Performance Evaluation on State of the Art Sequential Pattern Mining Algorithms

Thomas. Rincy. N
M. Tech (Scholar),
SIRT, Bhopal

Yogadhar Pandey
Assistant Professor, (CSE) Dept,
SIRT, Bhopal

ABSTRACT
Data mining refers to extracting or mining knowledge from large amounts of data. Among the various data mining tasks sequential pattern mining is one of the most important tasks. It has broad applications in several domains such as the analysis of customer purchase patterns, web access patterns, seismologic data, and weather observations. Sequential pattern mining consists of mining subsequences that appear frequently in a set of sequences. Sequential pattern mining was first introduced by Rakesh Agarwal and Ramakrishnan Srikant in 1995. Many novel approaches for sequential pattern mining were proposed like Apriori, AprioriALL, GSP, SPADE, SPAM and PrefixSpan.

In this paper, the performance of state-of-the-art sequential pattern mining algorithms PrefixSpan and SPAM is evaluated. "From the comprehensive experiments what have been done several phenomena were observed which are different from the traditional standpoint will be explained in this paper."

Keywords: Data Mining, Sequential Pattern Mining, PrefixSpan, SPAM.

1. INTRODUCTION
Sequential pattern mining was first introduced by Agrawal and Srikant [2]. "*Given a set of sequences, where each sequence consists of a list of events (or elements) and each event consists of a set of items, and given a user-specified minimum support threshold of min_sup, sequential pattern mining finds all frequent subsequences, i.e., the subsequences whose occurrence frequency in the set of sequences is no less than min_sup.*" Let $I = \{i_1, i_2, ..., i_n\}$ be the set of all items. An itemset is a nonempty set of items. A sequence is an ordered list of events. A sequence s is denoted $< e_1e_2e_3...e_l >$ where event e_1 occurs before e_2, which occurs before e_3, and so on. Event e_j is also called an element of s sequence. An event is an itemset, i.e., an unordered list of items. An itemset (or event) is denoted as $(x_1x_2\ldots\ x_q)$, where x_k is an item. For brevity, the brackets are omitted if an element has only one item, i.e, element (x) is written as x. The number of instances of items in a sequence is called an l-sequence. A sequence $s_a = (a_1,a_2, ..., a_n)$ is contained in another sequence $s_b = (b_1,b_2, ...,b_m)$ if there exist integers $1 \leq i_1 < i_2 < .....< i_n \leq m$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}$, ..., $a_n \subseteq b_{in}$. If sequence s_a is contained in sequence s_b, then it is called s_a a *subsequence* of s_b and s_b a *supersequence* of s_a.

A database D is a set of tuples (*cid, tid, X*), where *cid* is a customer-id, *tid* is a transaction-id based on the transaction time, and X is an itemset such that $X \subseteq I$. Each tuple in D is referred to as a transaction. A sequence database S, is a set of tuples, <*SID, s*> where *SID* is a sequence_ID and s is sequence.

The *absolute support* of a sequence s_a in the sequence representation of a database D is defined as the number of sequences $s \in D$ that contains s_a, and the *relative support* is defined as the percentage of sequences $s \in D$ that contain s_a.

The support of s_a in D is denoted by $sup_D(s_a)$. Given a support threshold *minsup*, a sequence s_a is called a *frequent sequential pattern* on D if $sup_D(s_a) \geq minsup$. The problem of mining sequential patterns is to find all frequent sequential patterns for a database D, given a support threshold *sup*.

Consider the sequence database, given in Table 2 which, will be used in examples throughout this section.

Table 1. Sample Sequence Database *D*

User ID	TID	Access Sequence
10	t_1	<*qrqpt*>
20	t_2	<*qprqpt*>
30	t_3	<*pqt*>
40	t_4	<*pqtqs*>
50	t_5	<*pqps*>

Table 2. Sequence Database *D*

Sequence ID for each Customer	Data Sequence
1	<*(p)(t)*>
2	<*(uv)p(uqzr)*>
3	<*(pwrs)*>
4	<*p(pqrs)t*>
5	<*t*>

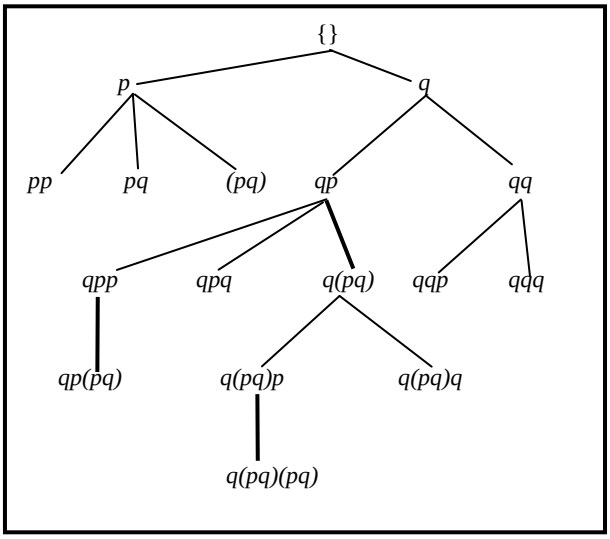


Figure 1: Lexicographic sequence subtree for items *p* and *q* only. Light lines mean sequence-extended sequence (S-Step), bold lines mean item-set extended sequence (I-Step).

2. SPAM ALGORITHM

SPAM [Ayres et al. 2002] [3]. SPAM: (Sequential PAttern Mining) algorithm is based on the Apriori property. SPAM uses some interesting efficient techniques and data structures. It uses a vertical bitmap database that allows for simple and efficient support count. It is designed to work with data in main memory, and it is the first algorithm for mining sequential patterns that traverse the lexicographical sequence tree in depth-first manner. For example, for SPAM, consider the database of Table 2. The lexicographical subtree for item *p* (abbreviated as *a*) is provided in Ayres et al. [2002], Figure 1 shows the lexicographical subtree for item *q*, assuming a maximum sequence size of 3. By traversing the tree if a sequence is generated then each node in the tree has sequence-extended children sequences generated in the *S-Step* of the algorithm, and itemset-extended children sequences generated by the *I-Step* of the algorithm at each node. SPAM traverses the sequence tree in depth-first search manner and checks the support of each sequence-extended or itemset-extended child against *minsup* recursively. If the support of a certain child *s* is greater than or equal to *minsup* that sequence is stored and depth-first search is repeated recursively. If the support of a certain child *s* is less than *min_sup* there is no need to repeat depth-first search on *s* by the Apriori property. For minimizing the number of children nodes Apriori-based pruning is applied at each *S-Step* and *I-Step* of the algorithm and this technique guarantees that all nodes corresponding to frequent sequences are visited. For efficient counting SPAM uses a *vertical bitmap* representation of data. Each bitmap has a bit corresponding to each element of the sequences in the database. Each bitmap partition of a sequence to be extended in the *S-Step* is first transformed using a lookup table, such that all the bits after the index of the first "1" bit (call it index y) are set to one and all the bits with index less than or equal to y are set to zero. Then, the resulting bitmap can be obtained by the ANDing operation of the transformed bitmap and the bitmap of the appended item. In the *I-Step*, ANDing is performed directly without transformation of the sequence. Now the support-counting it is a simple count of how many bitmap partitions, not containing all zeros. Each item in Table 2 will be represented by a vertical bitmap. The AND bitwise operation is used for checking support in the *S-Step* generation of, say <*pt*>, <*p*>'s bitmap, is transformed as described above and a regular AND operation is performed with the bitmap for <*t*> to generate the bitmap of <*pt*>. If *min_sup* = 2, then the candidate sequence <*pt*> will not be pruned because the number of nonzero bitmap partitions in this case is ≥ 2. SPAM it uses bitwise operations rather than regular and temporal joins. Here, Ayres et al. proposes to use a compressed bitmap representation in SPAM to save space, but do not elaborate on the idea.

Pseudocode for DFS with pruning

DFS-Pruning (node $n = (s_1, ..., s_k)$, S_n , I_n)

(1) $S_{temp} = \phi$.

(2) $I_{temp} = \phi$.

(3) **For each** ($i \in S_n$)

(4) **if** (($s_1, ..., s_k$, $\{i\}$) is frequent)

(5) $S_{temp} = S_{temp} \cup \{i\}$

(6) **For each** ($i \in S_{temp}$)

(7) DFS-Pruning(($s_1, ..., s_k, \{i\}$),S_{temp}, all elements in S_{temp} greater than i)

(8) **For each** ($i \in I_n$)

(9) **if** ((s_1, ... , $s_k \cup \{i\}$) is frequent)

(10) $I_{temp} = I_{temp} \cup \{i\}$

(11) **For each** ($i \in I_{temp}$)

(12) DFS-Pruning ((s_1, ..., $s_k \cup \{i\}$), S_{temp}, all elements in I_{temp} greater than i)

3. PrefixSpan ALGORITHM

PrefixSpan [Pei et al. 2004] [4]. PrefixSpan: (Prefix-Projected Sequential Patterns Mining) algorithm is a pattern-growth algorithm. It examines only the prefix subsequences and projects only their corresponding postfix subsequences into projected databases. The sequential patterns are grown in each projected database by exploring only local frequent sequences. To illustrate the idea of projected databases, consider <*u*>, <*(uv)*>, <*(uv)p*>, which are all prefixes of sequence <*(uv)p(uqzr)*> from Table 2, but neither <*up*> nor <*vp*> is considered a prefix while, <*(_v)p(uqzr)*> is the postfix of the same sequence w.r.t. <*(u)*>,and <*p(uqzr)*> is the postfix w.r.t. prefix <*(uv)*>. A running example of PrefixSpan on database *D* (Table 2) acts in three steps:

1. The first step is to scan the sequential database *D* to get the length-1 sequences. It generates *p:4, r:3, t:3, q:2, s:2* along with their support counts.
2. Sequential database is divided into different partitions according to the number of length-1 sequence to get projected databases. This example generates 5 disjoint subsets according to the 5 prefixes <*p*>, <*q*>, <*r*>, <*s*>, <*t*>.
3. Find subsets of sequential patterns; these subsets can be mined by constructing projected databases, and mining each one recursively. To find sequential patterns having prefix <*p*>, extend it by adding one item at a time. To add the next item *x*,

there are two possibilities: (1) the algorithm joins the last itemset of the prefix (i.e.,*<(px)>)* and (2) it forms a separate itemset (i.e.,*<px>*). So to produce *<p>* - projected database: if a sequence contains item *<p>* then the suffix following the first *<p>* is extracted as a sequence in the projected database. By looking at the second sequence (second row) of Table 2, *<(uv)p(uqzr)>*, which is projected to *<(qr)>* where *u* and *z* are removed because they are infrequent. The third sequence is projected to *<(_rs)>*, the fourth to *<(pqrs)t>*, eventually the final projected database for prefix *<p>* contains the following sequences: *t, (qr), (_rs), (pqrs)t*; for the other prefixes refer Table 3.

Table 3. Running PrefixSpan on Table 2

Prefix	Projected Database	Sequential Patterns
<p>	*<t>, <(qr)>,<(_rs)>,<(pqrs)t>*	*p, pq, pr, (pr), (ps), pt*
<q>	*<(_r)>, <(_rs) t>*	*q, (qr)*
<r>	*<(_s)>, <(_s)t>*	*(rs)*
<s>	*<t>*	ϕ
<t>	ϕ	ϕ

Now to find all frequent sequences of the form *<(px)>*, two templates are used: *<(_x)> and <px>* to match each projected sequences, to accumulate the support count for each possible *x* (*x* match any item). The second template uses the last itemset in the prefix rather than only its last item. In the example here, they are the same because there is only one item in the last itemset of the prefix. Then, it needs to find all frequent sequences of the form *<px>*. Table 3 contains all frequent sequential patterns generated for this example using PrefixSpan. Looking at the patterns generated for prefix *<p>*, after finding the frequent 2-sequences (namely, *pq, pr, (pr), (ps), pt),* it recursively create projected databases for them and start mining for frequent 3-sequences etc. The key advantage of PrefixSpan is that it does not generate any candidates. It only counts the frequency of local items. It utilizes a *divide-and-conquer* framework by creating subsets of sequential patterns (i.e. projected databases) that can be further divided when necessary.

Algorithm (PrefixSpan): Prefix-projected sequential pattern mining.

Input: A sequence database *S*, and the minimum support threshold min_support.

Output: The complete set of sequential patterns.

Method: Call *PrefixSpan* (< >, O, *S*).

Subroutine *PrefixSpan* (α, *l*, $S|_\alpha$)

The parameters are 1) α is sequential pattern; 2) *l* is the length of α; and 3) $S|_\alpha$ is the α–projected database if $\alpha \neq <>$, otherwise, it is the sequence database *S*.

Method:

1. Scan $S|_\alpha$ once, find each frequent item, *b*, such that

(a) *b* can be assembled to the last element of α to form a sequential pattern; or

(b) *<b>* can be appended to α to form a sequential pattern.

2. For each frequent item *b*, append it to α to form a sequential pattern α', and output α'.

3. For each α', construct α' - projected database $S|_{\alpha'}$, and call *PrefixSpan* (α', *l* + 1, $S|_{\alpha'}$).

The major cost of PrefixSpan is the construction of projected database.

Pseudoprojection:

If the number and/or the size of projected database can be reduced, the performance of sequential pattern mining can be further improved. Usually, a large number of projected databases will be generated in sequential pattern mining. The technique which may reduce the number and size of projected databases is pseudoprojection. Instead of performing physical projection, one can register the index (or identifier) of the corresponding sequence and the starting position of the projected suffix in the sequence. Then, a physical projection of a sequence is replaced by registering a sequence identifier and the projected position index point. Pseudoprojection reduces the cost of projection substantially when the projected database can fit in main memory.

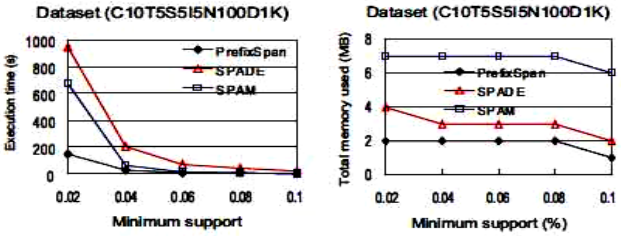
Table 4. Characteristics of PrefixSpan and SPAM algorithms

	PrefixSpan	SPAM
Database Layout	*Projected Databases, List of Items*	*Vertical, Bitmap*
Space-search Enumeration	*Bottom-Up*	*Bottom-Up*
Space-search Traversal	*Variable Length patterns*	*Depth-First*
Candidate Generation	*No*	*Yes*
Classes of Sequence Patterns	*All*	*All*
Taxonomies & Constraints	*Doesn't support*	*Doesn't support*

4. RELATED WORK

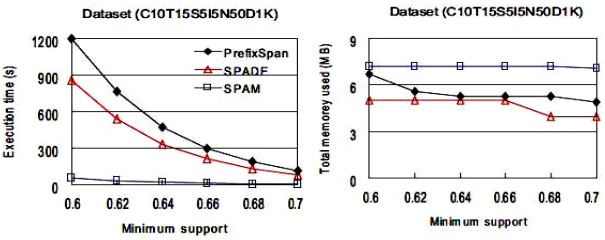
Sequential pattern mining is computationally challenging because such mining may generate and/or test a combinatorial explosive number of intermediate sequence. Many novel algorithms are proposed such as Apriori, AprioriALL, GSP, SPADE, SPAM and PrefixSpan. Apriori is seminal algorithm. The Apriori property states that any supersequence of a frequent sequence must not be frequent. AprioriAll [5] first finds all frequent itemsets transforms the database so that each transaction is replaced by all frequent itemsets it contains, and then finds patterns. They solve the problem of finding all sequential patterns in five phases: sort phase, itemset phase, transformation phase, sequence phase, and maximal phase. The main drawback of AprioriAll is that it performs many passes over the databases and it can generate a large amount of candidates which is time consuming. GSP (Generalized Sequential Patterns) [6] adopts a *candidate generate-and-test* approach using *horizontal data format* (where the data are represented as *<sequence_ID: sequence_ of _itemsets>*, as usual, where each itemset is an event). GSP reduces the search space; it typically needs to scan the database multiple times, as it will likely generate huge set of candidate sequences, especially when mining long sequences. SPADE (Sequential PAttern using Equivalent classes) [7] adopts a *candidate generate-and-test* approach using *vertical data format* (where the data are represented as *<itemset: (sequence_ID, event_ID)>).* The vertical data format can be obtained by transformed from a horizontally formatted sequence database in just one scan; however the basic methodology is breadth-first search and Apriori pruning. Despite the pruning SPADE have to generate large sets of candidates in breadth-first manner in order to grow longer sequences. SPAM (Sequential PAttern Mining) algorithm as described in previous section it uses a *vertical bitmap database* layout that allows for simple and efficient support count and it is the first strategy for mining sequential patterns to traverse the lexicographical sequence tree in depth-first manner. For efficient counting SPAM uses a vertical bitmap representation of data, and uses two pruning techniques: *S-step* pruning and *I-step* pruning based on Apriori heuristic to minimize the size of the candidate items. SPAM is very fast algorithm because it uses bitmap and other optimizations. PrefixSpan (Prefix-Projected Sequential Patterns Mining) algorithm as described in previous section is based on pattern-growth sequential pattern mining method. PrefixSpan examines only the prefix subsequences and projects only their corresponding postfix subsequences into projected databases. It utilizes a *divide-and-conquer* framework by creating subsets of sequential patterns (i.e. projected databases) that can be further divided when necessary. The key advantage of PrefixSpan is that it does not generate any candidates, on the other hand the major drawback of PrefixSpan is database projection, usually; a large number of projected databases will be generated in sequential pattern mining. The authors proposed the use of pseudoprojection technique to reduce the cost of projection substantially when the projected database can fit in memory.

Of all the novel algorithms discussed above, PrefixSpan and SPAM is interesting by considering, its various approach and optimization techniques. The research work lays a great interest in following topics "An UpDown Directed Acyclic Graph Approach for Sequential Pattern Mining" the author quoted that ("Among the various approaches, PrefixSpan was



(a) Execution time comparison (b) Memory usage comparison

Figure 2: One example shows PrefixSpan outperforms SPAM.



(a) Execution time comparison (b) Memory usage comparison

Figure 3: One example shows SPAM outperforms PrefixSpan in execution time but consumes more memory.

one of the most influential and efficient ones in terms of both time and space. Some approaches may achieve better performance under special circumstances; however the overall performance of PrefixSpan is among the best. For example LAPIN [8] is more efficient for dense data sets with long patterns but less efficient in other cases. Besides, it consumes much memory than PrefixSpan. SPAM outperforms the basic PrefixSpan but is much slower than PrefixSpan with pseudoprojection technique.") [9]. On another topic "Existing Sequential Pattern Mining Algorithms" the author quoted ("The main drawback of SPAM is the huge memory space. This disadvantage restricts SPAM as a best algorithm on mining large datasets in limited resource environments." Figure.2 shows a performance comparison in which PrefixSpan outperforms SPAM on execution time comparison and memory time comparison on a synthetic dataset. Figure.3 shows a performance comparison in which SPAM outperforming PrefixSpan in execution time but consumes more memory than PrefixSpan on a synthetic dataset) [10]. Although the author had referred about LAPIN, LAPIN is considered as an improved algorithm of SPAM which is popularly known as LAPIN-SPAM.

The traditional standpoint about SPAM is that "SPAM it generally consumes more memory than PrefixSpan and SPAM is faster on dense datasets with long patterns and less efficient on other dataset, besides it consumes more memory. SPAM is slower than PrefixSpan with pseudoprojection technique." To the best of knowledge, the traditional standpoint about SPAM still exists as on today. The related work is to fairly evaluate the performance of state-of-the-art sequential pattern mining algorithms PrefixSpan and SPAM on real-life datasets to see what's the actual trend is? The algorithms of PrefixSpan and

SPAM is implemented in C++ by their respective authors, so as decided, PrefixSpan and SPAM algorithms are implemented in Java programming language. The performance evaluation is done on real-life datasets having sparse and dense characteristics. "From the comprehensive experiments what have been done several phenomena were observed which are different from the traditional standpoint."

5. DATASETS CHARACTERISTICS

To compare the performance of PrefixSpan and SPAM algorithms a series of experiments are performed with real-life datasets.

If the performance is an issue, then the dataset characteristics should be considered. To determine, if a dataset is sparse or dense one should look at the dataset's basic characteristics such as the number of sequences, the number of items, the average sequence length and the average number of items per sequence.

The first dataset is "BMS-Webview1" the first of the three KDD-Cup datasets which are sometimes called "Gazelle" is a sequence database containing, several months of click-stream data from an e-commerce website. This dataset was used for the KDD-Cup 2001 competition and can be downloaded from http:/www.ecn.purdue.edu/KDD-CUP. The BMS-Webview1 originally contains 59,601 sequences, (59,601 lines in the file). There is a 497 different items for whole dataset. Each sequence in this database contains on average 2.51 items with a standard deviation of 4.85 and a variance of 23.54. It is possible that, the same item appears several times in a sequence. For this dataset, the average number of distinct items for each sequence is 2.51 with a standard deviation of 4.85 and a variance of 23.54. Each item in a sequence appears on average 1 time in the sequence with a standard deviation of 0 and a variance of 0. In this dataset, there is always only 1 item per itemset. The BMS-Webview1 is a sparse dataset, there is on average 2.51 different items in each sequence.

The second dataset is "BMS-Webview1 at 30,000 sequences" which is named and created, is another variation of BMS-Webview1 dataset to test the algorithms on shorter sequences. The BMS-Webview1 originally contains 59,601 sequences. The BMS-Webview1 at 30,000 sequences contains 30,000 sequences, (30,000 lines in the file). Each sequence in this database contains on average 2.36 items with a standard deviation of 4.31 and a variance of 18.63. For this dataset, the average number of distinct items for each sequence is 2.36 with a standard deviation of 4.31 and a variance of 18.63. Each item in a sequence appears on average 1 time in the sequence with a standard deviation of 0 and a variance of 0. In this dataset, there is always only 1 item per itemset. There is a 348 different items for whole dataset. The BMS-Webview1 at 30,000 sequences is a sparse dataset, there is on average 2.36 different items in each sequence.

The third dataset is "Toxin-Snake" [11], a sequence database from the domain of the biology. It contains 192 protein sequences. For these experiments only sequences containing more than 50 items were kept. Keeping only these sequences has been done to make the dataset more uniform; (because the original Toxin-Snake dataset contains a few sequences that are very short and many long sequences). This resulted in 163 long sequences containing an average 60.61 items. Besides having longer sequences, Toxin-Snake is a very dense dataset. Each of the item occurs in almost every sequence (there is on average 17.84 different items in each sequence and only 20 different items for the whole dataset).

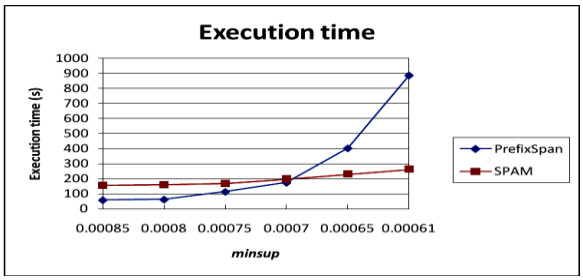
Table 5. Real Dataset Characteristics

	BMS-Webview1	**BMS-Webview1 at 30,000 sequences**	**Toxin-Snake**
Number of sequences	59,601	30,000	163
Number of items	497	348	20
Number of items per itemset	1	1	1
Average number of items per sequence	2.51 (σ =4.85)	2.36 (σ =4.31)	60.61(σ = 0.598)
Average number of different items per sequence	2.51(σ =4.85)	2.36 (σ =4.31)	17.84 (σ = 1.09)

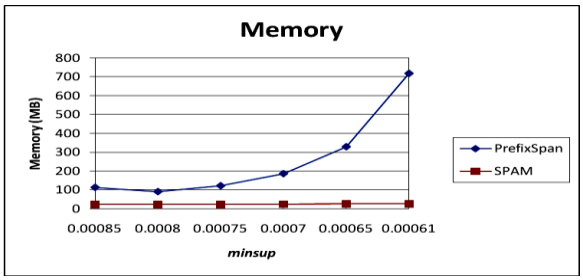
6. PERFORMANCE EVALUATION

An intensive experiment is conducted to evaluate the performance of PrefixSpan and SPAM algorithms in terms of computational costs and memory usage. The algorithms of PrefixSpan and SPAM are implemented in Java programming language and run in Eclipse SDK 4.2.0. Experiments were performed on a notebook computer with a 2.00 GHz T3200, Intel Dual Core Processor running Windows 7 Ultimate and 2 GB of free RAM. The size of virtual memory is set to 1GB, as by default Java virtual machine allocates 256 MB of memory.

Figure 4: shows the execution times and maximum memory usage of two algorithms PrefixSpan and SPAM on real dataset "BMS-Webview1." The BMS-Webview1 is a sparse dataset there is on average 2.51 items in each sequence. Both algorithms are applied with *minsup* = 0.00085, ... , 0.00061. As *minsup* value goes lower SPAM starts to grow faster than PrefixSpan. When *minsup* = 0.00065, PrefixSpan run time is 401.19 seconds while SPAM run time is 230.61 seconds which is more than 1.5 times faster than PrefixSpan. When *minsup* = 0.00061, SPAM has a runtime of 262.08 seconds which is more than 2.5 times faster than PrefixSpan (884.97 seconds). The memory usage trend is compared among the two algorithms PrefixSpan and SPAM. SPAM has stable memory usage than PrefixSpan for all *minsup* values for this dataset.



(a) Execution time comparison

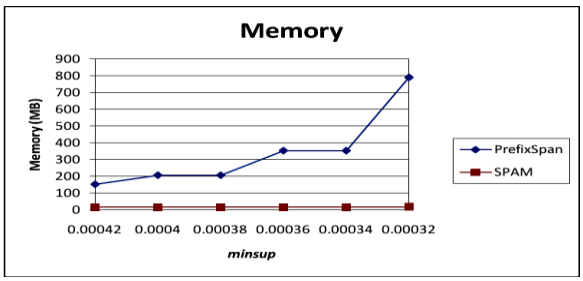


(b) Memory usage comparison

Figure 4: PrefixSpan and SPAM evaluated on BMS-Webview1 dataset.

To test the performance comparison among two algorithms PrefixSpan and SPAM on shorter sequences which is named and created is "BMS-Webview1 at 30,000 sequences" dataset, which contains 30,000 sequences as there is on average 2.36 different items in each sequence, as compared to BMS-Webview1 which originally contains 59,601 sequences, having on average 2.51 different items in each sequence.

(a) Execution time comparison

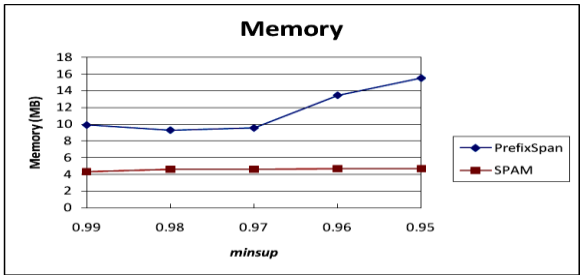


(b) Memory usage comparison

Figure 5: PrefixSpan and SPAM evaluated on BMS-Webview1 at 30,000 sequences dataset.

Figure 5: shows the performance comparison among two algorithms PrefixSpan and SPAM on real dataset "BMS-Webview1 at 30,000 sequences" with *minsup* = 0.00042, 0.00040, 0.00038, ... ,0.00032. When *minsup* < 0.00042 the SPAM started to become faster than PrefixSpan. At *minsup* = 0.00032, SPAM run time is 217.04 seconds, which is more than 1.25 times faster than PrefixSpan (533.39 seconds),while the memory usage trend clearly indicates that SPAM has a stable memory consumption than PrefixSpan at all *minsup* values for this dataset.

(a) Execution time comparison



(b) Memory usage comparison

Figure 6: PrefixSpan and SPAM evaluated on Toxin – Snake dataset.

Figure 6: shows the performance comparison among two algorithms PrefixSpan and SPAM on real dataset "Toxin-Snake." Besides having longer sequences the Toxin-Snake is a very dense dataset. Each item occurs in almost every sequence (there is average 17.84 different items in each sequence and only 20 different items for the whole dataset). Both algorithms are applied with *minsup* = 0.99, 0.98, 0.97, 0.96, 0.95. The performance gap between the algorithms was highest among all the experiments. When *minsup* = 0.95 SPAM is about two orders of magnitude faster than PrefixSpan, while comparing memory usage trend SPAM has a more stable memory consumption than PrefixSpan at all *minsup* values for this dataset.

By observing the performance evaluation trend it clearly shows that SPAM performs much better and has a better scalability than PrefixSpan in terms of execution time while in terms of memory usage the trend clearly indicates that SPAM has stable memory usage than PrefixSpan for all *minsup* values. PrefixSpan algorithm is implemented with pseudoprojection technique, still by observing the performance evaluation trend it clearly shows that SPAM can be faster on sparse and dense datasets, also the memory consumption is stable as compared to PrefixSpan which is contradictory to the traditional standpoint "SPAM it generally consumes more memory than PrefixSpan and SPAM is faster on dense datasets with long

patterns and less efficient on other dataset, besides it consumes more memory. SPAM is slower than PrefixSpan with pseudoprojection technique."

7. DISCUSSIONS

Database (1)

CID	Seq.
10	*p p*
20	*p q*
30	*p r*
40	*p s*
50	*p t*

Database (2)

CID	Seq.
10	*p p q r s t*
20	*q r s t p p*

(a) Two special type of databases

	Avg. Suffix length	Avg. local candidate list item	Suffix-oriented	LCI-oriented
Database (1)	1	5	5 times	25 times
Database (2)	2	1	4 times	0 times

(b) Effect on different types of databases

Figure 7: Performance of Suffix-oriented and LCI-oriented algorithms on different databases.

PrefixSpan belongs to *Suf fix-oriented* category of algorithm, because candidates are come from the suffix of the dataset. SPAM belongs to *LCI-oriented* category of algorithm, because the candidates are come from the local candidate item list. For example we have two sequence databases as shown in figure 7(a), the prefix sequence is *p*, and *minsup* = 1. To test the 2-length candidate sequences, whose prefix is *p* for database (1) the *Suf fix-oriented* algorithm scans the projected database which requires 1×5 = 5 scanning time. The *LCI-oriented* algorithm scans the local candidate item list for each sequence, which requires 5×5 = 25 scanning time. However, for database (2) suppose, to grow from <*pp*> to longer patterns, *Suf fix-oriented* algorithm requires a 4 scanning time (because there are four items *q, r, s, t* in the projected database of <*pp*>), and the *LCI-oriented* algorithm requires a, 0 scanning time (because only one candidate item, *p*, in the local list and no need to join). The effect of these two datasets on the two approaches is shown in Table 7(b). The above example illustrates that, if the average suffix sequence length is less than the average element length for those items in the local candidate list as in database (1) then *Suf fix-oriented* algorithm spends less time. However, if the average suffix sequence length is larger than the average element length for those items in the local candidate list as in database (2) *then LCI-oriented* algorithm is faster, which is different from traditional opinions that the *Suf fix-oriented* algorithm is always better and efficient than *LCI-oriented* algorithm. The two kinds of algorithms have their advantages and disadvantages with regard to different datasets. The reason that PrefixSpan is worse than SPAM is due to useless of scanning those items which are not frequent in the projected databases (i.e. *q,r,s* and *t)* in database (2) as shown in figure 7(a). In other words, PrefixSpan cannot fully utilize the Apriori heuristic because the intrinsic difference of the two algorithms.

8. CONCLUSION

By observing all the trends in the performance evaluation section, it concludes that SPAM performs better than PrefixSpan on particular datasets having sparse and dense characteristics, which is contradictory to the traditional opinion that ("SPAM it consumes more memory than PrefixSpan, SPAM is faster on dense dataset with long pattern but less efficient on other dataset besides, it consumes more memory. SPAM is much slower than PrefixSpan with pseudoprojection technique.").

Although PrefixSpan algorithm is implemented with pseudoprojection technique still, by observing all the trends of performance evaluation section SPAM is faster, performs better, and consumes less memory than PrefixSpan. Performance evaluation of PrefixSpan and SPAM is performed on real-life datasets. The datasets used for performance evaluation is determined by sparse datasets, ("BMS-Webview1", "BMS-Webview1 at 30,000 sequences") and a dense dataset, ("Toxin-Snake") which are real-life datasets which confirms the related work statement in aspects of accuracy and efficiency and proves the related method feasible and efficient. The performance evaluation section concludes that SPAM performs better and has a better scalability than PrefixSpan in terms of execution time, while in terms of memory usage comparison SPAM has more stable memory consumption than PrefixSpan at all *minsup* values.

9. ACKNOWLEDGEMENT

The authors would like to express sincere thanks to Philippe Fournier-Viger for providing all necessary help by taking his precious time. For more information about Philippe Fournier-Viger visit, http://www.philippe-fournier-viger.com/spmf/

10. REFERENCES

[1] R. AGRAWAL AND R. SRIKANT. "Fast Algorithms for Mining Association Rules," *Proc.* 1994. *Int'l Can! Very Large Data Bases (VLDB* 94), *pp. 487-499, Sept. 1994.*

[2] R.AGRAWAL AND R. SRIKANT. "Mining Sequential Patterns," *Proc. 1995. Int'l Can! Data Eng. (lCDE ' 95), pp. 3-14. Mar. 1995*.

[3] JAY AYRES, JOHANNES GEHRKE, TOMI YIU, JASON FLANNICK. Sequential pattern mining using a bitmap representation. *In Proceedings of the 8th ACM SIGKDD, International Conference on Knowledge Discovery and Data Mining.*

[4] J. PEI, J. HAN, B. MORTAZAVI-ASL, H.WANG, J. PINTO, Q. CHEN, U. DAYAL, AND M. HSU. 2004. Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. *pp. 1424 -1440. In Proceedings of IEEE TKDE.*

[5] R. AGRAWAL AND R. SRIKANT. "Mining Sequential Patterns," *In Proceedings of International conference on data engineering. pp. 3-14.*

[6] SRIKANT R. AND AGRAWAL R. "Mining Sequential Patterns: Generalizations and Improvements: *In Proceedings of the 5th International Conference Extending Database Technology, 1996, 1057, pp. 3-17.*

[7] M. J. ZAKI. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning. 42 (1/2): pp.31-60, 2001.*

[8] YANG, Z. AND KITSUREGAWA, M. 2005. LAPIN-SPAM: An improved algorithm for mining sequential pattern. *In Proceedings of the 21st International Conference on Data Engineering (ICDE '05). IEEE.*

[9] JINLIN CHEN 2010. An UpDown Directed Acyclic Graph Approach for Sequential Pattern Mining, pp 914, Section 2.2, Para 9, lines 1 to 8 & 12 to 13. *In Proceedings with IEEE Transactions on knowledge and Data Engineering.*

[10] ZHENGLU YANG, 2008. Fast Algorithms for Sequential Pattern Mining, pp 19, Section 2.2.4, Para 5, lines 1 to 4, Section 3.1.2, Figure 3.1 (a & b), pp 24. Figure 3.2 (a & b), pp 25.

[11] I. JONASSEN, J.F. COLLINS, AND D.G HIGGINS. Finding flexible patterns in unaligned protein sequences, Protein Science vol. 4, no. 8 pp 1587-1595, Wiley-Blackwell, 1995.

[12] UNIL YUN, JOHN J.LEGGETT. WSpan: Weighted Sequential pattern mining in large sequence databases. *In proceedings of the 3rd International IEEE conference Intelligent Systems, September 2006. pp. 512 – 517.*

[13] VEERA BOONJING, PANIDA SONGRAM. Efficient Algorithms for Mining Closed Multidimensional Sequential Patterns. *In Proceedings of the Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007).*

[14] JEN.WEI HUANG, CHI-YAO TSENG, JIAN-CHIHOU AND MING-SYAN CHEN. A General Model for Sequential Pattern Mining with a progressive Database *In Proceedings with IEEE Transaction on Knowledge and Data Engineering, Vol. 20 No. 9, September 2008. pp. 1153 – 1167.*

[15] YI SUI, FENGJING SHAO, RENCHANG SUN, JINLONG WANG. A Sequential Pattern Mining Algorithm Based on Improved FP-tree. *In Proceedings with Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing 2008. pp. 440 – 444.*

[16] TONY, CHENG–KUI HUANG. Developing an Efficient Knowledge Discovering Model for Mining Fuzzy Multi-level Sequential Patterns in Sequence Databases. *In Proceedings with International Conference on New Trends in Information and Service Science. 2009. pp. 362 - 371.*

[17] SHIN-YI WU, YEN-LIANG CHEN. Discovering hybrid temporal patterns from sequences consisting of point- and interval-based events. *In Proceedings of Data and knowledge Engineering 68 (2009). pp.1309-1330. Elsevier.*

[18] R.J. KUO, C.M. CHAO, C.Y. LIU. Integration of K-means algorithm and AprioriSome algorithm for fuzzy sequential pattern mining. *In Proceedings of Applied Soft Computing 9(2009). pp. 85-93.Elsevier.*

[19] NASEER AHMED SAJID, SALMAN ZAFAR, SOHAIL ASGHAR. Sequential Pattern Finding: A Survey. *In Proceedings with IEEE transaction.2010.*

[20] DMITRIY FRADKIN, FABIAN MOERCHEN. Margin-Closed Frequent Sequential Pattern Mining. *In Proceedings with UP'10, July 25th, 2010 Washington, DC, USA. pp 45-54. ACM.*

[21] HAIFENG LI. A Stream Sequential Pattern Mining Model. *In Proceedings with International Conference on Computer Science and Network Technology.2011. pp. 704-707.*

[22] KEN KANEIWA, YASUO KUDO. A sequential pattern mining algorithm using rough set theory. *In proceedings of International Journal of Approximate Reasoning 52(2011). pp. 881-893.*

[23] YANG TANG, FEIFEI LI, HONGYAN LI. Mining Scalable Pattern Based on Temporal Logic over Data Streams. 2012, *9th International conference on Fuzzy Systems and knowledge discovery (FSKD) 2012.*

[24] JUNFU YIN, ZHIGANG ZHENG, LONGBING CAO. USpan: An efficient Algorithm for Mining High Utility Sequential Patterns. *In Proceedings with KDD'12, August 12-16, Beijing, China. pp. 660-668. ACM.*

[25] ZHOU ZHAO, DA YAN AND WILFRED NG. Mining Probabilistically Frequent Sequential Patterns in Uncertain Databases. *In Proceedings with, EDBT 2012, March 26-30, 2012, Berlin, Germany. pp. 74-85. ACM.*

[26] CHIH-HUNG WU, CHIH-CHIN LAI, YU-CHIEH LO. An empirical study on mining sequential patterns in a grid computing environment. *In proceedings of Expert Systems with Applications 39 (2012). pp. 5748-5757. Elsevier.*