A Rough Sets Partitioning Model for Incremental Mining of Sequential Patterns in Large Databases

Jigyasa Bisaria, Namita Srivastava and K. R. Pardasani

Department of Mathematics
Maulana Azad National Institute of Technology (A Deemed University)
Bhopal M.P. (India)
jigyasa@bisaria.com

Abstract

Frequent sequential patterns mining finds interesting event sequences from various real world databases under study. However, most of real world databases are subject to updations with advancement of time. The problem of incremental mining of sequential patterns is about finding frequent sequential patterns in updated sequence database by reusing the mining results from the previous mining task. The problem of incremental mining not only requires the information of frequent patterns in database instance at time stamp T but also requires results of infrequent patterns as they many become frequent in the revised database instance in time stamp T+Δt. This causes maintainence of too many patterns for the incremental mining tasks.
In this paper we have proposed a novel perspective to the problem of incremental mining of sequential patterns. The proposed model works on the concept of maintaining a knowledge base of a limited number of patterns to mine the updated database instance. The mining method is based on indiscernibility relation from the theory of rough sets. It is confirmed by experiments that the proposed method outperforms previous well known method ISE (incremental sequence extraction) in algorithm execution times by over a magnitude.

Keywords: data mining, incremental mining, Sequential patterns, indiscernibility relation, partitioning etc.

1.0 Introduction

Sequential patterns are temporally ordered patterns corresponding to an object under study in a given information system. Most of the real world databases are subject to updations with advancement of time. Some of the examples of databases that grow incrementally are database of fault signals from telecom
access networks, database of customer transaction in a retail store, database of web browsing patterns, database of patient treatment etc. This causes two types of operations [2] [3], (i) an existing sequence associated with the object under study is appended and (ii) fresh sequences are generated. Tackling append operation is tricky since the interaction of appended sequence with the original sequence can cause a lot of infrequent subsequences to turn frequent with the updation. It is desirable to mine and store results of frequent sequences and use these results to generate frequent sequential patterns in updated database. But it is expensive to save all mining results. To reflect the current state of the database where previous sequential patterns would become irrelevant and new sequential patterns might appear, there is a need for efficient algorithms to update, maintain and manage the information discovered.

If the transactions from an existing customer appear with database updation, the existing sequence corresponding to this customer gets appended with events corresponding to transactions in time $\Delta t$. Also new sequential patterns appear with transactions by new customers who enter the database of transactions in time $\Delta t$. It is utter wastage of computation time and effort if fresh set of sequential patterns have to be refined with every updation. Incremental mining of sequential patterns is about deducing the space of frequent patterns in the updated database by reusing information of previous mining task. Moreover, knowledge of only frequent sequence in database instance $D$ in time $T$ is insufficient for mining patterns in updated database $D'$ in time $T+\Delta t$. The snapshot of the database in time $\Delta t$ is denoted as $\Delta d$. Since there can be many infrequent patterns which become frequent with database updations. If an existing sequence is appended it will cause creation of new subsequence, which might render previous infrequent patterns frequent with the updation. Buffering or storing all the patterns in the database is not a solution since it will cause storage of an exponential number of patterns which would many times exceed the size of the database itself. Storing candidate patterns and full set of frequent patterns also wastes a lot of memory. So, there is need for a method which stores significant mining results in a compressed format and reuses this knowledge reserve to mine further data increment.

In this paper we have presented a novel framework for addressing the problem of incremental mining of sequential patterns. We have used the concept of partitioning of search space using indiscernibility relation from the theory of rough sets to first deduce the space of minimal generators and frequent closed patterns in the database under study. Minimal generators are those patterns which do not have any subpatterns in the database under study, closed patterns are the ones such that they have no super pattern with the same support in the database under study. The knowledge of maximal length infrequent patterns is also stored in pattern-frequency format in an alternate table. A maximal length pattern is the sequence set with highest cardinality. This knowledge base of minimal generators, closed patterns and maximal patterns with their occurrence frequency is used to deduce the space of frequent patterns in the updated database. The algorithm I-PCI is based on algorithm PCI [6] which deduces closed patterns and their
generators in user defined time constraints and outperformed ISE by 3 to 7 times on execution time depending on the size of the database.

2.0 Related Research

Various computational methods for incremental mining of sequential patterns are proposed by various authors in recent literature. Most of these algorithms are based on methods for mining sequential patterns in [5,11,16]. Sequential pattern mining find patterns of antecedent is followed by consequent form in order of time. Suffix tree [2] is a technique that creates a suffix tree of frequent sequences. This method is thus very appropriate for incremental sequence extraction; because it only has to continue the data reading after the update. The method however suffers from the drawback of huge memory overhead and effectiveness of the method depends on the size of the frequent pattern space. FASTUP [2] is based on GSP [11]. It is yet another candidate generates test strategy and uses information of support threshold of patterns to generate new candidates and validate them in updated database. Parthsarthy et.al [10] have proposed ISM a method based on SPADE [16] and negative border. The algorithm builds a lattice of frequent patterns and buffers possible candidate sequences in negative border. This negative border is made of sequences which are not frequent but which are generated by frequent subsequences. When the database is incremented, some elements from negative border go to the bucket of frequent elements and the ones which are found infrequent on updations are pruned. Fresh candidates are added to the negative border, which are generated by joining or extending existing frequent patterns. The drawback of this method is that the candidate set can be huge. Also many of these candidates will have no existence in the increment under consideration.

Masseglia [2] gave an algorithm ISE (Incremental Sequence Extraction) which is candidate generate test method based on GSP [11]. It first mines the full set of frequent sequences in the database under study by employing GSP. In addition to the knowledge of frequent sequences it stores the following set of patterns:
1. Frequent 1 sequence embedded in $\Delta d$ and validated on $D'$
2. Candidate sequences generated from $\Delta d$
3. Frequent sequences obtained from candidates in $\Delta d$ and validated on $D'$
4. FreqSeed set which stores all frequent subsequences of $D$ extended with and item from length 1 patterns in $\Delta d$
5. CandInc contains all candidate sequences generated by appending sequences obtained in 3 and 4
6. FreqInc is the set of all frequent sequences obtained from CandInc and validated on $D'$

The method ISE suffered from the drawback of generation of a huge set of candidates which have to be checked for being frequent with the increment. Also its levelwise search causes multiple scans of database, which can be costly in case of long sequences. The method is superior to ISM [10] as only those candidates
are considered which have at least a single occurrence in db. Also there is some amount of pruning by non considerations of patterns infrequent in original database D. IncSpan [3] is based in PrefixSpan [5]. It is based in the concept of buffering semi-frequent patterns in the DB. Hong et al. [3] defined the term semi-frequent patterns as: Given a user defined value $0 \leq \mu \leq 1$, a pattern is semi-frequent if its support is $\mu \times \text{minimum support}$ of the patterns in buffered semi-frequent pattern space are used to deduce the frequent pattern in the incremented database. The method is very efficient and outperforms ISE and ISM. The authors have given comprehensive analysis of various situations that arise out the database increment. However, it suffers from the basic drawback of storing too many patterns. Also computational methodology underlying the support accumulation process is PrefixSpan [5] which requires multiple scans of the database and is inefficient for long patterns and large databases.

We identify the following two issues from above literature survey and focus on them in our proposed algorithm. (i) Most of the methods being candidate generate and test methods; lot of computation time and memory is utilized in candidate generation and testing. (ii) A huge set of pattern space needs to be maintained for generation of patterns in the incremented database.

This motivated us to design an algorithm that generates maximum information by utilizing compressed representation of limited set of mining results on D and also reduce the extent to which the database is accessed. Our method is based on alternate storage of support information of closed patterns [4,13] and their minimal generators [10].

### 3.0 Proposed Model and Method

Consider a database instance D formed by transaction database between start time $t_s$ and end time $t_e$. Consider the advancement in time $\Delta t$. The new database instance in time $T+\Delta t$ is represented by $D'$. From theory of rough sets, an information system is given as: $D=\{U,A,V,f\}$ where $U$ is a finite set of objects, $U=\{x_1,x_2,\ldots,x_n\}$ $\bigcup_{p \in A} V_p$ is a domain of attribute $p$ $f: U \times A \rightarrow V$ is a total function such that $f(x,q) \in V_q$ for every $q \in A$ and $x \in U$. Consider an example of a sample sequence database $D$ as in TABLE I. The information system is $D=\{U,E\}$ where $U=\{x_1,x_2,\ldots,x_n\}$ $E=\{e_1,e_2,e_3,\ldots,e_m\}$ is a sequence or serial episode defined as a set of events associated with the object under study ordered in time $\forall e_i \in E$ $e_i = \{i_1,i_2,\ldots,i_l\}$. The length of a sequence is the number of items it contains. A $k$-sequence contains $k$ items $k = \sum |e_i|$. The absolute support of a sequence $e_i$ is defined as the number of transaction that contain it and relative support is defined as $\text{sup} (e_i) = \text{absolute support}/\text{no. of objects in the dataset}$. A
pattern is frequent if it crosses a user specified frequency threshold called minimum support threshold. Given a support threshold \( \text{min}_\text{sup} \) a sequence \( e_\alpha \) is a frequent sequence and there is no supersequence \( s_\beta \) such that \( \text{Support}(e_\alpha) = \text{Support}(e_\beta) \) then the pattern is called closed. A pattern is a minimal generator if it does not have any subpattern in the dataset under study. A sequence \( e_\alpha = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle \) is said to be a subsequence of sequence \( e_\beta = \langle \beta_1, \beta_2, \ldots, \beta_m \rangle \) \( \exists i_1, i_2, i_3, \ldots, i_k \in I \) such that \( \alpha_1 \subseteq \beta_1, \quad \alpha_2 \subseteq \beta_2 \ldots \quad \alpha_n \subseteq \beta_m \).

Property 1: [18] If a pattern \( e_\alpha \) is a subpattern of \( e_\beta \), \( e_\alpha \subseteq e_\beta \) then \( \text{Support}(e_\alpha) \geq \text{Support}(e_\beta) \).

Consider the advancement in time \( \Delta t \). The new database instance in time \( T + \Delta t \) is represented by \( D' \). \( D' = \{U', A', V', f'\} \) the problem of incremental mining of sequential pattern is about deriving the frequent space in \( D' \) from information of frequent patterns in \( D \). Clearly \( U \subseteq U' \) if in any case \( U = U' \) it implies that there are no transactions from new objects and only existing sequences are appended with transaction from objects in \( U \). Example of \( D, D' \) are depicted in TABLE I and TABLE II.

**TABLE I Database D**

<table>
<thead>
<tr>
<th>Object Id</th>
<th>Sequence of events</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>1 4 1</td>
</tr>
<tr>
<td>x2</td>
<td>2 2 1 1</td>
</tr>
<tr>
<td>x3</td>
<td>2 1 1</td>
</tr>
<tr>
<td>x4</td>
<td>1 2 1 1</td>
</tr>
<tr>
<td>x5</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>x6</td>
<td>1 2 3 2</td>
</tr>
<tr>
<td>x7</td>
<td>1 6 *</td>
</tr>
</tbody>
</table>

**TABLE II Database D'**

<table>
<thead>
<tr>
<th>Object Id</th>
<th>Sequence of events</th>
<th>Appended sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>1 4 1</td>
<td></td>
</tr>
<tr>
<td>x2</td>
<td>2 2 1 1</td>
<td></td>
</tr>
<tr>
<td>x3</td>
<td>2 1 1</td>
<td></td>
</tr>
<tr>
<td>x4</td>
<td>1 2 3 4</td>
<td></td>
</tr>
<tr>
<td>x5</td>
<td>1 2 3 2</td>
<td></td>
</tr>
<tr>
<td>x6</td>
<td>1 6 *</td>
<td></td>
</tr>
<tr>
<td>x7</td>
<td>1 6 *</td>
<td></td>
</tr>
</tbody>
</table>

Theorem 1: The set of frequent generators and closed patterns is sufficient to deduce the frequent pattern space.

Proof: From Property 1 the support count of any pattern in a partition created prefix indiscernibility will definitely lie between the support of its generator and closed pattern. Moreover, if a subpattern has the same support as that of its superpattern it is absorbed by it. Also if the support of any pattern is higher than its superpattern and superpattern is also frequent then both patterns join the set of frequent closed patterns.

Illustrative Example: Consider the example database given in TABLE I. Method PCI [6] for deducing the space of frequent closed patterns and generators uses multiple definitions of “concept” to partition the search space of frequent generators and closed sequential patterns into various equivalence classes. The method assumes that the database (data structures in which the database elements are fetched) can fit into main memory. This assumption is realistic since in modern days the main memory is in gigabytes and it is increasing with advancements in hardware technology. The method uses a tree traversal algorithm.
and uses the apriori based pruning technique which states that “any superpattern of an infrequent pattern cannot be frequent” in each partition (equivalence class) created by a concept:

1. Scan the sequence database in user defined time constraints and store the database instance in a data structure
2. Generate frequent minimal generators:
   This is done by storing all unique items in alternate data structure and applying indiscernibility mapping between the two data structures. The data structure \( M = \{1, 2, 3, 4, 6, 7\} \) and datastructure \( U \) having dataset \( D \). To mine the frequent minimal generators we define a concept as concept1 “all transactions that contain the minimal generator \( m_i \) form an equivalence class”. The support of a minimal generator will be the number of elements that exist in the equivalence class satisfying the condition in the concept1. If the minimum support threshold is 2,

The resultant table of frequent generators is stored in a knowledge base \( K \).

Table III depicts the elements in the knowledge base after this step.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

From Table III it is evident that the knowledge base is a vertical representation of data in Pattern vs. Support format.

Lower and upper approximation [12]: let \( K = \{U, R\} \) we define for each subset of \( x_i \subseteq X \) and an equivalence relation \( R \in \text{IND}(K) \)

\[
RX = \bigcup \{x_i \in X : R : x_i \subseteq X\} \quad --- (1)
\]

\[
\overline{RX} = \bigcup \{x_i \in X : R : x_i \cap X \neq \emptyset\} \quad --- (2)
\]

are called R-lower and R-upper approximation of \( X \) respectively [14]. The set \( RX \) is set of all the elements of \( U \) which can be with certainly classified as elements of \( X \) in the knowledge \( R \) and the set \( \overline{RX} \) is the set of elements in \( U \) which can be possibly classified as elements of \( X \) employing knowledge \( R \).

Considering the knowledge base \( K \) an equivalence relation on frequent patterns is created on the basis of support of patterns. There is a classification of data in lower and upper approximation in the knowledge base on the basis of support of patterns.

Theorem 2: An equivalence class in \( K \) formed by Set of all frequent generators is \( RX \) and called the positive region in \( X \) and is denoted by \( \text{POS}_R(X) \).

Proof: Any frequent generator in \( D \) would definitely be frequent in \( D' \). Also any infrequent or partially frequent pattern which is an extension of a sequence in \( D \)
and becomes frequent on database updation will definitely have a frequent generator in Knowledge base K.

3. To derive frequent closed patterns in D we build V the domain set of event attribute. The set V encapsulates all possible subpatterns encapsulated in D. Due to apriori property all frequent subsequences would be some pattern combinations of frequent generators. To Generate the frequent pattern space from D we partition V into various equivalence classes on the basis of concept 2 given as: All patterns with the same prefix generator belong to one equivalence class.

4. This partitions the set V such a way that each partition represents a lexicographical tree of patterns arranged in increasing order of pattern cardinality. Thus there are as many partitions in V as the number of unique generators in M. If there are k frequent generators then there are k partitions in the database. Each tree branch is a specialization and the union of all data in all categories is the generalization of concept in V.

\[ V = \bigcup_{i=1}^{k} y_i \]

\[ \text{Ind}(\text{node}_i) = \{ x_i, x_j \in D : \forall \text{node}_i(x_i) \subseteq e_j \} \]

\[ \text{Ind}(1, 4) = \{ x_1, x_3, x_4 \} \]

\[ \text{node}_i, \text{node}_{i+1}, \text{node}_j, \text{node}_{j+1} \]

From the theory of rough sets a partition should satisfy the following three conditions:

Condition 1: \( y_i \neq \emptyset \)

Condition 2: \( y_i \cap y_j = \emptyset \)

Condition 3: \( V = \bigcup_{i=1}^{k} y_i \)

Theorem 3: The equivalence class formed by concept 2 form a partition in V.

Proof: Condition 1 of partitioning is satisfied since each branch will correspond to one frequent generator. Condition 2 is also satisfied since partitions are built on the basis of common prefix in strict lexicographical order. Condition 3 is also satisfied since the partitions are derived from V and all patterns will definitely be a node of some branch.

Running Example: Figure 1.0 represent the tree representation of partitions in clearly each branch with elements in strict lexicographical order represent a partition.
Counting instance scheme PCI [6] for nodes in the tree: Again we revisit the indiscernibility relation from the theory of rough sets. Here we reformulate the concept of equivalence class. Now we find elements which are indiscernible on the basis of pattern existence. This is indiscernibility mapping defined as: Let \( x_i, x_j \in D \) the following defines a binary relation called Indiscernibility relation:

\[
\text{Ind}(node_i) = \{x_i, x_j \in D^2 : \forall node_i(x_i) \subseteq c_j \}
\]  

--- (4)

It can be said that \( x_i, x_j \) are indiscernible by a set of attributes \( E \) in \( S \) if and only if the pattern at node \( i \) is contained in transaction \( x_j \). Clearly, \( D \times D \) operation is performed on two alternate representations of \( D \). We traverse through each branch of the tree in a depth first search manner and apply counting instance scheme. The support of each node is the number of members in its indiscernibility relation.

Illustrative Example:

Suppose we read the node “1,4” the indiscernibility relation corresponding to “1,4” is \( \text{Ind}(1, 4) = \{x_1, x_3, x_4\} \). Thus the support of this pattern is 3 which is the number of elements in its equivalence class.

If any pattern at node \( i \) is found to be infrequent all its lexicographically super sequences are pruned due to apriori property. And the search continues in the next branch till we derive the frequent pattern space. Simultaneously we apply the closure check to deduce the space of frequent closed patterns. Method closure check is explained in the following steps:

6. If in the frequent pattern tree, the support of \( node_i \) in a branch is greater than support at \( node_{i+1} \) then both \( node_i, node_{i+1} \) patterns join the set of frequent closed patterns.

7. If in the frequent pattern tree, if \( \text{support}(node_i) = \text{support}(node_{i+1}) \) then pattern at \( node_{i+1} \) joins the closed pattern set. This is due to the property that if the support of a pattern is equal to that of its superpattern it is absorbed in its superpattern.
8. We derive a tentative set of closed patterns in each specialization (partition). Now we form the generalization which is the union of all the patterns in each specialization. This is required due to the reason that, we formulate partition (specialization) using concept1 that all subpatterns with the same generator prefix and in lexicographical order are in one equivalence class. Thus, there are subpatterns with different generator prefixes which are not directly in lexicographic order in other equivalence classes. For example, in the frequent pattern tree of example illustrated above both patterns “1,4” and “1,4,1” join the set of closed pattern. Now we form a generalization of patterns by union of all patterns, note that there are subpatterns of patterns at various branches in the tree for example the closed patterns “4,1” and “1,1” also enters the closed pattern set in this case since it is in a different branch then its superpattern. Form the generalization of frequent pattern space we check if there are any non closed patterns which can be absorbed into their superpatterns. This is found by evaluating subpatterns of higher cardinality members and checking their frequency against their subpatterns in tentative closed pattern space. The revised closed frequent set is derived by pruning all subpatterns with the same support as their higher cardinality superpatterns. The result is full set of closed patterns with their generators in user defined time interval.Set of all closed patterns is also stored in the knowledge base in pattern-support format. In addition we also store the set of maximal infrequent pattern in the knowledge base as they might get extended and join the frequent pattern space with the increment.Also other infrequent subpatterns are can be encapsulated in them as a results if an infrequent subpattern of a maximal pattern finds sufficient support in $\Delta d$ it becomes frequent and joins the knowledge base as either a closed patterns or gets absorbed in an existing closed patterns with increment in respective support in the knowledge base.

Any of the following situations arise when the database is incremented [3]:

1. A pattern frequent in D is still frequent in D’
2. A patterns which is partially frequent in D becomes frequent in D’
3. A pattern which is partially frequent in D is still partially frequent in D’
4. Appended database bring new items
5. A pattern infrequent in D becomes frequent in D’
6. A pattern infrequent in D becomes semi-frequent in D’

Theorem 4: Infrequent maximal patterns are members of $RX$ approximation of frequent pattern space.

Proof: Under case 5 the members of lower approximation of the frequent sequence space can possibly become frequent with the increment in the database. These members in this case are promoted to the $\text{Pos}(X)$ in this case. Also they might contribute to support of infrequent subsequences in D becoming frequent with increment. Situation under case 6 is a member of the lower approximation is still the member of the lower apporimation.

3.1 Incremental mining with I-PCI:
Step 1: Read the database instance $\Delta d$. This can be done by querying the database instance in $\Delta t$ time period.
Step 2: Find length 1 patterns and accumulate their support as in PCI [6]. Also find support all the maximal patterns in $\Delta t$.
Step 3: Query $D'$ for only the appended sequences. Generate subsequences, and check their support by mapping the patterns with their counterpart patterns in the knowledge base. If equivalent pattern does not exist in the knowledge base we check for the presence of its super pattern. For example, the subsequence “4,1” generated by portion of incremented database, its super pattern “1,4,1” exist in the knowledge base with support 2. It is clear that pattern “4,1” also has the same support since in other wise situation it would not be absorbed. Thus support of the pattern “4,1” in $D'$ is now 3. Subsequences which are infrequent in $\Delta d$ and have no super patterns in the knowledge base can be safely pruned since there is no chance of their joining the frequent pattern space.
Step 4: For new elements either they will be frequent in $\Delta d$ or they are subpatterns members of the $R_X$ in this case their supports are accumulated on basis of information of their counterpart patterns or super patterns in the knowledge base. Patterns which are still infrequent can be safely pruned from $R_X$.
Step 5: Knowledge base instance after step 3 contains compressed representations of full set of frequent sequential patterns in $D'$. The full pattern space can be found with ease due to property 1 and characteristics of closed pattern set. For example subpatterns of frequent closed patterns can be deduced as patterns with same support as that of its closed pattern.

The proposed method visits the increment to read only the appended patterns characterized by same identification as the object in D. This contributes to the efficiency of the method since the increment in small interval $\Delta t$ is often small.

4. Experimental Results of algorithm efficiency

Our method is comparable with only ISE in the class of methods for incremental mining of sequential patterns since it incorporates time constraint due to logic built on GSP [11]. Our method is based on PCI [6] which also enables time window adjustment. We have implemented both I-PCI and ISE in java- JDK1.3. The algorithm uses the java database connectivity interface to the back end used is MSSQL Server 2005. The sequence database is fetched in java data structures and computation logic of indiscernibility is built on the same. The machine used is HP Proliant DL580G5 with Intel Xeon CPU 1.6 GHZ processor with 8 GB RAM. The operating system is Ms Windows Server 2003 R2. We have used java’s core strength of reuse and harnessed the strength of search methods on java’s data structures Treeset and ArrayList. The search methods in the above data structures give rapid results in log (n) time where n is the number of elements. Thus finding all elements in database of size N will take Nlog (n) computational time to build treeset of frequent itemsets.
A rough sets partitioning model

We have tested the efficiency of our method on real and synthetic datasets. The synthetic datasets are similar to ones generated by the synthetic data generation program available at http://www.almaden.ibm.com/cs/quest. The following are the descriptions of the parameters of the dataset.

|D| size of the database (number of customers)
|C| Average number of transactions per customer
|I| Average size of itemset in maximal potentially large sequence
|N| Number of items

On synthetic datasets our algorithm was about 3-7 times faster than ISE. The runtime comparison of Incremental sequence extraction with (I-RSP) Incremental Rough Sets partitioning on synthetic dataset C15-I1-N15-D400 is as in Figure 2.0 The graphs clearly indicated I-RSP outperforms ISE by over 7 times in execution times. Due to logic built on subsequence evaluation, its performance degrades on very long sequences. Another example of runtime comparison is given in Figure 3.0 We have also implemented our algorithm on real data of network faults in telecom landline network of madhya pradesh state in India.

5. Conclusion

The following are the benefits of proposed model:
(i) Due to logic built on partitioning our algorithm looks for patterns in only the specified equivalence classes.
(ii) There is no buffering of patterns as in case of IncSpan.
(iii) There is no storage of huge candidate set as in case of ISE and ISM
(iv) The number of database scans is also limited to one or two while, GSP[11] the underlying mechanism of ISE recursively scans the database for enumeration of frequent sequence space.
(v) Compressed representation of mining results in alternate data table improves performance
Partitioning of knowledge base into upper and lower approximation reduces the search space of patterns.

Acknowledgment. We are grateful to Department of Biotechnology, New Delhi and MP Council of Science and Technology for providing infrastructure facility for MANIT Bhopal for carrying out this research work. We are also grateful to BHARTI AIRTEL LTD. (LANDLINE) Bhopal for giving us support for implementing our algorithm.

References


[3] Hong Cheng, Xifeng Yan, Jiawei Han “Incspan: Incremental Mining of Sequential Patterns in Large Databases” in Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining Seattle, WA, USA 2004, 527 – 532,


[5] Jian pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Qiming Chen, Umeshwar Dayal, Mei-Chun Hsu, “Mining Sequential Patterns-Growth: The PrefixSpan Approach”, IEEE Transactions on Data and knowledge engineering vol 16,no.11, 2004,1424-1440


[8] Li, H. Li, L. Wong. J.Pei and G. Dong “Minimum description length principle: Generators are preferable to closed patterns In AAAI, 2006
A rough sets partitioning model


[13] Xifeng Yan, Jiawei Han, Ramin Afshar, ”CloSpan: Mining closed Sequential patterns in Large datasets” In proceeding SIAM, International conference on Data mining SDM ’03 San Francisco, CA May 2003


Received: October, 2009