Sequential Pattern Analyzer
Research Report

Mudrageda Venkata Sesha Sowjanya
940329-5703
Vemu15@student.bth.se

Siva Venkata Prasad Patta
9312211-7137
sipa15@student.bth.se

I. GROUP MEMBER'S PARTICIPATION

Group Member	Idea Creation	Report writing
Mudrageda Venkata Sesha Sowjanya	50%	50%
Siva Venkata Prasad Patta	50%	50%

Abstract:

The following research report deals with the data analytics in sequential patterns, association rules and vertical algorithms. To solve the problems regarding the transactions, finding all the association rules with confidence and support greater than or equal to specify thresholds. Pruning the unwanted candidates to the maximum extinct. Making use of Spade, Clasp, Cm-Clasp and Cm-Spade algorithms and making use of JFree Chart for comparing the algorithms Cm-Clasp and Cm-Spade give results with full confidence while it prunes many unwanted candidates. Co-occurrence map is the best type of data type to use while the patterns are huge during sequential analysis and algorithms using Co-occurrence map can solve the problem of finding all the related association rules.

I. Introduction

Data Mining:

Generally, data mining (sometimes called data or knowledge discovery) is the process of analyzing data from different perspectives and summarizing it into useful information - information that can be used to increase revenue, cuts costs, or both. Data mining software is one of a number of analytical tools for analyzing data. It allows users to analyze data from

Data mining majorly consists the following steps:
1) Extract, transform, and load transaction data onto the data warehouse system.
2) Store and manage the data in a multidimensional database system.
3) Provide data access to business analysts and information technology professionals.
4) Analyze the data by application software.

Association rules:

What are they?

- Looking for common causal relationships in basket data

Where are they used?

- Store layout
- Catalog design
- Customer segmentation

Many different dimensions or angles, categorize it, and summarize the relationships identified. Technically, data mining is the process of finding correlations or patterns among dozens of fields in large relational databases.

Sequential Pattern Mining:

What is it?
Given a set of events, find frequently occurring patterns

Where is it used?
Analyzing basket data
Medical diagnosis

Sequential Pattern mining is a topic of data mining concerned with finding statistically relevant patterns between data examples where the values are delivered in a sequence. [1]It is usually presumed that the values are discrete, and thus time series mining is closely related, but usually considered a different activity. Sequential pattern mining is a special case of structured data mining.

There are several key traditional computational problems addressed within this field. These include building efficient databases and indexes for sequence information, extracting the frequently occurring patterns, comparing sequences for similarity, and recovering missing sequence members.

In general, sequence mining problems can be classified as string mining which is typically based on string processing algorithms and item set mining which is typically based on association rule learning. String mining typically deals with a limited alphabet for items that appear in a sequence, but the sequence itself may be typically very long.

The Problems:

Consider the situation where in we need to analyze a user's browsing history in order to develop or improvise a recommender system, then the major requirements could possibly be getting hash codes from the log file sets, arranging them accordingly in database, mapping the url's to numbers such that algorithm takes a easy convention, joining them with the programming code for taking the urls as parameters and implementing the algorithm.

a. Given a set of transactions, finding all the association rules with confidence and support greater than or equal to specified thresholds is a problem. In the huge set of repeated patterns and the databases, where in there exists very minute differences from one to another items and item sets, it is almost impossible to find the user specific appropriate pattern which represents the resources used by the user.

b. The vertical algorithms generate a large amount of infrequent candidates hence pruning the unwanted candidates is another major issue with respect to mining of the sequential patterns. The algorithms like Spade and Clasp though use vertical databases(which majorly depends on the schema of the table of a relational database),they generate a huge number of unwanted, infrequent candidates for a particular minimum support, for the expected confidence.

Eventually the memory being wasted mounts up than the results being stored.

Objectives:

A. To understand what type of data sets deals with the sequential patterns with the atmost confidence prescribed for a given sequence.

B. To analyze what kind of algorithms works for different types of sequential data sets.

C. To use co-occurrence map efficiently in order to get the least response time and also to prune the unnecessary candidates during candidate generation.

Methods:

The methods being used are data mining algorithms namely Cm-Clasp and Cm-Spade (using core Java) and we try to graphically represent which algorithm works better for a given data set using a visualization tool call JFreeChart.

Contributions:

The algorithms being used is modified according to the dataset being used, the UI is designed, keeping in view the user's perspective. The output chart which shows the comparative description about the turnaround time with respect to the minimum support value rendered. This involves embedding JFree library, involves usage of many predefined interfaces and shows an XY graph of JFree chart.

II. Background and Motivation

Definition 1 (sequence database). Let I = {i1, i2, ..., il} be a set of items
(symbols). An itemset Ix = {i1, i2, ..., im} ⊆ I is an unordered set of distinct items. The lexicographical order lex is defined as any total order on I. Without loss of generality, it is assumed in the following that all itemsets are ordered according to lex. A sequence is an ordered list of itemsets s = I1, I2, ..., In such that Ik ⊆ I (1 ≤ k ≤ n). A sequence database SDB is a list of sequences SDB = s1, s2, ..., sp having sequence identifiers (SIDs) 1, 2...p. Example. A sequence database is shown in Fig. 1 (left). It contains four sequences having the SIDs 1, 2, 3 and 4. Each single letter represents an item. Items between curly brackets represent an itemset. The first sequence {a, b}, {c}, {f,g}, {g}, {e} contains five itemsets. It indicates that items a and b occurred at the same time, were followed by c, then f,g and lastly e.

SID	Sequences
1	{{a, b},{c},{f, g},{g},{e}}
2	{{a, d},{c},{b},{a, b, e, f}}
3	{{a},{b},{f},{e}}
4	{{b},{f, g}}

ID	Pattern	Support
p1	{{a},{f}}	3
p2	{{a},{c}{f}}	2
p3	{{b},{f,g}}	2
p4	{{g},{e}}	2
p5	{{c},{f}}	2
p6	{{b}}	4

A sequence database (left) and some sequential patterns found (right)

Definition 2 (sequence containment). A sequence sa = A1, A2, ..., An is said to be contained in a sequence sb = B1, B2, ..., Bm iff there exist integers 1 ≤ i1 < i2 < ... < in ≤ m such that A1 ⊆ Bi1, A2 ⊆ Bi2, ..., An ⊆ Bin (denoted as sa sb). Example. Sequence 4 in Fig. 1 (left) is contained in Sequence 1.

Definition 3 (prefix). A sequence sa = A1, A2, ..., An is a prefix of a sequence sb = B1, B2, ..., Bm, ∀n<m, iff A1 = B1, A2 = B2, ..., An−1 = Bn−1 and the first |An| items of Bn according to lex are equal to An.

Definition 4 (support). The support of a sequence sa in a sequence database SDB is defined as the number of sequences s ∊ SDB such that sa s and is denoted by supSDB(sa).

Definition 5 (sequential pattern mining). Let minsup be a threshold set by the user and SDB be a sequence database. A sequence s is a sequential pattern and is deemed frequent iff supSDB(s) ≥ minsup. The problem of mining sequential patterns is to discover all sequential patterns [11].

Definition 6 (closed sequential pattern mining). A sequential pattern is a is said to be closed if there is no other sequential pattern sb, such that sb is a superpattern of sa, sa sb, and their supports are equal. The problem of closed sequential patterns is to discover the set of closed sequential patterns, which is a compact summarization of all sequential patterns [7,12].

Definition 7 (horizontal database format). A sequence database in horizontal format is a database where each entry is a sequence.

Definition 8 (vertical database format). A sequence database in vertical format is a database where each entry represents an item and indicates the list of sequences where the item appears and the position(s) where it appears.

Once the candidate generator generates the candidates and if it turns ineffective during pruning then the outcome is enormously large amount of itemsets and items which are very much similar to each other and thus the confidence of accessing a single pattern become hideous. Pattern matching for a human mind is called common sense but a machine apparently a system which purely works on bits and byte cannot understand the whole purpose of the algorithm, hence it is programmed into the machine in order to lessen the burden oven a human mind thus making it artificially intelligent.

All these standards are meant for analyzing the state of art algorithms regarding sequential patterns. The problems of big data and data analytics majorly depend on pruning the unnecessary data generated by the data generation algorithm and the compiler. Hence majority of the candidates gets pruned during the processing of the algorithm. Hence if the candidates are not pruned accordingly then the stack overflows and the program crashes.

III. Research Definition and Plan

According to the requirement specification, it is quite necessary to understand the concept of mining a sequence, a pattern, an item set and ultimately an item.
Hence we need to analyze "what" and "how" in order to decide whether the road map which we choose is going right way or not!

The basic questions which come forward while dealing with the analysis is:

1. Does the performance of the algorithms be altered over a range of algorithms?
2. What are sequential algorithms?
3. What are vertical and horizontal data based algorithms?
4. How to bind the UI and the algorithms?
5. Which visualization tool to use?
6. How to design the whole structure?

Units of analysis majorly are the turnaround time and the minimum support which is the independent variable since we try to assign a sequence with a confidence which is apparently termed as minimum support.

The analysis majorly depends on what type of data is being processed. The algorithms chosen i.e., Cm-Clasp and Cm-Spade majorly deals with the user navigational data. Hence the prerequisite is to know types of data sets being used.

Then the performance needs to analyzed using a standard, here the standard could possibly be the minimum support and representing it on JFeeChart.

IV. Research Operation

The operation first starts by analyzing the algorithms Cm-Spade and Cm-Clasp. Even before we start learning about the algorithms, the very essence lies in the tem 'Çm' i.e., Co-occurrence map.

The Co-occurrence Map:

Definition 9. An item k is said to succeed by i-extension to an item j in a sequence I1, I2, ..., In iff j, k ∊ Ix for an integer x such that 1 ≤ x ≤ n and k lex j.

Definition 10. An item k is said to succeed by s-extension to an item j in a sequence I1, I2, ..., In iff j ∊ Iv and k ∊ Iw for some integers v and w such that 1 ≤ v<w ≤ n.

Definition 11. A Co-occurrence MAP (CMAP) is a structure mapping each item k ∊ I to a set of items succeeding it. We define two CMAPs named CMAPi and CMAPs. CMAPi maps each item k to the set cmi(k) of all items j ∊ I succeeding k by i-extension in no less than minsup sequences of SDB. CMAPs maps each item k to the set cms(k) of all items j ∊ I succeeding k by s-extension in no less than minsup sequences of SDB. Example. The CMAP structures built for the sequence database of Fig. 1(a) are shown in Table 1, being CMAPi on the left part and CMAPs on the right part. Both tables have been created considering a minsup of two sequences. For instance, for the item f, we can see that it is associated with an item, cmi(f) = {g}, in CMAPi, whereas it is associated with two items, cms(f) = {e, g}, in CMAPs. This indicates that both items e and g succeed to f by s-extension and only item g does the same for i-extension, being all of them in no less than minsup sequences.

$CMAP_i$	
items	is succeeded by (i-extension)
a	{b}
b	∅
c	∅
e	∅
f	{g}
g	∅

$CMAP_s$	
item	is succeeded by (s-extension)
a	{b,c,e,f}
b	{e,f,g}
c	{e.f}
e	∅
f	{e,g}
g	∅

Table 1. CMAPi and CMAPs for the database of Fig. 1 and minsup = 2

Size Optimization. Let n = |I| be the number of items in SDB. To implement a CMAP, a simple solution is to use an n×n matrix (two-dimensional array) M where each row (column) correspond to a distinct item and such that each entry mj,k ☐ M represents the number of sequences where the item k succeed to the item i by i-extension or s-extension. The size of a CMAP would then be O(n^2). However, the size of CMAP can be reduced using the following strategy. It can be observed that each item is succeeded by only a small subset of all items for most datasets. Thus, few items succeed another one by extension, and thus, a CMAP may potentially waste large amount of memory for empty entries if we consider them by means of a n × n matrix. For this reason, in our implementations we instead implemented each CMAP as a hash table of hash sets, where an hashset corresponding to an item k only contains the items that succeed to k in at least minsup sequences.

Co-occurrence-Based Pruning:

The CMAP structure can be used for pruning candidates generated by vertical sequential pattern mining algorithms based on the following properties.

Property 1 (pruning an i-extension):

Let be a frequent sequential pattern A and an item k. If there exists an item j in the last itemset of A such that k belongs to cmi(j), then the i-extension of A with k is infrequent. Proof. If an item k does not appear in cmi(j), then k succeed to j by i-extension in less than minsup sequences in the database SDB. It is thus clear that appending k by i-extension to a pattern A containing j in its last itemset will not result in a frequent pattern.

Property 2 (pruning an s-extension):

Let be a frequent sequential pattern A and an item k. If there exists an item j ∈ A such that the item k belongs to cms(j), then the s-extension of A with k is infrequent. Proof. If an item k does not appear in cms(j), then k succeeds to j by s-extension in less than minsup sequences from the sequence database SDB. It is thus clear that appending j by s-extension to a pattern A containing k will not result in a frequent pattern.

Property 3 (pruning a prefix):

The previous properties can be generalized to prune all patterns starting with a given prefix. Let be a frequent sequential pattern A and an item k. If there exists an item j ∈ A (equivalently j in the last itemset of A) such that there is an item k ∈ cms(j) (equivalently in cmi(j)), then all super sequences B having A as prefix and where k succeeds j by s-extension (equivalently i-extension to the last itemset) in A in B are infrequent. Proof. If
an item k does not appear in cms(j) (equivalently cmi(j)), therefore k succeeds to j in less than minsup sequences by s-extension (equivalently i-extension to the last itemset) in the database SDB. It is thus clear that no frequent pattern containing j (equivalently j in the last itemset) can be formed such that k is appended by s-extension (equivalently by i-extension to the last itemset).

Candidate Generation in SPADE:

The SPADE procedure takes as input a sequence database SDB and the minsup threshold. SPADE first constructs the vertical database V (SDB) and identifies the set of frequent sequential patterns F1 containing frequent items. Then, SPADE calls the ENUMERATE procedure with the equivalence class of size 0. An equivalence class of size k is defined as the set of all frequent patterns containing k items sharing the same prefix of k − 1 items. There is only an equivalence class of size 0 and it is composed of F1. The ENUMERATE procedure receives an equivalence class F as parameter. Each member Ai of the equivalence class is output as a frequent sequential pattern. Then, a set Ti, representing the equivalence class of all frequent extensions of Ai is initialized to the empty set. Then, for each pattern Aj ∈ F such that i lex j, the pattern Ai is merged with Aj to form larger pattern(s). For each such pattern r, the support of r is calculated by performing a join operation between IdLists of Ai and Aj .
If the cardinality of the resulting IdList is no less than minsup, it means that r is a frequent sequential pattern. It is thus added to Ti. Finally, after all pattern Aj have been compared with Ai, the set Ti contains the whole equivalence class of patterns starting with the prefix Ai. The procedure ENUMERATE is then called with Ti to discover larger sequential patterns having Ai as prefix. When all loops terminate, all frequent sequential patterns have been output SPADE and SPAM are very efficient for datasets having dense or long sequences and have excellent overall performance since performing join operations to calculate the support of candidates does not require scanning the original database unlike algorithms using the horizontal format. For example, the well-known Pre-fixSpan algorithm, which uses the horizontal format, performs a database projection for each item of each frequent sequential pattern, in the worst case, which is extremely costly. The main performance bottleneck of vertical mining algorithms is that they use a generate-candidate-and-test approach and therefore spend lot of time evaluating patterns that do not appear in the input database or are infrequent. In the next section, we present a novel method based on the study of item co-occurrence information to prune candidates generated by vertical mining algorithms to increase their performance.

Integration in SPADE:

The integration in SPADE is done as follows. In the ENUMERATE procedure, consider a pattern r obtained by merging two patterns Ai = P ∪ x and Aj = P ∪ y, being P a common prefix for Ai and Aj . Let y be the item that is appended to Ai to generate r. If r is an i-extension, we use the CMAPi structure, otherwise, if r is an s-extension, we use CMAPs. If the last item a of r does not have an item x ∈ cmi(a) (equivalently in cms(a)), then the pattern r is infrequent and r can be immediately discarded, avoiding the join operation to calculate the support of r. This pruning strategy is correct based on Properties 1, 2 and 3.

Note that to perform the pruning in SPADE, we do not have to check if items of the prefix P are succeeded by the item y ∈ Aj . This is because the items of P are also in Aj . Therefore, checking the extension of P by y was already done, and it is not necessary to do it again.

```
SPADE(SDB, minsup)
1.  Scan SDB to create V(SDB) and identify $F_1$ the list of frequent items.
2.  ENUMERATE($F_1$).
```

```
ENUMERATE(an equivalence class F)
1.  FOR each pattern $A_i \in F$
2.      Output $A_i$
3.      $T_i := \emptyset$.
4.      FOR each pattern $A_j \in F$, with $j \geq i$
5.          $R$ = MergePatterns($A_i$, $A_j$)
6.          FOR each pattern $r \in R$
7.              IF $sup(R) \geq minsup$ THEN
8.                  $T_i := T_i \cup \{R\}$;
9.      ENUMERATE($T_i$)
```

In this section, we formulate and explain every step of our ClaSP algorithm.
ClaSP has two main phases. The first one generates a subset of FS (and superset of FCS) called Frequent Closed Candidates (FCC), that is kept in main memory; and the second step executes a post-pruning phase to eliminate from FCC all non-closed sequences to finally obtain exactly FCS.

```
Algorithm 1. ClaSP
1: $\mathcal{F}_1$ = {frequent 1-sequences}
2: $\mathcal{FCC} = \emptyset$, $\mathcal{FCS} = \emptyset$
3: for all $i \in \mathcal{F}_1$ do
4:     $\mathcal{F}_{ie}$ = {frequent 1-sequences greater than i}
5:     $\mathcal{FCC}_i$=DFS-PRUNING(i,$\mathcal{F}_1$,$\mathcal{F}_{ie}$)
6:     $\mathcal{FCC} = \mathcal{FCC} \cup \mathcal{FCC}_i$
7: end for
8: $\mathcal{FCS}$ = N-ClosedStep($\mathcal{FCC}$)
Ensure: The final closed frequent pattern set $\mathcal{FCS}$
```

Algorithm 1, *ClaSP*, shows the pseudocode corresponding to the two main steps. It first finds every frequent 1-sequence, and after that, for all of frequent 1-sequences, the method *DFS-Pruning* is called recursively to explore the cor-responding subtree (by doing a depth-first search). *FCC* is obtained when this process is done for all of the frequent 1-sequences and, finally, the algorithm ends removing the non-closed sequences that appear in *FCC*.

Algorithm 2, *DFS-Pruning*, executes recursively both the candidate genera-tion (by means of i-extensions and s-extensions) and the support checking, re-turning a part of *FCC* relative to the pattern *p* taken as parameter. The method takes as parameters two sets with the candidate items to do s-extensions and i-extensions respectively (S_n and I_n sets). The algorithm first checks if the current pattern *p* can be discarded, by using the method *checkAvoidable* (this algorithm is explained later in algorithm 5). Lines 4-9 perform all the s-extensions for the pattern *p* and keep in S_{temp} the items which make frequent extensions. In line 10, the method *ExpSiblings* (algorithm 4) is called, and there, *DFS-Pruning* is executed for each new frequent s-extensions. Lines 11-16 and 17 perform the same steps, with i-extensions. Finally, in line 19, the complete frequent patterns set (with a prefix *p*) is returned.

To store the patterns in memory, we use a lexicographic sequence tree. The elements in the tree are sorted by a lexicographic order according to extension comparisons (see section 2). In figure 3 we show the associated sequence tree for *FS* in our example and we denote an s-extension with a line, and an i-extension with a dotted line. This tree is traversed by algorithms 1, 2 and 4, using a depth-first traversal.

There are two main different changes added in ClaSP with respect to SPADE:

(1) the step to check if the subtree of a pattern can be skipped (line 3 of algorithm 2), and (2) the step where the remaining non-closed patterns are eliminated (line 6, algorithm 1).

To prune the space search, ClaSP used the method *CheckAvoidable* that is inspired on the pruning methods used in CloSpan. This method tries to find those patterns $p = \langle \alpha\ e_j \rangle$ and $p' = \langle \alpha\ e_i\ e_j \rangle$, such that, all of the appearances of *p* are in those of *p′*, i.e., if every time we find a sequence *α* followed by an item e_j, there exists an item e_i between them, then we can safely avoid the exploration of the subtree associated to the pattern *p*. In order to find this kind of patterns, we define two numbers: 1) *l*(*s*, *p*), is the size of all the suffixes with respect to *p* in sequence s and 2). $I(D_p) = \sum^{n}_{i=1} l(s_i,p)$, the total number of remaining items with respect with respect to *p* for the database *D*, i.e. the addition of all of *l*(*s*, *p*) for every sequence in the database. Using *I*(*D*) and the subsequence checking operation, in algorithm 5, ClaSP checks the equivalence between the *I*(*D*) values for two patterns: Given two sequences, *s* and *s′* , such that $s \preceq s'$, if $I(D_s) = I(D_{s'})$, we can deduce that the support for all of their descendants is just the same.

In algorithm 5 , the pruning phase is implemented by two methods: 1) Back-ward sub-pattern checking and 2) Backward super-pattern checking. The first one (lines 8-10) occurs when we find a pattern which is a subsequence of a pat-tern previously found with the same *I*(*D*) value. In that case, we can avoid exploring this new branch in the tree for this new pattern. The second method (lines 12-16 and 20-24) is the opposite situation and it occurs when we find a pattern that is a super-sequence of another pattern previously found with the same *I*(*D*) value. In this case we can transplant the descendants of the previous pattern to the node of this new pattern.

In figure 4 we show the ClaSP search tree w.r.t. our example without all pruned nodes. In our implementation, to store the relevant branches, we define a hash function with $I(D)$ value as key and the pattern (i.e. the node in the tree for that pattern) as value ($\langle I(D_p), p\rangle$). We use a global hash table and, when we find a sequence p, if the backward sub-pattern condition is accomplished, we do not put the pair $\langle I(D_p), p\rangle$, whereas, if the backward super-pattern condition is true, we replace all the previous pairs $\langle I(D_p), p'\rangle$ (s.t. $p' \preceq p$) with the new one $\langle I(D_p), p\rangle$. If instead we do not find any pattern with the same $I(D)$ value of the pattern p, or those patterns with the same value are not related with p by means of the subsequence operation, we put the pair $\langle I(D_p), p\rangle$ to the global hash table (line 28). Note that when one of the two pruning conditions is true, we also need to check if the support for s and s' is the same since two $I(D_p)$ and $I(D_{p'})$ values can be equal but they do not necessarily have the same support.

We also need to consider all of the $I(D)_s$ for every appearance in a sequence. For instance, in our example shown in table 1, regarding the three first sequences, if we consider just the first $I(D_s)$ for the first appearance, if we have the pattern $\langle(a)(b)\rangle$ in our example, we deduce that $I(D\_{\langle(a)(ab)\rangle}\_) = I(D\_{\langle(a)(b)\rangle}\_)$ (both with value $I(D) = 5$), so we can avoid generating the descendants of $\langle(a)(b)\rangle$ because these are the same as in $\langle(a)(ab)\rangle$. However, we can check that $\langle(a)(bc)\rangle$ is frequent (with support 3), whereas $\langle(a)(abc)\rangle$ is not (support 1). This forces us to count in $I(D_s)$, all the number of items after every appearance.

Finally, regarding the non-closed pattern elimination (algorithm 3), the pro-cess consists of using a hash function with the support of a pattern as key and the pattern itself as value. If two patterns have the same support we check if one contains the other, and if this condition is satisfied, we remove the shorter pattern. Since the support value as key provoke a high number of collisions in the hash table (implemented with closed addressing), we use a $T(D_p) = \sum^{n}_{i=1} id(s_i)$ value, defined as the sum of all sequence ids where a pattern p appears. How-ever, as the equivalence of $T(D_p)$ does not imply the equivalence of support, after checking that two patterns have the same $T(D_p)$ value, those patterns have to have the same support to remove one of them.

Algorithm 2. DFS-Pruning(p, S_n, I_n)

```
Require: Current frequent pattern $p = (s_1, s_2, \ldots, s_n)$, set of items for s-
      extension $S_n$, set of items for i-extension $I_n$
 1: $S_{temp} = \varnothing$, $I_{temp} = \varnothing$
 2: $F_i = \varnothing$, $P_s = \varnothing$, $P_i = \varnothing$
 3: if (not checkAvoidable( $p$, $I(D_p)$ )) then
 4:     for all i $\in S_n$ do
 5:         if ($p = (s_1, s_2, \ldots, s_n, \{i\})$ is fre-quent) then
 6:             $S_{temp} = S_{temp} \cup \{i\}$
 7:             $P_s = P_s \cup \{p\}$
 8:         end if
 9:     end for
10:     $F_i = F_i \cup P_s \cup$
        ExpSiblings($P_s$,$S_{temp}$,$S_{temp}$)
11:     for all i $\in I_n$ do
12:         if ($p = (s_1, s_2, \ldots, s_n \cup \{i\})$ is fre-
            quent) then
13:             $I_{temp} = I_{temp} \cup \{i\}$
14:             $P_i = P_i \cup \{p\}$
15:         end if
16:     end for
17:     $F_i = F_i \cup P_i \cup$
        ExpSiblings($P_s$,$S_{temp}$,$I_{temp}$)
18: end if
19: return $F_i$
Ensure: Frequent pattern set $F_i$ of this node and its siblings
```

Algorithm 3. N-ClosedStep(FCC)

```
Require: A frequent closed candidates set
      $FCC$
 1: $FCS = \varnothing$
 2: A hash table $H$ is created
 3: for all $p \in FCC$ do
 4:     Add a new entry _$T(D_p)$, $p$_ in $H$
 5: end for
 6: for all entry $e \in H$ do
 7:     for all $p_i \in e$ do
 8:         for all $p_j \in e, j > i$ do
 9:             if ($p_i.support() = p_j.support()$)
                then
10:                 if ($p_i$ _ $p_j$) then
11:                     Remove $p_i$ from $e$
12:                 else
13:                     if ($p_j$ _ $p_i$) then
14:                         Remove $p_j$ from $e$
15:                     end if
16:                 end if
17:             end if
18:         end for
19:     end for
20:     $FC_e$ = all patterns $p \in e$
21:     $FCS = FCS \cup FC_e$
22: end for
23: return $FCS$
Ensure: The final closed frequent pattern set
      $FCS$
```

Algorithm 4. ExpSiblings(P, S_s, S_i)

```
Require: The pattern set $P$ which contains all the patterns whose children
      are going to be explore, the set of valid items $S_s$ which gen-erate the
      $P$ set by means of s-extensions, the set of valid items $S_i$ which
      generate the
      $P$ set by means of i-extensions
 1: $F_s = \varnothing$
 2: for all p $\in P$ do
 3:     $I$ = Elements in $S_i$ greater than the last item $e_i$ in $p$
 4:     $F_s = F_s \cup$ DFS-Pruning($p$,$S_s$,$I$)
 5: end for
 6: return $F_s$
Ensure: Frequent pattern set $F_s$ for all of the patterns' siblings
```

Algorithm 5. CheckAvoidable(p, k)

```
Require: a frequent pattern $p$, its hash-key $k$, hash table $H$
 1: $M_k$ = Entries in the hash table with key $k$
 2: if ($M_k = \varnothing$) then
 3:     Insert a new entry _$k$,$p$_ in $H$
 4: else
 5:     for all pair $m \in M_k$ do
```

```
6:        p = m.value()
7:        if (p.support()=p .support()) then
8:            //Backward sub-pattern
9:            if (p _ p ) then
10:               p has the same descendants as
                  p , so p points to p  descendants
11:               return  true
12:           else
13:               //Backward super-pattern
14:               if (p  _ p) then
15:                   p has the same descendants
                      as p , so p points to p de-scendants
16:                   Remove the current entry
                      _k, p _ from H
17:               end if
18:           end if
19:       end if
20:   end for
21:   //Backward super-pattern executed?
22:   if (Pruning method 2 has been accom-plished) then
23:       Add a new entry _k, p_ in H
24:       return  true
25:   end if
26: end if
27: //k does not exist in the hash table or it does exist but the
    present patterns are not related with p
28: Add a new entry _k, p_ in H
29: return  false
Ensure: It answers if the generation of p can be avoided
```

V. Data Analysis and Interpretation

These two standards can thus help in solving the problems being faced lately.

A. Finding a candidate with at most confidence:

Since these are frequent sequential patterns, they keep track of all the data generated items and the amount of diversity between them. These algorithms define a variable called 'minsup' which simply means minimum support which is the key parameter. It is taken interms of percentage of the total population of the generated candidates or can be given interms of fractions. If the input is given as 0.4% then it means, the minimum support is 0.4 percent of the total candidates generated, whereas is it is simply given 0.4 then it simply means the minsup must be at least 40% times repeated pattern out of the total candidates generated. Hence it becomes easy for a user to define the confidence of the support which he wants to analyze.
The algorithm then takes care of all the necessary steps being processed, ranging from function calls, inheriting the properties, pruning the infrequent candidates, analyzing the patterns and keeping the track of time which is the most important component which defines the efficiency.

B. Pruning the unwanted candidates:

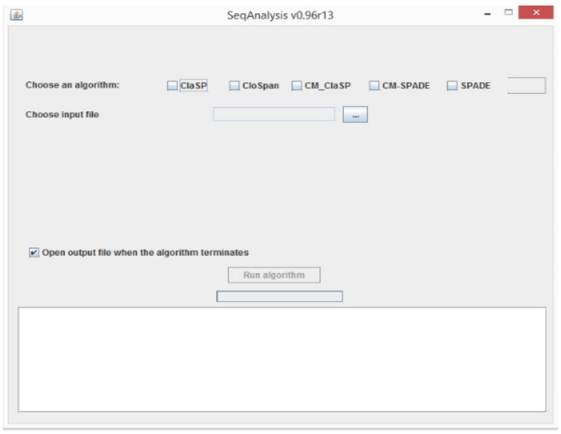
'Cm' is the crucial and the very essence of the whole document. It forms the basis for solving these two problems. It stands for Co-occurrence map and it simply states that, change in the data type being used, itself is sufficient for pruning the unnecessary infrequent items. In order to reduce the ambiguity, we can clearly say that we are trying to keep a track of a user's navigational data and collecting the hashcodes of the consistent repeated resources of the user's browsing history. Thus the Cmap being used is used to prune the resources which are not in regular use.

This map can simply be a nxn matrix where n is the total number of candidates generated. But the problem with matrix is it takes more space than expected since all the elements of the matrix are not being used, only few items of the different item sets are being in use, but the whole matrix needs to be stored since a matrix is a single unit which is defined and initialized. We use hash map which is made in such a manner that it contains a key value pair. This is nothing but the hash code of the resource and its url mapped number assigned and stored in the database.

VI. Discussion

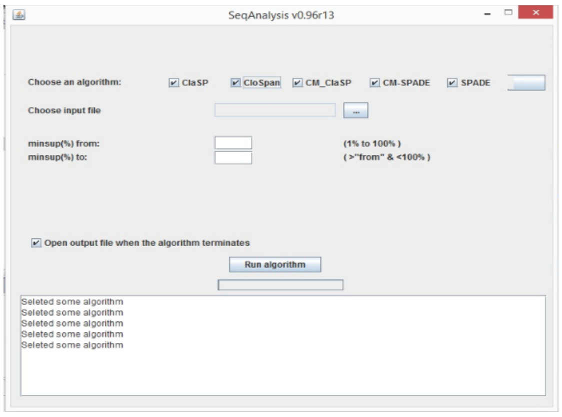
Let us try to analyze the flow of control from the very beginning of a user's perspective.

Step1: Run the program so as to get the following output, it will display the checkboxes regarding the algorithms to be selected and the minsup to be taken care of.



[✓] Open output file when the algorithm terminates

step2: Select the algorithms to be analyzed



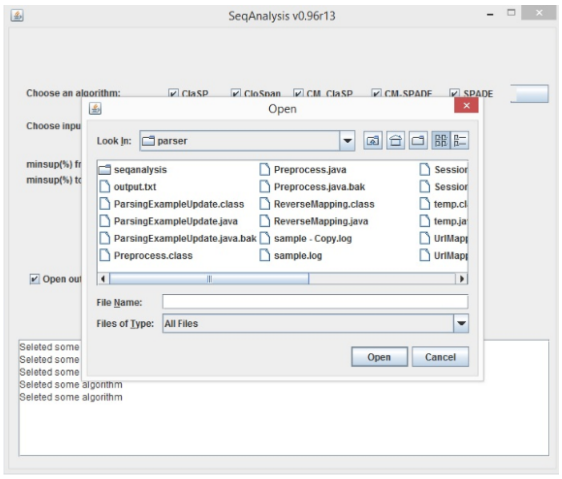
Choose an algorithm: [✓ ClaSP] [✓ CloSpan] [✓ CM_ClaSP] [✓ CM-SPADE] [✓ SPADE]

minsup(%) from: (1% to 100%)
minsup(%) to: (>"from" & <100%)

[✓] Open output file when the algorithm terminates

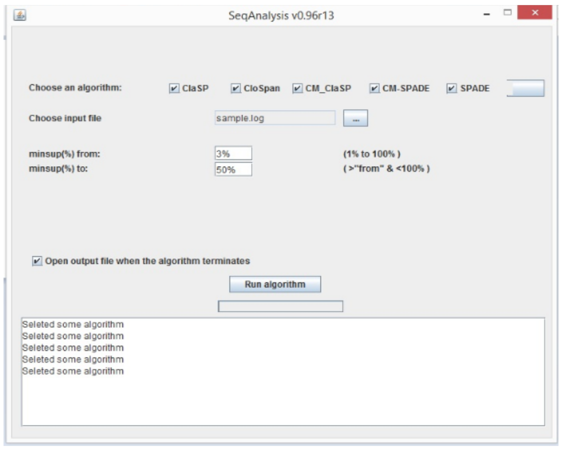
```
Seleted some algorithm
Seleted some algorithm
Seleted some algorithm
Seleted some algorithm
Seleted some algorithm
```

step3: Select the input files,make sure it is a log file



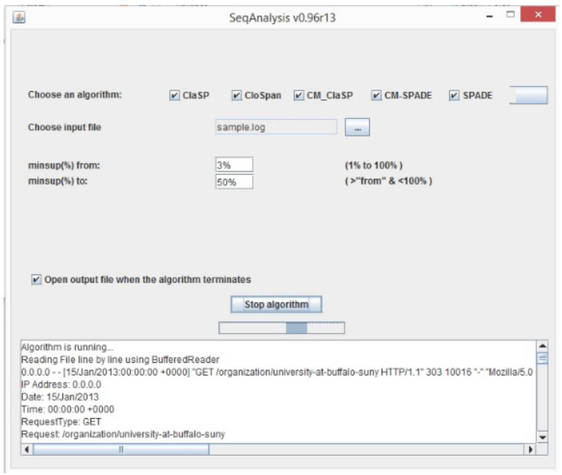
```
SeqAnalysis v0.96r13
Choose an algorithm:   [✓] ClaSP  [✓] CloSpan  [✓] CM_ClaSP  [✓] CM-SPADE  [✓] SPADE
Choose inpu
minsup(%) fr
minsup(%) tc
[✓] Open out

Open
Look In: parser
seqanalysis                    Preprocess.java         Sessior
output.txt                     Preprocess.java.bak     Sessior
ParsingExampleUpdate.class     ReverseMapping.class    temp.cl
ParsingExampleUpdate.java      ReverseMapping.java     temp.ja
ParsingExampleUpdate.java.bak  sample - Copy.log       UrlMapp
Preprocess.class               sample.log              UrlMapp
File Name:
Files of Type: All Files
Open   Cancel

Seleted some
Seleted some
Seleted some
Seleted some algorithm
Seleted some algorithm
```

Step4: Proceed to further analysis by giving the range of minimum support



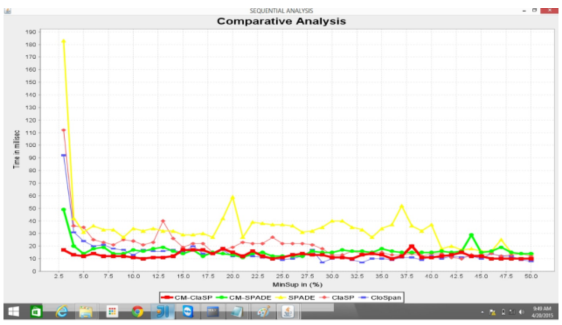
```
SeqAnalysis v0.96r13
Choose an algorithm:   [✓] ClaSP  [✓] CloSpan  [✓] CM_ClaSP  [✓] CM-SPADE  [✓] SPADE
Choose input file      sample.log  ...
minsup(%) from:        3%     (1% to 100% )
minsup(%) to:          50%    ( >"from" & <100% )
[✓] Open output file when the algorithm terminates
Run algorithm
Seleted some algorithm
Seleted some algorithm
Seleted some algorithm
Seleted some algorithm
Seleted some algorithm
```

Step5: Run the algorithm so as to get the result at run time.



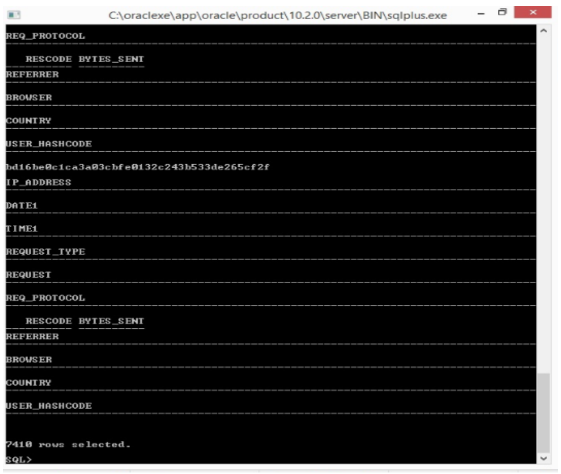
```
SeqAnalysis v0.96r13
Choose an algorithm:   [✓] ClaSP  [✓] CloSpan  [✓] CM_ClaSP  [✓] CM-SPADE  [✓] SPADE
Choose input file      sample.log  ...
minsup(%) from:        3%     (1% to 100% )
minsup(%) to:          50%    ( >"from" & <100% )
[✓] Open output file when the algorithm terminates
Stop algorithm
Algorithm is running...
Reading File line by line using BufferedReader
0.0.0.0 - - [15/Jan/2013:00:00:00 +0000] "GET /organization/university-at-buffalo-suny HTTP/1.1" 303 10016 "-" "Mozilla/5.0
IP Address: 0.0.0.0
Date: 15/Jan/2013
Time: 00:00:00 +0000
RequestType: GET
Request: /organization/university-at-buffalo-suny
```

Result

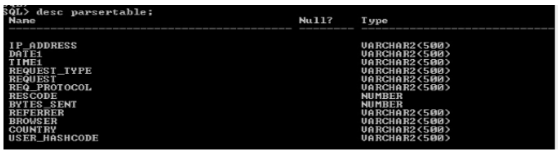


Thus the comparative analysis shows that for a taken log file Cm-Clasp gives the best results in a very less time compared to other sequential algorithms.

Following are the data base schemas being used and contributed to the research analysis.

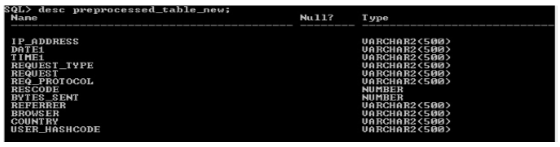


```
C:\oraclexe\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
REQ_PROTOCOL
--------------------------------------------------------------------------------
   RESCODE BYTES_SENT
---------- ----------
REFERRER
--------------------------------------------------------------------------------
BROWSER
--------------------------------------------------------------------------------
COUNTRY
--------------------------------------------------------------------------------
USER_HASHCODE
--------------------------------------------------------------------------------
bd16be0c1ca3a03cbfe0132c243b533de265cf2f
IP_ADDRESS
--------------------------------------------------------------------------------
DATE1
--------------------------------------------------------------------------------
TIME1
--------------------------------------------------------------------------------
REQUEST_TYPE
--------------------------------------------------------------------------------
REQUEST
--------------------------------------------------------------------------------
REQ_PROTOCOL
--------------------------------------------------------------------------------
   RESCODE BYTES_SENT
---------- ----------
REFERRER
--------------------------------------------------------------------------------
BROWSER
--------------------------------------------------------------------------------
COUNTRY
--------------------------------------------------------------------------------
USER_HASHCODE
--------------------------------------------------------------------------------


7410 rows selected.
SQL>
```



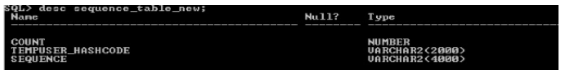
```
SQL> desc parsertable;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 IP_ADDRESS                                         VARCHAR2(500)
 DATE1                                              VARCHAR2(500)
 TIME1                                              VARCHAR2(500)
 REQUEST_TYPE                                       VARCHAR2(500)
 REQUEST                                            VARCHAR2(500)
 REQ_PROTOCOL                                       VARCHAR2(500)
 RESCODE                                            NUMBER
 BYTES_SENT                                         NUMBER
 REFERRER                                           VARCHAR2(500)
 BROWSER                                            VARCHAR2(500)
 COUNTRY                                            VARCHAR2(500)
 USER_HASHCODE                                      VARCHAR2(500)
```

The parser table



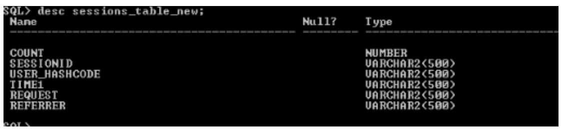
```
SQL> desc preprocessed_table_new;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 IP_ADDRESS                                         VARCHAR2(500)
 DATE1                                              VARCHAR2(500)
 TIME1                                              VARCHAR2(500)
 REQUEST_TYPE                                       VARCHAR2(500)
 REQUEST                                            VARCHAR2(500)
 REQ_PROTOCOL                                       VARCHAR2(500)
 RESCODE                                            NUMBER
 BYTES_SENT                                         NUMBER
 REFERRER                                           VARCHAR2(500)
 BROWSER                                            VARCHAR2(500)
 COUNTRY                                            VARCHAR2(500)
 USER_HASHCODE                                      VARCHAR2(500)
```

The Preprocessed table



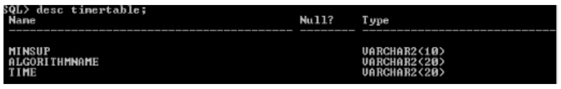
```
SQL> desc sequence_table_new;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 COUNT                                              NUMBER
 TEMPUSER_HASHCODE                                  VARCHAR2(2000)
 SEQUENCE                                           VARCHAR2(4000)
```

The Sequence table



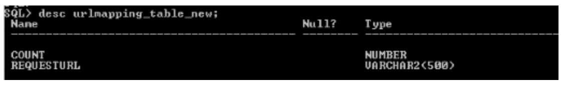
```
SQL> desc sessions_table_new;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 COUNT                                              NUMBER
 SESSIONID                                          VARCHAR2(500)
 USER_HASHCODE                                      VARCHAR2(500)
 TIME1                                              VARCHAR2(500)
 REQUEST                                            VARCHAR2(500)
 REFERRER                                           VARCHAR2(500)

SQL>
```

The Sessions table



```
SQL> desc timertable;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 MINSUP                                             VARCHAR2(10)
 ALGORITHMNAME                                      VARCHAR2(20)
 TIME                                               VARCHAR2(20)
```

The Timer table



```
SQL> desc urlmapping_table_new;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 COUNT                                              NUMBER
 REQUESTURL                                         VARCHAR2(500)
```

The URL mapping table

VII. Summary and Conclusions

Summing up the flow, the prerequisites for the analysis are the logfile, sequence vertical algorithm, usage of co-occurrence map, url mapping, connectivity to a visualization tool. Updating and truncating the database at regular intervals is also important since it doesn't cause any extra overhead.

We can conclude that the problems of confidently analysing a pattern and reporting the frequent patterns with the confidence can be easily promoted using these algorithms and the co-occurrence map can thus help in pruning the infrequent candidates which results in less turnaround time. The graph shows that the algorithms which has Co-occurrence map embedded within has rendered the results in less amount of time than the routine state of art algorithms. Thus we can prove that the two majorly faced problems can be handled using these types of algorithms.

Although these algorithms constitute a large amount of risk-free environment to the sequential mining, this analysis is restricted to generally log file type of datasets. The type of data being used impacts a lot on an algorithm than the data type being embedded inside the algorithm. Thus we need to make sure about the type of the datasets being processed. Finally we can say that co-occurrence map works well with vertical datasets rather than the horizontal datasets. Hence we need to have an understanding over the type of data being chosen.

References

1. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Proceedings of the Eleventh International Conference on Data Engineering, pp. 3–14. IEEE (1995)
2. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 429–435. ACM (2002)
3. Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q.: FreeSpan: frequent pattern-projected sequential pattern mining. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 355–359 (2000)
4. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering Frequent Closed Itemsets for Association Rules. In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 398–416. Springer, Heidelberg (1998)
5. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: Mining sequential patterns by pattern-growth: the PrefixSpan approach. IEEE Transactions on Knowledge and Data Engineering 16(11), 1424–1440 (2004)
6. Srikant, R., Agrawal, R.: Mining Sequential Patterns: Generalizations and Performance Improvements. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) EDBT 1996. LNCS, vol. 1057, pp. 3–17. Springer, Heidelberg (1996)
7. Wang, J., Han, J., Li, C.: Frequent closed sequence mining without candidate maintenance. IEEE Transactions on Knowledge and Data Engineering 19(8), 1042–1056 (2007)
8. Yan, X., Han, J., Afshar, R.: CloSpan: Mining closed sequential patterns in large datasets. In: Proceedings of SIAM International Conference on Data Mining, pp. 166–177 (2003)
9. Zaki, M.J.: SPADE: An efficient algorithm for mining frequent sequences. Machine Learning 42(1), 31–60 (2001)
10.http://en.wikipedia.org/wiki/Sequential_Pattern_Mining
11.. Aseervatham, S., Osmani, A., Viennet, E.: bitSPADE: A Lattice-based Sequential Pattern Mining Algorithm Using Bitmap Representation. In: Proc. 6th Intern. Conf. Data Mining, pp. 792–797. IEEE (2006)
12.Fournier-Viger, P., Gomariz, A., Gueniche, T., Mwamikazi, E., Thomas, R.: TKS: Efficient Mining of Top-K Sequential Patterns. In: Motoda, H., Wu, Z., Cao, L., Zaiane, O., Yao, M., Wang, W. (eds.) ADMA 2013, Part I. LNCS, vol. 8346, pp. 109–120. Springer, Heidelberg (2013)
13.http://www.philippe-fournier-viger.com/spmf/clasp.pdf
14.http://www.philippe-fournier-viger.com/spmf/PAKDD2014_sequential_pattern_mining_CM-SPADE_CM-SPAM.pdf
15.. Fournier-Viger, P., Nkambou, R., Tseng, V.S.: RuleGrowth: Mining Sequential Rules Common to Several Sequences by Pattern-Growth. In: Proc. ACM 26th

Symposium on Applied Computing, pp. 954–959
(2011)
16. Fournier-Viger, P., Wu, C.-W., Tseng, V.S.:
Mining Maximal Sequential Patterns
without Candidate Maintenance. In: Motoda, H.,
Wu, Z., Cao, L., Zaiane, O., Yao,
M., Wang, W. (eds.) ADMA 2013, Part I. LNCS,
vol. 8346, pp. 169–180. Springer,
Heidelberg (2013)