Journal of Computer Science 7 (9): 1325-1329, 2011
ISSN 1549-3636

Fast Algorithms for Discovering Sequential Patterns in Massive Datasets

[1]S. Dharani, [2]Justus Rabi and [3]Nanda Kumar and [4]Darly
[1]Department of CSE, St.Peter's University, Avadi, 54, Chennai, India
[2]Department of EEE, Toc H institute of Science and Technology, 682 313, Cochin,
[3]Department of CSE, A.C.S. College of Engineering, 560074, Bangalore, India
[4]Department of IEEE, Anna University, Chennai, India

Abstract: Problem statement: Sequential pattern mining is one of the specific data mining tasks, particularly from retail data. The task is to discover all sequential patterns with a user-specified minimum support, where support of a pattern is the number of data-sequences that contain the pattern. **Approach:** To find a sequence patterns variety of algorithm like AprioriAll and Generalized Sequential Patterns (GSP) were there. We present fast and efficient algorithms called AprioriAllSID and GSPSID for mining sequential patterns that were fundamentally different from known algorithms. **Results:** The proposed algorithm had been implemented and compared with AprioriAll and Generalized Sequential Patterns (GSP). Its performance was studied on an experimental basis. We combined the AprioriAllSID algorithm with AprioriAll algorithm into a Hybrid algorithm, called AprioriAll Hybrid. **Conclusion:** Implementation shows that the execution time of the algorithm to find sequential pattern depends on total no of candidates generated at each level and the time taken to scan the database. Our performance study shows that the proposed algorithms have an excellent performance over the best existing algorithms.

Key words: Data mining, sequential pattern mining, apriori all hybrid, proposed algorithm, temporary database, candidate sequences, minimum support

INTRODUCTION

Data Mining is the process of extracting useful information which is hidden in large databases. The knowledge or pattern mined could be used to make decisions. Sequential pattern mining is one of the major areas of research in the field of data mining. Sequential pattern mining is used to discover frequent sequences as patterns in a database. Several algorithms have been proposed to find sequential pattern (Changsheng *et al.*, 2009; Zhang *et al.*, 2009). First AprioriAll algorithm was introduced to find all sequential patterns. For finding generalized sequential patterns GSP (Generalized Sequential Patterns) was presented. To find sequential patterns from large amount of transaction data requires multiple passes over the database. We propose efficient algorithms namely AprioriAllSID and GSPSID to improve the performance by reducing the scale of the candidate item set C_k and the spending of I/O (Wang, 2010; Yong-Qing *et al.*, 2009; Yang *et al.*, 2009).

The original database is read only one time and we introduce a new temporary database D' for the next iterations. After completing the first iteration, we can find the candidate sequence of size-2 using temporary database D. Then we can find the candidate k-size sequences until the candidate sequence or temporary database size is empty. At this stage the database size is reduced as well as the number of candidate sequences are also reduced (Suneetha and Krishnamoorti, 2010; Liu, 2010). This feature is used for finding sequential patterns easily and efficiently reduced the time complexity. So the proposed methods are efficient than all other methods like AprioriAll and Generalized Sequential Patterns (GSP). Relative performance study of AprioriAllSID and GSPSID is given.

Problem statement: The problem of mining sequential patterns can be stated as follows: Let I = $\{i_1,i_2,..,i_m\}$ be a set of m distinct attributes, also called items. An itemset is a non-empty unordered collection of items (without loss of generality, we assume that items of an itemset are sorted in increasing order). All items in an itemset are assumed to occur at the same time. A sequence is an ordered list of itemsets. An itemset i is denoted as $(i_1,i_2,\ldots,i_k)$, where i_j is an item. An itemset with k items is called a k-itemset. A sequence s is denoted as $(s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_q)$, where the sequence element s_j is an itemset. A sequence with k-items (k =

Corresponding Author: S. Dharani, Department of CSE, St.Peter's University, Avadi, Chennai, 54, India

$\sum_j |\alpha_j|$) is called a k-sequence. For example, (B→ AC) is a 3-sequence. An item can occur only once in an itemset, but it can occur multiple times in different itemsets of a sequence.

A sequence p = ($p_1 \rightarrow p_2 \rightarrow \ldots \rightarrow p_n$) is a subsequence of another sequence q = ($q_1 \rightarrow q_2 \rightarrow \ldots \rightarrow q_n$), denoted as $p \rightarrow q$, if there exist integers $i_1 < i_2 < \ldots < i_n$, such that $p_j \subseteq q_{ij}$ for all p_j. For example the sequence (B→AC) is a subsequence of (AB→E →ACD), since the sequence elements B→ AB and AC→ACD. On the other hand the sequence (AB→ E) is not a subsequence of (ABE) and vice-versa. We say that p is a proper subsequence of q, denoted as $p \subset q$, if $p \subset q$ and $p \subset q$.

A transaction T has a unique identifier and contains a set of items, i.e., $T \subseteq I$. A customer C has a unique identifier and has associated with it a list of transactions {T_1, T_2,…,T_n}. We assume that no customer has more than one transaction with the same time-stamp, so that we can use the transaction-time as the transaction identifier. We also assume that the list of customer transactions is stored by the transaction-time. Thus the list of transactions of a customer is itself a sequence $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_n$ called the customer sequence. The database D consists of a number of such customer sequences.

A customer sequence C is said to contain a sequence p, if $p \subseteq q$ i.e., p is a subsequence of the customer sequence C. The support or frequency of a sequence C is denoted as σ (p), is the total number of customers that contains this sequence. Given a user-specified threshold called minimum support (denoted min-sup) we say that a sequence is frequent if it occurs more than minimum support times. The set of frequent k-sequences is denoted as F_k. A frequent sequence is maximal if it is not a sub sequence of any other sequence.

The problem of finding sequential patterns can be decomposed into two parts:

- Generate all combinations of customer sequences with fractional sequence support (i.e., support_D (C)/|D|) above a certain threshold called minimum support m
- Use the frequent sequences to generate sequential patterns
- The second sub problem is straightforward. However discovering frequent sequences is a non-trivial issue, where the efficiency of an algorithm strongly depends on the size of the candidate sequences

MATERIALS AND METHODS

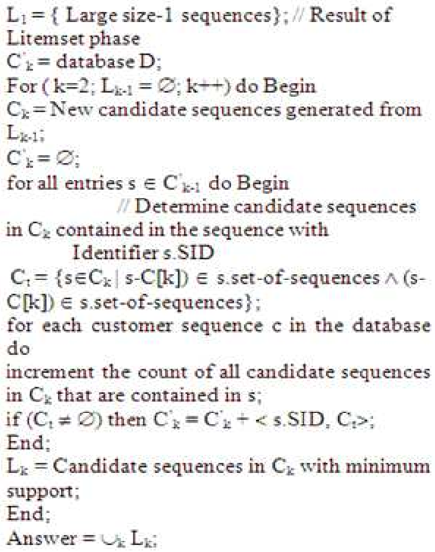
AprioriAllSID: The AprioriAllSID algorithm has shown in Fig. 1. An interesting feature of the proposed algorithm is that the given customer transaction database D is not use for counting support after the first pass. Rather the set C_k is used for determining the candidate sequences before the pass begins.

Each member of the set C_k is of the form < SID, {S_k} > where each S_k is a potentially frequent k-sequence present in the sequence with identifier SID. For k=1, C_1 corresponds to the database D, although conceptually each sequence i is replaced by the sequence {i}. For k>1, C_k is corresponding to customer sequence S is< s.SID, {s<C_k | s contained in t}>. If s customer sequence does not contain any candidate k-sequence, then C_k will not have an entry for this customer sequence.

Thus, the number of sequences in the database is greater than the number of entries in C_k. The number of entries in C_k may be smaller than the number of sequences in database especially for large value of k. In addition, for large values of k, each entry may be smaller than the corresponding sequence because very few candidate sequences may be contained in the sequence.

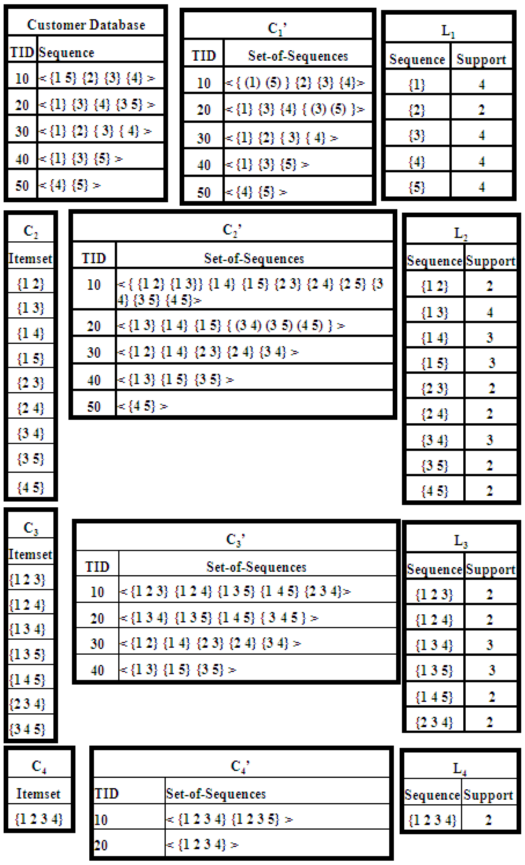
However, for small values of k, each may be larger than the corresponding sequence because an entry in C_k includes all candidate k-sequences contained in the sequence.

Algorithm AprioriAllSID: In Fig. 1, we present an efficient algorithm called AprioriAllSID, which is used to discover all sequential patterns in large customer database.



```
$L_1$ = { Large size-1 sequences}; // Result of
Litemset phase
$C'_k$ = database D;
For ( k=2; $L_{k-1} = \emptyset$; k++) do Begin
$C_k$ = New candidate sequences generated from
$L_{k-1}$;
$C'_k = \emptyset$;
for all entries s $\in C'_{k-1}$ do Begin
                    // Determine candidate sequences
in $C_k$ contained in the sequence with
          Identifier s.SID
 $C_t$ = {s$\in C_k$ | s-C[k]) $\in$ s.set-of-sequences $\wedge$ (s-
C[k]) $\in$ s.set-of-sequences};
for each customer sequence c in the database
do
increment the count of all candidate sequences
in $C_k$ that are contained in s;
if ($C_t \neq \emptyset$) then $C'_k = C'_k$ + < s.SID, $C_t$>;
End;
$L_k$ = Candidate sequences in $C_k$ with minimum
support;
End;
Answer = $\cup_k L_k$;
```

Fig. 1: Algorithm AprioriAllSID



Customer Database	
TID	Sequence
10	< {1 5} {2} {3} {4} >
20	< {1} {3} {4} {3 5} >
30	< {1} {2} { 3} { 4} >
40	< {1} {3} {5} >
50	< {4} {5} >

C_1'	
TID	Set-of-Sequences
10	< { (1) (5) } {2} {3} {4}>
20	< {1} {3} {4} { (3) (5) }>
30	< {1} {2} { 3} { 4} >
40	< {1} {3} {5} >
50	< {4} {5} >

L_1	
Sequence	Support
{1}	4
{2}	2
{3}	4
{4}	4
{5}	4

C_2
Itemset
{1 2}
{1 3}
{1 4}
{1 5}
{2 3}
{2 4}
{3 4}
{3 5}
{4 5}

C_2'	
TID	Set-of-Sequences
10	< { {1 2} {1 3}} {1 4} {1 5} {2 3} {2 4} {2 5} {3 4} {3 5} {4 5}>
20	< {1 3} {1 4} {1 5} { (3 4) (3 5) (4 5) } >
30	< {1 2} {1 4} {2 3} {2 4} {3 4} >
40	< {1 3} {1 5} {3 5} >
50	< {4 5} >

L_2	
Sequence	Support
{1 2}	2
{1 3}	4
{1 4}	3
{1 5}	3
{2 3}	2
{2 4}	2
{3 4}	3
{3 5}	2
{4 5}	2

C_3
Itemset
{1 2 3}
{1 2 4}
{1 3 4}
{1 3 5}
{1 4 5}
{2 3 4}
{3 4 5}

C_3'	
TID	Set-of-Sequences
10	< {1 2 3} {1 2 4} {1 3 5} {1 4 5} {2 3 4}>
20	< {1 3 4} {1 3 5} {1 4 5} { 3 4 5 } >
30	< {1 2} {1 4} {2 3} {2 4} {3 4} >
40	< {1 3} {1 5} {3 5} >

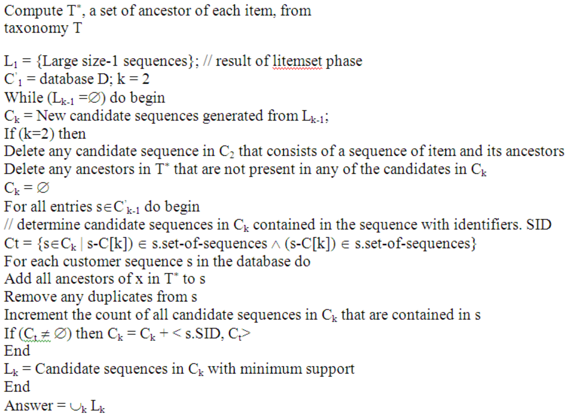
L_3	
Sequence	Support
{1 2 3}	2
{1 2 4}	2
{1 3 4}	3
{1 3 5}	3
{1 4 5}	2
{2 3 4}	2

C_4
Itemset
{1 2 3 4}

C_4'	
TID	Set-of-Sequences
10	< {1 2 3 4} {1 2 3 5} >
20	< {1 2 3 4} >

L_4	
Sequence	Support
{1 2 3 4}	2

Fig. 2: Example



```
Compute $T^*$, a set of ancestor of each item, from
taxonomy T

$L_1$ = {Large size-1 sequences}; // result of litemset phase
$C'_1$ = database D; k = 2
While ($L_{k-1} = \varnothing$) do begin
$C_k$ = New candidate sequences generated from $L_{k-1}$;
If (k=2) then
Delete any candidate sequence in $C_2$ that consists of a sequence of item and its ancestors
Delete any ancestors in $T^*$ that are not present in any of the candidates in $C_k$
$C_k = \varnothing$
For all entries $s \in C'_{k-1}$ do begin
// determine candidate sequences in $C_k$ contained in the sequence with identifiers. SID
Ct = {$s \in C_k$ | s-C[k]) $\in$ s.set-of-sequences $\wedge$ (s-C[k]) $\in$ s.set-of-sequences}
For each customer sequence s in the database do
Add all ancestors of x in $T^*$ to s
Remove any duplicates from s
Increment the count of all candidate sequences in $C_k$ that are contained in s
If ($C_t \neq \varnothing$) then $C_k = C_k$ + < s.SID, $C_t$>
End
$L_k$ = Candidate sequences in $C_k$ with minimum support
End
Answer = $\cup_k L_k$
```

Fig. 3: Algorithm GSPSID

For example, consider the database in Fig. 2 and assume that minimum support is 2 customer sequences. By using candidate-gen procedure with size-1 of frequent sequences gives the candidate sequence in C_2 by iterating over the entries in $C^{'}_2$ and generates $C^{'}_2$ in step 6-11 of Fig. 1. The first entry in $C_1^{'}$ is < {(1) (5)} {2} {3} {4}> corresponding to customer sequence 10. The C_t at step 7 corresponding to this entry s is {{(1) (5)} {2} {3} {4}} are members of s.set-of-sequences.

By using Candidate-gen procedure with L_2 gives C_3. Making pass over the data with $C^{'}_2$ and C_3 generates $C_3^{'}$. This process is repeated until there is no sequence in the customer sequence database.

Algorithm GSPSID: In Fig. 3, we propose an efficient algorithm called GSPSID, which is used to discover all generalized sequential patterns in large customer database.

We add optimizations to GSP algorithm, which gives the algorithm GSPSID. In GSPSID algorithm, given original database D is not used for counting after the first pass. The first pass of algorithm determines the support of each item, like GSP algorithm. At the end of first pass, the algorithm knows which items are frequent, i.e., has minimum support. We introduce the temporary database $D^{'}$ which is used to determine the candidate sequences before the pass begins. The member of that temporary database is of the form <SID, {S_k}>, where each S_k is a potentially frequent k-sequence present in the sequence with identifier SID.

For k = 1, the C_1 is the corresponding temporary database $D^{'}$. If k = 2, then we add three optimizations, to reduce the size of the database. If a customer sequence does not contain any candidate k-sequence, then $C_k^{'}$ will not have an entry for this customer sequence. Thus, the number of sequences in the database is greater than the number of entries in $C_k^{'}$. Conversely, the number of entries in $C_k^{'}$ may be smaller than the number of sequences in database especially for large value of k. In addition, for large values of k, each entry may be smaller than the corresponding sequence because very few candidate sequences may be contained in the sequence. For small values of k, each may be larger than the corresponding sequence because an entry in C_k includes all candidate k-sequences contained in the sequence.

Apriori All Hybrid algorithm: We combine the AprioriAllSID and AprioriAll algorithms to get the Apriori All Hybrid. This shows that the first iteration of AprioriAll algorithm and in the later iteration with AprioriAllSID gives the Apriori All Hybrid algorithm. Both algorithms are efficient, but when compared with AprioriAllSID, Apriori All Hybrid is faster.

Both algorithms use same data structures. Each candidate sequence is assigned a unique number called its SID. Each set of candidate sequence $C_k^{'}$ is kept in an array indexed by the IDs of the sequences in C_k. So, a member of $C_k^{'}$ is of the form ⟨SID, {ID}⟩. Each $C_k^{'}$ is stored in a sequential structure.

There are two additional fields maintained for each candidate sequence. They are:

- Generators: This field of sequence C_k stores the IDs of the two maximal (k-1) sequence whose join generated C_k
- Extensions: This field stores IDs of all the sequences C_{k+1} obtained as an extension of C_k

Now, s.set-of-sequence of C'_{k-1} gives the IDs of all the (k-1)-candidate sequence contained in transaction s.SID. For each such candidate sequence $C_{k-1}^{'}$ the extensions field gives S_k the set of IDs of all the candidate k-sequences that are extensions of C_{k-1}. For C_k in S_k the generators field gives the IDs of the two sequences that generated C_k. If these sequences are present in the entry for s.set-of-sequences, C_k is present in customer sequence s.SID. Hence we add C_k to C_t, by using this data structure we can efficiently stored and processed the candidate sequences.

RESULTS

We describe the experiments and the performance results of AprioriAllSID algorithms. We also compare the performance with the AprioriAll and GSP algorithms. We performed our experiments on an IBM Pentium machine. Using data set generator, we have simulated the data and test algorithms like AprioriAll, AprioriAllSID, GSP and GSPSID

Performance evaluation: We have used the simulated data for the performance comparison experiments. The data sets are assumed to simulate a customer-buying pattern in a retail environment.

In the sets. The Table 1 and 2 shows the performance of AprioriAll, GSP, AprioriAllSID and GSPSID for minimum support 1-5% for different volume of data. Even though AprioriAllSID and GSPSID seems to bperformance comparison, we used the five different date nearly equal, for massive volume of data, the performance of AprioriAllSID and GSPSID will be for better than AprioriAll and GSP algorithms.

DISCUSSION

In Table 1 and 2 shows the execution times for the five data sets for an increasing value of minimum support (say 1-5%). The execution times increase for both AprioriAllSID and AprioriAll algorithms and GSP and GSPSID as the minimum support is decreased because the total number of candidate sequence increases. AprioriAll algorithm and GSP are the multiple passes over the data.

CONCLUSION

The execution time is increased with increase of the customer transactions in the database. In Table 1 and 2, we can conclude that the AprioriAllSID algorithm is 2 times faster than AprioriAll algorithm and GSPSID algorithm is 3 times faster than GSP for small volume of data and more than order of magnitude for the large volume of data. The data sets ranges from giga bytes to tera bytes the proposed algorithms will be very much faster than AprioriAll and GSP. Thus we conclude that the proposed algorithms are very much suitable for massive databases.

Table 1: Performance evaluation between AprioriAll and AprioriAllSID algorithms

	AprioriAll (Execution time in sec) (%)					AprioriAllSID (Execution time in sec) (%)				
DB size	1	2	3	4	5	1	2	3	4	5
100 K	187	199	211	228	245	98	121	148	164	183
200 K	325	339	351	367	384	174	192	221	246	265
300 K	428	447	465	489	510	269	281	301	324	356
400 K	559	587	611	638	669	346	371	392	415	458
500 K	678	691	726	758	793	489	514	561	592	636

Table 2: Performance evaluation between GSP and GSPSID algorithms

	GSP (Execution time in sec) (%)					GSPSID (Execution time in sec) (%)				
DB size	1	2	3	4	5	1	2	3	4	5
100 K	149	188	213	249	274	67	98	121	149	162
200 K	251	289	308	329	368	123	146	178	204	238
300 K	339	368	392	421	467	212	258	298	332	378
400 K	426	467	493	527	569	320	357	398	435	481
500 K	512	541	570	518	536	404	449	481	523	556

REFERENCES

Changsheng, Z., L. Zhongyue and Z. Dongsong, 2009. An improved algorithm for apriori. Proceedings of the 1st International Workshop on Education Technology. Computer Science, Mar. 7-8, IEEE Xplore Press, Wuhan, Hubei, pp: 995-998. DOI: 10.1109/ETCS.2009.227

Liu, Y., 2010. Study on application of apriori algorithm in data mining. Proceedings of the 2nd International Conferences on Computer Modeling and Simulation, Jan. 22-24, IEEE Xplore Press, Sanya, Hainan, pp: 111-114. DOI: 10.1109/ICCMS.2010.398

Suneetha, K.R. and R. Krishnamoorti, 2010. Advanced version of a priori algorithm. Proceedings of the 1st International Conference on Integrated Intelligent Computing, Aug. 5-7, IEEE Xplore Press, Bangalore, pp: 238-245. DOI: 10.1109/ICIIC.2010.64

Wang, X., 2010. Study of data ming based on Apriori algorithm. Proceedings of the 2nd International Conference on Software Technology and Engineering, Oct. 3-5, IEEE Xplore Press, San Juan, PR. pp: 400-403. DOI: 10.1109/ICSTE.2010.5608780

Yong-Qing, W., Y. Ren-Hua and L. Pei-Yu, 2009. An improved Apriori algorithm for association rules of mining. Proceedings of the IEEE International Symposium on IT in Medicine and Education, Aug. 14-16, IEEE Xplore Press, Jinan, pp: 942-946. DOI: 10.1109/ITIME.2009.5236211

Yang, G., H. Zhao, L. Wang and Y. Liu, 2009. An implementation of improved apriori algorithm. Proceedings of the International Conference on Machine Learning and Cybernetics, Jul. 12-15, IEEE Xplore Press, Baoding, pp: 1565-1569. DOI: 10.1109/ICMLC.2009.5212246

Zhang, C. and J. Ruan, 2009. A modified apriori algorithm with its application in instituting cross-selling strategies of the retail industry. Proceedings of the International Conference on Electronic Commerce and Business Intelligence, Jun. 6-7, IEEE Xplore Press, Beijing, pp: 515-518. DOI: 10.1109/ECBI.2009.121