Sequential Pattern Mining With Multiple Minimum Supports by MS-SPADE

K.M.V.Madan Kumar[1], P.V.S.Srinivas[2] and C. Raghavendra Rao[3]

[1] Research Scholar, CMJ University, Shillong, Meghalaya, India
Assoc Professor, TKR College of Engg &Tech, Hyderabad, AP, India
[2] Professor, Dept. of CSE, Geethanjali College of Engineering & Technology, Hyderabad, India
[3] Professor, Dept. of CIS, University of Hyderabad, Hyderabad, India

Abstract

Although there may be lot of research work done on sequential pattern mining in static, incremental, progressive databases, the previous work do not fully concentrate on support issues. Most of the previous approaches set a single minimum support threshold for all the items or item sets. But in real world applications different items may have different support threshold to describe whether a given item or item set is a frequent item set. This means each item will contain its own support threshold depends upon various issues like cost of item, environmental factors etc. In this work we proposed a new approach which can be applied to any algorithm independent of whether the particular algorithm may or may not use the process of generating the candidate sets for identifying the frequent item sets. The proposed algorithm uses the concept of "percentage of participation" instead of occurrence frequency for every possible combination of items or item sets. The concept of percentage of participation will be calculated based on the minimum support threshold for each item set. Our proposed algorithm by name "MS-SPADE", which stands for Multiple Support Sequential Pattern Discovery using Equivalent classes , which discovers sequential patterns b by considering different multiple minimum support threshold values for every possible combinations of item or item sets.

Keywords

Multiple minimum supports, sequential patterns, Percentage of participation, Frequent Patterns

1. Introduction

Sequential Pattern mining is one of the most important research issues in data mining. Which was first introduced by Agarwal and Srikanth [2] and can be described as follows: we are given a set of data sequences which will be used as input data. Each sequence consists of a list of ordered item sets containing a seta of different items. The sequential pattern mining finds all Sub sequences with frequencies lower than min support which is given by user. After that there have been a significant research work was done by various researchers on sequential pattern mining and defined number of algorithms not only for static data base [2],[3] Where data do not change over time but also for incremental data bases[6] where there will be new data arriving as the time goes by. In addition to above researchers some other researchers derived other kind of sequential patterns like closed sequential patterns [7], [8], [9]. Constraint sequential pattern [10] maximal sequential patterns [11] spatio temporal sequential patterns [12]. Sequential pattern on specific type of data [13] on stream data. But all the works which was discussed above are by considering uniform min.sup
But considering uniform min.sup, implicitly assumes that all items in the data base have similar frequency. However some items may appear very frequently in the data base while other rarely appears. Under such circumstances, if we set the value of min support too high we will not find those rules involving rare items in the data base. On the other hand if we set that value too low, it will generate huge amounts of meaningless patterns. Therefore lieu et.al [4] first address this and propose the concept of multiple minimum supports (MMS in Short)in association rule mining. In this study we first extend the definition of sequential pattern by considering the concept of MMS, which allows users to specify multiple min.sup for each item. Items with low frequencies will be specified with

lower minimum support and pattern involving these items can be easily retrieved for further decision support.

But the major problem on frequent pattern mining with MMS is that the *downward closure property* no longer holds in the mining process, which means that a super-pattern of an infrequent pattern might be frequent pattern. To effectively reduce the search space in a level –wise methods, Liu.et.al proposes the sorted closure property, where all item in data sets are sorted in ascending order by their MIS values. The sorted closure property however is invalid in sequential pattern mining since the order in the data sequences cannot be altered. Therefore to discover complete set of sequential patterns with MMS is not straight forward. Based on the new definitions of sequential patterns with MMS, we first proposed the concept called *Percentage of participation (POP)* where POP is the percentage of participation of the item or item set with respect to the no of sequences in the POI i.e. |Db| and minimum item support (MIS).

The structure of this paper as follows. In section 2, we first review the concept of SPADE as the basis of our approach. Section 3 introduces the definition of the sequential pattern mining with multiple minimum supports (MMS) by involving the Percentage of participation (POP).Our algorithm called MS-SPADE will be discussed in section 4. Result analysis was organized in section 5 and we have conclusion in section 6.

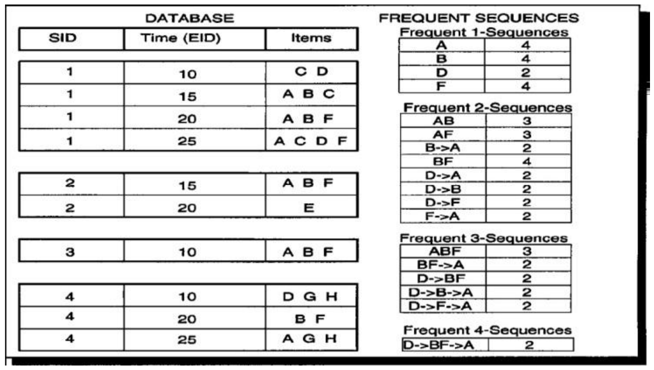
2. Related work

Mohammed J.Zaki proposed an algorithm which works for mining of sequential patterns in static data bases by utilizing combinational properties to decompose the original problem in to smaller sub problems . The smaller sub problems can be independently solved in main memory using efficient lattice search techniques and using simple join operations. By using the SPADE algorithm they discovered all the frequent sequences within three database scans. It avoids the process of making repeated scans for every k-sequence so that improve the efficiency of the processes in terms of time and space. The SPADE not only decreases the no of scans and avoids the complex hash structure which have poor locality. The inputs of sequential pattern mining by SPADE are the set of atoms of sub- lattice S, along with their id-list and user-defined minimum support threshold. The algorithm SPADE will work on the following three features.

1. We use a *vertical id-list* database format, where we associate with each sequence a list of objects in which it occurs, along with the time-stamps. We show that all frequent sequences can be enumerated via simple temporal joins (or intersections) on id-lists.

2. We use a lattice-theoretic approach to decompose the original search space (lattice) into smaller pieces (sub-lattices) which can be processed independently in main-memory. Our approach usually requires three database scans, or only a single scan with some pre-processed information, thus minimizing the I/O costs.

3. We decouple the problem decomposition from the pattern search. We propose two different search strategies for enumerating the frequent sequences within each sub-lattice: breadth-first and depth-first search.



DATABASE

SID	Time (EID)	Items
1	10	C D
1	15	A B C
1	20	A B F
1	25	A C D F
2	15	A B F
2	20	E
3	10	A B F
4	10	D G H
4	20	B F
4	25	A G H

FREQUENT SEQUENCES

Frequent 1-Sequences

A	4
B	4
D	2
F	4

Frequent 2-Sequences

AB	3
AF	3
B->A	2
BF	4
D->A	2
D->B	2
D->F	2
F->A	2

Frequent 3-Sequences

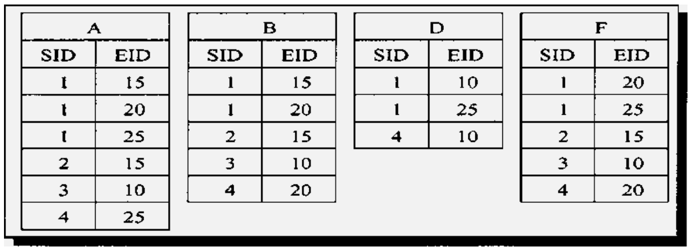
ABF	3
BF->A	2
D->BF	2
D->B->A	2
D->F->A	2

Frequent 4-Sequences

D->BF->A	2

Figure 1:Example Data Base

For the above example shown in figure1 the algorithm SPADE constructs the candidate sets as follows which is shown in Figure2 (a) and Figure 2(b).



A

SID	EID
1	15
1	20
1	25
2	15
3	10
4	25

B

SID	EID
1	15
1	20
2	15
3	10
4	20

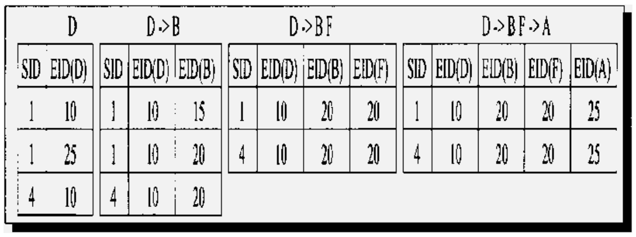
D

SID	EID
1	10
1	25
4	10

F

SID	EID
1	20
1	25
2	15
3	10
4	20

Figure 2(a):ID List for example database

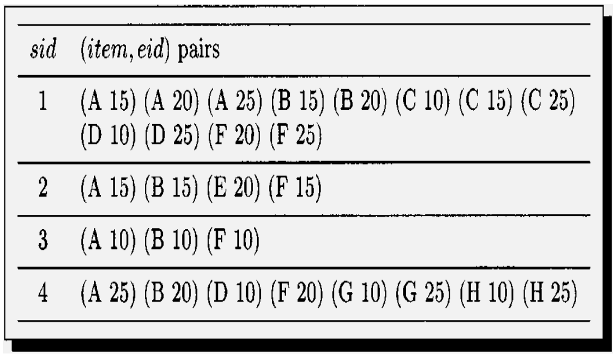


D		D->B			D->BF				D->BF->A				
SID	EID(D)	SID	EID(D)	EID(B)	SID	EID(D)	EID(B)	EID(F)	SID	EID(D)	EID(B)	EID(F)	EID(A)
1	10	1	10	15	1	10	20	20	1	10	20	20	25
1	25	1	10	20	4	10	20	20	4	10	20	20	25
4	10	4	10	20									

Figure 2(b):Id list for temporal items

To find frequent sequences SPADE is using efficient lattice search techniques and simple joins. All the sequences are discovered with only three passes over the database, it also decomposes the mining problem into smaller sub problems, which can be fitted in the main memory.

In this approach, the sequential database is transformed into a vertical id-list database format, in which each id is associated with corresponding items and the time stamp. The vertical database of figure 1 is shown in figure 2(a). For item A, the support count is 4, it occurs with SID 1, 2, 3 and 4 at EID 15,15,10 and 25 respectively. By scanning the vertical database, frequent 1-sequences can be generated with the minimum support. For the 2-sequences, the original database is scanned again and the new vertical to horizontal database is created by grouping those items with SID and in increase order of TID. The result vertical to horizontal database is shown in figure 2(c). By scanning the vertical to horizontal database 2-sequences are generated. All the 2-item sequence found are used to construct the lattice, which is quite large to be fitted in main memory. However the lattice can be further decomposed to different classes, sequences that have the same prefix items belong to the same class. By decomposition the lattice is partitioned into small partitions that can be fitted in main memory. During the third scanning of the database all those longer sequences are enumerated by using *temporal join*
.
There are two methods of enumerating frequent sequences of a class: Breadth- First Search (BFS) and Depth-First Search (DFS). In BFS the classes are generated in a recursive bottom-up manner. For example to generate the 3-item sequences all the 2-item sequences have to be processed, but in DFS only one 2-item sequence and a k-item sequence are further needed to generate (k+1)-item sequence. BFS need much bigger main memory to store all the consecutive 2-item sequences, but DFS just need to store the last 2-item sequence of the newly generated k-item sequences. However BFS has more information to prune the candidate k-item sequences, and the possibility of being large of those sequences generated by BFS is much bigger than those generated by DFS. All the k-item patterns are discovered by temporal or equality joining the frequent (k-1)-item patterns which have the same (k-2)-item prefix, while there are three possible joining results as candidate generation process in GSP [Srikant and Agrawal 1996], the Apriori property pruning technique is also employed in SPADE.



sid	(*item*, *eid*) pairs
1	(A 15) (A 20) (A 25) (B 15) (B 20) (C 10) (C 15) (C 25) (D 10) (D 25) (F 20) (F 25)
2	(A 15) (B 15) (E 20) (F 15)
3	(A 10) (B 10) (F 10)
4	(A 25) (B 20) (D 10) (F 20) (G 10) (G 25) (H 10) (H 25)

Figure 2(c) Vertical to horizontal database recovery

Before this many researchers have developed various methods to find frequent sequential patterns with a static database. Apriori All [2], GSP [3] are the milestones of sequential pattern mining algorithms based on traditional association rules mining technique. Han et al. and Pie et al. brought up Free Span and Prefix Span, which found sequential patterns by constructing sub databases of the entire database. Zhang et al. proposed two algorithms GSP+ and MFS+ based on static algorithms GSP and MFS (also derived by the same authors).But as discussed early in section1 all the researchers considered uniform min.sup to describe whether the sequence is frequent or not.

3. Problem Definition

In this section, we formally give definitions used in the discovery of sequential pattern with MMS. Let *I* denote the set of items in the database, and a subset of *I* is called an item set. A customer's data-sequence is an ordered list of items with time stamps. Therefore, a sequence, say *α*, can be represented as <(a_1: t_1), (a_2: t_2), (a_3: t_3), …, (a_n: t_n)>, where a_j is an item, and t_j stands for the time when a_j occurs, 1≤ j≤n, and tj-1≤tj for 2≤j≤ *n*.

If several items occur at the same time in t_j for 2 *j* the sequence, they are ordered alphabetically.

Definition 1. Given an item set $I_q = (i_1, i_2...i_m)$, we say item set I_q occurs in α if integers $1 \le k_1 < k_2 < \ldots < k_m \le n$ exist such that, $i_1 = a_{k1}$, $i_2 = a_{k2}, \ldots, i_m = a_{km}$ and $t_{k1} = t_{k2} = \ldots = t_{km}$. We refer to k_1 and t_{k1} jointly as the position and the time that I_q occurs in *α*, respectively.

Definition 2. Let $\beta = <I_1, I_2...I_s>$ (I_q subset or equal to *I* for $1 \le q \le s$) be a sequence of item set. Assume that each I_q in *β* occurs in *α*. Then we say sequence *β* occurs in *α*, or is a subsequence of *α* if $t_{I1} < t_{I2} < \ldots < t_{Is}$, where t_{Iq} $(1 \le q \le s)$ is the time, at which I_q occurs in *α*.

Definition 3. A sequence database *S* is formed by a set of records <*sid*, *s* >, where *sid* is the identifier of this data-sequence and *s* is a data-sequence. For a given sequence *β*, the support count of sequence β in S are defined as follows:

supp (β) = |{(sid, s)| (sid, s) €S ^β is a subsequence in s}|

The following definitions are related to the concept of MMS. In this model, the definition of the minimum support is changed. Each item in the database can have its *minsup,* which is expressed in terms of *minimum item support* (MIS). In other words, users can specify different MIS values for different items.

Definition 4: Let MIS (*i*) denote the MIS value of item *i* (*i* subset or equal to *I*). Given an item set $I_q = (i_1, i_2, \ldots, i_m)$, the MIS value of item set $I_q = (i_1, i_2, \ldots, i_m)$ $(1 \le k \le m)$, denoted as MIS(I_q), is equal to:

$$\min [\text{MIS}(i_1), \text{MIS}(i_2), \ldots, \text{MIS}(i_m)]$$

Definition 5: Given a sequence $\beta = <I_1 I_2...I_s>$ (I_q subset or equal to *I* for $1 \le q \le s$), the minimum support threshold of *β*, denoted as MIS (*β*), is equal to:

$$\min [\text{MIS}(I_1), \text{MIS}(I_2), \ldots, \text{MIS}(I_s)]$$

Definition 6: Given a sequence database *S* and a sequence *β*, we call *β* is frequent in *S* or *β* is a sequential pattern in *S* if *supp* (*β*) ≥MIS (*β*).

Definition 7: Let I be an item or item set and MIS (I) be the minin number of sequences In the POI. Timestamps between p ,q.

$$\text{Percentage Of Participation(POP)} = 100/ |Db^{p,q}| * \text{MIS(I)}$$

With the provision of different minimum item support thresholds for different items, user can effectively express different support requirements for different data-sequences. MMS allow users to have higher minimum supports for sequences involving high frequent items, and also allows us to have lower minimum supports for sequences involving rare items. Given a sequence Database S and a set of MIS values for all items in S, we discover all sequential patterns that satisfy MIS (*β).*

4. The Proposed Algorithm

SPADE: Algorithm design and implementation

In this section we describe the design and implementation of MS-SPADE. Figure 4(a) shows the high level structure of the algorithm. The main steps include the computation of the frequent 1-sequences and 2-sequences, the decomposition into prefix-based parent equivalence classes, and the enumeration of all other frequent sequences via BFS or DFS search within each class. We will now describe each step in some more detail.

```
MS-SPADE (pop, D);
F_1= {frequent items or 1-sequences};
F_2= {frequent 2-sequences};
E= {equivalence classes [X] Ω_1};
for all [X] €E do Enumerate –Frequent-set ([X];
```

Figure 4(a): Algorithm MS-SPADE

```
Algorithm POP (Items, MIS(I))
                    2
1. var POP;
2. var I, I_min;// Items
3. For (all combinations of items in the ele);
4. If (No of items >1);
5. Check and identify the item I_min having less support;
6. POP (ele)=100/|Db^{p,q}|*MIS(I_min): // |Db^{p,q}| is no of
sequences from p to q
7. else
8. POP (ele)=100/|Db^{p,q}|*MIS(I): // |Db^{p,q}| is no of
sequences from p to q

END
```

Figure 4(b): Algorithm for Percentage of Participation

3.1.Computing frequent 1-sequences and 2-sequences

Most of the current sequence mining algorithms assume a *horizontal* database layout such as the one shown in figure 1. In the horizontal format the database consists of a set of input-sequences. Each input-sequence has a set of events, along with the items contained in the event. In contrast our algorithm uses a *vertical* database format, where we maintain a disk-based id-list for each item, as shown in figure 2(a). Each entry of the id-list is a (*sid,eid*) pair where the item occurs. This enables us to check support via simple id-list joins.

Computing POP: For computing pop of the elements we first check the no of items for the element, if the no of items are more than one then the algorithm consider the MIS value of item which is having least .Then it will calculate the Pop. For example if the element is AB or A->B and the MIS values are 0.2 and 0.3 then the MIS of element is 0.2. Then the pop will be 100/0.2 * no of sequences i.e. 100/0.2*4 =125 % for it's each occurrence.

Computing F_1: Given the vertical id-list database, all frequent 1-sequences can be computed in a single database scan. For each database item, we read its id-list from the disk into memory. We then scan the id-list, incrementing the *pop* for each new *sid* encountered as shown in figure 4(d) for the example shown in figure 1.But here we are taking the values of min.sup for A,B,C,D,E,F,G and H are 4,3,2,4,1,3,2 and 2. We are considering an item as a binary variable. We consider only once per each sequence occurrence.

Computing F_2: Let $N_2 = |F_1|$ be the number of frequent items, and A the average id-list size in bytes. A naive implementation for computing the frequent 2-sequences requires id-list joins for all pairs of items. The amount of data read is $A \cdot N \cdot (N - 1)/2$, which corresponds to around $N/2$ data scans. This is clearly inefficient. Instead of the naive method we propose two alternate solutions:

1. Use a preprocessing step to gather the *pop* of all 2-sequences above a user specified lower bound. Since this information is invariant, it has to be computed once, and the cost can be amortized over the number of times the data is mined.

2. Perform a vertical-to-horizontal transformation on-the-fly. This can be done quite easily, with very little overhead. For each item i, we scan its id-list into memory. For each (sid, eid) pair, say (s, e) in $L(i)$, we insert (i, e) in the list for input-sequence s. For example, consider the id-list for item A, shown in figure 7. We scan the first pair $(1, 15)$, and then insert $(A, 15)$ in the list for input-sequence 1. Figure 2(c) shows the complete horizontal database recovered from the vertical item id-lists. Computing F_2 from the recovered horizontal database is straight-forward. We form a list of all 2-sequences in the list for each sid, and update pop in a 2-dimensional array indexed by the frequent items.

4.2. Enumerating frequent sequences of a class

Figure 4(c) shows the pseudo-code for the breadth-first and depth-first search. The input to the procedure is a set of atoms of a sub-lattice S, along with their id-lists. Frequent sequences are generated by joining the id-lists of all pairs of atoms (including a self-join) and checking the cardinality of the resulting id-list against *POP*. Before joining the id- lists a pruning step can be inserted to ensure that all subsequences of the resulting sequence are frequent. If this is true, then we can go ahead with the id-list join, otherwise we can avoid the temporal join. The sequences found to be frequent at the current level form the atoms of classes for the next level. This recursive process is repeated until all frequent sequences have been enumerated. In terms of memory management it is easy to see that we need memory to store *intermediate* id-lists for at most two consecutive levels. The depth-first search requires memory for two classes on the two levels. The breadth-first search requires memory of all the classes on the two levels. Once all the frequent sequences for the next level have been generated, the sequences at the current level can be deleted.

```
Enumerate-Frequent-Seq (S);
        for all atoms $A_i$ Є S do
            $T_i$= $;
            for all atoms $A_j$ Є S, with j ≥ I do
                R=$A_i$ V $A_j$;
                If (Prune (R) == FALSE) then
                    L(R) =L ($A_i$) ∩ L ($A_j$);
                If total.POP (R) ≥ 100 then
                    $T_i$ =$T_i$ U {R}; $F_{|R|}$ = $F_{|R|}$ U {R};
end
if (Depth-First –Search) then Enumerate-Frequent–Seq ($T_i$);
end
if (Breadth-First –Search) then
for all $T_i$ ≠ $ do Enumerate - frequent-Seq ($T_i$);
```

Figure 4 (c): Pseudo code for breadth-first and depth-first search

A		
SID	**EID**	**POP**
1	15	25
1	20	
1	25	
2	15	25
3	10	25
4	20	25

B		
SID	**EID**	**POP**
1	15	33.3
1	20	
2	15	33.3
3	10	33.3
4	20	33.3

E		
SID	**EID**	**POP**
2	20	100

F		
SID	**EID**	**POP**
1	20	33.3
1	25	
2	15	33.3
3	10	33.3
4	20	33.3

Figure 4 (d): Id –lists for atoms of frequent 1-sequence with pop

5. Result Analysis

Our proposed algorithm is very effectively working for test dataset and we have analyzed the test dataset for different parameters like period of interest, execution time etc.

5.1 Impact of period of interest (POI) on execution time

Execution time is the time required to execute all the instructions in the proposed algorithm. Here we can observe that the execution time will increase a little bit with respect to the time taken for identifying the item having minimum MIS value. For this we applied the quick sorting technique .It can be noted that execution time is directly proportional to POI. The reason is that, the increase in POI required more time for process as the no of new elements will be added to the existing one.
Figure 5(a) shows the impact of POI over the execution time.

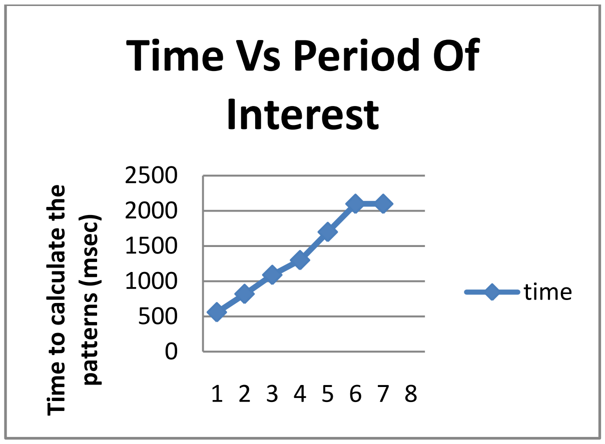


Figure 5(a)

5.2 Impact of POI over number of patterns

Number of patterns is dependent on period of interest a. As from Fig5(b) we can see that as the period of interest increases, the number of patterns also increases. This is because as the period of interest increases, the algorithm has more items to process and so they give more number of patterns.

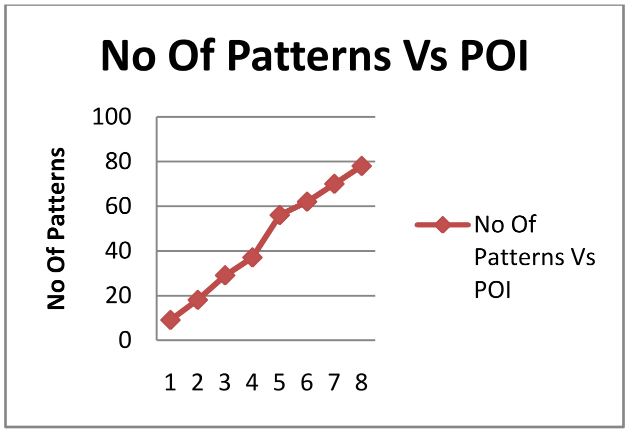


Figure 5(b)

5.3 Comparison between MS-SPADE and SPADE in terms of time in progressive database

Generally mining frequent sequential patterns with multiple minimum supports will take more time when compared with mining frequent sequences with uniform support. In case of percentage of participation (POP), the time for execution of Ms-**SPADE** will take little bit more time than the SPADE. The difference of time between these two will be the time for identifying the item having minimum MIS value from the newly arriving element and accumulating the POP. We can observe the difference in the Figure5(c).

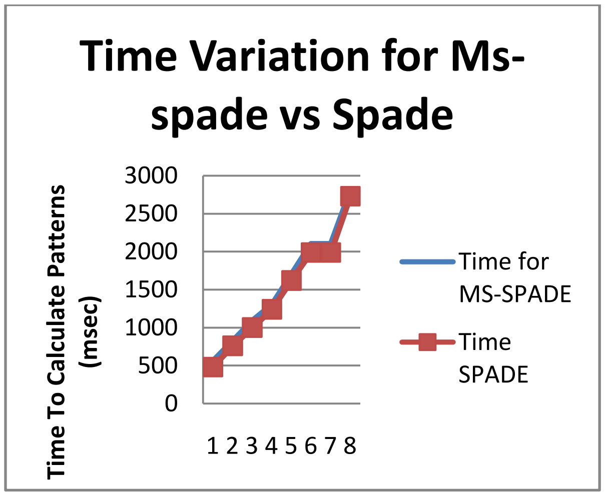


Figure 5(C)

6. Conclusion

The proposed MS SPADE algorithm works efficiently by using the concept of percentage of participation (POP) for mining frequent sequences with multiple minimum supports. Even though the algorithm is taking little more time for execution when compared with SPADE (which finds the frequent sequential patterns considering uniform min.support for different items), it works on frequent sequences with multiple minimum supports for different items. As the algorithm is finding the sequences with multiple minimum supports we can ignore the little increase in time of execution. But in order to calculate the POP of all candidate sequential patterns, it keeps all the candidate sets for all sequences. This involves huge memory usage and involves lots of computation time.

References

[1] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” Proc. 20th Int’l Conf. Very Large Data Bases (VLDB ’94), pp. 478-499, Sept. 1994.

[2] R. Agrawal and R. Srikant, “Mining Sequential Patterns,” Proc. 11th Int’l Conf. Data Eng. (ICDE ’95), pp. 3-14, Feb. 1995.

[3] R. Srikant and R. Agrawal, “Mining Sequential Patterns: Generalizations and Performance Improvements,” Proc. Fifth Int’l Conf. Extending Database Technology (EDBT ’96), Mar. 1996.

[4] B.Liu, W.Hsu and Y.Ma, “Mining association rules with multiple minimum supports”, Proceedings of the fifth ACM, SIGKDD conference Sandiego,CA,USA,August 15-18,1999,P.341

[5] M.J. Zaki, “Efficient Enumeration of Frequent Sequences,” Proc. Seventh ACM Int’l Conf. Information and Knowledge Management (CIKM ’98), Nov. 1998.

[6] Jen-Wei Huang, Chi-Yao Tseng, Jian-Chih Ou, and Ming-Syan Chen, Fellow, IEEE: A General Model for Sequential Pattern Mining with a Progressive Database IEEE Transactions On Knowledge and Data Engineering VOL. 20, NO. 9, SEPTEMBER 2008

[7] S Cong, J. Han and D. Padua,“Parallel Mining of Closed Sequential Patterns,” Proc. 11th ACM

SIGKDD Int'l Conf. Knowl- edge Discovery and Data Mining (KDD '05), pp. 562-567, Aug. 2005.

[8] J. Wang and J. Han, "Bide: Efficient Mining of Frequent Closed Sequences," Proc. 20th Int'l Conf. Data Eng. (ICDE '04), pp. 79-91, 2004.

[9] X. Yan, J. Han, and R. Afshar, "Clospan: Mining Closed Sequential Patterns in Large Datasets," Proc. Third SIAM Int'l Conf. Data Mining (SDM '03), pp. 166-177, May 2003.

[10] M.N. Garofalakis, R. Rastogi, and K. Shim, "Spirit: Sequential Pattern Mining with Regular Expression Constraints," Proc. 25th Int'l Conf. Very Large Data Bases (VLDB '99), pp. 223-234, 1999.

[11] C. Luo and S.M. Chung, "Efficient Mining of Maximal Sequential Patterns Using Multiple Samples," Proc. Fifth SIAM Int'l Conf. Data Mining (SDM), 2005.

[12] H.Cao, N .Mamoulis, and D.W. Cheung, "Mining Frequent Spatio-Temporal Sequential Patterns," Proc. Fifth Int'l Conf. Data Mining (ICDM '05), pp. 82-89, Nov. 2005.

[13] J.-K. Guo, B.-J. Ruan, and Y.-Y. Zhu, "A Top-Down Algorithm for Web Log Sequential Pattern Mining," Proc. Ninth Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD), 2005.