Mining Sequential Patterns by PrefixSpan algorithm with approximation

ANKHBAYAR YUKHUU* SANSARBOLD GARAMRAGCHAA
and HWANG YOUNG SUP**
Department of Computer Science,
Sun Moon University,
100 Kalsan-ri, Tangjeongmyeon, Asan ; Chugnam, 336-708
SOUTH KOREA

Abstract- We want to find sequential patterns in a long continues noisy DNA sequence. Sequential pattern mining, which discovers frequent subsequences as patterns in a sequence database, is an important data mining problem with broad applications, including the analysis of customer purchase patterns or Web access patterns and analysis of DNA sequences, and so on. We investigated sequential pattern mining algorithms for long continues DNA sequences. Most previously proposed mining algorithms follow the exact matching with a sequential pattern definition. They are not able to work in noisy environments and inaccurate data in practice. We investigated approximate matching method to deal with those cases. In this paper, we develop and apply Pattern–Growth PrefixSpan algorithm to find most repeated patterns, for example, motifs in DNA sequence. Our algorithm gains its efficiency by using pattern growth and approximation methodologies. The algorithm is based on the observation that all occurrences of a frequent pattern can be classified into groups, which we call approximated pattern. We developed algorithms to quickly find out all relative frequents by a pattern growth method and to determine approximated patterns from those frequents. Our experimental studies demonstrate that our algorithm is efficient in mining repeated approximate sequential patterns that would have been missed by existing methods.

Key-Words: - sequential pattern mining, searching sequential pattern, pattern growth method, PrefixSpan algorithm with approximation, long DNA sequence mining, searching subsequences.

1 Introduction

The sequential pattern mining problem was first introduced by Agrawal and Srikant in [1]: Given a set of sequences, where each sequence consists of a list of elements and each element consists of a set of items, and given a user-specified minimum support (user-specified minimum length frequent) threshold, sequential pattern mining is to find all frequent subsequences, i.e., the subsequences whose occurrence frequency in the set of sequences is no less than the minimum support. Sequential pattern mining is an important data mining problem with broad applications, including the analyses of customer purchase behavior, Web access patterns, scientific experiments, disease treatments, natural disasters, DNA sequences, and so on [2]. There are too many previously proposed pattern mining methods based on a priori-like methods: typical a priori-like method such as GSP (Generalized Sequence Pattern) adopts a multiple-pass, candidate generation, test approach in sequential pattern mining [2][3][4][5]. Also Freespan and PrefixSpan extremely run faster than the a priori-based GSP algorithm [3][4][5]. Almost existing mining algorithms follow the exact matching with a sequential pattern definition. They are not able to work in noisy environments and inaccurate data in practice. We studied approximate matching method to deal with such as DNA sequence. The development of efficient and scalable algorithms for mining approximate sequential patterns is a challenging and practically useful direction to pursue. It has been shown that the capacity to accommodate approximation in the mining process has become critical due to inherent noise and imprecision in data or gene mutations in genomic DNA sequences [8]. The notion of approximate sequential pattern has been proposed in [8], in which an algorithm called ApproxMap is designed to mine consensus patterns. While mining consensus patterns provides one way to produce compact mining result under general distance measures, it remains a challenge how to efficiently mine the complete set of approximate
sequential patterns under some distance measure which is stricter yet equally useful in many cases.

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 couldn’t 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.

The purpose of this study is efficiently mining long sequential patterns in noisy environments. In this paper, we develop and apply Pattern-Growth PrefixSpan algorithm to find most repeated patterns, motifs in DNA sequence. We establish a more general model. The sequences in its support set may have different patterns of substitutions; they can in fact be classified into groups, which we call approximated patterns. Each approximated pattern is a set of sequences sharing a unified pattern representation together with its support.

The idea is that by pattern growth algorithm the mined sets of the patterns discover their related approximated patterns, and we are able to classify them. These initial approximated patterns will then be iteratively assembled into longer approximated patterns in a local search fashion, until no longer ones can be found. In the second stage, generalize the longest or most repeated patterns from grouped based on their constituting sequences to form a support set so that the frequent approximate patterns would be identified. Instead of mining only the patterns repeating within a sliding window of fixed sizes, our algorithm is able to mine all globally repeating approximate patterns.

Traditionally in data mining, sequential patterns have been defined as a subsequence that appears frequently in a sequence database. Such definition leads to a support model based sequential pattern mining methods [3]. Currently as far as we know, the only methods available for analyzing sequences of sets are based on such a support model.

Although the support model based sequential pattern mining has been extensively studied and many methods have been proposed, there are some inherent obstacles within conventional model. These methods meet inherent difficulties in mining databases with long sequences and noise. These limitations are discussed with our algorithm.

The most common sequential pattern mining method is Mining Patterns by Pattern growth [3]: divide sequential patterns to be mined based on the subsequences obtained so far and project the sequence database based on the partition of such patterns. The general idea is outlined as follows: Instead of repeatedly scanning the entire database, generate and test large sets of candidate sequences. One can recursively project a sequence database into a set of smaller databases associated with the set of patterns mined so far and, then, mine locally frequent patterns in each projected database.

2 Term definition

**Definition 1** - If sequence X is corresponded to \( (x_1, x_2, ..., x_i) \) where \( x_i \) is an element of the sequence. The value of instances of items in a sequence is length \( (l) \) of the sequence.

**Definition 2** - If X sequence is subsequence of another sequence \( Y=<y_1,y_2, ..., y_m> \), if there exist integers \( 1<i_1<i_2<...<i_n<m \) such that \( x_1=y_{i_1}, x_2=y_{i_2}, ..., x_n=y_{i_n} \).

**Definition 3** - An X sequence and a user-specified error threshold minimum support \( \theta \), a sequence is said to be frequent if it is contained in most times with \( \theta \) frequents in the sequence. A sequential pattern is a maximal sequence that is frequent.

In the case of DNA sequence: The element of sequence \( X=A,C,G,T \). If X sequence includes a P sequence at least \( \theta \) times, a sequence P is called a sequential pattern and integer \( \theta \) is called user-specified repeat threshold.

In our case we studied to pattern recognition algorithm to find a DNA pattern sequence in long sequence with noise by pattern growth algorithm with error recognition by user specified error threshold.

3 Related Algorithms

In this section, we first outline a projection-based sequential pattern mining method, called FreeSpan, and then introduce an improved PrefixSpan. Both methods generate projected databases, but they differ at the criteria of database projection: FreeSpan creates projected databases based on the current set of frequent patterns without a particular ordering (i.e., growth direction), whereas PrefixSpan projects databases by growing frequent prefixes. PrefixSpan is substantially faster than FreeSpan in most sequence databases.

3.1 FreeSpan

FreeSpan is based on the following property: If an itemset \( X \) is infrequent, any sequence whose projected itemset is a superset of \( X \) cannot be a sequential pattern. FreeSpan mines sequential patterns by partitioning the search space and projecting the sequence subdatabases recursively based on the projected itemsets.

Given the database \( S \) and \( \min\_support \theta_1 \), FreeSpan first scans \( S \), collects the support for each item, and finds the set of frequent items.

Frequent items are listed in support descending order.
PrefixSpan algorithm’s main attribute is that PrefixSpan only grows longer sequential patterns from the shorter frequent this was our useful source of idea.

Algorithm of PrefixSpan as follows:

Input: A sequence database $S$, and the minimum support threshold $\theta$

Output: The complete set of sequential patterns

Method: Call PrefixSpan ($\langle \rangle$, $\theta$, $S$)

Subroutine PrefixSpan ($\alpha$, $l$, $S|\alpha$ )

Parameters: $\alpha$ sequential pattern; $l$ the length of $S|\alpha$ -projected database, if $\alpha\neq\langle\rangle$; otherwise the sequence database $S$.

### 4 PrefixSpan algorithm with approximation

One particularly important problem in the area of sequential pattern mining is the problem of discovering all subsequences that appear on a given sequence database and have minimum support threshold. The difficulty is in figuring out what sequences to try and then efficiently finding out which of those are frequent [2].

Pattern-growth methods are a more recent approach to deal with sequential pattern mining problems. The key idea is to avoid the candidate generation step altogether, and to focus the search on a restricted portion of the initial database.

PrefixSpan is the most promising of the pattern-growth methods and is based on recursively constructing the patterns, and simultaneously, restricting the search to projected databases[3]. A database is the set of subsequences, which are suffixes of the sequences that have prefix $\alpha$. In each step, the algorithm looks for the frequent sequences with prefix $\alpha$, in the correspondent projected database.

In this way, the search space is reduced at each step, allowing for better performances in the presence of small support thresholds. In the presence of gap constraints, the algorithm has to be adapted, and this claim is no longer valid. However, as been shown, the new version of PrefixSpan remains faster than other methods.

### 4.1 Approximate matching

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. Even with moderate noise, a frequent long pattern can be mislabelled as an infrequent pattern [8]. Not much work has been done on approximate pattern mining.

We can extend the conventional sequential pattern mining model to get an approximate sequential pattern mining model. Given a minimum threshold distance $\sigma$,
the patterns generated with approximate matching depend on the number of string matching transition σ. Instead of generating the possible sequences, considering possible errors, we verify if each sequence element performs a valid transition in the automaton, beginning on the initial state. Whenever an element does not correspond to a valid transition, the algorithm tries to replace it (which corresponds to apply a Replacement), and if this fail, then tries to ignore (which corresponds to a Deletion) and finally by trying to introduce a valid transition (which corresponds to an Insertion). Those insertions and replacements try correct element by every possible element (frequent length is constant). The operation does not depend on the size of the number of different elements in the database, but only on the DNA nucleotides of sequences.

In our algorithm we use edit operations in approximate string matching, 3 transitions as follows:

- **Insertion - Ins** \((x, i)\) - adds symbol \(x\) to the position \(i\);
- **Deletion - Del** \((x, i)\) - deletes the symbol \(x\) from the sequence at position \(i\);
- **Replacement - Repl** \((x, y, i)\) - substitutes the symbol \(x\), that occurs at position \(i\), by symbol \(y\)

**Example 3:** If we are searching ATAT– motif, we will find all occurrences with approximate string matching with

- **Insertion** ATAAT,
- **Deletion** ATT, and
- **Replacement** ATGT ... etc

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 native output of the complete set of approximate patterns in the above model may generate many (trivial) patterns and thus obstruct the mining of information.

### 4.2 PrefixSpan algorithm with approximate matching

We have used PrefixSpan algorithm to find exact patterns, the approximate matching can identify which approximate pattern is related.

We created **Algorithm 2**, PrefixSpan, approximate matching algorithms which to mine sequential patterns. Input data is a long continuous DNA sequence.

Our algorithm is divided into two main sections, the first creating patterns

```
CreateSeqPref(char *pattern) {
  // Creating pattern for searching function by pattern growth approach
  ... return pattern;
}
```

and the second is searching patterns

```
SearchPrefix(sequence, pattern) {
  // Searching pattern in sequence for find out repeat of pattern
  ...
  Boolean approxAcc(Sequence, int i, pattern, int e, int Th);
  return repeat;
}
```

```
Boolean approxAcc(Sequence, int i, pattern, int e, int Th) {
  // Approximate (by Th - error threshold) in searching process
  ... if (e > Th)
    return false; // Exceeds maximal error
  if (I < |s|)
    ok = approxAcc(s,i+1,pattern,Th,e);
    // found the transition
  else if (pattern in Sequence)
    return true; // Try to achieve a final state
  else
    return approxAcc(S, i+1, pattern, Th, e);
}
```

```
Boolean approxAcc(Sequence, int i, pattern, int e, int Th) {
  // Check edits operations/replacement
  if (e > Th)
    return false; // Exceeds maximal error
  if (|s[i]| = |a in s[i]|) ok = approxAcc(s,i+1,pattern,Th,e); // found the transition
  else
    ok = approxAcc(s,i+1,pattern,e+dif,Th); //Check edits operations/insertion
  return true;
}
```

```
if (ok) {
  return false;
  ... }  
```

```
CreateSeqPref(char *pattern), the second is searching patterns
```
First, creating section defines which frequent is next searching process. Creating pattern step is creating longer. Second, searching pattern section is finding subsequence from main sequence with approximate matching algorithm, the section returns a pattern repeat. The rule is comparing two sequences with approximation transitions.

When searching short patterns (length=1; length=2; length=3) the algorithm may meet collision. Furthermore, if pattern length is too long, pattern mining result would be reduced. However, we are using user defined coefficient minimum length, min_length and maximum length, max_length to solve those problems.

CreateSeqPref(char *pattern) is creating patterns to search sequence by pattern growth approach, and add to all possibilities of one element for pattern. SearchPrefix(sequence, pattern) function returns how many times pattern is repeated in sequence.

Boolean approxAcc(Sequence, int i, pattern, int e, int Th) is checking error exceed and final states, Boolean accTrans(Sequence, S[i], int i, pattern, int Th, e) is finding and recognizing transitions. Th is user defined error threshold and e is a temporary error.

Whenever an element doesn’t correspond to valid transition, the algorithm tries to replace it (Replacement), and if this fail, then tries to ignore (Deletion) and finally by trying to introduce a valid transitions (Insertion) instead of trying every possible elements(A, C, G, T). Transitions are limited by user defined threshold Th can be 1 or 2, because Th increased by 1 searching process time would be increase at least 3 times.

5. Experimental Results

Our goal is to validate our claim that approximations allow the discovery of unknown information, keeping the mining process centered on the user. In order to do it, we compared general PrefixSpan algorithm and the PrefixSpan algorithm with an approximation approach when finding a motif in a DNA sequence. This PrefixSpan pattern growth method has been used in most studies in a sequential pattern mining, and it generates datasets that mimic real-world transactions, where customers tend to make a sequence of transactions involving a set of items. Sequence and transaction sizes are clustered around a mean and some of them may have larger sizes.

Input data was a long continuous DNA sequence text file. The first 100 element of sequence is as follows:

Example 4:

```
AAGTATTGCGATACTTTATGTTGAGATCCAATATCCcdc
CAGTACATTAGTTGACTCTCCTTAACTGTTTCCCTTGTACGATTATTTATGTTTAAAC...
```

A text file consists A, G, C, T nucleotides. Algorithm outputs were most repeated patterns and their repeats. We have used 27,126 length DNA sequence and user defined error threshold. It means that one transition is possible.

Table 2 illustrates experimental results of two algorithms. When a pattern length was three (length=3), output pattern of the proposed algorithm was repeated 2 times more than PrefixSpan algorithm’s and last nucleotides were different {TTT, TTA}. Two algorithms are observed that they can find motif can be dissimilar at length to length. Different patterns between the outputs showed hidden unknown information to us. Because, many patterns can include some transitions in continues long noisy DNA sequence.

Figure 1 compares of the repeats of PrefixSpan algorithm and approximated PrefixSpan algorithm. The repeat desperately rose with approximation. One and two length pattern is not approximated with our algorithm. The 3 length repeats has increased 936 to 2,015, when length was 4 repeats were 270 and 1,350 and when finding 5 length PrefixSpan algorithm repeat was 90, our algorithm showed 520 times repeated pattern ATAGT.

The repeats of our algorithm were always more than other algorithm and showed dissimilar patterns during the experiment. It can be observed that PrefixSpan with approximation algorithm can effectively uncover the hidden trend in the DNA sequence. That comparison shows that our algorithm can discover more patterns, trends than PrefixSpan algorithm. In result, we are able to find approximately the motifs, most repeated patterns from a long noisy DNA sequence.

In summary our experiment shows that the PrefixSpan with approximation algorithm is more efficient than non approximated PrefixSpan in finding a motif in a long repeated pattern.

<table>
<thead>
<tr>
<th>Pattern length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrefixSpan Pattern</td>
<td>T</td>
<td>TT</td>
<td>TTT</td>
<td>TTT</td>
<td>TTTT</td>
</tr>
<tr>
<td>Pattern with Approximation</td>
<td>T</td>
<td>TT</td>
<td>TTA</td>
<td>TTAT</td>
<td>TTATT</td>
</tr>
<tr>
<td>PrefixSpan Repeat</td>
<td>9,288</td>
<td>2,646</td>
<td>936</td>
<td>270</td>
<td>90</td>
</tr>
<tr>
<td>Repeat with Approximation</td>
<td>9,288</td>
<td>2,646</td>
<td>2,015</td>
<td>1,350</td>
<td>520</td>
</tr>
</tbody>
</table>
6. Conclusions
We proposed the definition of closed frequent approximate sequential patterns to solve the problem of mining the complete set of frequent approximate sequential pattern mining. Our study shows that PrefixSpan algorithm with approximation mines the complete set of patterns and it is more efficient than PrefixSpan algorithm. Our algorithm is based on the notion of classifying a pattern’s support set into approximated patterns. We combine approximation approach mining and iterative pattern growth. We adopt a local search optimization technique to completeness of the mining result. Our performance study shows that our algorithm is able to mine out globally repeating approximate patterns in biological genomic DNA data with great efficiency (when length=4, 5 times better etc.). It is important to explore how to further develop such a pattern-growth based sequential mining methodology for effectively mining DNA databases.

Acknowledgments
This work was supported by the Korea Research Foundation. Grant funded by the Korean Government (MOEHRD), (KRF-2006-311-D00833).

References