European Journal of Computer Science and Information Technology
Vol.4, No.1, pp.63-76, January 2016
Published by European Centre for Research Training and Development UK (www.eajournals.org)

A NEW APPROACH TO DISCOVER FREQUENT SEQUENTIAL PATTERNS WITHOUT ORDERING THE SUB SEQUENCES USING DAIS ALGORITHM

[1]S.Suryaa and [2]Dr. K. Subramanian,

[1]Ph.D. Research Scholar, J.J. College of Arts and Science, Pudukkottai, Tamil Nadu, India.
[2]Research Guide, Assistant Professor, VSS Govt. Arts College, Poolankurichi, Tamil Nadu, India.

ABSTRACT: *This paper primarily focuses on discovering the frequent sequential patterns without sorting the subsequences present in the sequence of the dataset. The proposed algorithm utilizes numerically converted values of subsequence item to identify the exact order of the patterns efficiently. Most of the existing algorithms for brevity consider only the sorted data or initially sorts the unsorted data to find the patterns. But there are certain circumstances where the data has to be presented and mined without ordering the data. This paper proposes a new algorithm named "DATA as it IS Algorithm" to find frequent sequential patterns and prune away the infrequent items at the beginning stages of the process. The experimental evaluation portrayed that the proposed DAIS algorithm performs effectively and effectively and outscores the existing algorithms by an order of magnitude.*

KEYWORDS: Sequential patterns, without ordering, frequent patterns

---

INTRODUCTION

Finding sequential patterns in large transaction databases is an important data mining problem. Mining, also known as knowledge discovery in databases, has been recognized as a promising new area for database research. This area can be denned as efficiently discovering interesting rules from large databases. A new data mining problem, discovering the frequent sequential patterns without sorting the subsequences present in the sequence of the dataset. The input data is a set of sequences, called data-sequences. Each data-sequence is a list of transactions, where each transaction is a set of literals, called items. Typically there is a transaction-time associated with each transaction.

A sequential pattern also consists of a list of sets of items with a user-specified minimum support. It is used in various domains such as medical treatments, natural disasters, and customer shopping sequences, DNA sequences and gene structures.

LITERATURE REVIEW

The pioneer in this frequent sequential pattern mining is Agarwal [1] who introduced and solved the problem of frequent sequential mining. For a given sequential data, the problem is to find all sequential patterns with a user-defined minimum support, also named frequent sequential patterns. But Agarwal either considers sorted data or sorts the data before processing. A modified Apriori named Apriori-All [1] was also introduced by Agarwal but these two algorithms doesn't discover patterns without sorting the data. Numerous algorithms based on the working principle of Apriori were introduced by researchers but none was

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

designed to find the patterns of unsorted data without sorting the data. The Apriori-like sequential pattern mining methods suffer from the costs to handle a potentially huge set of candidate patterns and scan the database repeatedly. The main disadvantage of Apriori based approach is voluminous candidate generation especially 2-itemset candidates.

The SPAM [2] algorithm uses bitmap representations to find the I-Extended sequences and S-Extended sequences but SPAM algorithm assumes the dataset sequences as a sorted one or it explicitly sorts the sequences before finding the sequential patterns.

Sequential pattern mining algorithms using the vertical format are very efficient, because they can calculate the support of candidate patterns by avoiding costly. Fast Vertical Mining of Sequential Patterns[3]database scans. However, the main performance bottleneck of vertical mining algorithms is that they usually spend lot of time evaluating candidates that do not appear in the input database or are infrequent. This can be used for pruning candidates generated by vertical mining algorithms.

An improved apriori algorithm for association rules[4] is proposed through reducing the time consumed in transactions scanning for candidate itemsets by reducing the number of transactions to be scanned. Whenever the k of k-itemset increases, the gap between our improved Apriori and the original Apriori increases from view of time consumed, and whenever the value of minimum support increases, the gap between our improved Apriori and the original Apriori decreases from view of time consumed. The time consumed to generate candidate support count in our improved Apriori is less than the time consumed in the original Apriori.

Analysis sequential patterns mining[5,6] approaches such as Apriori-based algorithms encounter the problem that multiple scans of the database are required in order to determine which candidates are actually frequent. Most of the solutions provided so far for reducing the computational cost resulting from the apriori property use a bitmap vertical representation of the access sequence database and employ bitwise operations to calculate support at each iteration. The transformed vertical databases, in their turn, introduce overheads that lower the performance of the proposed algorithm, but not necessarily worse than that of pattern-growth algorithms.

PRELIMINARIES

Let I = $\{i_1, i_2, i_3, \dots . i_n\}$ be a set of unique items. A sequence S is an unordered list of events, denoted as $<e_1, e_2, e_3\dots en>$ where e_i is an item, (i.e.) $e_i \in I$ for $1\leq i \leq n$. For brevity, the brackets are omitted if the element has only one element, (i.e.) (*a*) is written as *a*. An item can occur multiple times in different event of a sequence. The number of events in a sequence is called the length of a sequence and a sequence of *l* length is *l*-sequence. A sequence $S_a = \{a_1, a_2, a_3.. a_n\}$ is contained in another sequence $S_b=\{b_1,b_2,b_3,.. b_m\}$, if there exist integers $1 \leq i_1< i_2< i_3 \dots < i_n\leq m$ such that $a_1= b_{i1}, a_2=b_{i2}, \dots. a_n= b_{in}$. If sequence S_a is contained in another sequence S_b, then S_a is called subsequence of S_b and S_b a super-sequence of S_a, denoted by $S_a \subseteq S_b$.

From the table 1, the input sequence database S is a set of tuples (*sid, s*), where *sid* is the sequence identifier and *s* is the input sequence. The number of tuples in S database is called base size of the database S, and denoted as |S|. A tuple (*sid,s*) is said to contain in sequence S_a, If $S_a$is a subsequence of *s*. The support of a sequence $S_a$in the database S is the number of tuples in the database containing S_a, denoted as sup(S_a).

For a given positive integer *min-sup*, as the support threshold, a sequence $S_a$is called frequent sequential pattern in database S, if sup(S_a) ≥*min-sup*. Otherwise the pattern is infrequent.

PROPOSED APPROACH

Table 1: Sample database of unordered events

SeqID	Sequences
S1	[a (dc) ad]
S2	[acae]
S3	[cad(cbd)]
S4	[bbc]
S5	[(bcd)d]

The proposed approach first scan the database to find the unique items present in the database. A table is constructed to identify whether the item is present in the sequence and if the item is present then it is denoted by “1” else denoted by “0”. Along with this the support count of every unique item is calculated and stored. A numerical value for the items are provided and marked in this table as shown in the table 2.

Let us consider S1= [a (dc) ad], here there are four sequences and for brevity, the brackets are omitted and [(a) (dc) (a) (d)] is written as [a (dc) ad]. The sequence S1 contains four events and the second event (dc) is unsorted and this event or subsequence is not sorted to find the frequent sequential patterns in the proposed approach.

After finding the unique items in the database the following table is constructed initially to recognize whether the sequences contain a particular item or not.

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

European Journal of Computer Science and Information Technology
Vol.4, No.1, pp.63-76, January 2016
Published by European Centre for Research Training and Development UK (www.eajournals.org)

Table 2: Identification of unique items with count and numerical value

UNIQUE ITEM	S1	S2	S3	S4	S5	SUPPORT COUNT	NUMERICAL VALUE
a	1	1	1	0	0	3	1
b	0	0	1	1	1	3	2
c	1	1	1	1	1	5	3
d	1	0	1	0	1	3	4
e	~~0~~	~~1~~	~~0~~	~~0~~	~~0~~	~~1~~	~~5~~

ITEM LOCATION AND EVENT INDEXING

Now the unique items exact location in each sequence is found and marked with the item name or simply that location is left empty. Along with this marking of location, the number of items in every event is found and the numerical values corresponding to the items are stored in the table as shown in the figure 1.

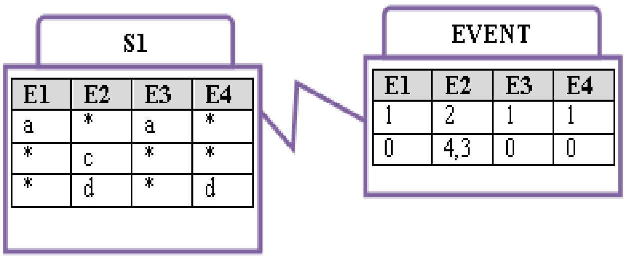


Figure 1: Locating items table for sequence S1

Here in figure 1, the sequence S1 is considered and S1 consists of 4 events [E1,E2,E3,E4] and the unique items in this sequence is identified and located in the corresponding events as shown in the above figure 1. Similarly number of items in each event is found and if it found value exceeds 1, the corresponding item's numerical value is stored as shown. Here in event indexing 4 denotes "d" and 3 denotes "c".

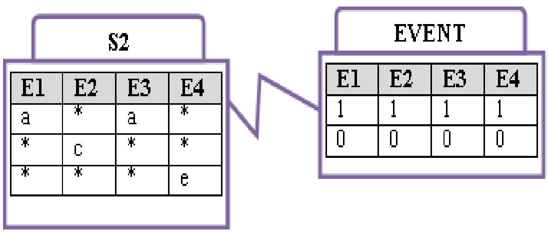


Figure 2: Locating item table for sequence S2

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

S3

E1	E2	E3	E4
*	a	*	*
*	*	*	b
c	*	*	c
*	*	d	d

EVENT

E1	E2	E3	E4
1	1	1	3
0	0	0	3,2,4

Figure 3: Locating item table for sequence S3

S4

E1	E2	E3
b	b	*
*	*	c

EVENT

E1	E2	E3
1	1	1
0	0	0

Figure 4: Locating item table for sequence S4

S5

E1	E2
b	*
c	*
d	d

EVENT INDEXING

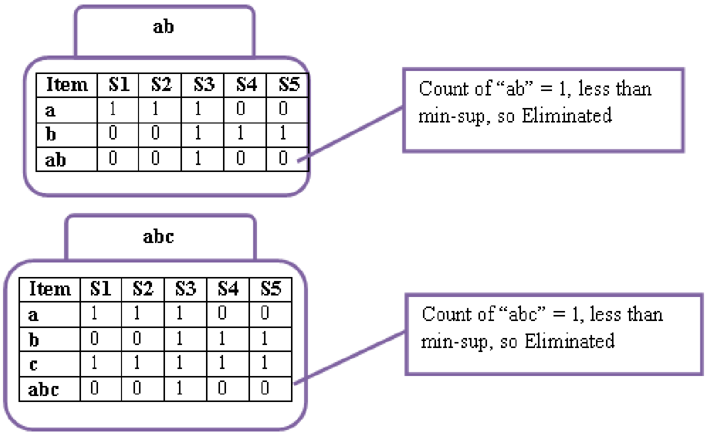
E1	E2
3	1
2,3,4	0

Figure 5: Locating item for sequence S5

FORMATION OF ITEM COMBINATIONS

After locating the items in the sequence, the item combinations are found to check whether the combinations are higher than the minimum support value provided by the user assuming mun-sup value is 2. First item “a” is considered and the probable combinations are formed for the entire unique items.

The combinations formed are “ab”, ”abc”, ”abd”, ”abcd”, ”ac”, “acd”, ”ad”, ”bc”, ”bcd”, ”cd”, are found. Here since “e” item support count is less than the minimum support value, it is eliminated. To check whether the combination items produce patterns, AND operation is performed as shown below.

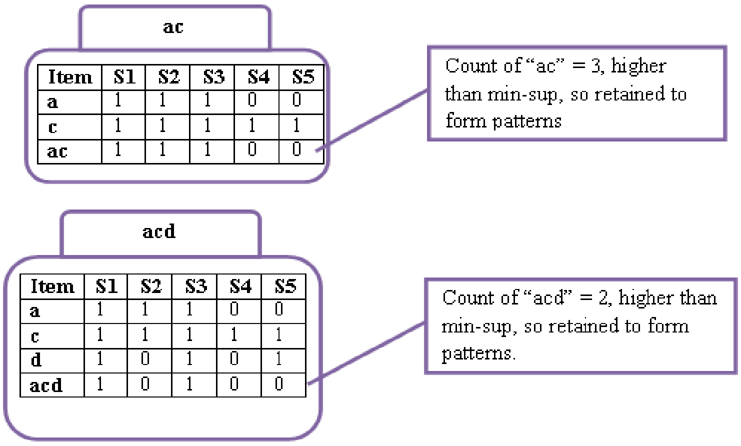


Item	S1	S2	S3	S4	S5
a	1	1	1	0	0
b	0	0	1	1	1
ab	0	0	1	0	0

Item	S1	S2	S3	S4	S5
a	1	1	1	0	0
b	0	0	1	1	1
c	1	1	1	1	1
abc	0	0	1	0	0

Figure 6: Combination formation of “ab” to generate frequent patterns

Since “ab” support count is 1, the corresponding itemsets which contains “ab” will also be 1. So the entire “ab” combinations are eliminated and pruned away.



Item	S1	S2	S3	S4	S5
a	1	1	1	0	0
c	1	1	1	1	1
ac	1	1	1	0	0

Item	S1	S2	S3	S4	S5
a	1	1	1	0	0
c	1	1	1	1	1
d	1	0	1	0	1
acd	1	0	1	0	0

Figure 7:Combination formation of “ac” to generate frequent patterns

Since “e” is eliminated at the first stage, the 4-itemset combination is not considered.

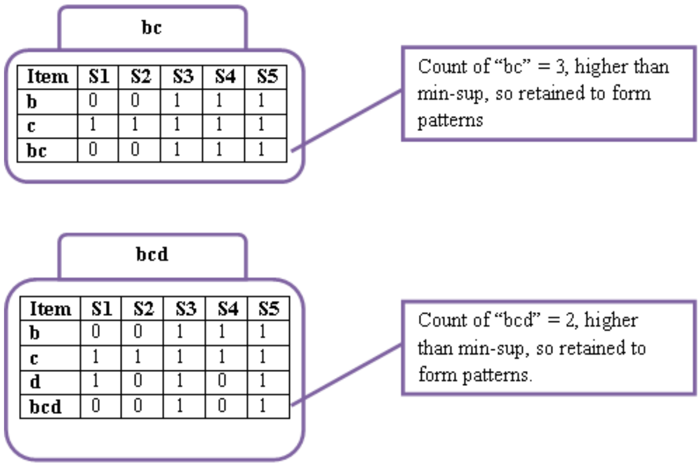


Figure 8: Combination formation of "bc" to generate frequent patterns

Similarly all the unique item combinations are formed and checked whether the count is higher or equal to the minimum support value provided. Those combinations which are higher or equal to the minimum support values are further processed top discover the frequent sequential patterns.

SEQUENTIAL PATTERN GENERATION

Since "ac" combination has a higher min-sup value than the threshold value provided, the result of "ac" is scrutinized. The items "a" and "c" are present in three sequences namely S1,S2, and S3 as shown in figure 3.The corresponding S1, S2 and S3 locating item table is fetched for processing to find the frequent sequential patterns as shown in figure 5. Here the minimum support is assumed to be 2.

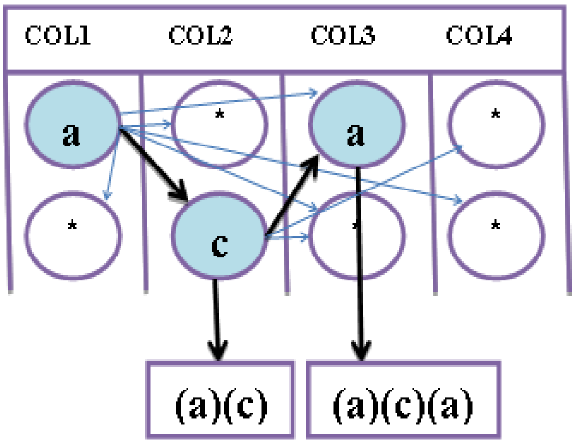


Figure 9: Pattern discovering mechanism for S1

First the column 1 (COL1) is checked for a non-empty value, if found the process starts from that row. Move from the first column towards right, up or down to identify a non-empty value to concatenate and form the sequential patterns.

Here the event indexing value is not considered because the numerical values present in the event indexing table is not matched with these items present here. The numerical values in event indexing table for the second event or the second column is (4,3) that is equal to (d,c). Here the patterns (a)(c) , (a)(c)(a) are formed and stored separately to check the min_sup.

Patterns found in S1 = (a)(c), (a)(c)(a)

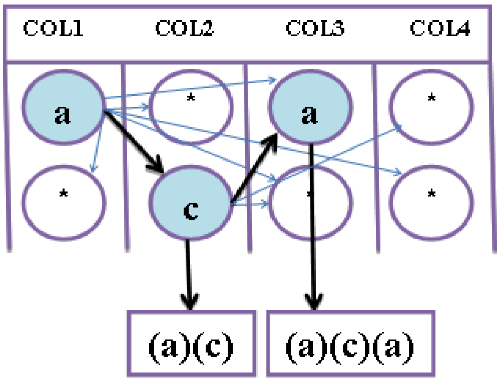


Figure 10: Pattern discovering mechanism for S2

Patterns found in S2 = (a)(c), (a)(c)(a)

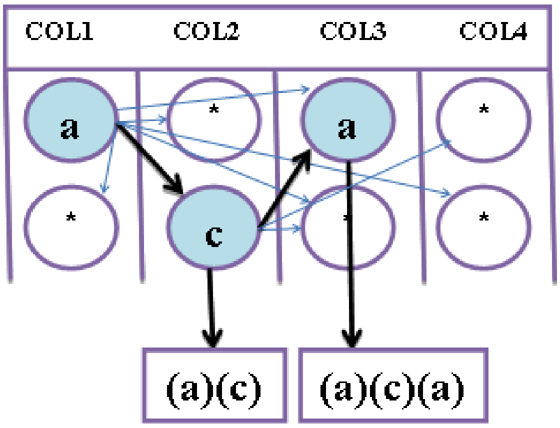


Figure 11: Pattern discovering mechanism for S3

Patterns found in S3 = (a)(c), (c)(a)(c), (c)(a), (c)(c)

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

Hence (a)(c) = 3, (a)(c)(a) = 2, are frequent sequential patterns whereas (c)(a) = 1, (c)(c) = 1 and (c)(a)(c) = 1 are less than the min-sup value provided are considered infrequent.

Similarly for "acd" the patterns are formed and checked. S1 and S3 contain the items "a", "c" and "d". These two sequences are computed to form a sequential pattern which has a higher min-sup value.

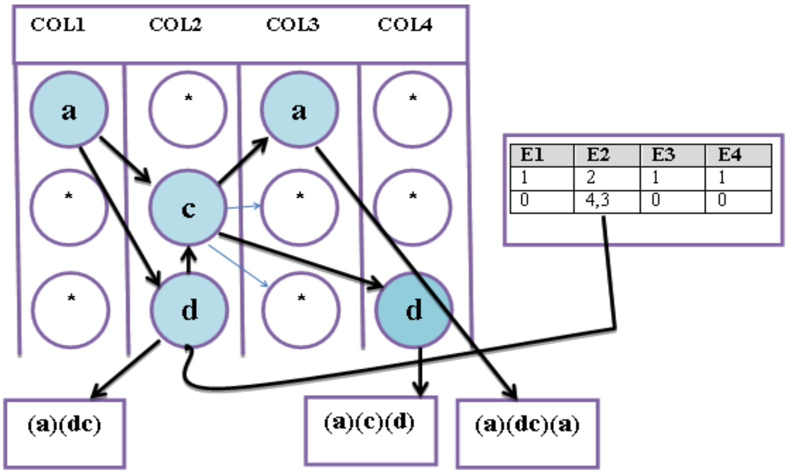


Figure 12: Pattern discovering mechanism for S1 -3items

Here the event indexing table plays its part by not considering (cd) instead it considers (dc) in the second column as the numerical value is fetched and accordingly the items in the sub-sequences are grouped to form the patterns. From the first row, since the column 1 of row 1 is not empty, the value present in col1 of row 1 is concatenated with row 3 and then moves up to row 2. The sequential patterns found here are (a)(dc), (a)(c)(d) and (a)(dc)(d). Similarly for the sequence S3 the patterns are found. The patterns found for sequence S3 = (c)(a)(d), (a)(d)(c), (a)(cd), (c)(a)(d)(c), (c)(a)(d)(d). None of the patterns found here are frequent as the values are less than the minimum support values. Considering the sequence S1, where the second event is (dc) and in the sequence S3, the fourth event is (cbd). In the existing algorithms like Spade, Apriori-All, and SPAM, the sequences will be sorted and as a result of this ordering (a)(cd) will be a frequent sequential pattern. But in the proposed methodology, (a)(dc) is different from (a)(cd), hence the items "a","c","d" are not formed as a frequent pattern.

PROPOSED ALGORITHM

The algorithm consists of many procedures and the procedures are enumerated here,

Procedure InitialProcess(Dataset D, min-sup ς)
Input: Dataset D OutPut: Normalized Converted Data *Begin:* 1. Load and Scan the Dataset D

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

```
        2.      ∀DatarowDr∈ D do
        Mark UniqueItems in InitialTable
        Mark Numerical Value for UniqueItems in InitialTable
        Mark Items in ItemLocation Table
        Find NumberOfEventsnE∈Dr
        Index the Item numerical values to ItemLocation Table
        IF [nE = 1] Mark 0
        Else Mark Numerical Values separated by comma
        End IF
        End For
        3.      Mark Count of UniqueItems, IF[ count <ς ] Prune UniqueItem
End Procedure
```

Figure 13: Pseudo code for InitialProcess procedure

In this proposed method the procedure initial process first scans the database to identify the unique items. Step 2 finds the location of the item support in a dataset. After identify the location a numerical value is assigns for every data row. If the items were present in the sequence assign 1 else assign 0. Step 3 produce the result of support count and finally prune the unique items.

```
Procedure FindProbableCombinations(InitialTableInT)
Input: InitialTable InT
Output: Probable combinations
Begin:
        1.      Find The totalItems Tot in InT
        2.      ∀Index I ∈Tot
                        3.      ∀ Index I+1 ∈ Tot
        ConcateInT[I],InT[I+1]→ Combination
        4.      End For
        5.      End For
        6.      Return Combination
End procedure
```

Figure 14: Pseudo code to Find Combinations

This algorithm proposed a new procedure to find a probable combination of unique items. Step 1 finds the total number of unique item to identify the combination. Step 2 identify the index value to locate the items. In Step 3 the Item combinations are found to check whether the combinations are higher than the minimum support value provided by the user. The result is the combination of items produce a patterns.

```
Procedure Data-as-it-IS( Dataset D, min-sup ς)
Input: Dataset D, minimum support ς
Output: Frequent Sequential patterns
Begin:
        1.      Call InitialProcess(D,ς)
        2.      Call FindProbableCombinations(T)
```

```
        3.      ∀ Combination C ∈ Combination do
        Loop Until (Sequences in C ≠ ϕ)
        Load Sequence from ItemLocationTable according to C
        Check for a non-empty value in the first Column
        IF Row[Column1] = Non-empty, Start from that row
        Concatenate the non-empty value by moving UP,Right, Down, Diagonal
        IF (Items found in same Column)
        Check the EventIndex table to find the numerical values
        Concatenate according to numerical values as a single sequence
        End if
        Store the patterns → temp
        End if
        End Loop
        Count the patterns and check ≥ς
        Store frequent sequentialpatterns
        Else
        Ignore infrequent patterns
        End For
        Return Frequent sequential patterns
End Procedure
```

Figure 15: Pseudo code of the DAIS algorithm

The proposed algorithm DAIS is the way to discover the frequent sequential patterns without ordering the items. At first the column 1 is checked for a non-empty value. If the non-empty value found, the process starts from that row. Move from the first column towards right, up or down to identify a non-empty value. After identify the non-empty value, all the items were concatenate and form the sequential patterns. This method employs event item indexing for each sequence and uses numerical values for events which contains more than one item. This methodology efficiently finds and discovers the frequent sequential patterns without disturbing the data present in the events of a sequence.

EXPERIMENTAL EVALUATION

The proposed DAIS algorithm was implemented Microsoft C#.NET programming language on a personal computer with 2.66GHz Intel Pentium core2duo processor with 1GB RAM running on windows 7 ultimate.

The evaluations were performed on synthetic data generated by the IBM synthetic market-basket data generator. The inputted parameters used for comparison are given below in the table 3.

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

Table 3: Dataset parameters

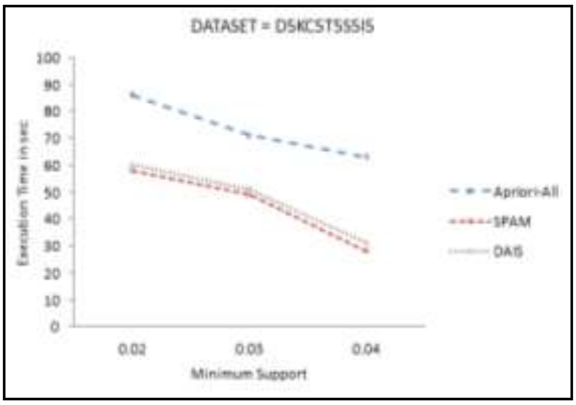
Parameters	Description of parameter	Dataset utilized for evaluation	
D	Number of sequences in the dataset	3K	12K
C	Average events per sequence	5	12
S	Average length of potentially frequent sequential patterns	5	8
I	Average length of itemsets in maximal potentially frequent patterns	5	8

The evaluation is done using different minimum support values and varying number of total sequences in the dataset, the proposed algorithm clearly outscores Apriori-All and matches nearly with the execution speed of SPAM. The proposed algorithm is executed on small, medium and large datasets and it clearly performs better on most of the situations with respect to speed and memory space. The DAIS and SPAM algorithm performed well in large dataset due to the recursive steps it performs during pattern finding. The proposed algorithm performed quite well on small and medium datasets when compared to its counterparts. As far as the memory footprints are concerned, SPAM performed reasonably better than the proposed algorithm mainly due to the bitmap representation of data. The number of frequent sequential patterns found in the proposed algorithm is definitely lower than that of the existing algorithm since the proposed algorithm never sorts the sub sequences.

COMPARISON WITH RESPECT TO EXECUTION TIME (sec)						
Dataset	**D5KC5T5S5I5** (Synthetic Data set)			**D12KC12T12S8I8** (Synthetic Data set)		
Algorithm	**Minimum Support values**			**Minimum Support values**		
	2	**3**	**4**	**5**	**6**	**7**
Apriori-All	86	71	63	112	103	81
SPAM	58	49	28	37	29.2	23
DAIS	60	51	31	38	28	23.32

Table 4: Evaluated results with respect to execution time

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)



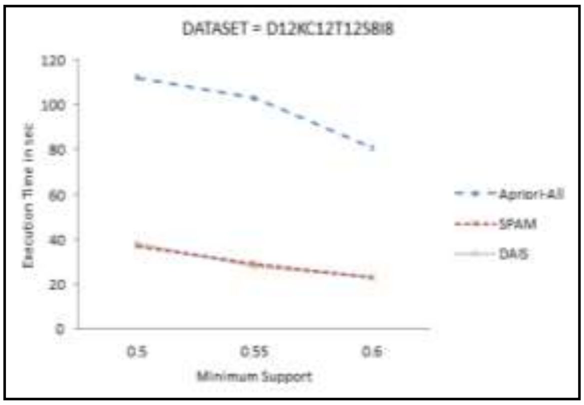


Figure 16(a): Execution time evaluation

Figure 16(b): Execution time evaluation

From the table 4 and from the figure 16(a),16(b) it is quite clear that the proposed algorithm DAIS performed well when compared to Apriori-All and matched the performance of SPAM. SPAM algorithm performance is better for small dataset, and DAIS performed extremely well for large dataset.

CONCLUSION

A new algorithm to meet the challenge of discovering sequential patterns without ordering the events in a sequence is proposed in this paper and the proposed algorithm DAIS performed increasing well for larger datasets without sorting by employing event item indexing using numerical values. The proposed algorithm should be executed on denser datasets with very small min-sup values to test the efficiency and the accuracy.

REFERENCES

[1] R. Agarwal and R. Srikant. Mining Sequential Pattern, Fast Algorithm for Mining Association Rule in Large Databases.In Proc.1995,1994 Int. Conf. Data Engineering, pages 3-10, 1995., Int. Conf. Very Large DataBases, pp. 487-499, 1994.

[2] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential PAttern Mining Using A Bitmap Representation. In Proc. 2002 Int. Conf. Knowledge Discovery and Data Mining, 2002.

[3] Philippe Fournier-Viger1, Antonio Gomariz2,Manuel Campos2, and Rincy Thomas3: In Proc.2014 Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information,University de Moncton

[4] Mohammed Al-Maolegi1: In Proc.2014 An improved apriori algorithm for association rules. International Journal on Natural Language Computing (IJNLC) Vol. 3.

[5] Nikhil Gundawar1, Venkatesh Akolekar2, Piyush Phalak3, Akshay Gujar4 and
L. A. Bewoor5: In Proc.2014 Analysis of Sequential Pattern Mining: international journal for research in emerging science and technology, volume-1, issue-6.

[6] Dr. Sunita Mahajan 1, Prajakta Pawar 2and Alpa Reshamwala: In Proc.2014 Performance Analysis of Sequential Pattern Mining Algorithms on Large Dense Datasets-
International Journal of Application or Innovation in Engineering & Management

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

[7] Bhawna Mallick, Deepak Garg and Preetam Singh Grover. Constraint-Based Sequential Pattern Mining: In Proc. 2014 The International Arab Journal of Information Technology, Vol. 11, No. 1, January 2014.

[8] Goswami, Chaturvedi Anshu, Raghuvanshi: In Proc.2010 An Algorithm for Frequent Pattern Mining Based On Apriori. International Journal on Computer Science and Eng

[9] Ms Shweta, Dr. Kanwal Garg2:In Proc.2013Mining Efficient Association Rules Through Apriori Algorithm Using Attributes and Comparative Analysis of Various Association Rule Algorithms. International journal of advanced research in computer science and software engineering.

[10] C K Bhensdadia, Y P Kosta: In Proc.2012 An Efficient Algorithm for Mining Frequent Sequential Patterns and Emerging Patterns with Various Constraints International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-1, Issue-6.

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)