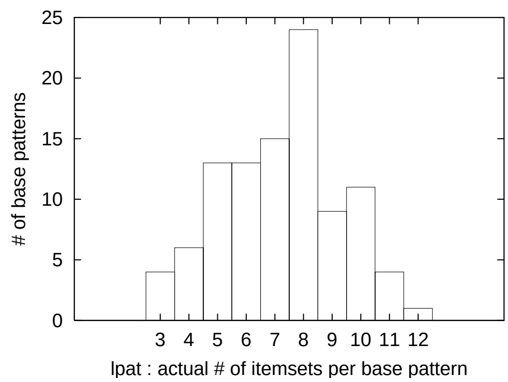
1. There are $\|I\|$=1,000 unique items in the synthetic database.

2. Using these $\|I\|$=1,000 unique items $\|\Lambda\|$=5,000 itemsets were generated at random. These are the potentially frequent itemsets used to construct the base patterns.

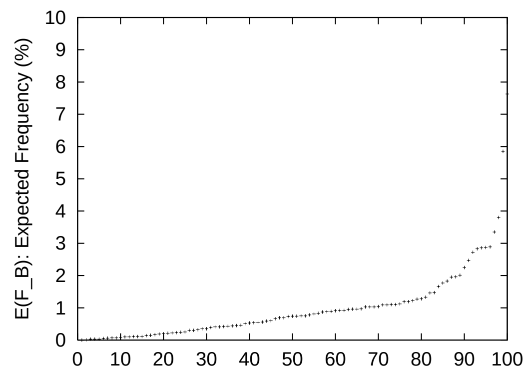
Table 5.4: A common configuration of the synthetic dataset used in the experiments

Notation	Meaning	Default value
$\|I\|$	# of items	$1,000$
$\|\Lambda\|$	# of potentially frequent itemsets	5,000
N_{seq}	# of data sequences	$10,000$
N_{pat}	# of base pattern sequences	100
L_{seq}	Avg. # of itemsets per data sequence	10
L_{pat}	Avg. # of itemsets per base pattern	7
I_{seq}	Avg. # of items per itemset in the database	2.5
I_{pat}	Avg. # of items per itemset in base patterns	2

Figure 5.1: Distributions from the synthetic database specified in Table 5.4



(a) Distribution of the actual l_{pat}



(b) Distribution of the $E(F_B)$

3. On average there are I_{pat}=2 items per itemset in these 5,000 potentially frequent itemsets.

4. N_{pat}=100 base patterns were randomly constructed using the 5,000 potentially frequent itemsets.

5. On average there are L_{pat}=7 itemsets per base pattern. The acutal distribution of the number of itemsets for each of the 100 base patterns, l_{pat}, is given in Figure 5.1(a). Remember that l_{pat} has a possion distribution with mean at L_{pat}. When L_{pat}=7, 10% of the patterns have between 3 or 4 itemsets in the base patterns. On the other hand 5% of the patterns have more than 10 itemsets per base pattern. The remaining 85% of base patterns have between 5 to 10 itemsets per pattern. Values of $L_{pat} < 7$ starts to introduce base patterns of less then 3 itemsets per pattern. Thus, L_{pat}=7 is the practical minimum value that will embed sequential patterns of more then 2 itemsets into the synthetic data.

6. N_{seq}=10,000 data sequences were constructed using the 100 base patterns.

7. The distribution of the expect frequencies, $E(F_B)$, of the 100 base patterns is given in

Figure 5.1(b). Of 100 base patterns, 11 have $E(F_B) < 0.1\%$ (0.1% * 10,000=10 seq). Of them, 1 base pattern has expected frequency less than 0.01% (0.01% * 10,000=1 seq). As discussed above these are the practical outliers that occur rarely in the database. On the other hand, there are 12 base pattern with $E(F_B) > 2\%$ (2% * 10,000=200 seqs). Of these the four largest $E(F_B)$ are 7.63%, 5.85%, 3.80%, and 3.35% respectively. The other 8 are all between 2% and 3% ($2\% < E(F_B) \leq 3\%$). The majority, 77 base patterns, have $E(F_B)$ between 0.1% and 2% (10 seq= $0.1\% < E(F_B) \leq 2\%$ =200 seqs).

8. The base patterns were combined so that on average there are L_{seq}=10 itemsets per data sequence and I_{seq}=2.5 items per itemset in a data sequence. Note that since L_{pat}=7 is the practical minimum for embedding sequential patterns into the synthetic data, L_{seq} should be greater than 7. Thus, in many of our experiments which require us to test on a range of L_{seq}, we vary L_{seq} from 10 to 50.

5.1.3 Patterned data with varying degree of noise

Noise occurs at the item level in sequential data. Therefore, to introduce varying degree of controlled noise into the IBM patterned data, we use a corruption probability α. Items in the patterned database are randomly changed into another item or deleted with probability α. This implies that $1 - \alpha$ is the probability of any item remaining the same. Hence, when $\alpha = 0$ no items are changed, and higher values of α imply a higher level of noise [64].

5.1.4 Patterned data with varying degree of outliers

Outliers are sequences that are unlike most other sequences in the data. That is there are very few sequences similar to the outlier sequence in the data. A randomly generated sequence, such as the sequences generated for the random data, can be such an outlier sequence. Thus, we introduce controlled level of outliers into the data by adding varying number of random sequences to the IBM patterned data. The random sequences are generated using the same parameters L_{seq}, I_{seq}, and $\|\mathcal{I}\|$ as those used to generate the patterned data. In the rest of the dissertation, random sequences added to patterned data is referred to as outliers.

5.2 Evaluation Criteria

The effectiveness of a sequential pattern mining method can be evaluated in terms of how well it finds the real underlying patterns in the data (the base patterns), and whether or not it generates any confounding information. However, the number of base patterns found or missed is not alone an accurate measure of how well the base patterns were detected because it can not take into account how much of a base pattern was detected (which items in the base pattern were detected) or how strong (frequent) the pattern is in the data. Instead, we

Table 5.5: Confusion Matrix

		predicted (Result Patterns Generated)	
		negative	postive
actual (Base Patterns Embedded)	negative	a (NA)	b (Extraneous Items)
	positive	c (Missed Items)	d (Pattern Items)

report a comprehensive view by measuring this information at two different levels; (1) at the item level and (2) at the sequence level.

5.2.1 Evaluation at the item level

At the item level, we adapt the ROC analysis to measure recoverability and precision. ROC analysis is commonly used to evaluate classification systems with known actual values [46]. The confusion matrix contains information about the actual and predicted classifications [46]. The confusion matrix for our evaluation is given in Table 5.5. The actual patterns are the base patterns that were embedded into the database. The predicted patterns are the result patterns generated from any sequential pattern mining algorithm. Then the true positive items, called *pattern items*, are those items in the result patterns that can be directly mapped back to a base pattern. The remaining items in the result patterns, the false positive items, are defined as *extraneous items*. These are items that do not come from the embedded patterns, but rather the algorithm falsely assumes to be part of the base patterns. There are a couple of reasons why this occurs. We discuss this in more detail in section 5.4. The items from the base pattern that were missed in the result patterns, the false negative items, are the *missed items*. In this context, there are no true negative items (cell a). Thus, we use only the cells b, c, and d in our evaluation.

Using the confusion matrix we measure two criteria at the item level. *Recoverability* measures how much of the base patterns have been found. *Precision* measures how precise are the predictions made about the base patterns. That is, precision measures how much confounding information (extraneous items) are included with the true pattern items.

Normally recall, $(\frac{d}{c+d})$, the true positive rate, is used to measure how much of the actual pattern has been found. However, recall is not accurate in this application because base patterns are only *potentially* frequent sequential patterns in the data. The actual occurrence of a base pattern in the data, which is controlled by $E(F_{B_i})$ and $E(L_{B_i})$, varies widely.

$E(F_{B_i})$ is exponentially distributed then normalized to sum to 1. Thus, some base patterns have tiny $E(F_{B_i})$. These base patterns do not exist in the data or occur very rarely. Recovering these patterns are not as crucial as recovering the more frequent base patterns.

$E(L_{B_i})$ controls how many items on average in the base patterns are injected into one occurrence of the base pattern in a sequence. This means that one sequence in the database is not expected to have all the items in the embedded base pattern. Remember that before

Table 5.6: Item counts in the result patterns

Notation	Meaning	Equation
N_{item}	total number of items	$\sum_{\text{rslt pat } \{P_j\}} \|P_j\|$
N_{patI}	total number of pattern items	$\sum_{\text{rslt pat } \{P_j\}} (\max_{\text{base pat } \{B_i\}} \|B_i \otimes P_j\|)$
N_{extraI}	total number of extraneous items	$\sum_{\text{rslt pat } \{P_j\}} (\|P_j\| - \max_{\text{base pat } \{B_i\}} \|B_i \otimes P_j\|)$

a base pattern is embedded into a data sequence we corrupt the base pattern by randomly deleting items from it. $E(L_{B_i})$ controls how many items on average are deleted in this process. Thus, we can not expect to find all the items in a base pattern in one database sequence.

Therefore, taking $E(F_{B_i})$ and $E(L_{B_i})$ into account, we designed a weighted measure, *recoverability*, which can more accurately evaluate how much of the base patterns have been recovered. Specifically, given (1) a set of base patterns, $\{B_i\}$, along with $E(F_{B_i})$ and $E(L_{B_i})$ for each base pattern, and (2) a set of result patterns, $\{P_j\}$, let each result pattern map back to the most similar base pattern. That is, the result pattern, P_j, is matched with the base pattern, B_i, if the longest common subsequence between P_j and B_i, denoted as $B_i \otimes P_j$, is the maximum over all base patterns. We indicate this matching by referring to the matched result patterns with two indices. $P_j(i)$ denotes that pattern P_j has been mapped to base pattern B_i.

Now let $P_{max}(i)$ be the max pattern for base pattern B_i. A *max pattern*, $P_{max}(i)$, is the result pattern that shares the most items with a base pattern, B_i, over all result patterns mapped to the same base pattern. Furthermore, at least half of the items in P_j has to come from the base pattern B_i. Thus, $\max_{\text{rslt pat } \{P_j(i)\}} \|B_i \otimes P_j\|$[1] is the most number of items recovered for a base pattern B_i. In essence, max patterns recovered the most information about a particular base pattern. Note that, there is either one or no max pattern for each base pattern. There could be no max pattern for a base pattern if none of the result patterns recovered enough of the items from the base pattern.

Since $E(L_{B_i}) \cdot \|B_i\|$ is the expected number of items (from the base pattern B_i) in one occurrence of B_i in a data sequence, $\frac{\max \|B_i \otimes P_j\|}{E(L_{B_i}) \cdot \|B_i\|}$ would be the fraction of the expected number of items found. $E(L_{B_i})$ is an expected value, thus sometimes the actual observed value, $\max_{\{P_j(i)\}} \|B_i \otimes P_j\|$ is greater than $E(L_{B_i}) \cdot \|B_i\|$. In such cases, we truncate the value of $\frac{\max \|B_i \otimes P_j\|}{E(L_{B_i}) \cdot \|B_i\|}$ to one so that recoverability stays between 0 and 1. Intuitively, if the recoverability of the mining is high, major portions of the base patterns have been found. Then, we define the recoverability as follows,

$$\text{Recoverability } \mathcal{R} = \sum_{\text{base pat } \{B_i\}} E(F_{B_i}) \cdot \min \begin{cases} 1 \\ \left(\frac{\max_{\text{rslt pat } \{P_j(i)\}} \|B_i \otimes P_j\|}{E(L_{B_i}) \cdot \|B_i\|} \right) \end{cases} \tag{5.2}$$

[1] $\|seq_i\|$=length of seq_i denotes the total number of items in seq_i

In ROC analysis, *precision*, $\frac{d}{b+d}$, is the proportion of the predicted positive items that were correct. It is a good measure of how much of the prediction is correct [46]. In sequential pattern mining, precision measures how much confounding information (extraneous items) are mixed in with the pattern items in the result pattern. Remember that when the result pattern P_j is mapped to base pattern B_i, the items in both the result pattern and the base pattern, $B_i \otimes P_j$, are defined as pattern items. Note that the result pattern P_j is mapped to base pattern B_i, when $\|B_i \otimes P_j\|$ is maximum over all base patterns. Thus, the number of pattern items for a result pattern, P_j, is $\max_{\{B_i\}} \|B_i \otimes P_j\|$. The remaining items in the result pattern, P_j, are the extraneous items. The different item counts in the result patterns are give in Table 5.6.

Denoted as $\mathcal{P}$, precision can be calculated using Table 5.6 as either

$$\text{Precision } \mathcal{P} = \frac{\sum_{\text{rslt pat } \{P_j\}} (\max_{\text{base pat } \{B_i\}} \|B_i \otimes P_j\|)}{\sum_{\text{rslt pat } \{P_j\}} \|P_j\|} \times 100\% \tag{5.3}$$

or

$$\text{Precision } \mathcal{P} = 1 - \frac{\sum_{\text{rslt pat } \{P_j\}} (\|P_j\| - \max_{\text{base pat } \{B_i\}} \|B_i \otimes P_j\|)}{\sum_{\text{rslt pat } \{P_j\}} \|P_j\|} \times 100\% \tag{5.4}$$

We tend to report using Equation 5.4 to indicate the exact number of extraneous items in the results.

5.2.2 Evaluation at the sequence level

At the sequence level, the three criteria that we measure are (1) the total number of result patterns, (2) the number of spurious patterns, and (3) the number of redundant patterns. To do so, we categorize the result patterns into spurious, redundant, or max patterns depending on the composition of pattern items and extraneous items. We do not report on the number of max patterns because it can be easily calculated by

$$N_{max} = N_{total} - N_{spur} - N_{redun}$$

Spurious patterns are those that were not embedded into the database, but what the algorithms incorrectly assumed to be sequential patterns in the data. In this evaluation, spurious patterns are defined as the result patterns that have more extraneous items than pattern items. As discussed in the previous section, max patterns are those that recover the most pattern items from each of the base patterns. The remaining sequential patterns are redundant patterns. These are result patterns, P_a, which match with a base pattern, B_i, but there exists another result pattern, P_{max}, that match with the same base pattern but better in the sense that $\|B_i \otimes P_{max}\| \geq \|B_i \otimes P_a\|$. Therefore these patterns are redundant data that clutter the results.

Table 5.7: Evaluation Criteria

criteria	Meaning	Level	Unit
$\mathcal{R}$	Recoverability: the degree of the base patterns detected (Eq. 5.2)	item	%
$\mathcal{P}$	Precision: 1-degree of extraneous items in the result patterns (Eq. 5.4)	item	%
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	seq	# of patterns

5.2.3 Units for the evaluation criteria

Recoverability and precision is reported as a percentage of the total number of items in the result ranging from 0% to 100%. In comparison, the spurious patterns and redundant patterns are reported as number of patterns. These measures can easily be changed to percentage of the total number of result patterns as needed.

We report the actual number of patterns because the number of spurious patterns can be tiny as a percentage of total number of result patterns. In fact, by definition we expect that there will be only a few spurious patterns if the algorithm is reasonably good. In such situations, we would want to be see exactly how few spurious patterns are in the result rather than its proportion in the result. For example, one of our experiments on the support model [2] had over 250,000 result patterns of which 6,648 (2.66%) were spurious patterns.

Unlike spurious patterns, redundant patterns are not incorrect patterns. Sometimes, they can even have additional information, such as suggesting slight variations of a strong pattern in the data. The most negative effect of redundant patterns is the confounding effect it can have on understanding the results when there are too many of them. Hence, the exact number of redundant patterns is directly related to the interference factor. For example, it is easy to glean some information and/or ignore 10 redundant patterns of 20 result patterns but not so easy to work through 50% of 250,000 patterns.

The five evaluation criteria is summarized in Table 5.7.

5.3 Example

Let Table 5.8 be the base patterns used to construct a sequence database. The expected frequence $E(F_{B_i})$, the expected length after corruption $E(L_{B_i})$, and the actual length $\|B_i\|$ of the base patterns are also given. In addition, let Table 5.9 be the result patterns returned by a sequential pattern mining algorithm. The actual length of the result patterns is given in column $\|P_j\|$. Then, the evaluation is done in the following steps:

1. *Identify total number of result patterns,* $N_{total} = \|\{P_j\}\|$. N_{total}=5 in this example.

[2] We did a comparison study between our method and the conventional support model reported in chapter 7.

Table 5.8: Base Patterns $\{B_i\}$: $N_{pat} = 3, L_{pat} = 7, I_{pat} = 2$

ID	Base Patterns B_i	$E(F_{B_i})$	$E(L_{B_i})$	$\|B_i\|$
B_1	(PR) (Q) (IST) (IJ) (U) (D) (NT) (I)	0.566	0.784	13
B_2	(FR) (M) (GK) (C) (B) (Y) (CL)	0.331	0.805	10
B_3	(D) (AV) (CZ) (HR) (B)	0.103	0.659	8

Table 5.9: Result Patterns $\{P_j\}$

ID	Result Pattern P_j	$\|P_j\|$
P_1	(PR) (Q) (I) (IJ) (IJU) (U) (D) (T)	12
P_2	(GKQ) (IT) (IJ) (D) (NT)	10
P_3	(P) (IS) (U) (DV) (NT)	8
P_4	(FR) (M) (C) (BU) (Y) (CL)	9
P_5	(F) (AV) (CL) (BSU) (I)	9

Table 5.10: Worksheet : $\mathcal{R} = 84\%, \mathcal{P} = 1 - \frac{12}{48} = 75\%, N_{total} = 5, N_{spur} = 1, N_{Redun} = 2$

ID	Base Pattern B_i	$\|B_i\|$	$E(F_{B_i})$	$E(L_{B_i})$	$E(L_{B_i}) \cdot \|B_i\|$
ID	Result Pattern P_j	$\|P_j\|$	N_{patI}	N_{extraI}	$\mathcal{R}(B_i)$
B_1	(PR) (Q) (IST) (IJ) (U) (D) (NT) (I)	13	0.566	0.784	10
P_1	(PR) (Q) (I) (IJ) (IJU) (U) (D) (T)	12	9	3	9/10=0.9
P_2	(GKQ) (IT) (IJ) (D) (NT)	10	8	2	Redundant
P_3	(P) (IS) (U) (DV) (NT)	8	7	1	Redundant
B_2	(FR) (M) (GK) (C) (B) (Y) (CL)	10	0.331	0.805	8
P_4	(FR) (M) (C) (BU) (Y) (CL)	9	8	1	8/8=1
B_3	(D) (AV) (CZ) (HR) (B)	8	0.103	0.659	5
P_5	(F) (AV) (CL) (BSU) (I)	9	4	5	Spurious

Table 5.11: Evaluation Results

criteria	Meaning	Results
$\mathcal{R}$	Recoverability: the degree of the base patterns detected	84%
$\mathcal{P}$	Precision: 1-degree of extraneous items in the result patterns	75%
N_{spur}	# of spurious patterns	1
N_{redun}	# of redundant patterns	2
N_{total}	total # of patterns returned	5

2. *Map result patterns to base patterns.* Each result pattern, P_j, is mapped to the best matching base pattern B_i such that $\|B_i \otimes P_j\|$ is maximized over all base patterns B_i in Table 5.10. For result pattern, P_5, $\|B_1 \otimes P_5\| = \|\{(\mathsf{U})(\mathsf{I})\}\| = 2, \|B_2 \otimes P_5\| = \|\{(\mathsf{F})(\mathsf{C})(\mathsf{B})\}\| = 3$, and $\|B_3 \otimes P_5\| = \|\{(\mathsf{AV})(\mathsf{C})(\mathsf{B})\}\| = 4$. Thus, result pattern P_5 is mapped to base pattern B_3.

3. *Count the number of pattern items and extraneous items for each result pattern.* For each result pattern in Table 5.10, the number of pattern items and extraneous items are given in the fourth and fifth column labeled N_{patI} and N_{extraI}. Result pattern P_1 has 9 pattern items it shares with B_1 ((PR)(Q)(I)(IJ)(U)(D)(T)) and 3(=12-9) extraneous items ((IJU)).

4. *Calculate precision, $\mathcal{P}$.* The total number of pattern items is 9+8+7+8+4=36. The total number of items in the result pattern is 12+10+8+9+9=48. Thus, the total number of extraneous items is 48-36=12.

$$\mathcal{P} = (1 - \frac{12}{48}) \times 100\% = 75\%$$

5. *Identify spurious patterns, N_{spur}.* If a result pattern, P_j, has more extraneous items than pattern items, it is classified as a spurious pattern. P_5 is a spurious pattern because $4 < 5$. Therefore, $N_{spur} = 1$.

6. *Identify max patterns, $P_{max}(i)$.* Of the remaining result patterns, for each base pattern, B_i, identify the max result pattern such that $\|B_i \otimes P_j\|$ is maximized over all result patterns $P_j(i)$ mapped to B_i. In Table 5.10, result patterns are sorted by $\|B_i \otimes P_j\|$ for each base pattern. P_1 and P_4 are max patterns for B_1 and B_2 respectively. B_3 does not have a max pattern.

7. *Identify redundant patterns, N_{redun}.* Any remaining result patterns are redundant patterns. P_2 and P_3 are redundant patterns for B_1 that confound the results. Hence, $N_{redun} = 2$.

8. *Calculate recoverability, $\mathcal{R}$.* For each max pattern, calculate recoverability with respect to B_i, $\mathcal{R}(B_i) = \frac{\|B_i \otimes P_{max}(i)\|}{E(L_{B_i}) \cdot \|B_i\|}$. Truncate $\mathcal{R}(B_i)$ to 1 if necessary. Weight and sum over all base patterns.

$$\begin{aligned}
\mathcal{R} &= E(F_{B_1}) \cdot \mathcal{R}(B_1) + E(F_{B_2}) \cdot \mathcal{R}(B_2) + E(F_{B_3}) \cdot \mathcal{R}(B_3) \\
&= 0.566 \cdot \tfrac{9}{10} + 0.331 \cdot \tfrac{8}{8} + 0.103 \cdot 0 \\
&= 0.84 = 84\%
\end{aligned}$$

5.4 A Closer Look at Extraneous Items

Extraneous items are those items that are part of the result pattern the algorithms found, but were not embedded into the database intentionally (false positive items). That is, they are not part of the mapped base pattern. There are a couple of reasons why an algorithm would falsely assume extraneous items to be part of the base patterns.

The most obvious reason would be that the data mining algorithm is incorrect. Algorithms can inadvertently inject items into the real embedded patterns. These artificially created items are incorrect. Our evaluation method would correctly report them as extraneous items.

A more likely related reason would be that the data mining algorithm is inaccurate. Different from being incorrect, in these cases the extraneous items do occur *regularly* in the database. However, this is a random occurrence because these items do not come from the base patterns we embedded into the database. Furthermore, it is rooted on the definition of

Table 5.12: Repeated items in a result pattern

ID	len	N_{patI}	N_{extraI}	Patterns
P_1	16	13	3	(G) (E) (A E F) (A E) (H) (A D) (B H) (C I) (B H)
B_1	13			(G) (E F) (A) (H) (A D) (B H) (C I) (B H)

regular that is either explicitly or implicitly specified within the model used to define patterns in the algorithm. That means that the definition of patterns used in the algorithm is not accurate enough to differentiate between random occurrences and real patterns in the data. In any case, these inaccurate items are also reported as extraneous items.

In reality, the most common extraneous items are repeated items. We use the most conservative definition of extraneous items. Thus, the current definition will classify repeated items as extraneous items. That is when an item in the base pattern is reported in the result pattern more than once, we consider only one of them as being a pattern item. All other items are classified as extraneous items. Table 5.12 is an illustration from one of our experiments. The E in the second and fourth, and A in the third itemset are all repeated items that were classified as extraneous items. All are lightly colored to indicate its relatively weak presence in the data. Clearly, the E comes from the second itemset in the base pattern, and the A comes from the third itemset in the base pattern. However, there is a much stronger presence of E in the third itemset and A in the fourth itemset of the result pattern. Therefore, the three repeated items were classified as extraneous items.

Another possibility is due to the limitations of the evaluation method. The current evaluation method maps each result pattern to only one base pattern. Thus, any items which do not come from the designated primary base pattern is reported as extraneous items even though they might come from another base pattern. However, statistically with enough data and not enough base patterns, a new underlying pattern can emerge. Recall that we build sequences in the database by combining different base patterns. Consequently, when enough sequences are generated by combining multiple base patterns in a similar way, it will produce a new underlying pattern that is a mix of the basic base patterns. This new mixed pattern will occur regularly in the database and a good algorithm should detect it.

This phenomena can be best depicted through an example from our experiment. A result pattern P_1 with 11 extraneous items of 22 items in total is given in Table 5.13. In this table, color does not represent item weights. The dark items in P_1 are pattern items, and the light items in P_1 are extraneous items. P_1 is mapped to B_1 because they share the most items $\|B_1 \otimes P_1\| = 11$. The 11 light colored items in the result pattern P_1, which do no come from B_1, are reported as the extraneous items. Clearly 10 of the 11 extraneous items are from B_2. There is only one real extraneous item in the third itemset, F (underlined). F is a repeated item from B_1. This result pattern suggests that there is a group of sequences that combine

Table 5.13: A new underlying trend emerging from 2 base patterns

ID	len	Patterns							
P_1	22	(F, J)	(A,F,H,N)	(*F*,K,Q)	(A,E,G,L,P)	(A,O)	(G,M,R,S)	(A,B)	
B_1	14	(F)	(F)	(K)	(A,E,G,L)		(G,M,R)	(B)	~~(C,D,O)~~
B_2	14	(J)	(A,H,N)	(Q)	(P)	(A,O)	(S)	(A)	~~(G,L) (I) (R)~~

the two base patterns, B_1 and B_2, in a similar manner to create a new underlying pattern P_1. Essentially, a new underlying pattern can emerge when two base patterns combine in a similar way frequently enough. Such phenomena arise in the IBM data when N_{seq}/N_{pat} is large.

When algorithms detect such mixed patterns, the current evaluation method will incorrectly report all items not belonging to the primary base pattern as extraneous items. Fortunately, such situation could be easily detected. If we were to pick out all the extraneous items from a particular result pattern, then build a separate sequence with them, it should become a subsequence of considerable length of another base pattern. That is the result pattern minus all items in the primary base pattern should produce a sequence very similar to another base pattern.

Nonetheless, incorporating this information into the evaluation upfront is quite difficult. Mapping each result pattern to more than one base pattern for evaluation introduces many new complications. Not the least of which is the proper criteria for mapping result patterns to base patterns. That is how much of a base pattern is required to be present in the result pattern for a mapping to occur. Obviously, one shared item between the result pattern and the base pattern is not enough. But how many is enough? This and other issues will be studied in future work for a more comprehensive evaluation method. For now, whenever there is a large number of extraneous items, we investigate how much of the extraneous items could be mapped back to a non-primary base pattern and include this information in the results.

Chapter 6

Results

In this chapter, we report an extensive set of empirical evaluations. We study in detail various aspects of ApproxMAP using the evaluation method discussed in chapter 5. All experiments were run on a Dell with Dual 2GHZ Intel Xeon processors emulating 4 logical processors. The computer runs Red Hat Linux and has 2GB of memory. The program only uses one CPU in all experiments.

Unless otherwise specified, we use the version of ApproxMAP which includes the improvement discussed in section 4.6.1 but not section 4.6.2. That is, ApproxMAP uses reduced precision of the proximity matrix with k-nearest neighbor clustering.

For all experiments, we assumed that the pattern consensus sequences were the only results returned by ApproxMAP. In this section, we often refer to the consensus sequence pattern as simply *consensus pattern*. Thus, the two main parameters that affect the result are k and θ. In addition, there is an advanced parameter $min\_DB\_strength$ that also affects the pattern consensus sequence. However, $min\_DB\_strength$ is not meant to be changed unless it is necessary for the application. That is, $min\_DB\_strength$ should be left at the default unless the application is specifically trying to find patterns that occur in less than 10 sequences. Thus, we assume that $min\_DB\_strength$ is kept constant at the default, 10 sequences, for all experiments.

Furthermore, only the consensus sequences with more than one itemset in the sequence were considered as result patterns. That is all clusters with null consensus patterns (called *null clusters*) or cluster with consensus patterns that have only one itemset (called *one-itemset clusters*) were dismissed from the final result patterns because these are obviously not meaningful sequential patterns.

To accurately depict the full results of ApproxMAP we include additional information about (1) the total number of clusters, (2) the number of clusters with null consensus patterns, and (3) the number of clusters with one itemset consensus patterns. Table 6.1 gives the notations used.

Table 6.1: Notations for additional measures used for ApproxMAP

Measures	Meaning
$\|C\|$	number of clusters
$\|C_{null}\|$	number of clusters with null consensus patterns (null clusters)
$\|C_{one\_itemset}\|$	number of clusters with one itemset consensus patterns (one itemset clusters)

6.1 Understanding ApproxMAP

Before we do a full evaluation of ApproxMAP using the method developed in chapter 5, we take an in depth look at some important aspects of ApproxMAP. In summary, we found that ApproxMAP was robust to the input parameter k and θ as well as the order of alignment.

6.1.1 Experiment 1.1: k in k-nearest neighbor clustering

Table 6.2: Parameters for the data generator in Experiment 1

Notation	Meaning	Default value
$\|I\|$	# of items	$1,000$
$\|\Lambda\|$	# of potentially frequent itemsets	5,000
N_{seq}	# of data sequences	$10,000$
N_{pat}	# of base pattern sequences	100
L_{seq}	Avg. # of itemsets per data sequence	20
L_{pat}	Avg. # of itemsets per base pattern	14
I_{seq}	Avg. # of items per itemset in the database	2.5
I_{pat}	Avg. # of items per itemset in base patterns	2

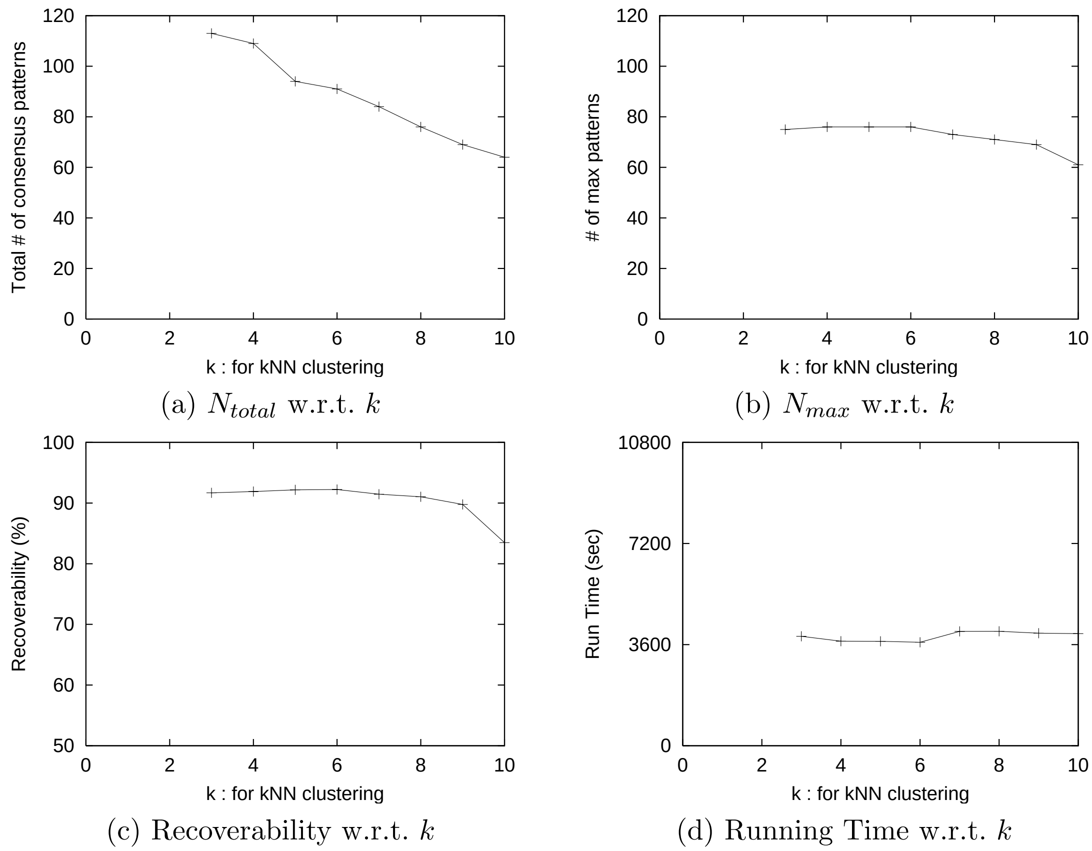
First, we studied the influence and sensitivity of the user input parameter k. We fix other settings and vary the value of k from 3 to 10, where k is the nearest neighbor parameter in the clustering step, for a patterned dataset. The parameters of the dataset are give in Table 6.2 and the results are shown in Table 6.3 and Figure 6.1. As analyzed before, a larger value of k produces less number of clusters, which leads to less number of patterns. Hence, as expected when k increases in Figure 6.1(a), the number of consensus patterns decreases.

However most of the reduction in the consensus patterns are redundant patterns for $k = 3..9$. That is the number of max patterns are fairly stable for $k = 3..9$ at around 75 patterns (Figure 6.1(b)). Thus, the reduction in the total number of consensus patterns returned does not have much effect on recoverability (Figure 6.1(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 6.1(b)). This causes loss of some weak base patterns and thus the recoverability decreases somewhat as shown in Figure 6.1(c). Figure 6.1(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.

Table 6.3: Results for k

k	Recoverability	Precision	N_{spur}	N_{redun}	N_{total}	N_{max}	Running Time(s)
3	91.69%	100.00%	0	38	113	75	3898
4	91.90%	100.00%	0	33	109	76	3721
5	92.17%	100.00%	0	18	94	76	3718
6	92.23%	100.00%	0	15	91	76	3685
7	91.45%	100.00%	0	11	84	73	4070
8	91.04%	100.00%	0	5	76	71	4073
9	89.77%	100.00%	0	0	69	69	4007
10	83.47%	100.00%	0	3	64	61	3991

Figure 6.1: Effects of k



(a) N_{total} w.r.t. k (b) N_{max} w.r.t. k

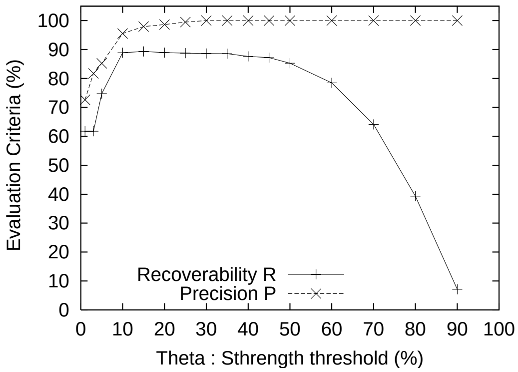
(c) Recoverability w.r.t. k (d) Running Time w.r.t. k

In terms of running time, Figure 6.1(d) indicates that the performance of ApproxMAP is not sensitive to parameter k. It is stable at a little over an hour for all $k = 3..10$.

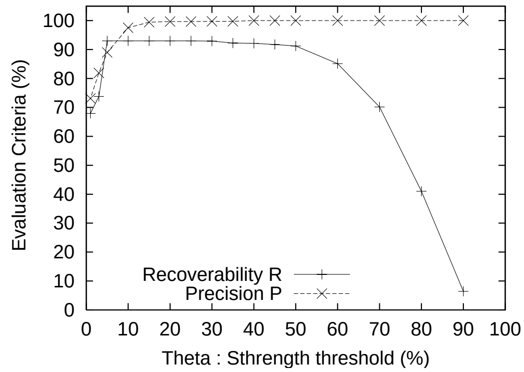
6.1.2 Experiment 1.2: The Strength Threshold

In ApproxMAP, we use two strength cutoff points, θ and δ, to filter out noise from weighted sequences. Here, we study the strength threshold to determine its properties empirically.

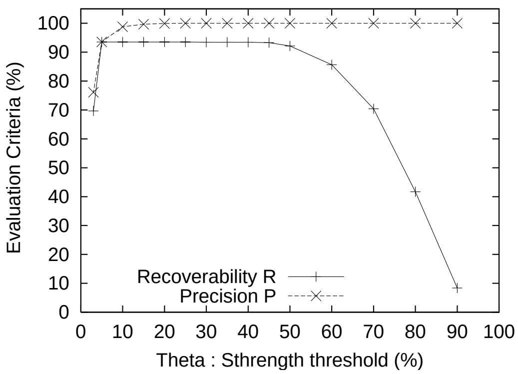
Figure 6.2: Effects of θ



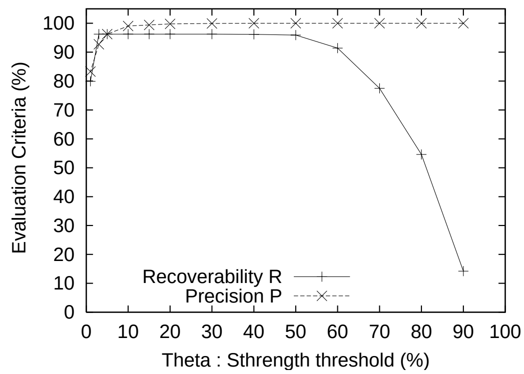
(a) $L_{seq} = 10$



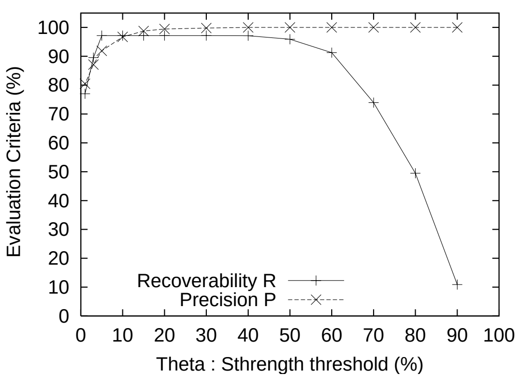
(b) $L_{seq} = 15$



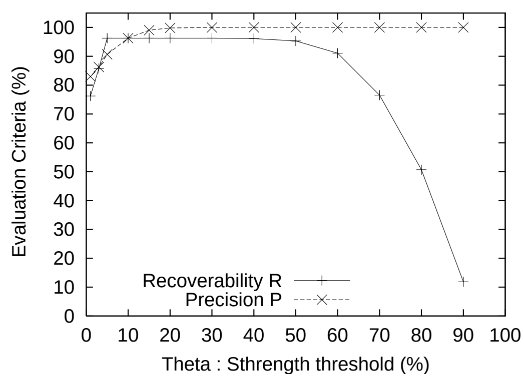
(c) $L_{seq} = 20$



(d) $L_{seq} = 30$



(e) $L_{seq} = 40$



(f) $L_{seq} = 50$

Since both cutoff points have the same property, we use θ as the representative strength cutoff parameter. We ran 6 experiments on 6 different patterned datasets. The patterned data was generated with the same parameters given in Table 6.2 except for L_{seq} and L_{pat}. We experimented on varying $L_{seq} = 10..50$ with $L_{pat} = 0.7 \cdot L_{seq}$.

We then studied the change in recoverability and precision as θ is changed for each dataset. Without a doubt, in Figure 6.2 the general trend is the same in all datasets.

In all datasets, as θ is decreased from 90%, recoverability increases quickly until it levels off at $\theta = 50\%$. We see that precision stays high at close to 100% until θ 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, we set the default value for pattern consensus sequence at 50%. In all our experiments, the default value of $\theta = 50\%$ is used unless otherwise specified.

On the other side, when θ is too low precision starts to drop. We further see that in conjunction with the drop in precision, there is a point at which recoverability drops again. This is because, when θ 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 dataset with $L_{seq} = 10$ this occurs at $\theta \leq 10\%$. In all other datasets, this occurs when $\theta \leq 5\%$.

The drop in precision starts to occurs when $\theta < 30\%$ for the dataset with $L_{seq} = 10$. In the datasets of longer sequences, the drop in precision starts near $\theta = 20\%$. This indicates that items with $strength \geq 30\%$ are probably items in the base patterns.

Moreover, in all datasets, when $\theta \leq 10\%$, we see a steep drop in precision. This indicates, that many extraneous items are picked up when $\theta \leq 10\%$. Since recoverability is close to 100% when $\theta = 10\%$, this means that most of the items with strength less than 10% are extraneous items. Hence, as a modest estimation, we set the default value for the variation consensus sequence at 20%. This modest estimate will allow ApproxMAP to detect almost all pattern items while picking up only a small number of extraneous items in the variation consensus sequence.

In summary, this experiment indicates that 20%-50% is in fact a good range for the strength threshold for a wide range of datasets.

6.1.3 Experiment 1.3: The Order in Multiple Alignment

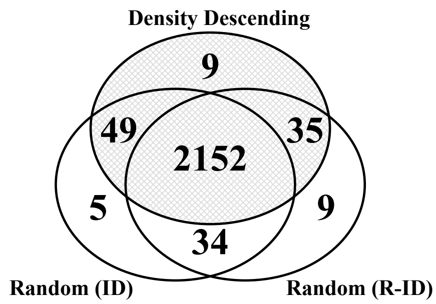
Now, we study the sensitivity of the multiple alignment results to the order of sequences in the alignment. We compare the mining results using the density-descending order, density-ascending order, and two random orders (sequence-id ascending and descending order) for the same dataset specified in Table 6.2. As expected, although the exact alignment changes slightly depending on the orders, it has very limited effect on the consensus patterns.

The result shows that (Table 6.4), 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%.

Table 6.4: 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 6.3: Comparison of pattern items found for different ordering



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 we tried gave comparable number of pattern items as the density descending order.

Figure 6.3 gives a detailed comparison of the pattern items detected by the two random orders and the density descending order. 2152 pattern items detected were identical in the three orders. 9 pattern items were detected by only the density descending order. 2201 (or 2187) pattern items were detected by both the density descending order and a random order. Essentially, a random order detected about 98% (49/2245=2%=35/2245) 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.

6.1.4 Experiment 1.4: Reduced Precision of the Proximity Matrix

As discussed in section 4.5, the straight forward ApproxMAP algorithm has time complexity $O(N_{seq}^2 \cdot L_{seq}^2 \cdot I_{seq})$. It can be optimized with respect to $O(L_{seq}^2)$ by calculating the proximity matrix used for clustering to only the needed precision. This was discussed in section 4.6.1. Here we study the speedup gained empirically. Figure 6.4 shows the speedup gained by using the optimization with respect to L_{seq} in comparison to the vanilla algorithm. We see that such optimization can reduce the running time to almost linear with respect to L_{seq}.

To investigate further the performance of the optimization, we looked at the actual number of cell calculations saved by the optimization. That is, with the optimization, the modified

Figure 6.4: Running Time w.r.t. L_{seq}

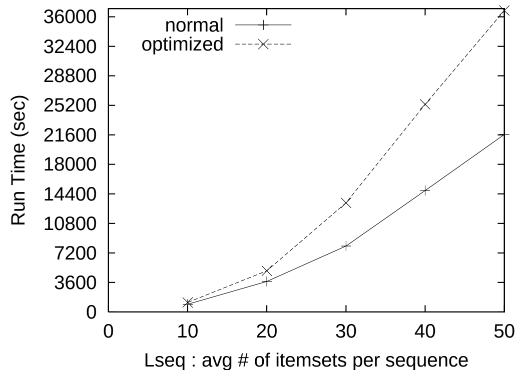
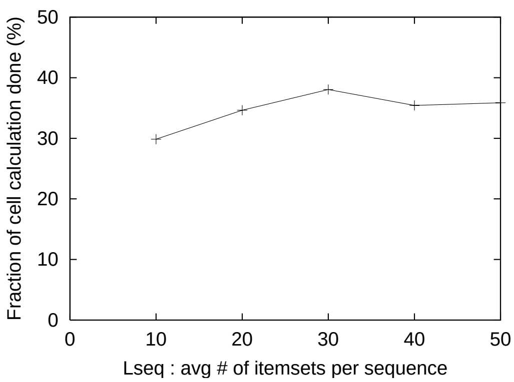
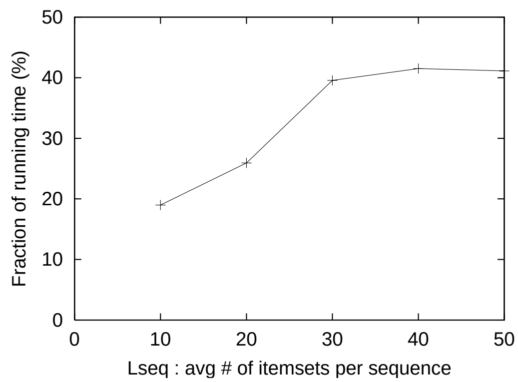


Figure 6.5: Fraction of calculation and running time due to optimization



(a) Fraction of cells calculation



(b) Fraction of running time

proximity matrix has mostly values of ∞ because $k << N$. For those $dist(seq_i, seq_j) = \infty$, we looked at the dynamic programming calculation for $dist(seq_i, seq_j)$ to see how many cells in the recurrence table could be skipped. To understand the savings in time, we report the following in Figure 6.5(a).

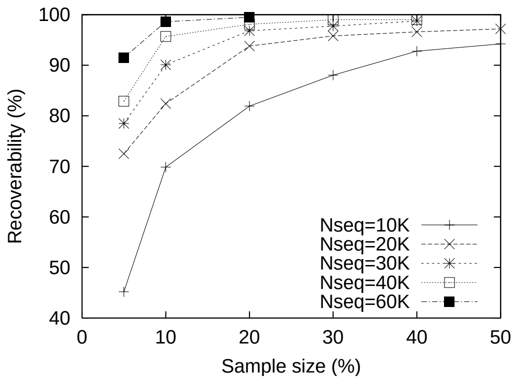
$$\frac{\sum \text{the number of cells in the recurrence table skipped}}{\sum \text{the total number of cells in the recurrence table}} \cdot 100\%$$

We see that when $10 \leq L_{seq} \leq 30$, as L_{seq} gets larger 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%.

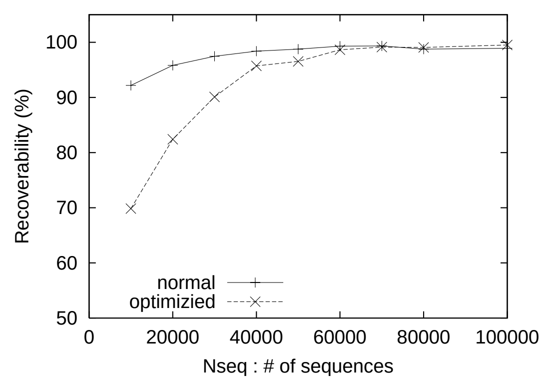
This is directly reflected in the savings in running time in Figure 6.5(b). Figure 6.5(b) reports the reduction in running time due to optimization as a proportion of the original running time. The proportion of savings in run time increases until $L_{seq} = 30$. At $L_{seq} = 30$ it levels off at around 40%. Thus, we expect that when $L_{seq} \geq 30$, the optimization will give a 40% reduction in running time. This is a substantial speedup without any loss in the accuracy of the results.

6.1.5 Experiment 1.5: Sample Based Iterative Clustering

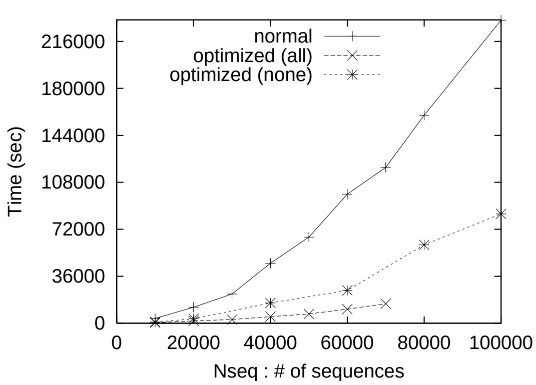
Figure 6.6: Sample based iterative clustering



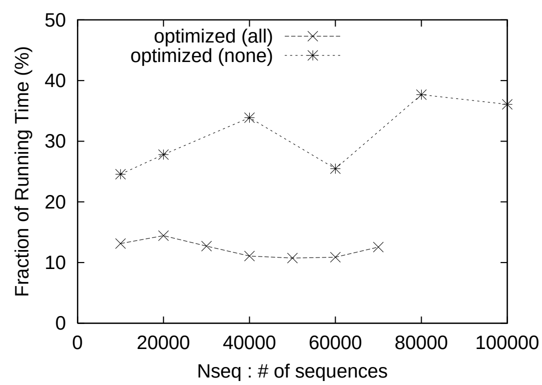
(a) Recoverability w.r.t. sample size



(b) Recoverability w.r.t. N_{seq} (sample=10%)



(a) Running time w.r.t. N_{seq}



(b) Fraction of running time w.r.t. N_{seq}

In section 4.6.2 we discussed how to speedup the clustering step by using a sample based iterative partitioning method. Such change can optimize the time complexity with respect to $O(N_{seq}^2)$ at the cost of some reduction in accuracy and larger memory requirements. Obviously, the larger the sample size the better the accuracy with slightly less gain in running time. In this section, we study the tradeoff empirically to determine the appropriate sample size. The experiments also suggests when such optimizations should be used.

Figure 6.6(a) presents recoverability with respect to sample size for a wide range of datasets. All runs use the default $k' = 3$. We see that when $N_{seq} \geq 40,000$, recoverability levels off at 10% sample size. When $N_{seq} < 40,000$, recoverability levels off at a larger sample size. However, when $N_{seq} = 40,000$ it takes less then 13 hours even without the optimization. Thus, the experiment suggests that the optimization should be used for datasets when $N_{seq} \geq 40,000$ with sample size 10%. For datasets with $N_{seq} < 40,000$, as seen in Figure 6.6(b), sample size 10% is too small. Thus, a larger sample size should be used if the normal algorithm is not fast enough for the application. The experiments indicate that the sample size should have at least 4000 sequences to get comparable results. As expected, in the smaller datasets with $N_{seq} < 40,000$, the running time is fast even when using a larger sample size. When $N_{seq} = 30,000$ and sample size=20% the running time was only 90 minutes.

Figure 6.6(b) and (c) show the gain in running time and the loss in recoverability with respect to N_{seq} with the optimization (sample size=10%, $k' = 3$). *optimized (all)* is a simple hash table implementation with all proximity values stored and *optimized (none)* is the implementation with none of the proximity values stored. The results clearly show that the optimization can speedup time considerably at the cost of negligible reduction in accuracy. Figure 6.6(d) show that the optimization can reduce running time to roughly 10%-40% depending on the given conditions.

6.2 Experiment 2: Effectiveness of ApproxMAP

In this section, we study the effectiveness of ApproxMAP using the full evaluation method discussed in chapter 5 on some manageable sized datasets. We ran many experiments with various synthetic data sets. The trend is clear and consistent. The evaluation results reveal that ApproxMAP returns a succinct but accurate summary of the base patterns with few redundant or spurious patterns. It is also robust to both noise and outliers in the data.

6.2.1 Experiment 2.1: Spurious patterns in random data

In this section, we study empirically under what condition suprious patterns are generated from completely random data ($\|\mathcal{I}\| = 100, N_{seq} = 1000, L_{seq} = 10, I_{seq} = 2.5$). For random data, because there are no base patterns embedded in the data, evaluation criteria recoverability, precision, and number of redundant patterns do not apply. The only important evaluation measure is the number of spurious patterns, N_{spur}, generated by the algorithm. We include the total the number of result patterns returned, N_{total}, for completeness. Since there are no base patterns in the data $N_{total} = N_{spur}$.

Recall that there are two parameters, k and θ in ApproxMAP. We first study the cutoff parameter θ. To determine the threshold at which spurious patterns will be generated, T_{spur}, we ran 8 experiments varying θ for a particular k. Each individual experiment has a constant k from 3 to 10.

Here we first report the full results of the experiment with the default value $k = 5$. When $k = 5$, there are 90 clusters of which only 31 clusters had 10 or more sequences. That means that the remaining 59 clusters will not return a consensus pattern no matter how low the cutoff is set because $min\_DB\_strength = 10$ sequences. Recall that when generating pattern consensus sequences, the greater of the two cutoff, θ or $min\_DB\_strength$ is used. Therefore, the following theorem shows that the lowest value of θ that can introduce the most spurious

Table 6.5: Results from random data ($k = 5$)

θ	N_{total}	N_{spur}	$\|C\|$	$\|C_{null}\|$	$\|C_{one\_itemset}\|$
50%	0	0	90	90	0
40%	0	0	90	89	1
30%	0	0	90	89	1
20%	0	0	90	86	4
16%	1	1	90	85	4

Table 6.6: Full results from random data ($k = 5$)

Cluster ID	Cluster size	$22\% \geq \theta \geq 19\%$	$18\% \geq \theta \geq 17\%$	$16\% \geq \theta \geq 0$
$Cluster_1$	42	⟨(A)⟩	⟨(A)⟩	⟨(A)⟩
$Cluster_2$	61	⟨(B)⟩	⟨(B)(E)⟩	⟨(B)(E)(B)⟩
$Cluster_3$	22	⟨(C)⟩	⟨(C)⟩	⟨(C)⟩
$Cluster_4$	50	⟨(D)⟩	⟨(D)⟩	⟨(D)⟩
$Cluster_5$	60			⟨(F G)⟩
$\|C_{one\_itemset}\|$		4	3	4
N_{spur}		0	1	1
$\|C_{null}\|$		86	86	85
$\|C\|$		90	90	90

patterns is 16%.

THEOREM 4. Given a weighed sequence, $wseq = \langle WX_1 : v_1, \ldots, WX_l : v_l \rangle : n$, when $min\_DB\_strength$ is kept constant, the pattern consensus sequences generated by ApproxMAP stay constant regardless of θ for $0 \leq \theta \leq \frac{min\_DB\_strength}{n}$.

Proof: According to Algorithm 2, when generating pattern consensus sequences the actual cutoff applied is the greater of the two specified cutoffs, θ or $\frac{min\_DB\_strength}{n}$. Thus, the cutoff applied is $\frac{min\_DB\_strength}{n}$ for $0 \leq \theta \leq \frac{min\_DB\_strength}{n}$, regardless of θ.

By Theorem 4, when θ is lowered enough such that it is smaller than $\frac{min\_DB\_strength}{cluster\_size}$ for all clusters, there is no change in the results regardless of θ. That is the result is constant for $0 \leq \theta \leq \frac{min\_DB\_strength}{size(C_{max})}$ where $size(C_{max})$ is the number of sequences in the biggest cluster.

In this experiment the biggest cluster had 61 sequences. Thus, all results are the same for $0 \leq \theta = 16\%$ ($\theta \leq \frac{min\_DB\_strength}{61} = \frac{10}{61} = 16.4\%$). Hence the smallest possible effective value of θ is 16%.

The results are given for $16\% \leq \theta \leq 50\%$ in Table 6.5. Noting that the first spurious pattern occurs when $\theta = 16\%$ we investigated the region $16\% \leq \theta < 20\%$ in more detail. The results for $0\% \leq \theta \leq 22\%$ are given in Table 6.6. It includes the actual consensus sequences generated for all non null clusters (the first five rows). The next two rows summarize how many of them are one itemset clusters and how many are spurious patterns. It is then followed by the number of null clusters in the result. These three add to the total number of clusters given next.

Table 6.7: Results from random data ($\theta = 50\%$)

k	N_{total}	N_{spur}	$\|C\|$	$\|C_{null}\|$	$\|C_{one\_itemset}\|$
3	0	0	134	134	0
4	0	0	103	103	0
5	0	0	90	90	0
6	0	0	86	84	2
7	0	0	57	56	1
8	0	0	61	61	0
9	0	0	59	59	0
10	0	0	56	55	1

Table 6.8: Results from random data at $\theta = T_{spur}$

k	T_{spur}	N_{total}	N_{spur}	$\|C\|$	$\|C_{null}\|$	$\|C_{one\_itemset}\|$
3	26%	1	1	134	132	1
4	19%	1	1	103	99	3
5	18%	1	1	90	86	3
6	27%	1	1	86	81	4
7	18%	1	1	57	47	9
8	23%	2	2	61	55	4
9	27%	1	1	59	54	4
10	26%	1	1	56	53	2

We see that up to $\theta = 19\%$ there are no spurious patterns returned and only four one-itemset consensus sequences returned. At $\theta = 18\%$ the first spurious pattern is generated when an additional itemset (E) is introduced in the consensus sequence from $Cluster_2$. Thus, the point at which the first spurious pattern occurs, T_{spur}, is 18% for $k = 5$. Then at $\theta = 16\%$ two more itemsets are introduced. First, one more itemset, (B), is introduced again to the consensus sequence from $Cluster_2$. Second $Cluster_5$, a null cluster up to this point, now has an itemset in its consensus sequence. The new consensus sequence $\langle$(F G)$\rangle$ is a one-itemset consensus sequence. By Theorem 4, for all $\theta < 16\%$, the result is the same as that when $\theta = 16\%$

In summary, when $k = 5$, there is no spurious pattern generated for $\theta > 18\%$ and only one spurious pattern for $\theta \leq 18\%$. The spurious pattern is $\langle$(B)(E)$\rangle$ for $17\% \leq \theta \leq 18\%$ and $\langle$(B)(E)(B)$\rangle$ for $0\% \leq \theta \leq 16\%$.

The summary of all 8 experiments varying k from 3 to 10 is given in Tables 6.7 and 6.8. Table 6.7 give the results for different k while keeping θ at the default value 50%. All experiments had no clusters with a meaningful (more than one itemset) consensus sequence. In fact, almost all clusters resulted in null consensus sequences. Five of eight experiments had all clusters with null consensus sequences. The other three experiments, k equal to 6, 7 and 10, gave only 2, 1, and 1 cluster with a one-itemset consensus sequence respectively. All of these one-itemset clusters, were small and had only one random item aligned. The exact cluster sizes that had a random item align to produce a one-itemset consensus sequence were 30, 31, 20 and 19 sequences. All other clusters could not align any items at all. These good results are not surprising when we look at each experiment in detail. In the eight experiments, the cluster sizes ranged from 1 to 153 sequences with many being tiny. In total, 63% had less than 10 sequences. These clusters could not generate any consensus sequences. This is discussed more later.

In Table 6.8, we report the threshold, T_{spur}, along with the other evaluation measures for each k. The threshold at which the first spurious pattern occurs ranges from 18% to 27%.

Clearly, ApproxMAP generates no spurious patterns from the random data set ($\|\mathcal{I}\| = 100, N_{seq} = 1000, L_{seq} = 10, I_{seq} = 2.5$) for a wide range of k ($3 \geq k \geq 10$) when $\theta > 27\%$. This is in line with our study of θ in section 6.1.2 that it is very unlikely that extraneous items appear in the consensus pattern when $\theta \geq 30\%$

Discussion : ApproxMAP and random data

ApproxMAP handles random data very well. Although the algorithm generated many clusters (all experiments generated from 56 to 134 clusters), when $\theta > 27\%$ all the clusters produced pattern consensus sequences with either 0 or 1 itemset which were easily dismissed.

There are two reasons why there is no pattern consensus sequence for a cluster. First, if the cluster is tiny it is not likely to have much item weights greater than $min\_DB\_strength$. Recall that the default value of $min\_DB\_strength$ is 10 sequences. Thus, clusters with less than 10 sequences had no pattern consensus sequence.

Second, not enough sequences in the cluster could be aligned to generate any meaningful consensus itemsets. That is in all our experiments at most one itemset could be aligned over all the itemsets. A one itemset sequence is clearly, not a meaningful sequential pattern. This is not surprising since the probability of two long sequences being similar by chance is quite small. Thus, it is very unlikely that enough sequences will align to produce patterns simply by chance.

6.2.2 Experiment 2.2: Baseline study of patterned data

The full parameters of the patterned database is given in Table 6.9. We optimized the parameters k and θ by running a set of experiments.

The first step was to find a reasonable θ that would work for a range of k. So for the first experiment, we varied k from 3 to 10 with θ set at the default value of 50%. The results, given in Table 6.10, showed that recoverability was not high enough in the region where the number of redundant patterns were small ($N_{redun} < 5$ which would be $k \geq 5$).

Therefore, we ran a second experiment with a lower $\theta = 30\%$ while varying k. As seen in the results given in Table 6.11 in general recoverability was more acceptable with good precision. So we used $\theta = 30\%$ to determine the optimal k.

Table 6.9: Parameters for the data generator for patterned data

Notation	Meaning	Default value
$\|I\|$	# of items	100
$\|\Lambda\|$	# of potentially frequent itemsets	500
N_{seq}	# of data sequences	1000
N_{pat}	# of base pattern sequences	10
L_{seq}	Avg. # of itemsets per data sequence	10
L_{pat}	Avg. # of itemsets per base pattern	7
I_{seq}	Avg. # of items per itemset in the database	2.5
I_{pat}	Avg. # of items per itemset in base patterns	2

Table 6.10: Results from patterned data ($\theta = 50\%$)

k	recoverability	Precision	N_{total}	N_{spur}	N_{redun}
3	91.85%	1/169=99.41%	15	0	7
4	88.75%	1/143=99.30%	13	0	6
5	88.69%	0/128=100.00%	11	0	4
6	87.45%	0/96=100.00%	8	0	1
7	78.18%	0/91=100.00%	8	0	2
8	82.86%	0/85=100.00%	7	0	1
9	81.09%	0/84=100.00%	7	0	1
10	64.40%	0/71=100.00%	6	0	1

As expected, as k increase, more sequences are clumped together to form less clusters. This is reflected in the reduction of N_{total}. Initially the clusters merged in this process are those with similar sequences built from the same base pattern. This can be seen for $k = 3..6$ where N_{redun} decreases from 7 to 1 and little change occur in recoverability from 92.36% to 91.16%. However, when k is increased beyond 6, small clusters (sequences built from less frequent base patterns) are merged together and we start to loss the less frequent patterns resulting in decreased recoverability. For $k = 6..10$, we see this phenomena where recoverability is decreased from 91.16% to 70.76%. Figure 6.7(a) depicts the drop in recoverability at $k = 6$. Figure 6.7(b) illustrates that the number of redundant patterns level off at the same point ($k = 6$). Hence, $k = 6$ is the optimal resolution for clustering this dataset.

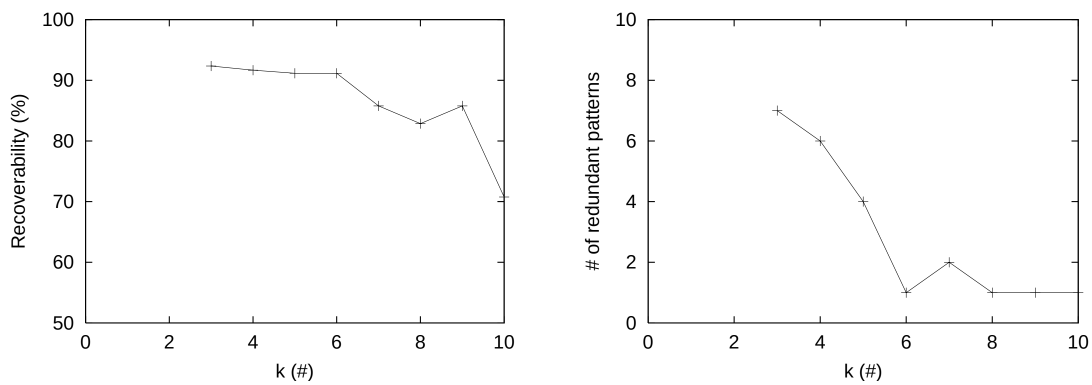
Now, we wanted to optimize θ for the optimal $k = 6$. Thus, we ran an experiment with $k = 6$ and varied θ from 20% to 50% (Table 6.12). For the optimal value, depending on the application, a user would choose either (1) the smallest θ with no extraneous items ($\theta = 40\%$) or (2) the largest θ with good recoverability and precision even though this point could include some extraneous items. We choose to go with the later option because the definition for extraneous items is the most conservative possible. That is, not all extraneous items are truly extraneous. See section 5.4 for details.

For this dataset, we decided that the precision=84.8% for $\theta = 20\%$ was too low. The region $25\% \geq \theta \geq 35\%$ had both good recoverability and precision. Since $\theta = 25\%$ had the

Table 6.11: Results from patterned data ($\theta = 30\%$)

k	recoverability	Precision	N_{total}	N_{spur}	N_{redun}
3	92.36%	1-5/179=97.21%	15	0	7
4	91.66%	1-2/153=98.69%	13	0	6
5	91.16%	1-4/136=97.06%	11	0	4
6	**91.16%**	**1-3/106=97.17%**	**8**	**0**	**1**
7	85.77%	1-1/100=99.00%	8	0	2
8	82.86%	1-4/90=95.56%	7	0	1
9	85.77%	1-0/90=100.00%	7	0	1
10	70.76%	1-4/82=95.12%	6	0	1

Figure 6.7: Effect of k, the nearest neighbor parameter.



(a) Recoverability w.r.t. k (b) Number of redundant patterns w.r.t. k

same number of pattern items found with $\theta = 30\%$ but two more extraneous items, $\theta = 25\%$ was dropped from consideration.

Now, let us compare the results from $\theta = 30\%$ and $\theta = 35\%$. Although the recoverability are the same, the actual number of pattern items found are different. The number of pattern items found are 103 and 100 respectively. There are two circumstances in which there is no change in recoverability even though more pattern items are found.

First, an increase in the pattern items for a non-max pattern has no affect on recoverability unless the non-max pattern becomes longer than the original max pattern because only the pattern items found in the max pattern is used to calculate recoverability.

Second, even if the additional pattern items found are from the max pattern, if $\max_{\{P_j(i)\}} \|B_i \otimes P_j\|$ is already greater than $E(L_{B_i}) \cdot \|B_i\|$, the recoverability does not change. This is because when the expected length of the base pattern, $E(L_B) \cdot \|B_i\|$ is smaller than the observed value $\max_{\{P_j(i)\}} \|B_i \otimes P_j\|$, we truncate $\frac{\max \|B \otimes P\|}{E(L_{B_i}) \cdot \|B_i\|}$ to 1.

To be precise, the optimal point is the one where the most pattern items are found with the least possible extraneous items. Thus, after debating between $\theta = 35\%$ and $\theta = 30\%$, we decided that $\theta = 30\%$ was the optimal point [1].

[1] The difference between the results produced by $\theta = 35\%$ and $\theta = 30\%$ are only 4(=106-102) items. One(=3-2) of them is an extraneous item and the other three are pattern items. All the pattern consensus

Table 6.12: Results from patterned data ($k = 6$)

θ	Recoverability	N_{item}	N_{patI}	N_{extraI}	Precision	N_{total}	N_{spur}	N_{redun}
20%	93.10%	125	106	19	84.80%	8	0	1
25%	91.16%	108	103	5	95.37%	8	0	1
30%	**91.16%**	**106**	**103**	**3**	**97.17%**	**8**	**0**	**1**
35%	91.16%	102	100	2	98.04%	8	0	1
40%	88.69%	98	98	0	100.00%	8	0	1
45%	88.69%	97	97	0	100.00%	8	0	1
50%	87.45%	96	96	0	100.00%	8	0	1

Table 6.13: Parameters for ApproxMAP

k	# of neighbor sequences	6
θ	the strength cutoff for pattern consensus sequences	28%..30%
δ	the strength cutoff for variation consensus sequences	20%
$min\_DB\_strength$	the optional parameter for pattern consensus sequences	10 sequences
$max\_DB\_strength$	the optional parameter for variation consensus sequences	10%

By investigating the region $25\% < \theta < 35\%$ in more detail we found that the consensus sequences are exactly the same in the region $28\% \geq \theta \geq 30\%$. That is the optimal results can be obtained with $k = 6$ and $28\% \geq \theta \geq 30\%$ for this dataset.

Now let us take a closer look at the mining result from the optimal run ($k = 6, \theta = 30\%$). Under such settings, ApproxMAP finds 8 pattern consensus sequences. The patterns and their variations are given along with the base pattern used to build the dataset in Table 6.14. For variation consensus sequences, we use the default settings as given in Table 6.13. Note that we have 9 clusters but only 8 consensus patterns because there is one null cluster ($PatConSeq_9$).

As shown in Table 6.14, each of the 8 pattern consensus sequences match a base pattern well. There were no spurious patterns. The pattern consensus sequences do not cover the three weakest base patterns ($BaseP_8, BaseP_9$, and $BaseP_{10}$). The recoverability is still quite good at 91.16%. In general, the pattern consensus sequences recover major parts of the base patterns with high expected frequency in the database.

The pattern consensus sequences cannot recover the complete base patterns (all items in the base patterns) because, during the data generation, only parts of base patterns are embedded into a sequence. Hence, some items in a particular base pattern may have much lower frequency than the other items in the same base pattern in the data. When we explored the weighted sequences by trying different cutoffs for θ, we saw that the full weighted sequences

sequences for $\theta = 30\%$ is given in Table 6.14. As θ changes from 35% to 30% an additional extraneous item, 58 in the fourth itemset in $PatConSeq_8$, is picked up. The missed pattern items when $\theta = 35\%$ are (1) the last item, 51, in $PatConSeq_3$, (2) and the two items in the last itemset, (2 74), in $PatConSeq_8$. The first does not affect recoverability because $PatConSeq_3$ is a redundant pattern. The second does not affect recoverability because the expected length of $PatConSeq_8$ is $0.6 * 13 = 7.8$. Thus, finding 11 items of the 13 items in the base pattern is the same as finding all 13 items when calculating recoverability.

Table 6.14: Consensus sequences and the Base Patterns for $k = 6$ and $\theta = 30\%$

$BaseP_i$ ($E(F_B)$:$E(L_B)$)	‖ P ‖	**Pattern <100: 85: 70: 50: 35: 20>**
$BaseP_1$ (0.21:0.66)	14	<(15 16 17 66)(15)(58 99)(2 74)(31 76)(66)(62)(93) >
PatConSeq$_1$	13	**<(15 16 17 66)(15)(58 99)(2 74)(31 76)(66)(62)>**
VarConSeq$_1$	18	**<(15 16 17 66)(15 22)(58 99)(2 74)(24 31 76)(24 66)(50 62)(93)>**
$BaseP_2$ (0.161:0.83)	22	<(22 50 66)(16)(29 99)(94)(45 67)(12 28 36)(50)(96)(51)(66)(2 22 58)(63 74 99) >
PatConSeq$_2$	19	**<(22 50 66)(16)(29 99)(94)(45 67)(12 28 36)(50)(96)(51)(66)(2 22 58)>**
VarConSeq$_2$	25	**<(22 50 66)(16)(29 99)(22 58 94)(2 45 58 67)(12 28 36)(2 50)(24 96)(51)(66)(2 22 58)>**
PatConSeq$_3$	15	**<(22 50 66)(16)(29 99)(94)(45 67)(12 28 36)(50)(96)(51)>**
VarConSeq$_3$	15	**<(22 50 66)(16)(29 99)(94)(45 67)(12 28 36)(50)(96)(51)>**
$BaseP_3$ (0.141:0.82)	14	< (22)(22)(58)(2 16 24 63)(24 65 93)(6)(11 15 74) >
PatConSeq$_4$	11	**<(22)(22)(58)(2 16 24 63)(24 65 93)(6)>**
VarConSeq$_4$	13	**<(22)(22)(22)(58)(2 16 24 63)(2 24 65 93)(6 50)>**
$BaseP_4$ (0.131:0.90)	15	<(31 76)(58 66)(16 22 30)(16)(50 62 66)(2 16 24 63) >
PatConSeq$_5$	11	**<(31 76)(58 66)(16 22 30)(16)(50 62 66)>**
VarConSeq$_5$	11	**<(31 76)(58 66)(16 22 30)(16)(50 62 66)(16 24)>**
$BaseP_5$ (0.123:0.81)	14	<(43)(2 28 73)(96)(95)(2 74)(5)(2)(24 63)(20)(93) >
PatConSeq$_6$	13	**<(43)(2 28 73)(96)(95)(2 74)(5)(2)(24 63)(20)>**
VarConSeq$_6$	16	**<(22 43)(2 28 73)(58 96)(95)(2 74)(5)(2 66)(24 63)(20)>**
$BaseP_6$ (0.121:0.77)	9	<(63)(16)(2 22)(24)(22 50 66)(50) >
PatConSeq$_7$	8	**<(63)(16)(2 22)(24)(22 50 66)>**
VarConSeq$_7$	9	**<(63)(16)(2 22)(24)(22 50 66)>**
$BaseP_7$ (0.054:0.60)	13	<(70)(58 66)(22)(74)(22 41)(2 74)(31 76)(2 74) >
PatConSeq$_8$	16	**<(70)(58)(22 58 66)(22 58)(74)(22 41)(2 74)(31 76)(2 74)>**
VarConSeq$_8$	18	**<(70)(58 66)(22 58 66)(22 58)(74)(22 41)(2 22 66 74)(31 76)(2 74)>**
PatConSeq$_9$	0	cluster size was only 5 sequences so no pattern consensus sequence was produced
VarConSeq$_9$	8	**<(70)(58 66)(74)(74)(22 41)(74)>**
$BaseP_8$ (0.014:0.91)	17	< (20 22 23 96)(50)(51 63)(58)(16)(2 22)(50)(23 26 36)(10 74) >
$BaseP_9$ (0.038:0.78)	7	< (88)(24 58 78)(22)(58)(96) >
$BaseP_{10}$ (0.008:0.66)	17	< (16)(2 23 74 88)(24 63)(20 96)(91)(40 62)(15)(40)(29 40 99) >

given from ApproxMAP accurately stores this information. The less frequent items in the base patterns were in the weighted sequence but with smaller weights. This is illustrated by the lighter (weaker) items in the longer variation consensus sequences. For example, $VarConSeq_1$ has 5 additional items (22, 24, 24, 50, and 93) compared to $PatConSeq_1$. Four of them are extraneous items (22, 24, 24, and 50) while the last one (93) is not. It comes from the base pattern $BaseP_1$, just with less frequency compared to the other items in the base pattern. These weaker items are not included in the pattern consensus sequences because their item strengths are not frequent enough to clearly differentiate them from extraneous items.

Precision is excellent at $\mathcal{P} = 1 - \frac{3}{106} = 97.17\%$. Thus, clearly all the pattern consensus sequences are highly shared by sequences in the database. In all the pattern consensus sequences, there is only three items (the lightly colored 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. See section 5.4 for a detailed discussion on this issue.

It is interesting to note that a base pattern may be recovered by multiple pattern consensus sequences. For example, ApproxMAP forms two clusters whose pattern consensus sequences 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. As discussed in chapter 5, sequences are generated by removing various parts of the base pattern and combining with other items. Thus, two sequences using the 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 our evaluation method ($N_{redun} = 1$).

Based on the above analysis, we can see that ApproxMAP provides a succinct summary of the database. That is, ApproxMAP is able to summarize the 1000 sequences into 16 consensus sequences (two for each partition) both accurately and succinctly. The 16 pattern consensus sequences resemble the base patterns that generated the sequences very well (recoverability=91.16%, precision=97.17%). No trivial or irrelevant pattern is returned ($N_{total} = 8, N_{spur} = 0, N_{redun} = 1$).

Note that in real applications, there is no way to know what the true underlying patterns are. Thus, the approach used in this section to find the optimal parameter setting can not be utilized. However, consistent with results in section 6.1, this section clearly shows that

Table 6.15: Effects of noise ($k = 6, \theta = 30\%$)

$1-\alpha$	running time	recoverability	Precision	N_{total}	N_{spur}	N_{redun}
0%	14	91.16%	1-3/106=97.17%	8	0	1
10%	16	91.16%	1-1/104=99.04%	9	0	2
20%	10	91.16%	1-1/134=99.25%	12	0	5
30%	10	90.95%	1-0/107=100.00%	9	0	2
40%	16	88.21%	1-0/95=100.00%	9	0	2
50%	10	64.03%	1-0/68=100.00%	8	0	3

ApproxMAP is robust with respect to the input parameters. That is many settings give results comparable to the optimal solution ($k = 5$ or $6; 25\% \geq \theta \geq 35\%$). More importantly, for a wide range of k and θ the results are at least a sub optimal solution. For k=3 to 9 and $25\% \geq \theta \geq 50\%$, all results had recoverability greater than 82.98% and precision greater than 95.56%.

The 16 consensus sequences give a good overview of the 1000 sequences. Thus, in a real application, a careful scan of the manageable sized results can give a domain expert good estimates (even with a sub optimal setting) about what possible patterns they can search for in the data. This is really what makes ApproxMAP a powerful exploratory data analysis tool for sequential data. Pattern search methods (sometimes called pattern matching) are much more efficient than pattern detection methods. Users can use pattern search methods to confirm and/or tune suggested patterns found from ApproxMAP.

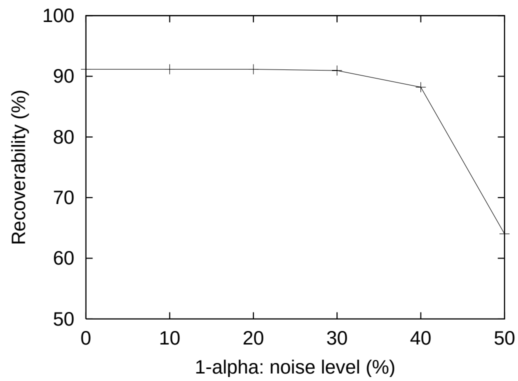
6.2.3 Experiment 2.3: Robustness with respect to noise

Here we evaluate the robustness of the ApproxMAP with respect to varying degrees of noise added to the pattern data used in the previous section. We use the parameters that optimized the results for the patterned data ($k = 6, \theta = 30\%$). Results are given in Table 6.15 and Figure 6.8.

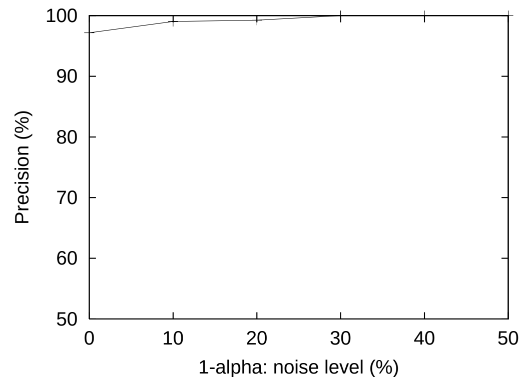
There are two aspects to how noise can interfere with the data mining process. First, if the data mining process cannot differentiate pattern items from the noise items, precision will decrease. In ApproxMAP, pattern items are identified as those items that occur regularly in a certain location after alignment. The probability of a random item showing up in a certain position frequently enough to be aligned is very low. Thus, even with large amounts of noise present in the data, these random noise are easily identified and ignored as noise.

In fact, Figure 6.8(b) shows that precision actually increases when there is more noise in the data. This is because with noise in the data, ApproxMAP is more likely to miss the weak signatures that it identified as patterns without noise. For recoverability, that means that some weak signatures that are true patterns will be missed resulting in some reduction in recoverability. On the other hand, for precision, that means that the weak signatures that were false positives will now be accurately identified as extraneous items resulting in higher

Figure 6.8: Effects of noise ($k = 6, \theta = 30\%$)



(a) Recoverability w.r.t. $1 - \alpha$



(b) Precision w.r.t. $1 - \alpha$

precision. Recently, researchers have looked into intentionally adding noise to data for such reasons [] [FROM KUM: **add reference**].

Second, noise can interfere with detecting the proper pattern items. This will reduce recoverability. As discussed in the previous paragraph, noise will cause ApproxMAP to miss some of the weak signatures and in turn reduce recoverability slightly. However, when the base pattern appears fairly frequent in the database ApproxMAP can still detect most of the base pattern even in the presence of noise.

To understand why, let us consider the behavior of ApproxMAP in the presence of noise. When there are noise in a data sequence, ApproxMAP will simply align based on the remaining pattern items. Then once the alignment is done ApproxMAP reports on the items that occur regularly in each position. Note that the items that are randomly changed are different in each sequence. Thus, a particular pattern item missing from some of the data sequences, is most likely still captured by ApproxMAP because there will be other data sequence with that particular pattern item in the proper position if the base pattern signature is strong enough.

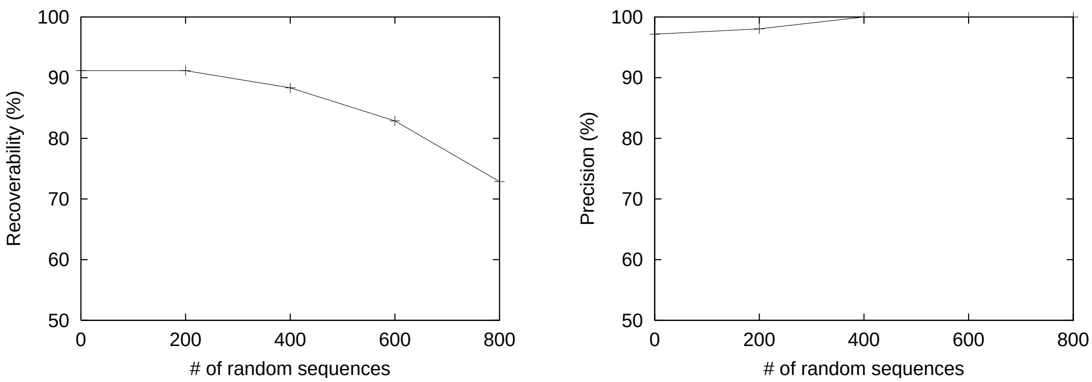
Furthermore, ApproxMAP aligns the sequences by starting with the most similar sequences and working out to the least similar. The weighted sequence built from the core of the sequences in the cluster forms a center of mass. That is, the items in the base pattern start to emerge in certain positions. Then the sequences with more noise can easily attach its pattern items to the strong underlying pattern emerging in the weighted sequence. In essence, ApproxMAP works only with those items that align with other sequences and simply ignores those that cannot be aligned well. This makes ApproxMAP very robust to noise in the data.

Table 6.15 and Figure 6.8(a) show that the results are fairly good with up to 40% of noise in the data. Despite the presence of noise, it is still able to detect a considerable number of the base patterns (i.e. recoverability is 88.21% when corruption factor is 40%) with no extraneous items (precision is 100%) or spurious patterns. At 50% of noise (that is 50% of

Table 6.16: Effect of outliers ($k = 6, \theta = 30\%$)

$N_{outlier}$	N_{PatSeq}	N_{PatI}	N_{ExtraI}	$\mathcal{R}$	$\mathcal{P}$	N_{total}	N_{spur}	N_{redun}	$\|C_{null}\|$	$\|C_{one\_itemset}\|$	$\|C\|$
0	1000	103	3	91.16%	97.17%	8	0	1	1	0	9
200	1000	100	4	91.16%	98.04%	8	0	1	2	0	10
400	1000	97	6	88.33%	100%	8	0	1	3	0	11
600	1000	92	10	82.87%	100%	8	0	1	4	0	12
800	1000	81	11	72.88%	100%	7	0	1	10	1	18

Figure 6.9: Effects of outliers ($k = 6, \theta = 30\%$)



(a) Recoverability w.r.t. outliers (b) Precision w.r.t. outliers

the items in the data were randomly changed to some other item or deleted), recoverability starts to degenerate (recoverability=64.03%).

Clearly, the results show that ApproxMAP is robust to noise in the data. ApproxMAP is very robust to noise because it looks for long sequential patterns in the data and simply ignores all else.

6.2.4 Experiments 2.4: Robustness with respect to outliers

This experiment is designed to test the effect of outliers in the data. To do so, we added varying number of outliers (random sequences) to patterned data used in experiment 2.2. The main effect of the outliers are the weakening of the patterns as a percentage of the database.

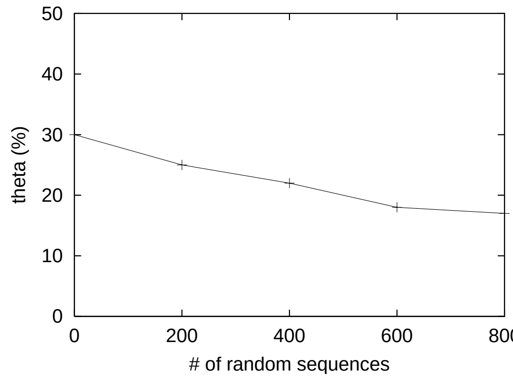
Again we start with the parameters that optimized the results for the patterned data ($k = 6, \theta = 30\%$). When k is maintained at 6, the added random sequences had no effect on the clusters formed or how the patterned sequences were aligned because the nearest neighbor list does not change much for the patterned data. Each cluster just picks up various amounts of the outliers which are aligned after all the patterned sequences are aligned. In effect, the outliers are ignored when it cannot be aligned with the patterned sequences in the cluster. The rest of the outliers formed small separate clusters that generated no patterns, as in experiment 2.1 on random data.

Nonetheless, the consensus sequences were shorter because the random sequences increased

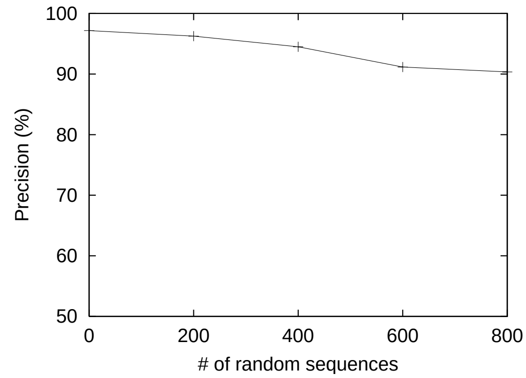
Table 6.17: Effect of outliers ($k = 6$)

$N_{outlier}$	N_{PatSeq}	θ	N_{PatI}	N_{ExtraI}	$\mathcal{R}$	$\mathcal{P}$	N_{total}	N_{spur}	N_{redun}	$\|C_{null}\|$	$\|C_{one\_itemset}\|$	$\|C\|$
0	1000	30%	103	3	91.16%	97.17%	8	0	1	1	0	9
200	1000	25%	103	4	91.16%	96.26%	8	0	1	2	0	10
400	1000	22%	103	6	91.16%	94.50%	8	0	1	3	0	11
600	1000	18%	103	10	91.16%	91.15%	8	0	1	4	0	12
800	1000	17%	103	11	91.16%	90.35%	8	0	1	10	0	18

Figure 6.10: Effects of outliers ($k = 6$)



(a) θ w.r.t. outliers



(b) Precision w.r.t. outliers

the cluster size (Table 6.16 and Figure 6.9). This in turn weakened the signature in the cluster resulting in reduced recoverability (Figure 6.9(a)) when the cutoff θ was kept at the same level (30%) as the pattern data. Similar to what we saw with noise in the data, precision slightly increases with more outliers (Figure 6.9(b)).

However, as seen in Table 6.17, we can easily find the longer underlying patterns by adjusting θ to compensate for the outliers in the data. That is θ can be lowered to pick up more patterned items. The tradeoff would be that more extraneous items could be picked up as well. We assessed the tradeoff by finding out how many extraneous items would be picked up by ApproxMAP in order to detect the same number of patterned items as when there were no outliers. In Table 6.17 the largest θ for which the exact same patterned items can be found are reported on each dataset along with the additional extraneous items picked up. We include the number of null and one-itemset clusters as well.

We can see in Figure 6.10, that with only a small drop in θ we can recover all of the base patterns detected without any outliers with only a small decrease in precision for all dataset. In general, the more outliers, the more extraneous items are found for the threshold that gives the same patterned items.

Again, clearly ApproxMAP is robust to outliers. Even with over 40% outliers in the data ($\frac{800}{1000} \cdot 100\% = 44.4\%$) precision is good at 90.35% when θ is set to give the same recoverability as the patterned data (91.16%). This is inline with the results of experiment 2.1 on random data. Random data has very little affect on ApproxMAP. It does not introduce spurious

Table 6.18: Parameters for the data generator for Experiment 3

Notation	Meaning	Default value
$\|I\|$	# of items	$1,000$
$\|\Lambda\|$	# of potentially frequent itemsets	5,000
N_{seq}	# of data sequences	$10,000$
N_{pat}	# of base pattern sequences	100
L_{seq}	Avg. # of itemsets per data sequence	20
L_{pat}	Avg. # of itemsets per base pattern	$14 = 0.7 \cdot L_{seq}$
I_{seq}	Avg. # of items per itemset in the database	2.5
I_{pat}	Avg. # of items per itemset in base patterns	$2 = 0.8 \cdot I_{seq}$

Table 6.19: Parameters for ApproxMAP for Experiment 3

k	# of neighbor sequences	5
θ	the strength cutoff for pattern consensus sequences	50%
$min\_DB\_strength$	the optional parameter for pattern consensus sequences	10 sequences

patterns and it does not hinder ApproxMAP in finding the true base patterns. The only real effect is the increased data size which in turn weakens the pattern as a proportion of the data. This can easily be compensated for by adjusting θ.

6.3 Database Parameters And Scalability

Now that we have a good understanding of ApproxMAP and its effectiveness, we go on to study its performance on different types of patterned data. Here we examine the effects of various parameters of the IBM patterned database on the results. We use much bigger datasets in these experiments. The default configuration of the data sets used is given in Table 6.18. The expected frequencies of the 100 base patterns embedded in the dataset range from 7.63% to 0.01%. The full distribution of $E(F_B)$is given in Figure 5.1(b). See section 5.3 for a detailed discussion on the properties of this synthetic database. We used all default parameters for ApproxMAP which are restated in Table 6.19.

We study the results on four factors, the total number of unique items in the data set $\|\mathcal{I}\|$ and the data set size in terms of (1) number of sequences, N_{seq}, (2) the average number of itemsets in a sequence, L_{seq}, and (3) the average number of items per itemset, I_{seq}. The following analysis strongly indicates that ApproxMAP is effective and scalable in mining large databases with long sequences.

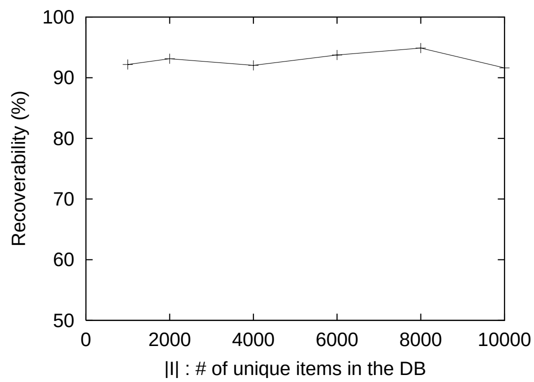
6.3.1 $\|\mathcal{I}\|$: Number of unique items in the database

We first studied the effects of the number of items in the set $\mathcal{I}$, $\|\mathcal{I}\|$. A smaller value of $\|\mathcal{I}\|$ results in a denser database (i.e., patterns occur with higher frequencies) because the

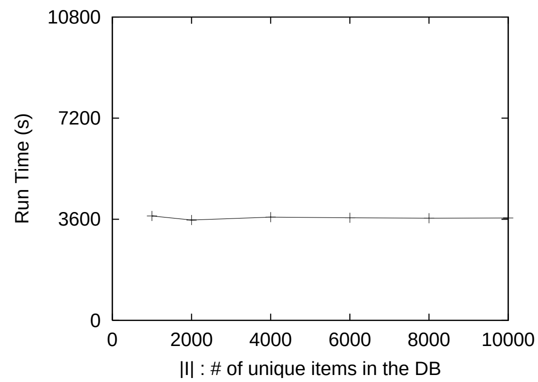
Table 6.20: Results for $\|\mathcal{I}\|$

$\|\mathcal{I}\|$	Recoverability	Precision	N_{spur}	N_{redun}	N_{total}	Running Time(s)
1000	92.17%	100.00%	0	18	94	3718
2000	93.13%	100.00%	0	15	91	3572
4000	92.03%	100.00%	0	22	96	3676
6000	93.74%	100.00%	0	15	94	3652
8000	94.89%	100.00%	0	16	92	3639
10000	91.62%	100.00%	0	15	85	3644

Figure 6.11: Effects of $\|\mathcal{I}\|$



(a) Recoverability w.r.t. $\|\mathcal{I}\|$



(b) Running Time w.r.t. $\|\mathcal{I}\|$

items come from a smaller set of literals. In multiple alignment, the positions of the items have the strongest effect on the results. Although same items may occur often (and belong to many different base patterns), if the item occurs in different locations in relation to other items in the sequence the items can be easily identified as coming from different base patterns. Thus, even when the density of the database changes, the alignment statistics does not change substantially in our experiments. The full results are given in Table 6.20 and Figure 6.11. We observe that there is no noticeable difference in the results of ApproxMAP in terms of the five evaluation measures as well as the running time. In this experiment, for a wide range of $\|\mathcal{I}\|$ between 1,000 to 10,000,

1. recoverability is consistently over 90%,
2. there is no extraneous items (precision=100%) or spurious patterns,
3. there is a manageable number of redundant patterns ($15 \geq N_{redun} \geq 22$),
4. and running time is constant at about 1 hour.

6.3.2 N_{seq}: Number of sequences in the database

In this section, we test the effect of data set size in terms of number of sequences in the data set. The results are shown in Table 6.21 and Figure 6.12. As the data set size increases with respect to N_{seq}

Table 6.21: Results for N_{seq}

N_{seq}	Recoverability	Precision	N_{spur}	N_{redun}	N_{max}	N_{total}	Running Time(s)
10000	92.17%	100.00%	0	18	76	94	3718
20000	95.81%	100.00%	0	39	87	126	12279
40000	98.39%	99.39%	0	54	93	147	45929
60000	99.29%	96.44%	0	107	95	202	98887
80000	98.76%	95.64%	0	134	95	229	159394
100000	98.90%	93.45%	0	179	97	276	232249

1. recoverability increases slightly,
2. precision decreases slightly,
3. the number of max patterns increase,
4. the number of redundant patterns increase,
5. there is no spurious patterns,
6. and running time is quadratic (very modest slope) with respect to N_{seq}.

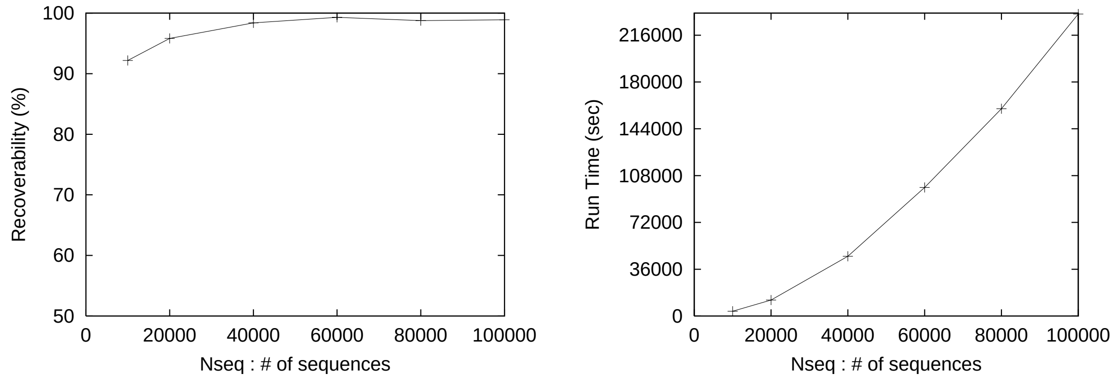
Let us look at each point in more detail. It is interesting to note that the recoverability increases as the data set size goes up, as shown in 6.12(a). It can be explained as follows. With multiple alignment, the more the sequences in the data set, the easier the recovery of the base patterns. In large data sets, there are more sequences approximating the patterns. For example, if there are 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 base pattern. It would be much easier for ApproxMAP to detect the general trend from $1,000$ sequences than from 10 sequences.

This is the same reason why overall there are more patterns returned, N_{total}, as N_{seq} increases. When there are more sequences, ApproxMAP is able to detect more of the less frequent patterns. Some of the extra sequences detected are categorized as redundant patterns, hence the increase in N_{redun}, and others are max patterns, N_{max}. Obviously, the increase in recoverability is directly related to the increase in max patterns.

Remember redundant patterns returned are almost never the exact same patterns being returned. As shown in the small example in Table 6.14, redundant patterns are the consensus sequences that come from the same base patterns ($PatConSeq_2$ and $PatConSeq_3$). Many times redundant patterns hold additional information about possible variations to the max pattern. For example, $PatConSeq_2$ and $PatConSeq_3$ in the small example suggests that there might be many sequences in the dataset that are missing the later part of the longer $PatConSeq_2$ [2]. As there are more sequences in the dataset, ApproxMAP is more likely to pick up these variations, which results in increased redundant patterns.

[2] Such hypothesis can be verified using more efficient pattern search methods.

Figure 6.12: Effects of N_{seq}



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

Table 6.22: A new underlying trend emerging from 2 base patterns

ID	len	Patterns
PatC	16	(22,43) (2,22,28,73) (96) (95) (2,74) (5,24,65,93) (2,6)
BP_1	14	(43) (2,28,73) (96) (95) (2,74) (5) (2) (24, 63) (20) (93)
BP_2	14	(22) (22) (58) (2, 16, 24, 63) (24,65,93) (6) (11, 15, 74)

Another reason for the increase in redundant patterns as N_{seq} increases is due to the limitations in our evaluation method discussed in section 5.4. That is, the current evaluation method maps each result pattern to only one base pattern. However, this is not aways accurate. Essentially, when N_{seq}/N_{pat} is large in the IBM data generator, a new underlying pattern can emerge as two base patterns combine in a similar way frequently enough. This new mixed pattern will occur regularly in the database and a good algorithm should detect it.

To demonstrate how ApproxMAP works in such situations, we look in detail at an example from one of our experiments [3]. In Table 6.22, $PatC$ is the pattern consensus sequence returned for a cluster. The underlined items are the 6 extraneous items as determined by the evaluation method. Clearly, $PatC$ originates from base pattern BP_1 since the two share 10 items. The color of the items in $PatC$ indicates the strength of the items while the dark items for BP_1 are shared items and the light items are missed items. However, it is also clear that the remaining 6 items in $PatC$ (the underlined items) originated from BP_2. This cluster seems to suggest that there is a group of sequences that combine the two base sequences, BP_1 and BP_2, in a similar manner to create a new underlying pattern similar to $PatC$.

[3] Instead of using an example from one of the experiments done in this section we use a smaller experiment because it is much easier to see. The example had the following parameters for the data generator: $N_{seq} = 5000, N_{pat} = 10, L_{seq} = 10, L_{pat} = 7, I_{seq} = 2.5$, and $I_{pat} = 2$.

Table 6.23: The full aligned cluster for example given in Table 6.22

BP_1	[43]	[2,28,73]	[96]		[95]	[2,74]	[5]	[2]	[24,63]	[20]	[93]
BP_2	[22]	[22]	[58]		[2,16,24,63]		[24,65,93]	[6]	[11,15,74]		
Wgt Seq 16	{15:1,16:1, 17:1, **22:16**, 31:2, **43:15**, 63:4, 66:1, 76:2}:16	{**2:16**,15:1, 16:1, 17:1, **22:13**, **28:16**, 66:1, **73:16**, 88:2}:16	{2:3,15:1, 16:1, **22:8**, **58:8**, 63:1, 66:1, **96:16**} :16	{58:1, 63:1, 95:1, 96:1} :1	{**2:9**,**16:9**,22:2,**24:9**, 30:1,**58:7**,**63:9**,66:1, 74:2,**95:16**}:16	{**2:16**,5:1,16:2, 22:1,24:6,31:1, 63:1,65:5,**74:16**, 76:1,93:5}:16	{**5:11**,6:2,16:1, **24:12**,31:1,58:1, **65:11**,66:1,76:1, **93:11**}:15	{**2:14**,**6:15**,11:2, 15:2,22:2,50:6, 58:2,62:2,66:4, 74:2,96:1}:16	{2:1,6:1,11:3, 15:3,16:1,24:3, 50:1,62:1,63:3, 74:3}:4	{20:1, 22:1, 50:1, 66:1} :1	
PatC	**{22,43}**	**{2,22,28,73}**	**{96}**		**{95}**	**{2,74}**	**{5,24,65,93}**	**{2,6}**			
VarC	**{22,43}**	**{2,22,28,73}**	**{22,58,96}**		**{2,16,24, 58,63,95}**	**{2,74}**	**{5,24,65,93}**	**{2,6}**			
763	{22,43}	{2,22,28,73}	{96}	{}	{58,95}	{2,74}	{24,65,93}	{2,6}	{}	{}	
762	{22,43}	{2,22,28,73}	{58,96}	{}	{2,16,24,63,95}	{2,74}	{5,24,65,93}	{2,6}	{}	{}	
764	{22,43}	{2,22,28,73}	{96}	{}	{58,95}	{2,74}	{5,24,65,93}	{2,6}	{}	{}	
767	{22,43}	{2,28,73}	{22,96}	{}	{58,95}	{2,74}	{5,24,65,93}	{2,6}	{}	{}	
761	{22,43}	{2,22,28,73}	{22,58,96}	{}	{2,16,24,63,95}	{2,24,65,74,93}	{5,6,24,65,93}	{2,6,11,15,74}	{}	{}	
775	{22,43,63}	{2,22,28,73}	{2,22,58,96}	{}	{2,16,24,63,95}	{2,74}	{5,24,65,93}	{2,6,50}	{}	{}	
718	{22}	{2,22,28,73}	{58,96}	{}	{95}	{2,74}	{24,65,93}	{6}	{}	{}	
766	{22,43}	{2,22,28,73,88}	{96}	{}	{58,95}	{2,22,74}	{24,65,93}	{2,6,58}	{}	{}	
1133	{22,43}	{2,22,28,73}	{22,96}	{}	{58,95}	{2,16,24,63,74}	{24,65,93}	{2,6}	{6,11,15,24, 63,74}	{}	
765	{22,43}	{2,22,28,73,88}	{22,58,63,96}	{}	{2,16,24,58,63,95}	{2,24,65,74,93}	{5,24,58,65,93}	{2,6,22,50,66,96}	{}	{}	
743	{22,31,43,76}	{2,22,28,73}	{58,96}	{}	{58,66,95}	{2,74}	{5,16}	{2,6,50,62,66}	{}	{}	
590	{15,16,17,22, 43,63,66}	{2,22,28,73}	{2,22,96}	{}	{2,16,22,24,63,74,95}	{2,31,74,76}	{5,66}	{2,6,50,58}	{}	{}	
776	{22,43,63}	{2,28,73}	{2,22,96}	{}	{2,16,24,63,95}	{2,24,65,74,93}	{5,6}	{2,11,15,74}	{}	{}	
742	{22,31,43,76}	{2,22,28,73}	{58,66,96}	{}	{2,16,22,24,30,63,95}	{2,16,24,65,74,93}	{}	{6,50,62,66}	{2,11,15,16, 24,63,74}	{}	
1909	{22,43}	{2,28,73}	{22,96}	{58,63, 95,96}	{2,16,24,63,74,95}	{2,5,74}	{5,24,65,93}	{2,6}	{11,15,74}	{20,22, 50,66}	
1145	{22,43,63}	{2,15,16,17,22 28,66,73}	{15,16,58,96}	{}	{2,16,24,63,95}	{2,24,65,74,93}	{5,24,31,76}	{2,6,22,50,66}	{24,50,62,63}	{}	

Table 6.24: Results for N_{seq}

N_{seq}	Precision $\mathcal{P}$	N_{item}	N_{patI}	N_{extraI}	N_{patI2}	$N_{extraI2}$	Precision $\mathcal{P}'$	N_{mult}
10000	100%	2245	2245	0	0	0	100%	0
20000	100%	3029	3029	0	0	0	100%	0
40000	99.39%	3590	3568	22	22	0	100%	1
60000	96.44%	5086	4905	181	180	1	100%	11
80000	95.64%	5662	5415	247	247	0	100%	13
100000	93.45%	7374	6891	483	479	4	99.95%	25

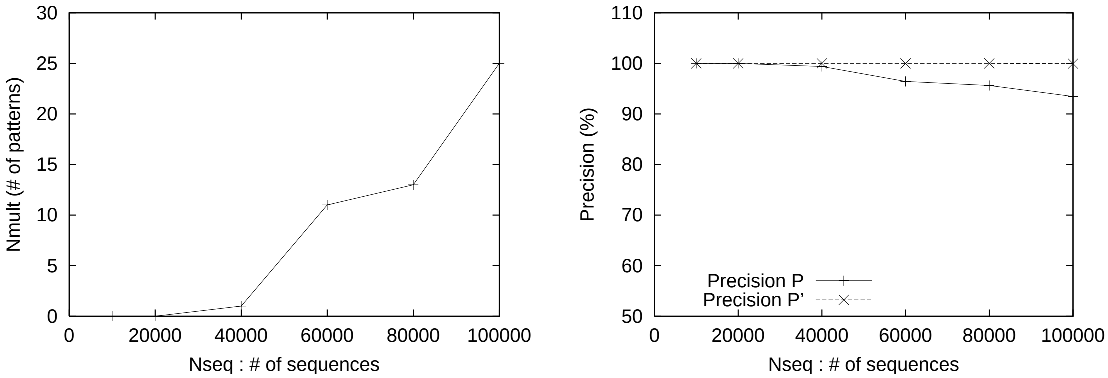
We investigate this cluster further in Table 6.23. Table 6.23 gives the full aligned cluster (size=16 sequences). $WgtSeq$ is the weighted sequence for the cluster. $PatC$ is the pattern consensus sequence with $\theta = 50\%$, and $VarC$ is the variation consensus sequence with $\delta = 40\%$. δ was set higher than the default because the cluster size was very small (16 sequences). Note that since $\theta = 50\% = 8$ seq $< min\_DB\_strength = 10$ seq, the cutoff for pattern consensus sequence is 10 sequences. The cutoff for the variation consensus sequence is $\delta = 40\% = 7$ sequences. The aligned sequences clearly show that the underlying pattern in the data is a combination of the two base patterns BP_1 and BP_2. We see that all 16 items in the consensus sequence are approximately present in all sequences. There are no real extraneous items.

Furthermore, the five items in the middle of BP_2, $\langle(58)(2, 16, 24, 63)\rangle$, are present in the data sequence with slightly less frequency (between 10 and 7 sequences). These items are accurately captured in the variation consensus sequence. In comparison, the seven items at the end of the two base patterns, $\langle(24, 63)(20)(93)\rangle$ for BP_1 and $\langle(11, 15, 74)\rangle$ for BP_2, barely exist in the data. All are present in less than 5 sequences. Thus, these items are not included in the consensus pattern because it does not exist frequently in the data. It is clear from looking at the aligned sequences that ApproxMAP has accurately mapped the 16 sequences to the combined pattern underlying the cluster.

Detecting such patterns could be quite useful in real applications for detecting interesting unknown patterns, which arise by systematically combining two known patterns. For example, we might know the separate buying patterns of those that smoke and those that drink. However if there are any systematic interactions between the two groups that were not known ApproxMAP could detect it.

In Table 6.24, we show the number of extraneous items that could be built into a second base pattern in column N_{patI2} and the remaining true extraneous items in $N_{extraI2}$. All but 1 of 181 extraneous items, when $N_{seq} = 60K$, could be built into a second base pattern. For $N_{seq} = 100K$, all but 4 of 483 extraneous items came from a second base pattern. In all other experiments, all the extraneous items came from a second base pattern. Column Precision P' gives the modified precision which exclude the extraneous items that map back to a second base pattern. Column N_{mult} is the number of consensus patterns that mapped back to 2

Figure 6.13: Effects of N_{seq}



(a) N_{mult} w.r.t. N_{seq} (b) Precision w.r.t. N_{seq}

Table 6.25: Results for L_{seq}

L_{seq}	Recoverability	Precision	N_{spur}	N_{redun}	N_{total}	Running Time(s)
10	85.29%	100.00%	0	26	92	934
20	92.17%	100.00%	0	18	94	3718
30	95.92%	100.00%	0	16	94	8041
40	95.93%	100.00%	0	11	93	14810
50	95.32%	100.00%	0	16	93	21643

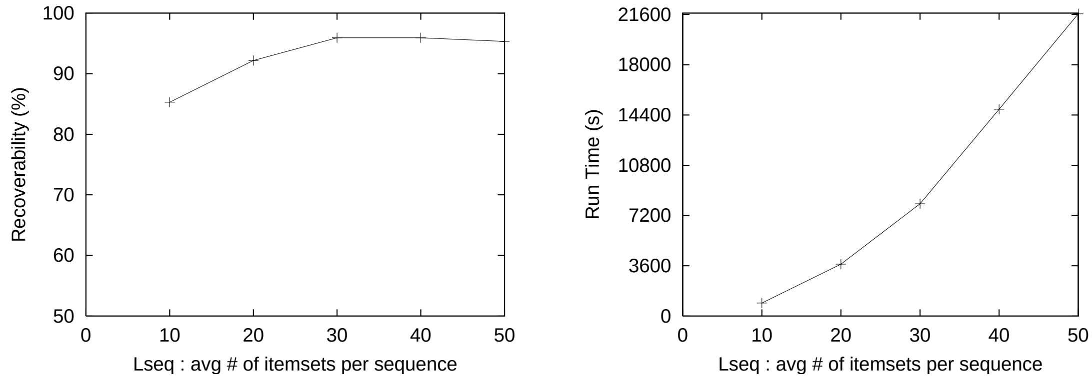
base patterns for each experiment. Not surprisingly, as N_{seq} increases (which will increase N_{seq}/N_{pat}), more interacting patterns (patterns that are a systematic mix of multiple base patterns) are found (Figure 6.13(a)).

In terms of the current evaluation method, there are two consequences of detecting these new patterns. First, the detected pattern will most likely be categorized as a redundant pattern. This is because, statistically it is very likely that there will be another cluster which represents the primary base pattern well. Thus, the small cluster giving the new emerging pattern is counted as a redundant pattern.

Second, the items that do not map back to the primary base pattern will be counted as extraneous items. Consequently, as N_{seq} increase, and more interacting patterns are found, precision decrease. However, when the second base pattern is properly taken into account we see that precision, $\mathcal{P}'$, is stable as N_{seq} increases (Figure 6.13(b)).

In terms of running time, we observe that ApproxMAP is scalable with respect to the data set size, as shown in Figure 6.12(b). Although asymptotically ApproxMAP is quadratic with respect to N_{seq}, we see that practically it is close to linear. Furthermore, when the dataset size is really big ($N_{seq} \geq$ 40K), ApproxMAP can use the sample based iterative clustering method discussed in section 4.6.2.

Figure 6.14: Effects of L_{seq}



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

Table 6.26: Results for I_{seq}

I_{seq}	Recoverability	Precision	N_{spur}	N_{redun}	N_{total}	Running Time(s)
2.5	92.17%	100.00%	0	18	94	3718
5	92.25%	100.00%	0	21	95	4990
10	94.52%	100.00%	0	21	98	7248
15	93.61%	100.00%	0	30	104	9583
20	95.05%	100.00%	0	26	105	11000

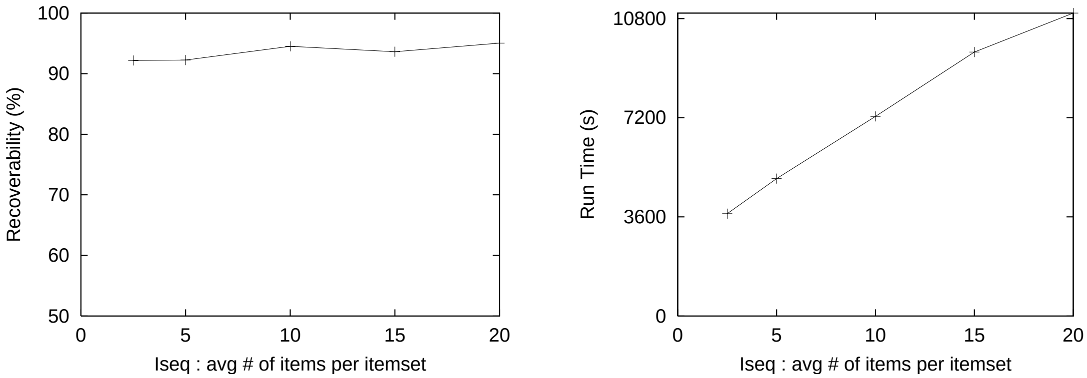
6.3.3 $L_{seq} \cdot I_{seq}$: Length of sequences in the database

We observe similar results with respect to the length of the sequences in the database. As the length increases, the results improve. There are two parameters that control the length (total number of items) of a sequence. L_{seq} is the average number of itemsets in a sequence and I_{seq} is the average number of items per itemset. Therefore, both change the total number of items of the data sequences as well as the embedded base patterns. As either parameter increases, the additional items provide more clues for proper alignment leading to better results.

More itemsets per sequence (larger L_{seq}), has a direct effect on improving alignment due to more positional information during alignment. That is, as the average number of itemsets in a sequence increase, recoverability tends to increase (Figure 6.14(a)). As the number of elements in the sequence (itemsets) increase the pattern signature becomes longer. That means there are more clues as to what is the proper alignment. This in turn makes it easier to capture the signature through alignment. The result are given in Table 6.25 and Figure 6.14.

On the other hand, more items per itemset (larger I_{seq}) has an indirect effect of improving alignment by providing more evidence on the location of a particular position. For example, if $I_{seq} = 2.5$, that means that on average there is only 2.5 items in any position to indicate its

Figure 6.15: Effects of I_{seq}



(a) Recoverability w.r.t. I_{seq} (b) Running Time w.r.t. I_{seq}

proper position. If one or two items in a particular itemset for a sequence is missing due to variations, there is not much more information left to use to identify the proper alignment. In comparison, if $I_{seq} = 10$, that means that on average there are 10 items in any position. So, even if one or two items are missing due to slight variations in the data, there are still a good number of items that can be used to locate the proper position in the alignment. The indirect effect of I_{seq} is not as pronounced as L_{seq}. The results are given in Table 6.26 and Figure 6.15.

Basically, the longer the pattern the more likely it is to detect the patterns. Thus, both L_{seq} and I_{seq} have similar affects, but L_{seq} has a stronger affect. In summary, for a wide range of L_{seq} between 10 to 50:

1. Recoverability tends to be better as the sequences get longer and levels off at around 96% at $L_{seq} = 30$.
2. There is no extraneous items (precision=100%) or spurious patterns.
3. There is a manageable number of redundant patterns ($16 \geq N_{redun} \geq 26$).
4. And the running time is close to linear with respect to L_{seq} with the optimization discussed in section 4.6.1 (Figure 6.14(b)).

The experiment of $I_{seq} = 2.5..20$ gave similar results as follows:

1. Recoverability has a slight upward tendency (Figure 6.15(a)).
2. There are no extraneous items (precision=100%) or spurious patterns.
3. And there is a manageable number of redundant patterns ($18 \geq N_{redun} \geq 30$).
4. However, unlike L_{seq}, the running time is linear with respect to the average number of items per itemset, I_{seq}. The time complexity analysis in section 4.5 show that ApproxMAP should be linear with respect to I_{seq}. Our experiment confirmed the relationship as shown in Figure 6.15(b).

Chapter 7

Comparative Study

Traditionally in data mining, sequential pattern has been defined as a subsequence that appears frequently in a sequence database [2]. Such definition leads to a support model based sequential pattern mining. Currently as far as we know, in KDD all methods available for sequential pattern mining is based on such a support model. In comparison, as shown in this dissertation ApproxMAP applies a very different model based on sequence alignment. ApproxMAP is an approximate solution to the multiple alignment sequential pattern mining problem given in section 3.4.

Much research has been done to efficiently find support based sequential patterns. But to the best of our knowledge, no research has examined in detail what patterns are actually generated from such a definition. In this chapter, we examined the results of the support model closely to evaluate whether it in fact generates interesting and understandable patterns. We then go on to compare the results from the support model to the multiple alignment model.

7.1 Conventional Sequential Pattern Mining

7.1.1 Support Model

Sequential pattern mining is commonly defined as finding the complete set of frequent subsequences in a set of sequences. The conventional support model finds all subsequences that meet a user specified threshold, *min_sup* [2]. We restate the exact definition, given in section 3.1, here for clarity.

Given a sequence database $\mathcal{D}$, the *support* of a sequence seq_p, denoted as $sup(seq_p)$, is the number of sequences in $\mathcal{D}$ that are supersequences of seq_p. seq_j is a supersequence of seq_i and seq_i is a subsequence of seq_j, if and only if seq_i is derived by deleting some items or whole itemsets from seq_j. Conventionally, a sequence seq_p is called a *sequential pattern* if $sup(seq_p) \geq min\_sup$, where min_sup is a user-specified *minimum support threshold*.

Much research has been done and many methods have been developed to efficiently find such patterns. The methods can be divided into breadth first search methods [2, 45, 55] and depth first search methods [4, 34, 48, 66]. These were reviewed in detail in chapter 2. All methods return the same results and only differ in how the frequent subsequences are counted.

7.1.2 Support Model Limitations

Although the support model based sequential pattern mining has been extensively studied and many methods have been proposed (e.g., GSP [55], SPADE [67], PrefixSpan [48], FreeSpan [34], and SPAM [4]), there are three inherent obstacles within the conventional model.

First, support along cannot distinguish between statistically significant patterns and random occurrences. When we applied the evaluation method, discussed in chapter 5, to the support model for sequential pattern mining, it revealed that empirically the model generates huge number of redundant and spurious patterns in long sequences, which bury the true patterns. Our theoretical analysis of the expected support of short patterns in long sequences confirms that many short patterns can occur frequently simply by chance along. To combat this problem in association rule mining, [10] has used correlation to find statistically significant associations. Unfortunately, the concept of correlation does not extend easily to sequences of sets.

Second, these methods mine sequential patterns with exact matching. The exact match based model is vulnerable to noise in the data. A sequence in the database supports a pattern if, and only if, the pattern is fully contained in the sequence. However, such exact matching approach may miss general trends in the sequence database. Many customers may share similar buying habits, but few of them follow exactly the same buying patterns. Let us consider a baby product retail store. Expecting parents who will need a certain number of products over a year constitute a considerable group of customers. Most of them will have similar sequential patterns but almost none will be exactly the same patterns. Understanding the general trend in the sequential pattern for expecting parents would be much more useful than finding all frequent subsequences in the group. Thus, to find non-trivial interesting long patterns, we must consider mining *approximate* sequential patterns.

Third, most methods mine the complete set of sequential patterns. When long patterns exist, mining the complete set of patterns is ineffective and inefficient, since every sub-pattern of a long pattern is also a pattern [63]. For example, if $\langle a_1 \cdots a_{20} \rangle$ is a sequential pattern, then each of its subsequence is also a sequential pattern. There are $(2^{20} - 1)$ patterns in total! On one hand, it is very hard for users to understand and manage a huge number of patterns. On the other hand, computing and storing a huge number of patterns is very expensive or even computationally prohibitive. In many situations, a user may just want the *long patterns* that cover many short ones.

Recently, mining compact expressions for frequent patterns, such as max-patterns [8] and frequent closed patterns [47], has been proposed and studied in the context of frequent itemset mining. However, mining max-sequential patterns or closed sequential patterns is far from trivial. Most of the techniques developed for frequent itemset mining "cannot work for frequent subsequence mining because subsequence testing requires ordered matching which is more difficult than simple subset testing" [63].

Just last year, Yan published the first method for mining frequent closed subsequences using several efficient search space prunning methods [63]. Much more work is needed in this area. Nonetheless, in a noisy sequence database, the number of max- or closed sequential patterns still can be huge, and many of them are trivial for users.

7.1.3 Analysis of Expected Support on Random Data

Implicit in the use of a minimum support for identifying frequent sequential patterns is the desire to distinguish true patterns from those that appear by chance. Yet many subsequences will occur frequently by chance, particularly if (1) the subsequence is short, (2) the data sequence is long, and (3) the items appear frequently in the database. When min_sup is set low, the support of those sequences that are frequent by chance can exceed min_sup simply by chance. Conversely, an arbitrarily large min_sup may miss rare but statistically significant patterns.

We can derive the expected support, $E\{sup(seq)\}$, for a subsequence under the statistical null hypothesis that the probability of an item appearing at a particular position in the sequence is independent of both its position in the sequence and the appearance of other items in the sequence. We will first consider simple sequences of items rather than sequences of itemsets. The analysis is then extended to sequences of itemsets.

The expected support (measured as a percentage) for a subsequence under our null hypothesis is $Pr\{seq_i \text{ appears at least once}\}$. We use the probability of the sequence appearing at least once because an observed sequence in the database can increment the support of subsequence only once, regardless of the number of times the subsequence appears in it.

$$E\{sup(seq_i)\} = Pr\{seq_i \text{ appears at least once}\} = 1 - Pr\{seq_i \text{ never appears}\} \tag{7.1}$$

Consider first the calculation of the probability that item A will appear at least one time in a sequence of length L. That is let us calculate the expected support of sequence A. Rather than calculate $Pr\{\mathsf{A}\}$, the probability of A appearing once, directly, it is easier to calculate $Pr\{\neg\mathsf{A}\}$, i.e. the probability of never seeing an A. $Pr\{\mathsf{A}\}$ is then found from 1-$Pr\{\neg\mathsf{A}\}$. The probability that A never appears, is

$$Pr\{\mathsf{A} \text{ never appears}\} = [1 - p\{\mathsf{A}\}]^L \tag{7.2}$$

where $p\{\mathsf{A}\}$ is the probability of item A appearing in the sequence at any particular position. Then the expected support of A, ie. the probability that A will appear at least one time in a sequence of length, L is

$$\begin{aligned} E\{sup(\mathsf{A})\} &= Pr\{\mathsf{A}\text{ appears at least once}\} \\ &= 1 - Pr\{\mathsf{A}\text{ never appears}\} \\ &= 1 - [1 - p(\mathsf{A})]^L \end{aligned} \tag{7.3}$$

Now consider the calculation of $Pr\{seq_i\}$ where the subsequence length is two. A and B arbitrarily represent two items, which need not be distinct. Here we calculate the expected support of sequence, AB. Again it is easier to first calculate $Pr\{\neg(\mathsf{A..B})\}$, i.e. the probability of never seeing an A followed by a B rather than to calculate directly $Pr\{\mathsf{A..B}\}$, the probability that item A will be followed by item B at least one time in a sequence of length L. To calculate $Pr\{\neg(\mathsf{A..B})\}$ divide all possible outcomes into the $L+1$ mutually exclusive sets:

$$\begin{array}{lcl} \mathsf{A}\text{ first appears in position }1 & = & first(\mathsf{A},1) \\ \mathsf{A}\text{ first appears in position }2 & = & first(\mathsf{A},2) \\ \ldots & & \\ \mathsf{A}\text{ first appears in position }j & = & first(\mathsf{A},j) \\ \ldots & & \\ \mathsf{A}\text{ first appears in position }L & = & first(\mathsf{A},L) \\ \mathsf{A}\text{ never appears} & & \end{array}$$

The probability that A first appears in position j

$$Pr\{first(\mathsf{A},j)\} = Pr\{\mathsf{A}\text{ first appears in pos }j\} = [1 - p(\mathsf{A})]^{j-1} p(\mathsf{A}) \tag{7.4}$$

for $j = 1..L$. The probability that A never appears is given in equation 7.2.

Next, we condition the probability of $\neg(\mathsf{A..B})$ upon these mutually exclusive events:

$$\begin{aligned} Pr\{\neg(\mathsf{A..B})\} = \; & Pr\{\neg(\mathsf{A..B})|\mathsf{A}\text{ first appears in pos 1}\} \cdot Pr\{\mathsf{A}\text{ first appears in pos 1}\} \\ & +Pr\{\neg(\mathsf{A..B})|\mathsf{A}\text{ first appears in pos 2}\} \cdot Pr\{\mathsf{A}\text{ first appears in pos 2}\} \\ & \ldots \\ & +Pr\{\neg(\mathsf{A..B})|\mathsf{A}\text{ first appears in pos j}\} \cdot Pr\{\mathsf{A}\text{ first appears in pos j}\} \\ & \ldots \\ & +Pr\{\neg(\mathsf{A..B})|\mathsf{A}\text{ first appears in pos L}\} \cdot Pr\{\mathsf{A}\text{ first appears in pos L}\} \\ & +Pr\{\neg(\mathsf{A..B})|\mathsf{A}\text{ never appears}\} \cdot Pr\{\mathsf{A}\text{ never appears}\} \\ = \; & \textstyle\sum_{j=1}^{L} Pr\{\neg(\mathsf{A..B})|first(\mathsf{A},j)\} \cdot Pr\{first(\mathsf{A},j)\} \\ & +Pr\{\neg(\mathsf{A..B})|\mathsf{A}\text{ never appears}\} \cdot Pr\{\mathsf{A}\text{ never appears}\} \end{aligned} \tag{7.5}$$

Where in general,

$$\begin{aligned} Pr\{\neg(\mathsf{A..B})|first(\mathsf{A},j)\} &= Pr\{\neg(\mathsf{A..B})|\mathsf{A}\text{ first appears in pos } j\} \\ &= Pr\{\mathsf{B}\text{ never appears in a sequence of length } L-j\} \\ &= [1-p(\mathsf{B})]^{L-j} \end{aligned} \tag{7.6}$$

By substituting equations 7.2, 7.4, and 7.6 into 7.5 and recognizing that the probability of not seeing A followed by B given that A never appears in the sequence is 1

$$Pr\{\neg(\mathsf{A..B})|\mathsf{A}\text{ never appears}\} = 1$$

we have,

$$Pr\left\{\neg(\mathsf{A..B})\right\} = p(\mathsf{A})\sum_{j=1}^{L}\left\{[1-p(\mathsf{B})]^{L-j}[1-p(\mathsf{A})]^{j-1}\right\} + [1-p(\mathsf{A})]^{L} \tag{7.7}$$

Then the probability that a sequence of length L will provide support for AB is

$$\begin{aligned} E\{sup(\mathsf{AB})\} &= Pr\left\{(\mathsf{A..B})\right\} \\ &= 1 - Pr\left\{\neg(\mathsf{A..B})\right\} \\ &= 1 - \left(p(\mathsf{A})\textstyle\sum_{j=1}^{L}\left\{[1-p(\mathsf{B})]^{L-j}[1-p(\mathsf{A})]^{j-1}\right\} + [1-p(\mathsf{A})]^{L}\right) \end{aligned} \tag{7.8}$$

The probability of observing longer subsequences can now be calculated recursively. To determine $Pr\{\mathsf{A..B..C}\}$ we again first calculate $Pr\{\neg(\mathsf{A..B..C})\}$ and condition on A appearing first in either the j^{th} position or not at all:

$$\begin{aligned} Pr\{\neg(\mathsf{A..B..C})\} = {} & Pr\{\neg(\mathsf{A..B..C})|\mathsf{A}\text{ first appears in pos 1}\}\cdot Pr\{\mathsf{A}\text{ first appears in pos 1}\} \\ & +Pr\{\neg(\mathsf{A..B..C})|\mathsf{A}\text{ first appears in pos 2}\}\cdot Pr\{\mathsf{A}\text{ first appears in pos 2}\} \\ & \ldots \\ & +Pr\{\neg(\mathsf{A..B..C})|\mathsf{A}\text{ first appears in pos j}\}\cdot Pr\{\mathsf{A}\text{ first appears in pos j}\} \\ & \ldots \\ & +Pr\{\neg(\mathsf{A..B..C})|\mathsf{A}\text{ first appears in pos L}\}\cdot Pr\{\mathsf{A}\text{ first appears in pos L}\} \\ & +Pr\{\neg(\mathsf{A..B..C})|\mathsf{A}\text{ never appears}\}\cdot Pr\{\mathsf{A}\text{ never appears}\} \\ = {} & \textstyle\sum_{j=1}^{L} Pr\{\neg(\mathsf{A..B..C})|first(\mathsf{A},j)\}\cdot Pr\{first(\mathsf{A},j)\} \\ & +Pr\{\neg(\mathsf{A..B..C})|\mathsf{A}\text{ never appears}\}\cdot Pr\{\mathsf{A}\text{ never appears}\} \end{aligned} \tag{7.9}$$

Note that the conditional probability

$$\begin{aligned} Pr\{\neg(\mathsf{A..B..C})|first(\mathsf{A},j)\} &= Pr\{\neg(\mathsf{A..B..C})|\mathsf{A}\text{ first appears in pos 1}\} \\ &= Pr\{\neg(\mathsf{B..C})\text{ in a sequence of length } L-j\} \end{aligned} \tag{7.10}$$

Applying equation 7.7 to a sequence of length $L-j$ we find that:

$$
\begin{aligned}
&Pr\{\neg(\mathsf{B..C}) \text{ in a sequence of length } L-j\}\\
&= p(\mathsf{B})\sum_{k=1}^{L-j}\left\{[1-p(\mathsf{C})]^{L-j-k}[1-p(\mathsf{B})]^{k-1}\right\} + [1-p(\mathsf{B})]^{L-j}
\end{aligned}
\tag{7.11}
$$

Noting that the probability of not seeing A followed by B followed by C given that A never appears in the sequence is 1,

$$Pr\{\neg(\mathsf{A..B..C})|\mathsf{A} \text{ never appears}\} = 1$$

and substituting equations 7.10, 7.4, and 7.2 into 7.9 produces an expression for $Pr\{\neg(\mathsf{A..B..C})\}$

$$
\begin{aligned}
&Pr\{\neg(\mathsf{A..B..C})\}\\
&= p(\mathsf{A})\sum_{j=1}^{L}\left(p(\mathsf{B})\sum_{k=1}^{L-j}\left\{[1-p(\mathsf{C})]^{L-j-k}[1-p(\mathsf{B})]^{k-1}\right\} + [1-p(\mathsf{B})]^{L-j}\right)\cdot[1-p(\mathsf{A})]^{j-1} + [1-p(\mathsf{A})]^{L}
\end{aligned}
\tag{7.12}
$$

from which $Pr\{\mathsf{A..B..C}\}$ can be found:

$$E\{sup(\mathsf{ABC})\} = Pr\{(\mathsf{A..B..C})\} = 1 - Pr\{\neg(\mathsf{A..B..C})\} \tag{7.13}$$

The analytical expression for the probability of observing a subsequence at least once quickly becomes cumbersome as the length of the sequence increases, but may be readily computed in a recursive manner as illustrated above.

Now consider a sequence of itemsets. Under some models for how an itemset is constructed the probability of seeing a set of n arbitrary items in the same itemset is simply

$$p(i_1)\cdot p(i_2)\cdots p(i_n).$$

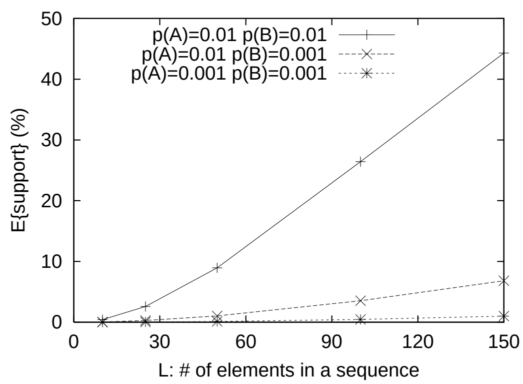
Then, by substituting the probability of an arbitrary element, $p(i_k)$, with $p(i_1)\cdot p(i_2)\cdots p(i_n)$, in any of the equations above we can derive the expected support of sequence of itemsets. For example, the expected support of a simple two itemset pattern, $\langle(i_1..i_n)(i_1..i_m)\rangle$, can easily be derived from equation 7.8 by replacing $p(\mathsf{A})$ with $p(i_1)\cdot p(i_2)\cdots p(i_n)$ and $p(\mathsf{B})$ with $p(i_1)\cdot p(i_2)\cdots p(i_m)$.

The quantities $p(\mathsf{A}), p(\mathsf{B})$, etc., are not known, but must be estimated from the observed database and will necessarily exhibit some sampling error. Consequently, the expression for the expected support of a subsequence, $E\{sup(seq_i)\}$, will also exhibit sampling error. It is, however, beyond the scope of this dissertation to evaluate its distribution. We merely propose as a first step that the choice of min_sup as well as the interpretation of the results should be guided by $E\{sup(seq_i)\}$. In Table 7.1, we calculated the expected support of $E\{sup(seq = \langle(\mathsf{A})(\mathsf{B})\rangle)\}$ with respect to L for two items with varying probabilities and for different lengths. As shown in Figure 7.1, the expected support grows linearly with respect to L.

Table 7.1: Example expected support

P(A)	P(B)	L=10	L=25	L=50	L=100	L=150
0.01	0.01	0.4266%	2.58%	8.94%	26.42%	44.30%
0.001	0.01	0.0437%	0.28%	1.03%	3.54%	6.83%
0.01	0.001	0.0437%	0.28%	1.03%	3.54%	6.83%
0.001	0.001	0.0045%	0.03%	0.12%	0.46%	1.01%

Figure 7.1: $E\{sup\}$ **w.r.t.** L



7.2 Results from the Small Example given in section 4.1

In chapter 4.1, we gave a small example of a sequence database $\mathcal{D}$ to demonstrate ApproxMAP. In Table 7.2 we present the results using the support model on the small example. We set $min\_sup = 20\% = 2$ sequences. The 6 sequences given in bold are the max seqential patterns.

We see that given the 10 sequences in Table 4.1, the support model returns 29 patterns of which 6 are max patterns when $min\_sup = 20\%$. In comparison, ApproxMAP returns 2 patterns $\langle$(A)(BC)(DE)$\rangle$ and $\langle$(IJ)(K)(LM)$\rangle$. See Tables 4.2 to 4.4 in section 4.1 for the full results.

Note that although nearly 3 times as many patterns (29 patterns) are returned than the original input data (10 sequences) from the support model, the two patterns detected by ApproxMAP is not found. This is because the two manually embedded patterns $\langle$(A)(BC)(DE)$\rangle$ and $\langle$(IJ)(K)(LM)$\rangle$ does not match any sequence in $\mathcal{D}$ exactly. Since the support pattern is based on exact match, the support model is not able to find the underlying trend that is not exactly matched by any sequence in the data.

7.3 Empirical Study

In this section, we conduct a detailed comparison of the traditional support sequential pattern mining model with an alternative approximate multiple alignment pattern mining model proposed in this dissertation. We employ the comprehensive evaluation method, pre-

Table 7.2: Results from database $\mathcal{D}$ given in Table 4.1 ($min\_sup = 20\% = 2$ seq)

id	pattern	support	id	pattern	support	id	pattern	support
1	⟨(A)(B)⟩	5	11	**⟨(B)(DE)⟩**	2	21	⟨(A)(B)(C)⟩	2
2	⟨(A)(C)⟩	3	12	⟨(C)(D)⟩	3	22	⟨(A)(B)(D)⟩	2
3	⟨(A)(D)⟩	4	13	⟨(C)(E)⟩	2	23	⟨(A)(B)(E)⟩	2
4	⟨(A)(E)⟩	3	14	⟨(I)(M)⟩	2	24	**⟨(A)(B)(BC)⟩**	2
5	⟨(A)(BC)⟩	3	15	⟨(J)(K)⟩	2	25	⟨(A)(C)(D)⟩	2
6	⟨(B)(B)⟩	3	16	⟨(J)(M)⟩	2	26	**⟨(A)(BC)(D)⟩**	2
7	⟨(B)(C)⟩	2	17	⟨(K)(M)⟩	2	27	⟨(B)(B)(E)⟩	2
8	⟨(B)(D)⟩	4	18	⟨(BC)(D)⟩	3	28	**⟨(J)(K)(M)⟩**	2
9	⟨(B)(E)⟩	4	19	**⟨(BC)(E)⟩**	2	29	**⟨(A)(B)(B)(E)⟩**	2
10	⟨(B)(BC)⟩	2	20	⟨(A)(B)(B)⟩	2			

sented in chapter 5, which can assess the quality of the mined results from any sequential pattern mining method empirically.

Note that all methods generate the same results for the support model. In contrast, the exact solution to the multiple alignment pattern mining is NP-hard, and in practice all solutions are approximate. Consequently the results of the alignment model are method dependent. We use the results of ApproxMAP to represent the alignment model. In general, most of these results were presented in detail in section 6.2. Here we recap the important points for comparison.

In the support based sequential pattern mining, one itemset frequent patterns, called large itemsets, are usually considered as part of the results. However, to be consistent in our comparison we only considered frequent patterns with more than one itemset as the result. Recall that in ApproxMAP we do not consider the one itemset patterns as sequential patterns and dismiss them along with the clusters that generate null patterns.

In summary, we demonstrate that the sequence alignment based approach is able to best recover the underlying patterns with little confounding information under most circumstances including those where the frequent sequential pattern model fails.

7.3.1 Spurious patterns in random data

We first compare the results from the two models on completely random data. We test empirically how many spurious patterns are generated from random data in the support model and the multiple alignment model. We generate random datasets with parameters $\|\mathcal{I}\| = 100, N_{seq} = 1000, I_{seq} = 2.5$, and varied L_{seq}.

The support model has no mechanism to eliminate patterns that occur simply by chance. As seen in the theoretical analysis in section 7.1.3, when sequences are long, short patterns can occur frequently simply by chance. The threshold where the first spurious pattern is generated, T_{spur}, depicts this well empirically (Table 7.3). As the sequence becomes longer,

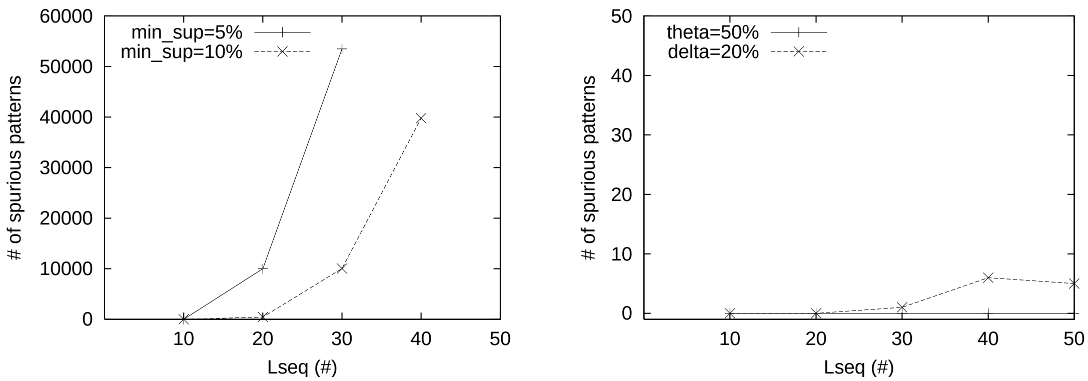
Table 7.3: Results for Random Data from the Support Model

L_{seq}	$min\_sup = 5\%$		$min\_sup = 10\%$		T_{spur}
	N_{spur}	$\|C_{one\_itemset}\|$	N_{spur}	$\|C_{one\_itemset}\|$	
10	9	100	0	100	4.9%
20	10000	100	436	100	20%
30	53471	100	10044	100	37%
40	out of mem	100	39770	100	52%
50	out of mem	100	out of mem	100	61%

Table 7.4: Results for Random Data from the Multiple Alignment Model ($k = 5$ & $\theta = 50\%$)

L_{seq}	N_{total}	N_{spur}	$\|C\|$	$\|C_{null}\|$	$\|C_{one\_itemset}\|$	T_{spur}
10	0	0	90	90	0	18%
20	0	0	93	92	1	17%
30	0	0	81	81	0	27%
40	0	0	95	94	1	35%
50	0	0	99	98	1	32%

Figure 7.2: Results for Random Data



(a) Support model (b) Multiple alignment model

T_{spur} increases quickly. When $L_{seq} = 50$, a simple sequential pattern, $\langle(\mathsf{A})(\mathsf{A})\rangle$, occurs in 610 of 1000 sequences simply by chance. Even when L_{seq} is only 20, the sequential pattern occurs in 200 of 1000 sequences simply by chance along.

Consistent with the theoretical analysis, the results show that support alone cannot distinguish between significant patterns and random sequences. The support model generates many spurious patterns given random data (Figure 7.2(a) and Table 7.3). When $min\_sup = 5\%$ and $L_{seq} = 30$, there are already 53,471 spurious patterns. In many real applications, since $min\_sup << 5\%$, and $L_{seq} > 30$, there could be many spurious patterns mixed in with true patterns.

Our implementation of the support model ran out of memory of when $min\_sup = 5\%$ and $L_{seq} > 30$. It is typical of the support based methods to have difficulty with long sequences [8]. To understand the trend better, we did a second experiment with a higher $min\_sup = 10\%$.

We were able to run upto $L_{seq} = 40$ when $min\_sup = 10\%$.

Clearly, Figure 7.2(a) shows that the number of spurious patterns increase exponentially with respect to L_{seq}. This follows naturally from Figure 7.1 which shows that $E\{sup\}$ of length 2-itemset sequences increase linearly with respect to L. Thus, the total of all patterns (that is the sum of all L-itemset sequences) should grow exponentially.

In contrast, as discussed in section 6.2.1 the probability of a group of random sequences aligning well enough to generate a consensus sequence is negligible. Thus, using default values ($k = 5$ and $\theta = 50\%$), the multiple alignment model found no spurious patterns in any datasets $L_{seq} = 10..50$ (Table 7.4 and Figure 7.2(b)). Although the algorithm generated many clusters (81 to 99), all the clusters were either very small, or not enough sequences in the cluster could be aligned to generate meaningful consensus itemsets.

In Figure 7.2(b), we also report the number of spurious patterns that occur at the cutoff for variation consensus sequences, $\delta = 20\%$. We see that when sequences are longer, there are a few negligible number of spurious patterns (1, 6, and 5 when L_{seq}=30, 40, and 50 respectively) generated at this low cutoff point. These few spurious patterns occur in small clusters where 20% of the cluster results in slightly more than 10 sequences. Since $min\_DB\_strength = 10$ sequences, ApproxMAP is unable to screen these patterns out automatically. Such spurious patterns only occur because there are no patterns in this random data to build an alignment on. It is unlikely you will see the same phenomena in patterned data.

For both models, the threshold where the first spurious pattern occurs, T_{spur}, suggests that as the sequences get longer, spurious patterns are more likely to occur at higher cutoff values. This is to be expected. However, the significant differences in the values of T_{spur} in the two models should be fully appreciated. In the support model, T_{spur} is the highest point at which a spurious pattern appears in the full database. On the other hand, in the multiple alignment model, T_{spur} is the highest point at which a spurious pattern occurs in any subgroup of the database (any cluster). That is support is based on the full database, whereas strength is based on clusters, subgroups of the database. Thus, in practice min_sup is usually very low and is almost always less than 10%. In the support model, when T_{spur} is greater 10% it suggests a significant problem in dealing with spurious patterns.

In comparison, since the strength cutoff, θ, is specified against a similar group of sequences, it is set high (20%-50%) in practice. For pattern consensus sequences, the default is set to 50%. The experiments in section 6 clearly demonstrate that recoverability is consistently over 90% for a wide range of datasets using such default value. Thus, $T_{spur} \leq 35\%$ for all datasets signifies that the first spurious pattern occurs at a considerably low point. Spurious patterns can clearly be differentiated from real patterns in such circumstances. Furthermore, as mentioned above these spurious patterns only occur because there are no patterns in the random data to build an alignment on. As shown in section 6.3.3, in patterned data the longer the sequence and the pattern embedded in the sequences, the better the recoverability

Table 7.5: Results from patterned data

	Multiple Alignment Model $k = 6$ & $\theta = 30\%$	Support Model $min\_sup = 5\%$
Recoverability	91.16%	91.59%
N_{item}	106	1,782,583
N_{extraI}	3	66,058
Precision	97.17%	96.29%
N_{total}	8	253,782
N_{spur}	0	58
N_{redun}	1	253,714

as well as precision.

7.3.2 Baseline study of patterned database

Now that we have an understanding of the behavior of the two models in random data, we move on to investigate patterned data. This experiment serves several purposes. First, it evaluates how well the models detect the underlying patterns in a simple patterned dataset. Second, it illustrates how readily the results may be understood. Third, it establishes a baseline for the remaining experiments. We generate 1000 sequences from 10 base patterns. We used the same patterned database as the one used in section 6.2.2. The parameters of the database is given in Table 6.9. We tuned both models to the optimal parameters. Results and the optimal parameters are given in Table 7.5.

The recoverability for both models is good at over 90%. However, in the support model it is difficult to extract the 10 base patterns from results that include 253,714 redundant patterns and 58 spurious patterns. Furthermore, although the precision seems to be reasonable at 96.29%, this accounts for 66,058 extraneous items of 1,782,583 total items. The precision seems reasonable because there are so many items (almost all of the result items belong to redundant patterns) that as a percentage, the number of extraneous items is small. In reality though, there is quite a bit of extraneous items in the results.

In fact, it would be impossible to list the full results, 253,782 sequences, from the support model. Most of these are redundant patterns. Many of the redundant patterns in the support model are either subsequences of a longer pattern or a small variation on it. They hold no additional information and instead bury the real patterns. Even if we were to find only the max patterns, there would still be 45,281 max patterns[1].

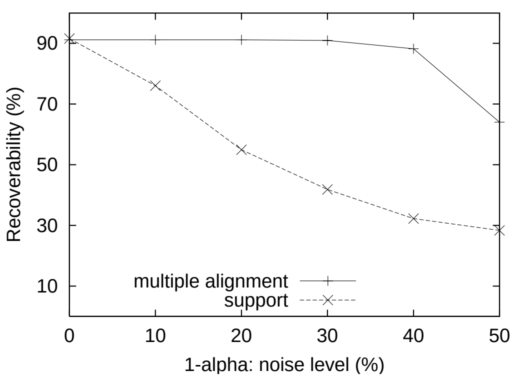
In contrast, the alignment model returned a very succinct but accurate summary of the base patterns. There were only one redundant pattern, no spurious patterns, and 3 extraneous items. The details were discussed in section 6.2.2. Table 6.14 in section 6.2.2 shows, in one page, all the pattern consensus sequences, $PatConSeq_i$, the variation consensus sequences,

[1] Of 45,281 max patterns returned 40 are spurious patterns and 45,231 are redundant patterns.

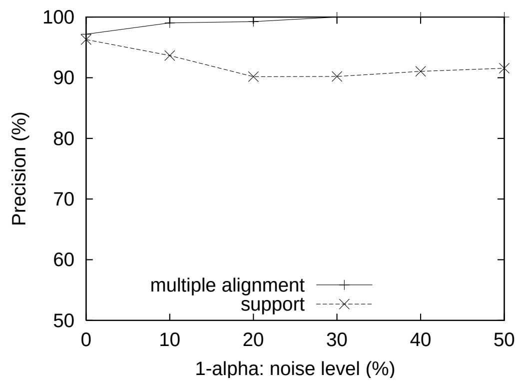
Table 7.6: Effects of noise ($min\_sup = 5\%$)

$1-\alpha$	Recoverability	N_{item}	N_{extraI}	Precision	N_{total}	N_{spur}	N_{redun}
10%	76.06%	235405	14936	93.66%	46278	9	46259
20%	54.94%	39986	3926	90.18%	10670	5	10656
30%	41.83%	11366	1113	90.21%	3646	1	3635
40%	32.25%	4021	360	91.05%	1477	0	1469
50%	28.35%	1694	143	91.56%	701	0	692

Figure 7.3: Effects of noise ($k = 6, \theta = 30\%$)



(a) Recoverability w.r.t. $1-\alpha$



(b) Precision w.r.t. $1-\alpha$

$VarConSeq_i$, with the matching base patterns, $BaseP_i$. In this small dataset, manual inspection clearly shows how well the consensus sequences match the base patterns used to generate the data. Each consensus pattern found was a subsequence of considerable length of a base pattern. The 16 consensus sequences provide a good overview of the 1000 data sequences.

Furthermore, as discussed in section 6.2.2 unlike the support model the small number of redundant patterns returned by ApproxMAP suggests useful information. Recall that in this experiment, there was one redundant pattern returned by the alignment model. This redundant pattern ($PatConSeq_3$) was a subsequence of a longer pattern($PatConSeq_2$). Sequences in the partition that generated $PatConSeq_3$ (Table 6.14) were separated out from the partition that generated $PatConSeq_2$ because they were missing most of the items at the end of $PatConSeq_2$. Thus, when the redundant patterns in the alignment model are subsequences of a longer pattern, it alerts the user that there is a significant group of sequences that do not contain some of the items in the longer pattern.

7.3.3 Robustness with respect to noise

In this section, we evaluate the robustness of the models with respect to varying degrees of noise added to the patterned data used in the previous section.

The results show that the support model is vulnerable to random noise injected into the data. As seen in Table 7.6 and Figure 7.3, as the corruption factor, $1-\alpha$, increases, the support model detects less of the base patterns (recoverability decrease) and incorporates more extraneous items in the patterns (precision decrease). When the corruption factor is 30%, the recoverability degrades significantly to 41.83%. We tried a lower *min_sup* to see if we could recover more of the base patterns. Even when *min_sup* was lowered to 2%, the recoverability was only 65.01% when the corruption factor is 30%. Such results are expected since the model is based on exact match. Note that even with recoverability at 41.83%, the model returns 3,646 patterns that include 1,113 extraneous items (precision=90.21 %).

In comparison, the alignment model is robust to noise in the data. Despite the presence of noise, as shown in section 6.2.3 and Figure 7.3(b), it is still able to detect a considerable number of the base patterns (i.e. recoverability is 90.95% when corruption factor is 30%) with high precision and no spurious patterns. The alignment model is robust to noise because it is an approximate method that reports the general underlying trend in the data.

7.3.4 Robustness with respect to outliers

This experiment is designed to test the effect of outliers added to patterned data. The main effect of the outliers is the weakening of the patterns as a percentage of the database.

Consequently, in the support model the recoverability along with the number of spurious patterns, the number of redundant patterns, and the number of extraneous items is decreased when *min_sup* is maintained at 5% (Table 7.7). On the other hand, if we maintain *min_sup*=50 sequences, obviously the recoverability can be maintained. The tradeoff is that the number of spurious patterns, redundant patterns, and extraneous items all increase slightly (Table 7.8).

Similarly, as discussed in detail in section 6.2.4, in ApproxMAP when θ is kept the same at 50% the results decline as the number of outliers increase. However, we can easily find the longer underlying patterns by adjusting θ to compensate for the outliers in the data. In summary, with a slight decrease in θ we can recover all of the base patterns detected without the outliers with only minor decrease in precision (Table 6.17).

Figure 7.4(a) compares the decrease in precision in the two models when the parameters are adjusted to compensate for the outliers. Although, on the surface it looks like the multiple alignment model is less robust to outliers since the precision is decreased more, this is misleading. Remember that precision is given as a percentage of the total number of items returned. Thus, when there are so many items returned as in the support model, precision does not give an accurate picture. The lower precision for the alignment model when $N_{outlier} = 600$, corresponds to 10 extraneous items of 113 total returned items. In comparison, the support model returned 66,571 extraneous items of 1,784,363 total items. The comparison is similar for $N_{outlier} = 800$.

As seen in Figure 7.4(b) in the support model, the number of extraneous items in the result increase linearly as we add more outliers. The increase in extraneous items for the

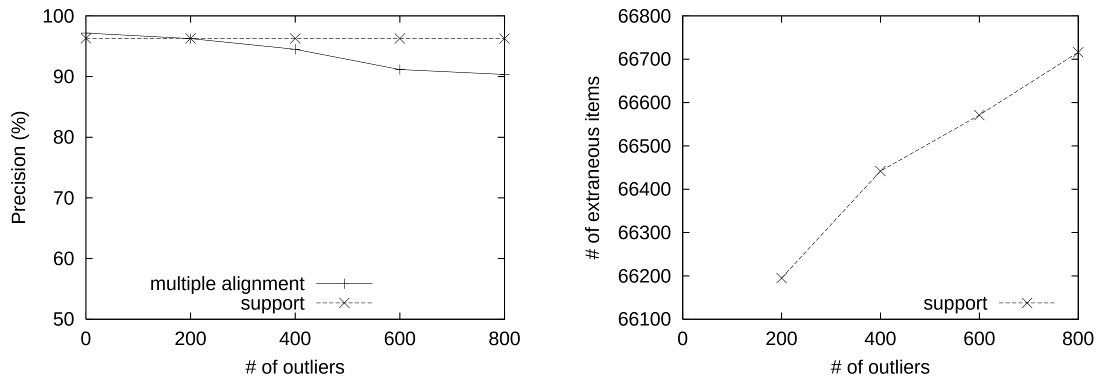
Table 7.7: Effects of outliers ($min\_sup = 5\%$)

$N_{outlier}$	N_{PatSeq}	Recoverability	N_{item}	N_{extraI}	Precision	N_{total}	N_{spur}	N_{redun}
0	1000	91.59%	1782583	66058	96.29%	253782	58	253714
200	1000	91.52%	882063	26726	96.97%	129061	16	129035
400	1000	87.67%	544343	12372	97.73%	82845	7	82828
600	1000	83.76%	286980	6404	97.77%	47730	4	47716
800	1000	80.38%	158671	3611	97.72%	28559	4	28546

Table 7.8: Effects of outliers ($min\_sup = 50$ sequences)

$N_{outlier}$	min_sup	Recoverability	N_{item}	N_{extraI}	Precision	N_{total}	N_{spur}	N_{redun}
0	10/1000=5%	91.59%	1782583	66058	96.29%	253782	58	253714
200	50/1200=4.2%	91.59%	1783040	66195	96.29%	253939	59	253870
400	50/1400=3.6%	91.59%	1783945	66442	96.28%	254200	59	254131
600	50/1600=3.1%	91.59%	1784363	66571	96.27%	254341	59	254272
800	50/1800=2.8%	91.59%	1784828	66716	96.26%	254505	61	254434

Figure 7.4: Effects of outliers



(a) Precision w.r.t. outliers (b) N_{extraI} w.r.t outliers

alignment model is not nearly as pronounced (Table 6.16).

7.4 Scalability

The time complexity of the support model is dependent on the algorithm. However, in general the support based algorithms are linear with respect to the size of the database but exponential with respect to the size of the pattern. The exponential time complexity with respect to the pattern is a lower bound since all algorithms have to enumerate the complete set of subsequences for any pattern by definition.

In contrast, the alignment model is not dependent on the size of the pattern because the patterns are detected directly through multiple alignment. Instead in the alignment model, computation time is dominated by the clustering step, which has to calculate the distance matrix and build the k nearest neighbor list. This inherently makes the time complexity $O(N_{seq}^2 \cdot L_{seq}^2 \cdot I_{seq})$. However, unlike the support model the time complexity can be improved by using a sampling based clustering algorithm and stopping the sequence to sequence distance calculation as soon as it is clear that the two sequences are not neighbors. These improvements were discussed and studied in this dissertation.

In summary, methods based on the support model has to build patterns, either in depth first or breath first manner, one at a time. In contrast, the methods based on the alignment model find the patterns directly through multiple alignment. Thus, not surprisingly when the patterns are relatively long, the alignment algorithms tend to be faster than support based algorithms that are exponential in nature to the size of the pattern.

Chapter 8

Case Study
Mining The Welfare Services Database

An extensive performance evaluation of ApproxMAP verifies that ApproxMAP is both effective and efficient. Now, we report the results on a real data set of welfare services accumulated over a few years in North Carolina State.

We started this research on sequential data mining in order to answer a policy question with our social welfare service data. What are the common patterns of services given to children with substantiated reports of abuse and neglect? What are its variations? Using the daysheet data of services given to these children, ApproxMAP confirmed much of what we knew about these children. These findings gave us confidence in the results. It further revealed some interesting unknown patterns that sent our clients back to investigate.

8.1 Administrative Data

There are three administrative databases used in this analysis. Most of the data comes from the NC social workers' daysheet data. It records activities of social workers in NC for billing purposes. County DSS (Department of Social Services) have numerous funding sources that are combined to pay social workers' salary. In order to properly bill the correct funding source, each social worker is required to report on their 40 hour work week. The daysheet data is a timesheet of the social worker's 40 hour week workload indicating what services were given when, to whom, and for how long. Time is accounted for in minutes. The data gives a fairly accurate picture on the various services given to clients each month. Therefore, we can convert the daysheet data into monthly services given to clients.

Now we need to identify the clients of interest. We can identify children who had a substantiated report of abuse and neglect using the abuse and neglect report database. And then, using the foster care database, we can further split the children with substantiated reports into those that were placed in foster care and those that were not. The children who were never placed in foster care received very little services in comparison to those who were placed. Thus, the interesting patterns were found in those who were placed in foster care. Here we report on our results from children who had a substantiated report of abuse and neglect and were placed in foster care.

There were 992 such children. Each sequence starts with the substantiated report and is followed by monthly services given to each child. The follow up time was one year from the report. In summary we found 15 interpretable and useful patterns.

8.2 Results

The most common pattern we detected was

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

In the pattern, RPT stands for a *Report*, INV stands for an *Investigation*, and FC stands for a *Foster Care service*. In total, 419 sequences were grouped together into one cluster, which gave the above consensus pattern. 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. Recall that the follow up time for the analysis was one year so 12 months in foster care means the child was in foster care for the full time of analysis. 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. The median length of stay in foster care in NC is about 1 year with many children staying in foster care for longer.

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

The rest of the sequences in this data set split into clusters of various sizes. Another obvious pattern was the small number of children who were in foster care for a short time. One cluster formed around the 57 children who had short spells in foster care. The consensus pattern was as follows.

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

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

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

In the pattern, 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. Home management services were meant to be given to those who where not placed in foster care. 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 over some time. 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? If so, 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_support* set to 20%, there is more than $100,000$ sequential patterns and the users just cannot identify the needle from the straws.

Chapter 9

Conclusions

We conclude the dissertation with a summary of our research and a discussion of areas for future work.

9.1 Summary

In any particular data mining problem, the first and most important task is to define patterns operationally. The algorithms are only as good as the definition of the patterns (model). In this dissertation, we propose a novel model for sequential pattern mining, *multiple alignment sequential pattern mining*. Its goal is to organize and summarize sequence of sets to uncover the underlying consensus patterns. We demonstrate that multiple alignment is an effective model to find such patterns that are approximately shared by many sequences in the data.

We develop an efficient and effective algorithm, ApproxMAP (for APPROXimate Multiple Alignment Pattern mining), for multiple alignment sequential pattern mining. ApproxMAP uses clustering as a preprocessing step to group similar sequences, and then mines the underlying consensus patterns in each cluster directly through multiple alignment. A novel structure, weighted sequences, is proposed to summarize and compress the alignment information. The consensus sequences are then extracted from the weighted sequences via strength cutoffs. The strength cutoff is a powerful and expressive mechanism for the user to specify the level of detail to include in the consensus patterns.

To the better of our knowledge, this is the first study on mining consensus patterns from sequence databases. It distinguishes itself from the previous studies in the following two aspects. First, it proposes the theme of approximate sequential pattern mining, which reduces number of patterns substantially and provides much more accurate and informative insights into the sequential data. Second, it generalizes the multiple alignment techniques to handle sequences of itemsets. Mining sequences of itemsets extends the application domain substantially. The method is applicable to many interesting problems, such as social science research, policy analysis, business analysis, career analysis, web mining, and security.

Our extensive evaluation demonstrates that ApproxMAP will effectively extract useful information by organizing the large database into clusters as well as give good descriptors (weighted sequences and consensus sequences) for the clusters using multiple alignment. We demonstrate that together the consensus patterns form a succinct but comprehensive and accurate model of the sequential data. Furthermore, ApproxMAP is robust to its input parameters, robust to noise and outliers in the data, scalable with respect to the size of the database, and in comparison to the conventional support model ApproxMAP can better recover the underlying patterns with little confounding information under most circumstances.

In addition, our case study on social welfare service patterns illustrates that approximate sequential pattern mining can find general, useful, concise, and understandable knowledge and thus is an interesting and promising direction.

9.2 Future Work

This dissertation is our first step towards the study of effective sequential pattern mining. Following the approximate frequent pattern mining model, many interesting research problems need to be solved.

First, more recent advances in multiple alignment come from Gibbs sampling algorithms, which use hidden Markov models [57]. These methods are better for local multiple alignment. Local multiple alignment is to find substrings of high similarity. Formally, given a set of strings, local multiple alignment first selects a substring from each string and then finds the best global alignment for these substrings. Since DNA sequences are very long, finding local similarity has many benefits. One possible future direction would be to expand ApproxMAP to do local alignment and investigate the benefits of local multiple alignment for sequences of sets in KDD applications.

Second, in the optimization of sample based iterative clustering the hash table implementation needs to be explored further. The optimization is made in order to speed up the running time for large datasets. But the current hash table implementation has a large memory requirement for large datasets. In our experiments, we ran out of memeory for datasets $N_{seq} > 70000$ given 2GB of memory. We already know that there are other more efficient implementations of hash tables. But ultimately, to make our method practically scalable, we need to explore an implementation that stores only the possible proximity values (limited by the given memory size), and recalculate the other distances when needed. This will make the application work with any memory size and still give a significant reduction in time (Figure 6.6(d)).

The most interesting future direction is to expand the distance metric to be more comprehensive. First, it could be expanded to handle sequences of multisets or sets with quantitative information. Many of the data mining applications have sets that have more than one of the same item (multiset). For example, people buy many packs of diapers at once. If ApproxMAP could be expanded to handle multisets, it can find quantitative sequential patterns.

Second, user specified taxonomies could be used to customize the replacement cost. For example, two toys should be considered more similar to each other than a toy and a piece of furniture. Under the current model, {doll}, {crib}, and {ball} are all equally distant. If a user specified a taxonomy tree putting doll and ball under the same ancestor and crib in a separate branch, the distance metric could be expanded to a weighted distance metric which can incorporate this information.

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

BIBLIOGRAPHY

[1] A. Abbott and A. Tsay. Sequence Analysis and Optimal Matching Methods in Sociology: Review and Prospect. In *Sociological Methods and Research* Vol 29 pp3-33, 2000.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE 95*, pages 3–14, Taipei, Taiwan, Mar. 1995.

[3] M. S. Aldenderfer and R. K. Blashfield. Cluster analysis Beverly Hills : Sage Publications, 1984.

[4] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *SIGKDD*, pages 429–435, July 2002.

[5] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *The fourth international conference on foundations of data organization and algorithms* 1993.

[6] P. Arabie and L. J. Hubert. An overview of combinatorial data analysis. In *Clustering and classification* Arabie, P. Hubert, L. J., & De Soete, G. (Eds.). River Edge, NJ: World scientific Publ. 1996.

[7] D. Barbara, Y. Li, and J. Couto COOLCAT: an entropy-based algorithm for categorical clustering. In *CIKM*, pages 582–589, 2002.

[8] R. J. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD*, pages 85–93, June 1998.

[9] Z. Bi, C. Faloutsos, F. Korn. The "DGX" distribution for mining massive, skewed data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM Press, 2001.

[10] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: generalizing association rules to correlations. In *Proc. of ACM International Conference On Management of Data (SIGMOD)*, pp265-276, 1997.

[11] P. Berkhin. Survey Of Clustering Data Mining Techniques. *Technical Report - Accrue Software*, San Jose, CA, 2002.

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

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

[14] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Dynamic Programming. In *Introduction to Algorithms.* pp 301-328. Cambridge, Mass. : MIT Press. 1990.

[15] R. O. Duda, P. E. Hart, and D. G. Stork. Unsupervised Learning and Clustering. In *Pattern Classification.*, New York: Wiley, pages 517–599, 2001.

[16] R. C. Dubes and A. K. Jain. Clustering methodologies in exploratory data analysis. In *Advances in Computers.* Vol 19, pages113–228. 1980.

[17] L. Ertoz, M. Steinbach, and V. Kumar. Finding Clusters of Dierent Sizes, Shapes, and Densities in Noisy, High Dimensional Data In *Third SIAM International Conference on Data Mining*, San Fransico. CA, 2003.

[18] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From Data Mining to Knowledge Discovery: An Overview. In *Advances in Knowledge Discovery and Data Mining*, Fayyad et al (Eds), pp 1-34. Cambridge, MA: AAAI/MIT Press, 1996.

[19] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The KDD process for extracting useful knowledge from volumes of data. In Communications of ACM Vol 36 Issue 11, pp 27-34. New York, NY: ACM Press, Nov. 1996.

[20] U. Fayyad and R. Uthurusamy. Data mining and knowledge discovery in databases. In *Communications of ACM* Vol 36 Issue 11, pp 24-26. New York, NY: ACM Press, Nov. 1996.

[21] J. Foley, A. VanDam, S. Feiner, and J. Hughes. Computer graphics : principles and practice Reading, Mass. : Addison-Wesley, 1996.

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

[23] V. Ganti, J. Gehrke, and R. Ramakrishnan CACTUS - Clustering Categorical Data Using Summaries. In *KDD*, pages 73–83, 1999.

[24] D. Gibson, J. M. Kleinberg, and P. Raghavan Clustering Categorical Data: An Approach Based on Dynamical Systems. In *VLDB*, pages 311–322, 1998.

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

[26] R. Goerge, D. Duncan, L. Usher, B. J. Lee, M. Reidy, B. Needell, and A. Brookhar. Dynamics of Children's Movement among the AFDC, Medicaid, and Foster Care Programs. *Technical report to U.S. Dept. of Health and Human Services (HHS).* 2000.

[27] S. Guha, R. Rastogi, and K. Shim ROCK: A Robust Clustering Algorithm for Categorical Attributes. In *ICDE*, pages 512-521, 1999.

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

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

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

[31] J. Han and M. Kamber. Data Mining: Concepts and Techniques. New York, NY: Morgan Kaufmann Publishers, 2001.

[32] A. Hinneburg and D. Keim. Clustering Techniques for Large Data Sets: From the past to the future. In *Tutorial notes fro ACM SIGKDD 1999 International Conference on Knowledge discovery and data mining*, Han et al, (Eds), ACM. 1999.

[33] D. Hand, H. Mannila, and P. Smyth. Principles of Data Mining. Cambridge, MA: MIT Press, 2001.

[34] J. Han, J. Pei, et al. FreeSpan: Frequent pattern-projected sequential pattern mining. In *SIGKDD*, pages 355–359, Aug. 2000.

[35] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. In *ACM Computing Surveys*, Vol. 31, No. 3, pages 264–323, Sep 1999.

[36] A. K. Jain and R. C. Dubes. Algorithms for clustering data. Englewood Cliffs, NJ: Prentice-Hall. 1988.

[37] L. Kaufman and P. J. Rousseeuw. Finding Groups in Data: An Introduction to Cluster Analysis. New York: Wiley. 1990.

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

[39] H.C. Kum, J. Pei, W. Wang, and D. Duncan. ApproxMAP : Approximate Mining of Consensus Sequential Patterns. *Technical Report TR02-031*, UNC-CH, 2002.

[40] J. Kruskal. An overview of sequence comparison. In *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison.* D. Snakoff, & J. Kruskal, (Eds.), pp 253-310. Addison-Wesley Pub. Co. MA. 1983.

[41] A. Large, L. Tedd, and R. Hartley. Information seeking in the online age: principles and practice. Bowker. 1999.

[42] B. Liu, M. Hu, and W. Hsu. Multi-level organization and summarization of the discovered rules. In *Proc. 6th ACM Intl. Conf. On Knowledge Discovery and Data Mining (SIGKDD)*, pp208-217, 2000.

[43] B. Liu, W. Hsu, and Y. Ma. Mining association rules with multiple minimum supports. In *Proc. 5th ACM Intl. Conf. On Knowledge Discovery and Data Mining (SIGKDD)*, pp337-341, 1999.

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

[45] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. In *Data Mining and Knowledge Discovery: An International Journal*, Vol 1(3):pp256-289, 1997.

[46] C. E. Metz. Basic principles of ROC analysis. In *Seminars in Nuclear Medicine*, vol. 8, pp283-298, 1978.

[47] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT*, pages 398–416, Jan. 1999.

[48] J. Pei, J. Han, et al. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *ICDE*, pages 215–224, April 2001.

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

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

[51] 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*, 2(2):169.194, 1998.

[52] D. Sankoff, R. Cedergren. Simultaneous comparison of three or more sequences related by a tree. In *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison.* D. Snakoff, & J. Kruskal, (Eds.), pp 253-310. Addison-Wesley Pub. Co. MA. 1983.

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

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

[55] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *EDBT*, pages 3–17, Mar. 1996.

[56] P. Stolorz and R. Musick (Eds). Guest Editorial. In *Data mining and knowledge discovery* Vol 1 No. 4. Special issue: Scalable High Performance Computing for Knowledge Discovery and Data Mining. Boston, MA: Kluwer Academic Publishers, 1998.

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

[58] C. L. Usher, K. A. Randolph, and H. C. Gogan. Placement patterns in foster care. In *Social Service Review*, vol 73, pp22-36. 1999.

[59] M. Vingron, M. Waterman. Sequence alignment and penalty choice. In *Journal of molecular biology*, Vol 235: pp1-12.

[60] K. Wang, Y. He, and J. Han. Mining frequent itemsets using support constraints. In Proc. 26th Intl. Conf. on Very Large Data Bases (VLDB), 2000.

[61] M. A. Wong and T. Lane. A kth Nearest Neighbor Clustering Procedure. In *Journal of the Royal Statistical Society*, Series B, 45, pp362-368, 1983.

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

[63] X. Yan, J. Han, and R. Afshar. CloSpan: Mining Closed Sequential Patterns in Larege Datasets. In *Third SIAM International Conference on Data Mining*, San Fransico. CA, 2003.

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

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

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

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

[68] Y. Zhang, A. W. Fu, C. H. Cai, and P. Heng Clustering Categorical Data. In *ICDE*, page 305, 2000.