An Advanced Model for Mining Time Interval Sequential Patterns in Stream data

K M V Madan Kumar

Abstract: Mainly existing sequential pattern mining algorithms are hard to find out long significant time-interval sequential patterns in information stream. In this paper, we propose a new bitmap-based algorithm of mining d time-interval sequential pattern in information stream called DSBMMS, which is based on binary bit counting and multiple time-interval sequential position. We transform the whole sequence database into bitmap and all operations into binary bit counting to get better progress regarding the efficiency of our algorithm. Moreover, instead of exact toning, we use multiple sequential positions to find out long significant time-interval sequential patterns. In the early stage of our algorithm, the information stream is separated into batches; we cluster the related sequences in each batch with our new distinct likeness or similarity. Then, we make sequential position on each cluster to symbol each item a weight. A new data structure called TIPT (time-interval sequence position table) is proposed to smooth the progress of the sequential position with time intervals. At last, the biased sequences are filtered with particular specified threshold to produce the d time-interval sequential pattern. Our experiments show that DSBMMS is fast and suitable to mining d sequential pattern in information stream.

Keywords: Data mining, Data stream, Time-interval sequential pattern, estimate sequential pattern, Bitmap.

1. Introduction

The sequential order is significant in numerous real-world applications. Agrawal and Srikant first introduced the concept of sequential pattern mining at 1995[1]. Intended for a sequence or a sequential pattern, not solitary the initiation arrange of data elements, except in addition their initiation time are imperative. The perception of mining time-interval sequential pattern was introduced by Chen, Chiang and Ko[2]. Pleasing time interval among consecutive item sets into account, some new algorithms were projected. AprioriLike time-interval mining algorithms [3] accomplish candidates’ initiation and check all the way through a breadth first rank by rank. During the first check, they discover the length-1 sequential patterns, that is, distinct frequent items in the database are achieved. During the second time, the length-2 sequential patterns are originated. On all level, merely possibility frequent candidates are created and experienced. Joong Hyuk Chang [4] projected a innovative sequence weight computation method support time intervals concerning every pair of succeeding item sets in sequence and a new projection-based depth-first search algorithm to mining time-interval sequence. It gets round the costly candidate invention by developing lengthy patterns as of short ones. Just the once a sequential pattern whose sequence weight is larger than individual porch is found, all sequences containing that blueprint are together as a proposed database to produce longer patterns. The uncontrolled and persistent characteristics of data torrent done various conventional algorithms immobilize [5].

ISSN: 2320 – 8007
Various modern algorithms for mining frequent item sets [6] and clustering [7] in data stream are projected period few for mining sequential pattern [8]. Inspired by the ideas for mining frequent item sets in data stream, Luiz F. Mendes projected the algorithm SS-BE and SS-MB [9] to extract sequential pattern with the scrupulous error bounce in data stream which extensive the PrefixSpan and Lossy Counting. Regrettably, it is additionally based exact matching and hard to discover the long meaningful sequences. To the paramount of our realization, SMDS [10] is the first algorithm that introduced the idea of multiple sequence alignment to mine sequential pattern in data stream.

In this paper, we have projected a new algorithm; in the direction of mine fairly accurate time-interval sequential patterns in data stream based bitmap and multiple time-interval sequence alignment called DSBMMS. Motivated by the initiative of bitmap [11] and the perception of mining d sequential pattern, our algorithm is designed to mine long meaningful time-interval sequences. It clusters the analogous sequences first, and then causes the weighted time-interval sequences by multiple sequence position with our new data structure TIAT (the time interval alignment table). At last, filtering the weighted time-interval sequences with specified threshold generates the d sequences. Additionally, all of the functions in our algorithm-based bitmap are renewed to expeditious bit counting that betters the efficiency of algorithm intensely.

The enduring of the paper is organized as follows. Section 2 introduces the foundational definitions. Section 3 develops DSBMMS algorithm and gives some examples. Section 4 presents our performance study. Section 5 contains concluding remarks.

2. Problem Definition

Let’s assume set of all items K= \{k1, k2... kn\}. Where Y is the Itemset and it is subset of K. where \(|Y|\) is the segment of item set are the items located in it. The order list of itemset is Sequence, represented like x=<x1, x2...xm>, where xk ∈ K and K = \{1... m\}. Here K=|x| is the segment of sequence is amount of itemset in the sequence, Intended in support of two sequence x1=<p1,p2,...,pn> and x2=<q1,q2,...,qm>, x1 is a subsequence of x2 if there be existent integers k1<k2<...kn, such that p1 ⊆ qk1,p2 ⊆ qk2,...,pn ⊆ qkn . Operation database as exposed in table 1 is a set of tuples <CID, TID, Itemset> represented like D.

Sequence database as exposed in table 2 is a set of tuples <CID, Sequence>, represented like S.

<table>
<thead>
<tr>
<th>CID</th>
<th>TID</th>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>{I1,I2,I4}</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>{I2,I3,I4}</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>{I2,I3,I4}</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>{I2}</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>{I1,I2,I3}</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>{I2,I3}</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>{I3}</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>{I2,I3,I4}</td>
</tr>
</tbody>
</table>

Table 1: Transaction database

Maintain of a sequence s is the amount of its super sequence in S, symbolize as support(s). A sequence is recurrent (or a sequence pattern) but it content
support(s) > minsup (minsup is a threshold particulate by user). Hire Sim be a resemblance compute by means of domain [0, 1]. For sequences x₁, x₂ and x₃, if Si(x₁, x₂) < Si(x₁, x₃), we call x₃ is extra similar to x₁ than x₂. d support is denoted as sup (xi) = ∑x ∈ Si(x, xi) and estimate sequential patterns are the sequences whose d support is superior than a précised threshold.

If we regard as the production time of itemset, a sequence A is represented as ((x₁,t₁),(x₂,t₂),(x₃,t₃)...(xₙ,tₙ)), where tⱼ named for the time stamp when xⱼ transpires,1≤ⱼ≤ₙ, and tⱼ-1≤tⱼ for 2≤ⱼ≤ₙ, hire t represent the time interval among two successive items, and let Tₙ be the given constants for 1≤l≤r-1. After that, the time interval is categorized into r+1 category, where: I₀ indicates the time interval t rewarding t=0.
I₁ denotes the time interval t satisfying 0≤t≤T₁
Iⱼ denotes the time interval t satisfying Tⱼ-1≤t≤Tⱼ for satisfying the condition that is: 1≤ⱼ≤r-1
Iᵣ denotes the time interval t satisfying Tr-₁< t<∞
Let the set of time intervals be represented as TI= {I₀, I₁, I₂... Iᵣ}. Then, the time-interval sequential pattern can be defined: a sequence B = { q₁, &₁, q₂, &₂,..., qₓ-₁, &₁-x, qₓ } is a time-interval sequence if qi ∈ I for 1≤i≤s and &i ∈ TI for 1≤i≤s-1.

The sequence database with the initiation time of items is shown in below table.

<table>
<thead>
<tr>
<th>CID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>&lt;(I₂):2, (I₁,I₂,I₃):4&gt;</td>
</tr>
</tbody>
</table>

Table 2: Sequence database with time stamp

<table>
<thead>
<tr>
<th>CID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;(I₁,I₂,I₄),2,(I₂,I₃,I₄),3, (I₂,I₃,I₄)&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;(I₂),2,(I₁,I₂,I₃)&gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt;(I₁,I₂),2,(I₃),4,(I₂,I₃,I₄)&gt;</td>
</tr>
</tbody>
</table>

Table 3: Time-interval Sequence Database (TSDB)

It’s effortless to renovate the sequence database like table 3 to the definitive time-interval sequence database as shown in table 4. Here the time interval value is virtual to the first itemset in sequence.

Our task is mining those estimated sequential patterns in the data stream, which is a very long subjective sequence list with time intervals as table 4 shown. We converted TSDB to bitmap as shown in Figure 1.

<table>
<thead>
<tr>
<th>Ti</th>
<th>I₁</th>
<th>I₂</th>
<th>I₃</th>
<th>I₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>011</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>000</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>000</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Bitmap of TSDB

In the bitmap, itemset is divided into each line; here different sequences are divided into with all zero lines. In this particular situation, 1 describes that corresponding item is enclosed and the 0 describes that item is not enclosed. And moreover the number of bit in a line depends on the number od different items in database. And Ti column means the time interval in between two itemset in sequence in binary. For instance, the value of Ti in line three is 011, it means the
time internal between the third itemset and the second itemset is 3 time units. Essentially we enlarge the number of bit in a line to an integer that can be divided by 8. We represent bitmap of sequence database as BS and bitmap of sequence as bs in this paper.

**Definition 1:** The density of sequence bitmap. For each bs, we denote STI(bs)=x/y as the density of sequence bitmap. x is the number of 1 in bs and y is the total number of bit.

**Definition 2:** Extended sequence bitmap. For two sequences si and sj (i, j≠0 and i, j∈N), Let row(bs) be the number of line in bs. If row(bsi)<row(bsj), we insert all zero line into bsi according to row(bsj). The result denoted as bsj’ is called extended sequence bitmap.

**Definition 3:** Absolute similarity between sequences. We denote bsij=bsi∧bsj as the result of AND operation between bsi and bsj. We call sim(si,sj)=STI(bsij) as the absolute similarity. Obviously, greater difference of sequence length leads lower similarity and longer common subsequence leads higher similarity.

**Definition 4:** Relative similarity between sequences. Relative similarity of si to sj is defined as

\[ \text{Sim}(si,sj)=\text{sim}(si,sj)/\text{STI}(si) \]

If \( \min(\text{Sim}(si,sj),\text{Sim}(sj,si))>\min\text{Sim} \), si and sj will share in the same cluster. Relative sequence similarity is shorted as sequence similarity in the rest of this paper.

### 3. Mining algorithm based on Bitmap

#### 3.1 Cluster progressions base on Bitmap

Algorithm 1 which is shown below explains Cluster algorithm. This algorithm is build up by using the concept of similarity of sequences and that does not consider time intervals. To improve the efficiency of our algorithm we implement two strategies.

1. **Random extraction:** One sequence of a cluster randomly to stand for the whole cluster to calculate similarity with the new arrival.

2. **Cycled calculation:** The algorithm needs to calculate the similarity with the existing clusters one by one. But instead of starting from the beginning we always continue the last comparing location.

**Table 4:** Sequence Database

<table>
<thead>
<tr>
<th>TID</th>
<th>CID</th>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Algorithm 1: Cluster progressions**

Input: sequences, simMin, batchsize
Output: a set of clusters
Let num be the current number of sequence;
Let cl point the current cluster to compare
Let lcs point the start location in this compare sequence
Begin:
1. while(num<batchsize){
2. //divide stream in batches with batchsize
3. for each sequence{
4. if( C is empty)
5. {
6. cl=new cluster;
7. //create a new cluster for the first sequence in
8. current batch
9. else{
10. lcs=cl;//record the start location
11. do{
12. sequenceTemp=random(cl)
13. if(min(Sim(x,ts),Sim(ts,x)<simMin)
14. //calculate the similarity and compare to
15. simMin threshold
16. cl. next();
17. //if the minimal similarity is less than the
18. Threshold, we’ll consider //next cluster.
19. else break;
20. }while(lcs<>cl)
21. if(lcs==cl) cl=new cluster;
22. //Algorithm will create a new cluster
23. //when it cannot find a similar one}
24. add s to cl;
25. }
26. }
27. End

Figure 1: The comprehensive bitmap of qx2

The input parameters simMin is the assessment threshold and batch size is the number of sequence in one batch. A set of clusters is the output consequence. When a new sequence is generated, a new cluster is produced if it is the first one; otherwise the similarity will be calculated by above two strategies (from line 13 and line 8 to line 22 shown in Algorithm 1). We must produce the absolute sequence if the two sequence by means of different row number prior to the calculation.

For example, as shown in Figure 1, row (qx2) <row (qx1), we must extend qx2 to qx2’ as shown in Figure 2. Let simMin=0.6 and batch size = 3, the sequence database as shown in table 2 will share the same batch. S1 is the first sequence of the batch, we ought to create a new cluster for it and indicate it as C1; Next, for S2, we will calculate Sim(x1, x2) and Sim(x2, x1); If min (Sim (x1, x2), Sim(x2, x1)) >= simMin, we should add x2 to c1, otherwise generate a new cluster for x2. First, we expand qx2 according to x1, denote it as qx2’, as shown in Figure 1. Next, we make AND operation between qx1 and qx2’ as shown in Figure 2.

<table>
<thead>
<tr>
<th>CID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;(I1,I2,I4),(I2,I3,I4),(I2,I3,I4)&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;(I2),(I1,I2,I3)</td>
</tr>
<tr>
<td>3</td>
<td>&lt;(I1,I2),(I3),(I2,I3,I4)&gt;</td>
</tr>
</tbody>
</table>

Figure 2: The consequence of qx1 AND qx2’

At last we calculated:
Sim(x1, x2) = sim (x1,x2) / STI (x1) = STI (qx1x2) / STI (x1)=0.25/0.75=0.33
Sim(x2,x1)= sim (x2,x1)/STI(x2) =STI(qx2x1)/ STI (x2) =0.25/0. 5=0.5
min (Sim (x1, x2), Sim (x2, x1)) =0.33<simMin
So, we should generate a new cluster c2 for s2.
Similarly, we calculated:
Sim(x1,x3)=sim(x1,x3)/STI(x1)=STI(qx1x3)/ STI(x1)=0.5/0.75=0.66;
Sim(x3,x1)=sim(x3,x1)/STI(x3)=STI(qx3x1)/ STI(x3)=0.5/0.5=1
min (Sim(x1,x3), Sim(x3,x1)) =min(0.66,1) =0.66> sim Min
So, we should add x3 to c1;
Now, there are two clusters: c1= {x1, x3} , c2= {x2}.

3.2 Bitmap-based time-interval chronological arrangement and filter through time interval position table

TIPT (the time interval position table) is a new data structure we proposed in this paper to arrange the time-interval sequence in each cluster as shown in figure 4.
The first line of TIPT is time intervals in TI, so there are r columns. The first column are setoff items at initial location of each sequence, we defined their time interval as 0. And the rest columns in TIPT record the subsequent setoff items in each sequence. For a time-interval sequence <(I1,I2,I3),2,(I2,I3,I4),3,(I2,I3,I4)> , we store it to TIPT as the second line shown in figure 3.

We summarize the procedure of filling TIPT as Algorithm 2 shown.

**Algorithm 2: Classifying Sequences with TIPT**

Input: a cluster of bbs (bit-based sequences)
Output: TIPT

1. Begin
2. int line Number;
3. int TI;
4. Bitset curSetOfItems= new Bitset();
5. for each(bbs in cluster){//dealing with each sequence in the cluster
6. for(i=0; i<cluster.size; i++) {//dealing with each itemset in the sequence
7. TI=cluster[i].TI; //the TI of sequence i in cluster as shown in figure 1
8. curSetOfitems=cluster[i].setOfItems;
9. TIPT[line Number][TI]=curSetOfItems;
10. // keep the setofitems in corresponding location in TIPT
11. }
12. lineNumber++;
13. }
14. End

In this algorithm, Bit set is a bit string to store one line of sequence bitmap. This means that a setofitems of sequence with its time interval as shown in figure 1. In fact, instead of converting the cluster outcome we finish the operations in the cluster. The upcoming section is an easy way that will show how to make position with TIPT. We accomplish sequence position on each cluster to produce the long biased sequence with TIPT. The number that appears after each item indicates the number of times it appeared during the operation. For example, at the location of first setofitems, item a appeared 2 times, so we denote it as a: 2 here. The time intervals between each pair of setofitems are joined together. Finally, by filtering the long biased sequence with user specified threshold k, we can generate the d sequential patterns as algorithm 3 shown.

**Algorithm 3 Bitmap-based chronological stance and categorize**

Input: TIPT
Output: d Time-interval Sequential Pattern

Begin
1. if(!TIPT.is Empty){
2. Sequence result=new Sequence();
3. Bitset temp Result SetofItems = new Bitset();
4. Counter SetofItems temp Counter SetofItems = new Counter SetofItems();
5. for each(column in TIPT){ //dealing with every column in TIPT
6. if(!column is Empty()){
7. for each(setofitems in column){//dealing with every setofitems in column

```plaintext
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I1,I2,I4)</td>
<td>(I2,I3,I4)</td>
<td>(I2,I3,I4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(I1,I2)</td>
<td></td>
<td>(I3)</td>
<td></td>
<td>(I2,I3,I4)</td>
</tr>
</tbody>
</table>
```

Figure 3: Position process with TIPT
8. for(int k=setofitems. nextSet
    Bit(0);k>=0;k=setofitems .nextSet
    Bit(k+1)){
9.    temp Counter Setofitems[k]++;//count the
times of item appearance
10.   if(temp Counter
    Setofitems[k]>threshold)//save the
    frequent one
11.   temp Result Setofitems[k].set();}
12. result .add New Setofitems(column. Get
    Number, temp Result Setofitems);}
13. return result;}
End

In this algorithm, Sequence is a part of
the bitmap of TSDB; Counter Setofitems is a
string of integer variables instead of bits, so we
can use it to count the appearance time of each
item. The function setofitems. Next SetBit (int a)
returns the index of the first bit that is set to true
that occurs on or after the specified starting index
and setofitems is also a Bitset here (for more
details, refer java API document).

In our programming, a big counter
sequence with int vector has to be used here. It’s
inefficient but necessary; we will make a
particular discussion later in the experiment. It’s
important that TIPT is also based bitmap. Every
setofitems in TIPT is a binary string and we
calculate each weight of item through counting
the corresponding bit.

3.3 The agenda of algorithm

The algorithm divides the data stream
into batches, then generate initial clusters of
sequences and each initial cluster was
compressed into biased time-interval sequence
through multiple sequence position. We filter
each biased time-interval sequence with the
specified threshold k and save the results at last.

Algorithm 4: Mining d time-intervals
chronological Pattern based Bitmap

Input: time-interval Sequence Stream, filter
threshold k, batchsize, simMin;
Output: Estimate time-interval sequence patterns;
Begin
1. while(NextBatch() is not null){
2.   B=NextBatch();
3.   //Get a new batch according to the
    batchsize
4.   For each s∈B//dealing with each
    sequence in B for clustering
5.   C=Clustering(B,simMin)
6.   //Cluster the batch according to the
    similarity as shown in algorithm 1
7.   For each c in C // C is a set of cluster
8.   c’= Align And Filter (c,k);
9.   //make multiple time-interval sequence
    position for each cluster and
10.  //filter each biased sequence with
    threshold as shown in algorithm 3
11. Approx-seq-pattern-set = Approx-seq-
    pattern-set +c’;
12. //save the estimate sequential pattern
13. End

In algorithm 4, k is the filter threshold, batch size
is the number of sequence for each batch, and
simMin is minimal similarity between two
sequences for clustering.
For example, we execute position on c1 (with 2
time-interval sequences in it);
Figure 4 is the general process of position and
figure 3 is position process with TIPT.
Figure 4: various time-interval progression positions

In the meantime, we filter the long biased sequence with user specified threshold k as shown in figure 5.

For example, set k=2, we’ll get sequence < 1, (I1, I2), 2, (I3)>, i.e. discard the items whose appearing times are lower than threshold, we call them feeble items.

4. Investigational Result

We encompass instigated the entire algorithm on PC (P4 2.19 GHz, 2GB memory, 500GB disk, Windows XP OS) with JAVA (JDK 1.6). The whole experimentations are accomplished at artificial sequence data engendered commencing IBM Synthetic Generator.

4.1 Space effectiveness

Support on bitmap, DSBMMS is a space efficient algorithm. Excluding in the subsequent two aspects we have to consider up among space efficiency and time efficiency. First, as declared over, we broaden the number of bit in a line to an integer that be able to be alienated by 8. For paradigm, we use 1024 bits to save a 1000 bits itemset essentially. Second, we have to use a big counter sequence as referenced exceeding to count biased sequence as a substitute of the bitmap based one. The big counter sequence is a counter list with 32bit breadth elements. But convenient are 1000 distinct items in the dataset; we need a 3.9MB (a common itemset is about 129B) counter sequence. Providentially, individual one such sequence is adequate. The groups in our algorithm allocate the same memory space, once the group size is known; the max memory detriment throughout the dispensation can be estimated. If not, TIPT is worn to systematize the time-interval sequence and facilitate multiple sequences position in our algorithm. The max time interval number in our implements is 10, so the corresponding column number of TIPT is 10. We list the estimate memory usage in table 3.

<table>
<thead>
<tr>
<th>BatchSize</th>
<th>Approximate Memory Usage (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>16.1</td>
</tr>
<tr>
<td>2000</td>
<td>28.3</td>
</tr>
<tr>
<td>3000</td>
<td>40.5</td>
</tr>
<tr>
<td>4000</td>
<td>52.7</td>
</tr>
<tr>
<td>5000</td>
<td>64.9</td>
</tr>
</tbody>
</table>

Table 5: Batch Size and Memory Usage

4.2 The runtime and numeral of progressions

Hire mean sequence length be 10, min sequence resemblance subsist 0.6, number of dissimilar items be 1000 and filter threshold k be 0.3; we experienced the algorithm on 10 diverse dimension datasets. The consequence is shown in
set to 0.7, for the reason that no prejudiced sequence fits the threshold in our dataset, it conduct the algorithm idling.

4.4 The characteristic of Mining Result

We deliberate the eminence of mining consequence of our algorithm at ultimate. By IBM synthetic data generator, we are able to produce sequence dataset by further clamour to several precise support sequences. So the finial dataset involving the base sequences. Stipulations our algorithm preserves ascertain this imaginative base sequences from first to last dataset, we are capable to articulate to facilitate the characteristic of our algorithm is assured.

For apiece base sequence, it have be enclosed when individual of our consequent d sequential patterns contribute to the precise correspondence threshold with it. We apply conceal capability to appraise the quality of our mining consequences, symbolized as R. The cover capacity enhances with the quantity of sequence as revealed in Figure 8. Since we acknowledged, the dataset is engendered from base sequences and our algorithm is accomplished to extract the inventive base sequences from enormous amount of noisy data. It can be appeared that more data will be further beneficial to ascertain the original base sequences (or original trend the dataset implicating).

5. Conclusion

In order to find out significant long time-interval sequences, we intend to recommend a new algorithm for mining d time-interval sequential pattern in data stream based bitmap and multiple sequential positions called DSBMMS in this paper. Initially, our algorithm segregates the data stream into batches and cluster sequences by similarity comparison. Then, each cluster was compressed into biased
sequence through multiple sequential positions. Using filtering approach we sort out each biased sequence with the specified threshold k and store them as result at last. All the operations in our algorithm are intended to facilitate fast bit counting on bitmap with a reduced amount of space. In the meantime, benefited from the multiple sequential positions mining technology, it avoids dealing with massive quantity of insignificant sequences. In short, it could achieve rapid mining skill inherent trend behind absolute data stream. Finally, the experimental consequences show that both the effectiveness and quality can be assured in this algorithm.

6. References
[9] Luiz F. Mendes, Bolin Ding, Jiawei Han,"Stream Sequential Pattern Mining with Precise Error Bounds", In Proceedings of Eighth IEEE International Conference on Data Mining, pp.941-946, 2008