GraSeq: A Novel Approximate Mining Approach of Sequential Patterns over Data Stream

Haifeng Li and Hong Chen

School of Information, Renmin University, Beijing, 100872, P.R. China
mydlhf@126.com,
chong@ruc.edu.cn

Abstract. Sequential patterns mining is an important data mining approach with broad applications. Traditional mining algorithms on database were not adapted to data stream. Recently, some approximate sequential pattern mining algorithms over data stream were presented which solved some problems except the one of wasting too many system resources in processing long sequences. According to observation and proof, a novel approximate sequential pattern mining algorithm is proposed named *GraSeq*. *GraSeq* uses directed weighted graph structure and stores the synopsis of sequences with only one scan of data stream; furthermore, a subsequences matching method is mentioned to reduce the cost of long sequences' processing and a conception *validnode* is introduced to improve the accuracy of mining results. Our experimental results demonstrate that this algorithm is effective and efficient.

Keywords: sequential pattern, data stream, directed weighted graph.

1 Introduction

A sequential pattern is a subsequence that appears frequently in the sequence database. The sequential patterns mining has shown its importance in many applications include business analysis, web mining, security, and bio-sequences analysis. For instance, a website wants to make users find their favorite contents conveniently, so they will get users' visit orders from web log. These visit orders should be seen as sequences and could be mined to sequential patterns so that the website's structure is improved according to these sequential patterns.

Before data stream appears, almost all of sequence pattern mining algorithms use the accurate matching method which waste a lot of system resources, and moreover, the results have to be got by multiple scans. There are two main kinds of accurate matching algorithms so far.

The first kinds are the appriori-like algorithms such as *GSP* [1] and *SPADE* [2], which need to create the candidate set and use multiple scans over database.

The second kinds are the projection-based algorithms such as *PrefixSpan*[3], *FreeSpan*[4] and *SPAM*[5], which avoid the process of creating candidate set and extend the local frequent itemsets to long sequential patterns according to the projection of sequential patterns.

R. Alhajj et al. (Eds.): ADMA 2007, LNAI 4632, pp. 401–411, 2007.

Data Stream is fast, unlimited, dynamic and continuous, so data can't be wholly stored in memory and can't be scanned for multiple times. Furthermore, the mining results always bring noises from data stream, so traditional mining methods are absolutely not adapted to data stream. The adaptation and approximation will be mainly considered in this environment. Two kinds of approximate sequential patterns mining methods have been developed: One is block data processing method, another is tuple processing method.

ApproxMap [6] uses the block data processing method, it clusters data stream into a series of blocks according to the similarity among sequences, and then compresses the similar sequences with multiple alignment method to reduce memory usage. The minimum support threshold is set to ignore the items which are not frequent so that noise is filtered. Finally the compression results are stored within a tree to make query convenient. *ApproxMap* can't get the real-time results because it can't compute until a group of data has arrived.

Hoong Hyuk Chang proposed an algorithm names *eISeq* [7] that uses the tuple data processing method. *eISeq* regards data stream as continuous transaction tuples and processes each tuple at once when it comes so that the real-time results are achieved. Five steps are in this algorithm: parameter update, count update, sequence insertion, sequential patterns selection and data pruning. *eISeq* computes the sequential patterns efficiently, but on the other hand, it wastes many time in scanning the tree to decide whether a new sequence can be inserted into the monitor tree, and also *eISeq* can't process long sequences because the longer a sequence is, the more system resources to create all its subsequences is used. For example, if $\langle a_1, \cdots, a_{20} \rangle$ is a sequence, there are $(2^{20} - 1)$ subsequences in total must be created. It is obviously difficult to compute and store all these subsequences.

In this paper, a novel sequential patterns mining algorithm names *GraSeq* is presented which uses the directed weighted graph structure so that memory usage is reduced a lot. *GraSeq* increasingly stores the whole information of the coming data from data stream. In this algorithm, a new subsequences matching method is proposed to create graph, and some relational data rules are introduced to filter most of the redundancy data. *GraSeq* has four steps to finish mining task:

1. Subsequences generating, Without creating all subsequences, only 1-subsequences set and 2-subsequences set of each sequence are created.
2. Sequence insertion and update. In this step, 1-subsequences and 2-subsequences of each sequence are inserted into graph as vertices and edges if they do not exist in graph, whereas the weight of vertices and edges are updated.
3. Sequential pattern mining. Users can traverse the directed weighted graph to acquire approximate sequential patterns with setting the minimum support threshold.
4. Data pruning. When system resources are not enough to support the running of *GraSeq*, some data will be erased according to given rules.

The main contributes of this paper are shown as follows: Firstly, a new directed weighted graph structure is presented to stored the synopsis of the sequences

of data stream; Secondly, in allusion to the characteristic of directed weighted graph, a new approximate sequential pattern mining method is proposed where a sequence is regarded as sequential pattern when all the 1-subsequences and 2-subsequences of which are frequent; Finally, to reduce the cost of system resources, a non-reclusive depth first search algorithm is introduced.

The rest of paper is organized as follows. Section 2 introduces the preliminaries of this algorithm and section 3 describes the data structure and implementation of *GraSeq* in detail. In section 4, a series of experiments are finished to show the performance of the proposed method. Finally, section 6 concludes this paper.

2 Preliminaries

Sequential pattern mining is the constraint of frequent patterns mining in data item's order which is presented in [8] firstly in 1995. There is a detailed description of sequential patterns definition:

An itemset is a non-empty set of items. Let $I = \{i_1, \cdots, i_l\}$ be a set of items. An itemset $X = \{i_{j_1}, \cdots, i_{j_k}\}$ is a subset of *I*. Conventionally, itemset $X = \{i_{j_1}, \cdots, i_{j_k}\}$ is also written as $\{x_{j_1}, \cdots, x_{j_k}\}$. A sequence $S = \langle X_1, \cdots, X_n\rangle$ is an ordered list of itemsets. A sequence database SDB is a multi-set of sequences.

A sequence $S_1 = \langle X_1, \cdots, X_n\rangle$ is a subsequence of sequence $S_2 = \langle Y_1, \cdots, Y_m\rangle$, and S_2 is a super-sequence of S_1, if $n \leq m$ and there exist integers $1 \leq i_1 < \cdots < i_n \leq m$ such that $X_j \subseteq Y_{i_j} (1 \leq j \leq n)$.

Given a sequence database SDB, the support of a sequence *P*, denoted as *sup(P)*, is the number of sequences in SDB that are super-sequences of *P*. Conventionally, a sequence *P* is called a sequential pattern if $sup(P) \geq S_{min}$, where S_{min} is a user-specified minimum support threshold.

In this paper, directed weighted graph is the data structure whose vertices denote itemsets and edges denotes order between itemsets. Some definitions are given as follows:

Definition 1. *For a sequence* $S = \langle s_1, \cdots, s_n\rangle (n \geq 1)$*, a 1-subsequence of* S *is* $oss_i = \langle s_i\rangle (1 \leq i \leq n)$*, which denotes a vertex in graph. A 1-subsequence set of* S *is* $OA = \{oss_i | i = 1 \cdots n\}$.

Definition 2. *For a sequence* $S = \langle s_1, \cdots, s_n\rangle (n \geq 2)$*, a 2-subsequence of* S *is* $tss_i = \langle s_i, s_j\rangle (1 \leq i < j \leq n)$*, which denotes an edge in graph. A 2-subsequence set of* S *is* $TA = \{tss_i | i = 1 \cdots n-1\}$.

Definition 3. *For a 2-subsequence* $tss_i = \langle s_i, s_j\rangle (i = 1 \cdots n-1, j = i+1 \cdots n)$*, s_i is parent of s_j and s_j is child of s_i.*

In data stream, a sequence support achieves its maximum value when all its subsequences happen in as many transactions as possible, so the support of a sequence must be not higher than the ones of all its subsequences.

Proposition 1. *For a sequence* $S = \langle s_1, \cdots, s_n\rangle (n \geq 2)$, C(S) *is support threshold of* S *and* $C_{max}(S)$ *is maximum support threshold of* S*, the approximate estimate formula of* $C_{max}(S)$ *is shown as follows:*

$$C_{max}(S) = min(\{C(a)|a \subseteq S \wedge |a| = n-1\}) \ . \tag{1}$$

Theorem 1. *For a sequence* $S = \langle s_1, \cdots, s_n \rangle (n \geq 1)$*, the approximate estimate formula of* $C_{max}(S)$ *is shown as follows:*

$$C_{max}(S) = \begin{cases} C(S) & |S| = 1 \\ min(\{C(a)|a \subseteq S \wedge |a| = 1\}) & |S| = 2 \\ min(\{C(a)|a \subseteq S \wedge |a| = 2\}) & |S| > 2 \end{cases} \tag{2}$$

Proof. For a sequence $S = \langle s_1, \cdots, s_n \rangle (n \geq 1)$, the results are obvious as $n \leq 2$. When $n > 2$, if $C_{max}(S) = min(\{C(a)|a \subseteq S \wedge |a| = n-1\})$, then $C_{max}(S_n) = min(C_{max}(S_{n-1}))$, and also $C_{max}(S_{n-1}) = min(C_{max}(S_{n-2})), \cdots, C_{max}(S_3) = min(C_{max}(S_2))$, so $C_{max}(S_n) = min(min(\cdots C_{max}(S_2) \cdots)) = min(C_{max}(S_2))$. Then,when $n > 2$,$C_{max}(S) = min(\{C(a)|a \subseteq S \wedge |a| = 2\})$,proof done.

3 *GraSeq* Method

From Theorem 1 we can find that if the support of all the 1-subsequences and 2-subsequences of one sequence S are acquired, the approximate support of this sequence S is acquired too. In other words, if the support of all the 1-subsequences and 2-subsequences of a sequence is higher than the minimum support threshold, the support of this sequence is higher than the minimum support threshold,and it means this sequence is frequent. So the main task in this paper is to store all the information of 1-subsequences and 2-subsequences of all sequences, and this guarantees the mining is almost valid. Users can traverse the graph to find all frequent sequences as sequential patterns on condition that their 1-subsequences and 2-subsequences are frequent.

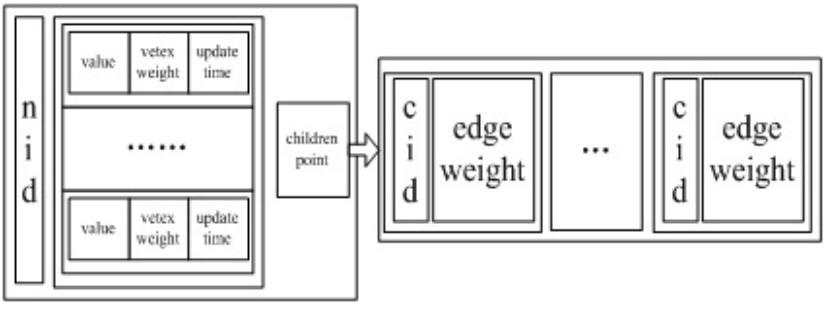

Fig. 1. Data structure of graph

3.1 Data Structure and Meaning

GraSeq uses directed weighted graph structure to share data information. Each vertex in graph denotes one itemset of sequence, and each edge denotes the order of two different itemsets. Each vertex is a 3-tuple $\langle nid, itm, ch \rangle$, in which *nid* is

the identification of vertex which is fixed to make it possible for quick visit; *itm* denotes the main information of vertex which is a collection of 3-tuple $\langle va, wt, ut\rangle$, where *va* is the real value of the vertex, *wt* is the weight of the vertex and *ut* is the latest update time stamp of the vertex; *ch* is a pointer to the children of vertex named *childrenlist* . The element in *childrenlist* is a 2-tuple $\langle cid, lwt\rangle$, *cid* is the identification of each child vertex and *lwt* denotes the edge weight between current vertex and its child vertex.Figure 1 shows the data structure of graph.

Example 1. After a series of sequences $\{\langle a, c\rangle$, $\langle a, d\rangle$, $\langle a, b, c\rangle$, $\langle a, b\rangle$, $\langle b, c\rangle$, $\langle b, d\rangle$, $\langle a, b, d\rangle\}$ have arrived, the data storage in memory is shown in figure 2.

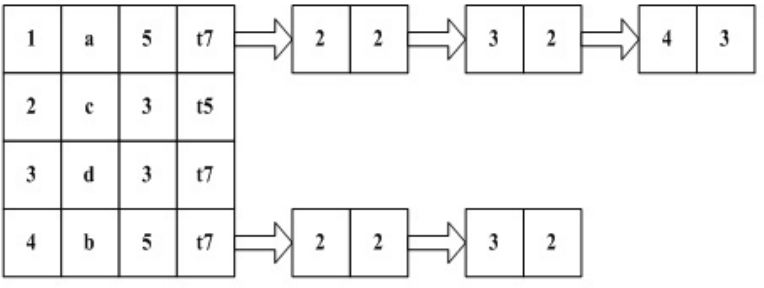

Fig. 2. Example of data storage in memory

3.2 Algorithm Description

GraSeq includes 4 steps to deal with each sequence: subsequences generating, sequence insertion and update, sequential patterns mining and data pruning. First two steps are graph establishment phase, step three is data mining phase, and they are parallel in running.

3.2.1 Subsequences Generating

In this section, the 1-subsequences set and 2-subsequences set are created from a sequence. For example, a sequence $\langle a, c, d, e\rangle$ has 1-subsequences set $\{\langle a\rangle, \langle c\rangle, \langle d\rangle, \langle e\rangle\}$ and 2-subsequences set $\{\langle a, c\rangle, \langle a, d\rangle, \langle a, e\rangle, \langle c, d\rangle, \langle c, e\rangle, \langle d, e\rangle\}$. From definition 1 and definition 2, it is clear to see that a n-sequence has only n 1-subsequences and $\sum_{i=1}^{n} i$ 2-subsequences. The count is much smaller than that of all subsequences $2^n - 1$ when sequence is longer.

3.2.2 Sequence Insertion and Update

The old sequence information has weakly affection with the coming of the new sequence. To differentiate the information of recently generated data elements from the obsolete information of old data elements which may be no longer useful or possibly invalid at present to make result reflect recent rule of data stream, a decay rate d [9] is used as follows:

$$d = b^{-(1/h)} \qquad (b \geq 1, h \geq 1)\ . \tag{3}$$

In this formula, b is decay-base and h is decay-base-life.

If decay-base is set to 1, frequent sequences are found as a mining result set as in the other mining algorithms of recent sequences, on the other hand, if a decay-base is set to be greater than 1, recently sequences are effective in mining. To avoid fluctuation in the set of recently frequent sequences, a decay-base-life h of a decay rate should be set to be greater than or equal to its lower bound h^{LB} , found as follows [9]:

$$h^{LB} = \lceil -\{\gamma / \log_b(1 - S_{min})\} \rceil \ . \tag{4}$$

In this formula, γ is safety factor and S_{min} is minimum support threshold.

Sequence insertion and update are to combine all 1-subsequences and 2-subsequences of one sequence with graph in the form of adding weight of vertices and edges. Two steps are in this section: The first step is to add weight of vertices, if a vertex corresponding one of 1-subsequences is not in graph, create and insert this vertex into graph, the real value is 1-subsequence's value and the initial weight *wt=1*; otherwise update the weight *wt=wt+1/d*. The second step is to add weight of edge as the same method as in step one.The optimized algorithm of sequence insertion and update is shown as follows.

```
Function createGraph(seq){
  create1subsequences(seq);
  create2subsequences(seq);
  for each item in 1-subsequences{
    findVertex(item);
    if(findVertex)
      getId(updateVertex(item));
    else
      getId(addNewVertex(item));
  }
  for each <preItem, nextItem> in 2-subsequences{
    findEdge(preItem,nextItem);
    if(findEdge)
      updateEdge(preItem,nextItem);
    else
      addNewEdge(preItem,nextItem);
  }
}
```

3.2.3 Sequential Pattern Mining

This step is independent from previous two steps. When the graph is constructing, users can provide the minimum support threshold S_{min} to get the sequential patterns anytime. Sequential patterns are obtained by depth first search over the graph. The mining process is to recursively traverse the graph with every vertex as beginner, and finally the real-time sequential patterns are acquired in which all the vertices are *validnode*.

Considering the efficiency of algorithm, a non-recursive traverse method is used. A stack named *nodestack* is introduced to store each *validnode*.

If there are no other rules in traverse, the results should be a superset of accurate sequential patterns. To eliminate most of the redundant sequential patterns, a concept *validnode* is introduced. For a vertex on the top of stack, if once there is no *validnode* in its children, output all vertices from the bottom to the top of stack as a sequential pattern.

Definition 4. *A vertex is* validnode *when this vertex satisfies the follow rules after it is pushed into stack.*

1. Every vertex in stack is greater than minimum support threshold S_{min}.

2. There are no repeated vertices in stack.

3. If the vertices from the bottom to the top of stack are regarded as a sequence, then every 2-subsequence of this sequence is frequent, i.e., the edge weight between each two vertices in stack is greater than the minimum support threshold S_{min}.

In traverse the current vertex may be searched through its parent vertex or ancestor vertex, the relationship between them will be recorded so that when all the parent and ancestor of the current vertex is popped from stack, the current vertex is surely visited. so a 32 bits binary integer *sign* of *validnode* in *nodestack* is imported to avoid data lose.

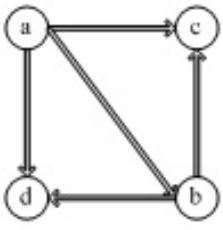

Fig. 3. Example of graph

Example 2. The sequences in example 1 is used here. The minimum support S_{min} is set to 2, then the graph where the weights of each vertex and edge greater than or equal to 2 is shown in Figure 3. They are denoted in data structure as $(a \rightarrow c \rightarrow d \rightarrow b, b \rightarrow c \rightarrow d, c, d)$, and Table 1 shows the process of sign operation with a as beginner.

As shown in Table 1, c is firstly signed by a because a is parent of c, and a is the *first* vertex in stack, so $Sign(c) = 1$, here c has no children, so output $\langle\{\langle a\rangle, \langle c\rangle\}$ in row 3 as sequential pattern and pop c in row 4. Next b is pushed in row 5 because it is the second child of a and also *Sign(b)=1*, Then c is pushed and signed as the first child of b in row 6, now b is the second vertex in stack, so *Sign(c)=11*. c is not unsigned as *Sign(c)=1* until b which is one of c's parent is popped from stack in row 9. The same as a's being popped.

The sequential patterns mining algorithm is shown as follows.From this algorithm we can find each sequence built from the bottom to the top of stack is a sequential pattern whose 1-subsequences and 2-subsequences are all frequent.

Table 1. The process of sign operation with vertex a as beginner

	Stack	Sign(a)	Sign(b)	Sign(c)	Sign(d)	Output
1		0	0	0	0	
2	a	0	0	0	0	
3	a,c	0	0	1	0	$\langle a \rangle, \langle c \rangle$
4	a	0	0	1	0	
5	a,b	0	1	1	0	
6	a,b,c	0	1	11	0	$\langle a \rangle, \langle b \rangle, \langle c \rangle$
7	a,b,d	0	1	11	10	$\langle a \rangle, \langle b \rangle, \langle d \rangle$
8	a,b	0	1	11	10	
9	a	0	1	1	0	
10	a,d	0	1	1	1	$\langle a \rangle, \langle d \rangle$
11	a	0	1	1	1	
12		0	0	0	0	

```
Function querySequence(Graph g){
 for(each vertex in g){
  if(vertex is valid){
    push(vertex);
    while(stack is not empty){
      nextvertex = findnextValidvertex(vertex);
      if(find){
        push(nextvertex);
        sign(nextvertex);
      }else{
        out(stack);
        unsign(all nextvertex of vertex);
        pop(topvertexofstack);
      }
    }
  }
}
```

3.2.4 Data Pruning

Generally, the smaller the weight of vertex is, the greater possibility a vertex will be erased, but in fact, some vertices with smaller weight may become frequent in future, so data can not be erased according to weight.

And on the other hand, the effect of old data will decay follows the new data's coming. If a vertex is not updated for a long time, we supposed that it would not be updated in future. Data pruning is to find the vertices that were not updated for a long time and erases them when system resources can't satisfy the running of algorithm.

4 Experimental Results

A set of experiments are designed to test the performance of this algorithm. The experiment uses synthetic supermarket business data created by the data

generator (*http://www.almaden.ibm.com/cs/quest*) of IBM. *PrefixSpan* is a multiple scan algorithm and creates accurate sequential patterns, so a precision and recall compare is based on the result of *PrefixSpan*. *eISeq* is a one scan algorithm and creates approximate sequential patterns, and it has the best performance in processing time and memory usage to the best of our knowledge, so it is chose as the comparative algorithm. The machine in experiment has PIV3.2G CPU and 512M memory, C++ is the implementary language under Windows environment.

Four data sets are created named *T8I10KD100K*, *T8I80KD100K*, *T15I10KD100K* and *T15I80KD100K*,where T is the average sequence length, I is the count of distinct sequence itemsets and D is the number of sequences.

Figure 4 shows the memory usage of two algorithms *GraSeq* and *eISeq* in sequences insertion and update phase over data set *T8I10KD100K* with $S_{min} = 100$, and figure 5 shows the memory usage over *T8I80KD100K* with $S_{min} = 10$. For *eISeq*, another parameter significant support threshold [7] $S_{sig} = 1$. It can be seen that on condition that significant support threshold is small, the memory usage of *eISeq* obviously increases a lot. To compare two figures, it is clear to see that *GraSeq* uses less memory when the count of distinct itemsets is smaller, that is because sequence information is stored in vertex of graph, and the same itemsets are stored in the same vertex so no redundant information are stored.

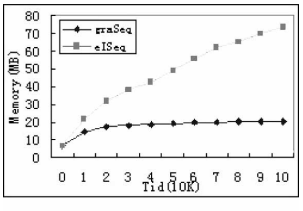

Fig. 4. Memory usage of *GraSeq* and *eISeq* over data set *T8I10KD100K*

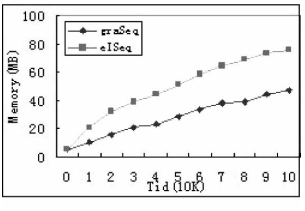

Fig. 5. Memory usage of *GraSeq* and *eISeq* over data set *T8I80KD100K*

Figure 6 shows the data processing time of *GraSeq* and *eISeq* in sequences insertion and update phase over data set *T15I10KD100K* with $S_{min} = 100$, and figure 7 shows data processing time of two algorithms over *T15I80KD100K* with $S_{min} = 10$. We can see that *eISeq* needs more time because the analysis of sequence spends much time when average length of sequence increases.

To compare the correctness of results of *GraSeq* and *eISeq*, two concepts precision and recall are introduced. For two result sets R_1 and R_2, define precision and recall as follows:

$$Precision(R_1|R_2) = \frac{|R_1 \bigcap R_2|}{|R_2|}. \tag{5}$$

$$Recall(R_1|R_2) = \frac{|R_1 \bigcap R_2|}{|R_1|}. \tag{6}$$

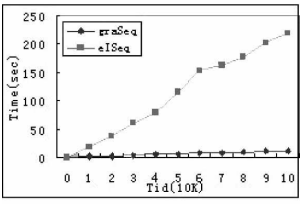

Fig. 6. Data processing time of *GraSeq* and *eISeq* over data set *T15I10KD100K*

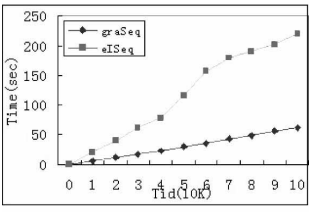

Fig. 7. Data processing time of *GraSeq* and *eISeq* over data set *T15I80KD100K*

Figure 8 shows both $Precision(R_{GraSeq}|R_{PrefixSpan})$ and $Precision(R_{eISeq}|R_{PrefixSpan})$, and figure 9 shows both $Recall(R_{GraSeq}|R_{PrefixSpan})$ and $Recall(R_{eISeq}|R_{PrefixSpan})$ with different S_{min}, both figures use the same data set *T15I10KD100K*. In *eISeq*, the significant support threshold is also set as $S_{sig} = 1$. We can find both precisions and recalls of *GraSeq* and *eISeq* are almost same and become uniform follows the increasing minimum support threshold.

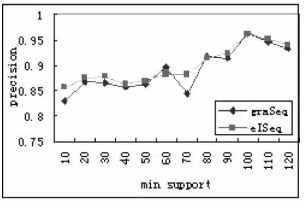

Fig. 8. Precision compare of *GraSeq* and *eISeq* over data set *T15I10KD100K*

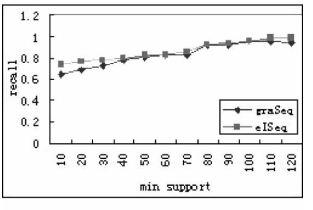

Fig. 9. Recall compare of *GraSeq* and *eISeq* over data set *T15I10KD100K*

All of the above experiments show that when the average length of sequence is greater or the count of distinct itemsets are smaller, *GraSeq* can save more system resource with less lose of precision and recall than *eISeq*.

5 Conclusions

This paper investigates the problem of sequential patterns mining over data stream and proposes a novel algorithm named *GraSeq* based on directed weighted graph structure. *GraSeq* is a one scan algorithm, and it stores the synopsis of sequences. With subsequence matching method, final approximate sequential patterns in different minimum support threshold are obtained dynamically with deep traverse over graph. The experiments show that *GraSeq* has a better performance in processing time and in memory usage than *eISeq* with the accordant accuracy of mining results. In this algorithm, data sharing brings less resources usage but a little more imprecise results, more adaptive rules may further improve the data correctness. Moreover, graph compression is also an interesting topic for future research.

References

1. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) EDBT 1996. LNCS, vol. 1057, pp. 3–17. Springer, Heidelberg (1996)
2. Zaki, M.: SPADE: An Efficient Algorithm for Mining Frequent Sequences. Machine Learning 40, 31–60 (2001)
3. Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.-C.: PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In: ICDE'01. Proceeding of the International Conference on Data Engineering, pp. 215–224 (2001)
4. Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., Hsu, M.-C.: FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. In: KDD '00. Proceeding of ACM SIGKDD International Conference Knowledge Discovery in Databases, August 2000, pp. 355–359 (2000)
5. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: KDD'02. Proceeding of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 2002, pp. 429–235 (2002)
6. Kum, H.C., Pei, J., Wang, W., Duncan, D.: Approx-MAP: Approximate Mining of Consensus Sequential Patterns. Technical Report TR02-031, UNC-CH (2002)
7. Chang, J.H., Lee, W.S.: Efficient Mining method for Retrieving Sequential Patterns over Online Data Streams. Journal of Information Science, 31–36 (2005)
8. Agrawal, R., Srikant, R.: Mining Sequential Patterns. In: ICDE'95. Proceedings of the 11th International Conference on Data Engineering, March 1995, pp. 3–14 (1995)
9. Chang, J.H., Lee, W.S.: Finding recent frequent itemsets adaptively over online data streams. In: Getoor, L., et al. (eds.) Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 2003, pp. 487–492 (2003)