# A Survey on Sequential Data Mining: Exploring the Redundant Patterns from Sequences to Minimize the Overall Patterns

**Mr. Tamboli Suraj M[1] Prof. Prabhudev Irabashetti[2]**
[1]M.E Student [2]Assistant Professor
[1,2]Department of Computer Engineering
[1,2]VACOE, A. Nagar. Pune University (MH), Ahmednagar, India

*Abstract*— Especially in genomics and proteomics discovering patterns from sequence data has plays an most important role in many aspects of science and society. Multiple strings as input sequence data and substrings as patterns. In the real world, their will be a large set of patterns would discovered many of them are repeated, thus degrading the output quality. For sequential data mining different method and techniques are implemented. These papers make study on different types of sequential data mining techniques (CISP mining technique). Sequence synthesis and recognition of patterns for multiple sequences was proposed by various algorithm such as A-close algorithm, Associaation Rules, MineTCFI, CFI2TCFI Algorithms, Biomolucer sequence i.e DNA and RNA sequence, GSP and Apriori All algorithm, Prefix span algorithm. To efficiently discover these patterns in very large sequence data, two efficient algorithms have been developed through innovative use of suffix tree. Discovery of delta closed patterns (DDCP) and Discovery of non-induced patterns (DNIP) are better as compare to existing algorithm and methods for sequential data mining of data patterns.

***Key words:*** Sequence Pattern Discovery; Delta Closed Patterns; Statistically Induced Patterns; Random Sequence; Suffix Tree

## I. INTRODUCTION

Today, a vast amount of such data from the genomic, proteomic, and business arenas has been acquired. The discovery of new interesting knowledge from these enormous sequence data has important applications and great value in many sectors but the discovery process recommend to carried out in an efficient and effective way. This survey paper addresses the basic comparative study of problem of mining interesting yet previously unknown patterns in sequences.

Non-delta-closed patterns and statistically induced pat-terns are redundant patterns with respect to their proper super patterns and sub patterns. Similar to the challenge of efficiently searching for supersets of an itemset in delta closed itemsets mining, and we also need an efficient method to identify these proper superpatterns and subpatterns so as to evaluate whether a pattern is redundant or not. This challenge is even greater in the sequence domain. In other terms, it is most computationally expensive to determine the superpattern-subpattern relation between two patterns since the elements in sequence pattern are in sequential order. Here Naive methods that implement the technique of delta closed patterns and statistically induced patterns in a straightforward way are computationally intense. Their runtimes are at least cubic for input size. Their algorithmic complexities are analyzed in the methodology section. This paper contains two useful algorithms developed for discover nonredundant patterns from sequences. One is for discovering delta closed patterns and the other is for discovering nonstatistically induced patterns, called noninduced patterns in abbreviation. They start to use for suffix tree to identify the exact superpatterns and subpatterns so they are able to discover delta closed patterns and noninduced patterns in linear time. They are scalable to handle very large sequence data. By extracting nonredundant patterns, the proposed methods produce a more compact set of patterns.

## II. RELATED WORK

In Discovering frequent closed item sets for association rule systems occurs the problem in recovering frequent itemset in a database. Here closed itemset is framework used. It shows the problem used to minimize problem of finding closed itemset. This system can build best contain mining system algorithm by reducing the search space for the closed item set than the subset lattice. System shows that the all frequent closed item sets suffices to determine a reduce set of association rule, also addressing the important problem i.e. limiting the number of rule produced without information lost.

## III. A-CLOSE ALGORITHM

### *A. A-Close Principle:*

A close algorithm generates all frequent itemset from database D through the following:

- Discover all frequent closed itemset in D i.e. itemset those are closed and have backup greater or equal minsup.
- Derive all frequent itemset from the frequent itemset found in previous. That is generating all subset of the most frequent closed itemset and derives their support from frequent closed itemset support.

Using the result of A-close directly generate the reduced set of valid association rule instead of determining all frequent itemset.

The procedure as follows:

- Discover all frequent closed itemset.
- Determine the exact valid association rule basis: determine the pseudo closed itemset in D and then generate all rules r.
- Construct the reduced set of approximate proper association rules.

 

**Algorithm 1** A-Close algorithm

```
1) generators in G1 ← {1-itemsets};
2) G1 ← Support-Count(G1);
3) forall generators p ∈ G1 do begin
4)     if (support(p) < minsup) then delete p from G1;   // Pruning infrequent
5) end
6) level ← 0;
7) for (i ← 1; Gi.generator ≠ ∅; i++) do begin
8)     Gi+1 ← AC-Generator(Gi);                  // Creates (i+1)-generators
9) end
10) if (level > 2) then begin
11)    G ← ⋃{Gj | j < level-1};                  // Those generators are all closed
12)    forall generators p ∈ G do begin
13)        p.closure ← p.generator;
14)    end
15) end
16) if (level ≠ 0) then begin
17)    G' ← ⋃{Gj | j ≥ level-1};                 // Some of those generators are not closed
18)    G' ← AC-Closure(G');
19) end
20) Answer FC ← {c.closure,c.support|c ∈ G ∪ G'};
```

Fig. 1: A Close Algorithm

**Algorithm 2** Support-Count function

```
1) forall objects o ∈ O do begin
2)     Go ← Subset(Gi.generator, f({o}));
3)     forall generators p ∈ Go do begin
4)         p.support++;
5)     end
6) end
```

Fig. 2: Support-Count Function

δ -Tolerance Closed Frequent Itemsets[3] system, study an inherent problem of mining Frequent Itemsets (FIs): the number of FIs mined is often too large. Their Frequent Is not only directly affects the mining performance, also severely area the application of FI mining. Here the literature survey, Closed FIs and Maximal Frequent Is are proposed as concise representations of FIs. But the number of CFIs is still too high in many cases that's why MFIs miss information about the frequency of the Frequent itemsets. To address these problem, the adjusted definition of CFIs and presents the δ-Tolerance CFIs (δ-TCFIs).So δ-TCFIs reduce all subsets not able within a frequency size bounded by δ. These system presents two algorithms are CFI2TCFI and Mine TCFI, to mine δ-TCFIs. So CFI2TCFI gather very high percent on the calculated frequency of that recovered FIs but is minor efficient when the number of CFIs is high, so it is based on CFI mining. That's why Mine TCFI is calculated faster and less memory than the algorithms of those state of the art efficient representations of FIs, while the accuracy of MineTCFI this is little bit lower than that of CFI2TCFI.

*1) Algorithm CFI2TCFI:*

Mining CFIs is in general much more efficient than mining FIs. Since the set of CFIs is a lossless representation of FIs, an algorithm which takes advantage of the efficiency of mining CFIs. The algorithm first generates the CFIs and then computes the δ-TCFIs from the CFIs.

1) Mine the set of all CFIs;
2) Let Ci be the set of CFIs of size i;
3) for each i ≥ 1 do
4) for each X ∈ Ci do
5) Find X's closest CFI superset, Y ;
6) if(∃Y s.t. freq (Y ) ≥ (1 − δ)|Y |−|X| freq (X) )
7) Update ext (Y ) with freq (X) and ext (X);
8) Delete X;
9) else
10) T ← T ∪ {X};
11) return T;

*2) Algorithm MineTCFI:*

After constructing the global FP-tree T ∅, MineTCFI invokes the recursive pattern-growth procedure GenTCFI, which is shown in Procedure.

1) Construct the global FP-tree, T ∅;
2) Initialize the global δ-TCFI tree, C∅;
3) T ← ∅;
4) Invoke GenTCFI(T∅, C∅, T );
5) Return T ;

*B. Procedure:*

The processing of IsCovered (Lines 4 and 14), IsCondCovered (Line 15) and the search for the closest δ-TCFI superset (Lines 5 and 16) are discussed in Coverage Testing. Procedure can be divided into two parts: when the input conditional FP-tree, TX, consists of only one single path (Lines 1-9), and when TX has more than one path (Lines 10-23).

GenTCFI(TX , CX , T )

1) if(TX is a single path, P )
2) Generate all local δ-TCFIs from P ;
3) for each local δ-TCFI, Y , generated do
4) if( IsCovered (Y, CX) = true )
5) Find Y 's closest δ-TCFI superset, Z;
6) Update ext (Z) with freq (Y );
7) else
8) T ← T ∪ {Y };
9) Insert Y into all CX's predecessor δ-TCFI trees in the recursive-call stack;
10) else
11) for each x in TX.header do
12) Y ← X ∪ {x};
13) Let H be the set of frequent items in BY ;
14) if( IsCovered (Y, CX) = true )
15) if( IsCondCovered (Y ∪ H, CX) = true )
/∗ Prune all supersets of Y ∗/
16) Find Y 's closest δ-TCFI superset, Z;
17) Update ext (Z) with freq (Y );
18) else
19) Construct Y 's conditional FP-tree, TY , and Y 's conditional δ-TCFI tree, CY ;
20) GenTCFI(TY , CY , T );
21) else /∗ IsCovered (Y, CX) = false ∗/
22) Construct Y 's conditional FP-tree, TY , and Y 's conditional δ-TCFI tree, CY ;
23) GenTCFI(TY , CY , T );

*C. Comparison Algorithms:*

We compare algorithms CFI2TCFI and MineTCFI with the following algorithms:

– FPclose: the winner of FIMI 2003 and one of the fastest public implementations for mining CFIs.

 

- NDI: the algorithm (the faster DFS approach) for computing the set of non-derivable FIs (NDIs).
- MinEx: the algorithm for mining the set of frequent δ-free-sets.
- RPlocal: the faster algorithm (than RPglobal) for computing the representative patterns of the δ-clusters.

*D. Advantages:*

The notion of δ-tolerance allows us to flexibly tune δ to enjoy the benefits of both MFIs and CFIs. They can prune a great amount of redundant patterns from the mining result as do MFIs, while they can retain the frequency information of the recovered FIs as do CFIs.

The actual error rate of the estimated frequency of the recovered FIs is much lower than the theoretical error bound. Algorithm CFI2TCFI attains an error rate significantly lower than δ in all cases. CFI2TCFI is also shown to be very efficient cases except when the number of CFIs is large.

Algorithm MineTCFI attains accuracy slightly lower than that of CFI2TCFI; however, MineTCFI is significantly faster than all other algorithms in all cases and also consumes small memory in cases.

In reference paper[5] place a methodology which is able to synthesize a class of biomolecular sequences into a relative pattern determine random sequence for that class and (2) use the random sequence to search and detect subsequences pertaining to that class from a much longer sequence. The detection is achieved through an optimal matching of the random sequence against segments of the search sequence. Since the random sequence contains probabilistic characteristics of many sequences in the class, its comparison with search sequence segments is much more reliable than between two single sequences. The paper presents both the basic notion as well as an algorithm of the synthesis process. It also describes an experiment for detecting transfer RNA sequences embedded in a long DNA sequence derived from bovine mitochondrial genome. The successful detection is based on the optimal matching of the DNA sequence segments with the random sequence synthesized from 12 transfer RNA sequences.

In reference paper [6] Sequential Patterns Performance Improvements that problem of mining sequential patterns was recently presented. We are given a database of number sequences so there is sequence is a list of all transactions ordered by transaction details, and each transaction is a set of items. Here problem is to discover sequential patterns with a user point minimum support their support of a pattern is the number of data-sequences that with the pattern. So an example of a sequential pattern is five percentage of customers bought ‘Foundation' and `Ring world' in particular transaction, followed by `Second Foundation' in a next transaction". We reduce the problem as follows. First, we add time constraints that specify a minimum and/or maximum time period

*E. Limitations:*

- Absence of time constraints.
- Rigid definition of a transaction.
- Absence of taxonomies.

## IV. GSP Algorithm

The basic structure of the GSP algorithm for finding sequential patterns is as follows. The algorithm makes multiple passes over the data. The first pass determines the support of each item, that is, the number of data-sequences that include the item. At the end of the first pass, the algorithm knows which items are frequent, that is, have minimum support. Each such item yields a 1-element frequent sequence consisting of that item. Each subsequent pass starts with a seed set: the frequent sequences found in the previous pass. The seed set is used to generate new potentially frequent sequences, called candidate sequences. Each candidate sequence has one more item than a seed sequence; so all the candidate sequences in a pass will have the same number of items. The support for these candidate sequences is found during the pass over the data. At the end of the pass, the algorithm determines which of the candidate sequences are actually frequent. These frequent candidates become the seed for the next pass. The algorithm terminates when there are no frequent sequences at the end of a pass, or when there are no candidate sequences generated.

We need to specify two key details:

1) Candidate generation: how candidates’ sequences are generated before the pass begins. We want to generate as few candidates as possible while maintaining completeness.
2) Counting candidates: how the support count for the candidate sequences is determined.

Algorithm is not a main-memory algorithm. If the candidates do not in memory, the algorithm generates only as many candidates as will t in memory and the data is scanned to count the support of these candidates. Frequent sequences resulting from these candidates are written to disk, while those candidates without minimum support are deleted. This procedure is repeated until all the candidates have been counted.

## V. Comparison of GSP and AprioriAll

On the synthetic datasets, GSP was between 30% to 5 times faster than AprioriAll, with the performance gap often increasing at low levels of minimum support. The results were similar on the three customer datasets, with GSP running 2 to 20 times faster than AprioriAll.

There are two main reasons why GSP does better than AprioriAll.

- GSP counts fewer candidates than AprioriAll.
- AprioriAll has to first find which frequent itemset are present in each element of data sequence during the data transformation and then find which candidate sequence is present in it. This is typically somewhat slower than directly finding the candidate sequence.

*A. PrefixSpan:*

Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth [6] propose a novel sequential pattern mining method, called PrefixSpan (i.e., Prefix-projected Sequential pattern mining), which explores prefix projection in sequential pattern mining. PrefixSpan mines the complete set of patterns but greatly reduces the efforts of candidate subsequence generation. Moreover, prefix-projection substantially reduces the size of projected databases and leads to efficient processing. Our performance study shows

 

that PrefixSpan outperforms both the -based GSP algorithm and another recently proposed method, FreeSpan, in mining large sequence databases.

*B. Advantages:*

As supported by analysis and performance study, both PrefixSpan and FreeSpan are faster than GSP, and PrefixSpan is also faster than FreeSpan. Both PrefixSpan and FreeSpan are pattern-growth methods; their searches are more focused and thus efficient. Pattern-growth methods try to grow longer patterns from shorter ones. Accordingly, they divide the search space and focus only on the subspace potentially supporting further pattern growth at a time. Thus, their search spaces are focused and are confined by projected databases. A projected database for a sequential pattern contains all and only the necessary information for mining sequential patterns that can be grown from. As mining proceeds to long sequential patterns projected databases become smaller and smaller. In contrast, GSP always searches in the original database. Many irrelevant sequences have to be scanned and checked, which adds to the unnecessarily heavy cost.

Prefix-projected pattern growth is more elegant than frequent pattern-guided projection. Com-paring with frequent pattern-guided projection, employed in FreeSpan, prefix-projected pattern growth is more progressive. Even in the worst case, PrefixSpan still guarantees that projected databases keep shrinking and only takes care postfixes. When mining in dense databases, FreeSpan cannot gain much from projections, whereas PrefixSpan can cut both the length and the number of sequences in projected databases dramatically. Generalization of Pattern-growth Methods for Sequential Pattern Mining with Gap Constraints [7] presents a generalization of the PrefixSpan algorithm to deal with gap constraints. A gap constraint imposes a limit on the separation of two consecutive elements of an identified sequence. This type of constraints is critical for the applicability of these methods to a number of problems, especially those with long sequences and small alphabets. The method we propose is based on the introduction of a new method to generate projected databases that efficiently stores the subsequences of all occurrences of each frequent element.

In this system we have presented the generalization of the PrefixSpan algorithm to deal with gap constraints. In order to achieve that goal, we have proposed a new method to generate projected databases that store the subsequences of all occurrences of each frequent element.

The modified PrefixSpan method keeps its performance advantages relatively to apriori-based algorithms in the more difficult situation of low support thresholds, although its relative advantage over these methods is reduced when compared with the high support thresholds situation.

*C. Disadvantages:*

The problem of sequential pattern mining is one of the several that has deserved particular attention on the general area of data mining. Despite the important developments in the last years, the best algorithm in the area (PrefixSpan) does not deal with gap constraints and consequently doesn't allow for the introduction of background knowledge into the process

*D. Comparison:*

A comparative study between Apriori-based and pattern-growth approaches with and without the presence of gap constraints. In order to do that, we use the AprioriAll, GSP and PrefixSpan algorithms in the absence of gap constraints, and the GSP and GenPrefixSpan algorithms in the presence of these restrictions. The sequences were generated and maintained in main memory during the algorithms processing.

*E. Application:*

The generalization of projection based methods to gap constrained sequential pat-tern problems is very important in many applications, since Apriori-based methods are inapplicable in many problems where low support thresholds are used. In fact, the imposition of a gap restriction is critical for the applicability of these methods in areas like bioinformatics, which exhibit limited size alphabets and very long sequences. We are actively working in applying this methodology to the problem of motif finding in bioinformatics sequences, an area that can benefit very much from more sophisticated methods for sequential pattern analysis.

In [8][9] paper the problem of Contiguous Item Sequential Pattern (CISP) Mining is presented as a sequential pattern mining problem under two constraints. First, each element in a sequence consists of only one item. Second, items appearing in the sequences that contain a pattern must be adjacent with respect to the underlying order as they appear in the pattern. Even though the problem of CISP mining can be solved by using previous approaches on sequential pattern mining under a general constraint description framework, this may lead to poor performance due to the large searching space. To efficiently solve this problem, a new data structure, UpDown Tree, is proposed for CISP mining. UpDown Tree based approach can greatly improve the efficiency of CISP mining in terms of both time and memory comparing to previous approaches. An extensive experimental study has shown promising results with our approach.

In Discovery of Delta Closed Patterns and Non-induced Patterns from Sequences [1] consider multiple strings as input sequence data and substrings as patterns. This paper improves the output quality by removing two types of redundant patterns. First, the notion of delta tolerance closed itemset is employed to remove redundant patterns that are not delta closed. Second, the concept of statistically induced patterns is proposed to capture redundant patterns which seem to be statistically significant yet their significance is induced by their strong significant sub patterns. It is computationally intense to mine these non-redundant patterns (delta closed patterns and non-induced patterns). To efficiently discover these patterns in very large sequence data, two efficient algorithms have been developed through innovative use of suffix tree. Three sets of experiments were conducted to evaluate their performance. They render excellent results when applying to genomics. The experiments confirm that the proposed algorithms are efficient and that they produce a relatively small set of patterns which reveal interesting information in the sequences.

 

## VI. CONCLUSION

This paper employs study of algorithm and techniques for sequence patterns and proposes the notion of statistically induced patterns to capture redundant patterns. The discovery of those nonredundant patterns is computation-ally intense for large sequences. This paper presents comparative study of different algorithms for discovering input sequence patterns and noninduced patterns from large sequence data linear time. Also study their advantages, disadvantages, application and limitations.

## REFERENCES

[1] Andrew K.C. Wong, Dennis Zhuang, Gary C.L. Li, and En shuin Annie Lee, "Discover of delta closed pattern and noninduced pattern from sequences," IEEE Transactions on Knowledge and Data Engineering, Vol. 24, No. 8, August 2012.
[2] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering Frequent Closed Itemsets for Association Rules," Proc. Seventh Int'l Conf. Database Theory,pp. 398-416, 1999.
[3] J. Cheng, Y. Ke, and W. Ng, "Delta-Tolerance Closed Frequent Itemsets," Proc. Sixth Int'l Conf. Data Mining, pp. 139-148, 2006.
[4] A.K.C. Wong, D.K.Y. Chiu, and S.C. Chan, "Pattern Detection in Biomolecules Using Synthesis Random Sequence," J. Pattern Recognition,vol. 29, no. 9, pp. 1581-1586, 1995
[5] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," Proc. Fifth Int'l Conf. Extending Database Technology, pp. 3-17, 1996.
[6] J. Pei and J. Han, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth,"Proc. 17th Int'l Conf. Data Eng.,pp. 215-224, 2001.
[7] C. Antunes and A.L. Oliveira, "Generalization of Pattern-Growth Methods for Sequential Pattern Mining with Gap Constraints," Proc. Int'l Conf. Machine Learning and Data Mining,pp. 239-251, 2003.
[8] J. Chen, "Contiguous Item Sequential Pattern Mining using UpDown Tree," J. Intelligent Data Analysis, vol. 12, no. 1, pp. 25-49, Jan. 2008.
[9] X. Yan, J. Han, and R. Afshar, "CloSpan: Mining Closed Sequential Patterns in Large Databases," Proc. Third SIAM Int'l Conf. Data Mining, pp.166-177, 2003.

 