Efficient Method for Mining Patterns from Highly Similar and Dense Database based on Prefix-Frequent-Items

Meng Han[1,2,] Zhihai Wang[1], Jidong Yuan[1]
[1]School of Computer and Information Technology
Beijing Jiaotong University, Beijing, 100044, P.R. China
[2]School of Computer Science and Engineering
Beifang University of Nationalities, Yinchuan, 750021, P.R. China
Email: compute2006_2@126.com

Abstract—**In recent years, there are a great deal of efforts on sequential pattern mining, but some challenges have not been resolved, such as large search spaces and the ineffectiveness in handling highly similar, dense and long sequences. This paper mainly focuses on how to design some effective search space pruning methods to accelerate the mining process. We present a novel structure, Prefix-Frequent-Items Graph (PFI-Graph), which presents the prefix frequent items of other items in sequential patterns. An efficient algorithm PFI-PrefixSpan (Prefix-Frequent-Items PrefixSpan) based on PFI-Graph is proposed in this paper. It avoids redundant data scanning, and thus can effectively speed up the discovery process of new patterns. Extensive experimental results on some synthetic and real sequence datasets show that the proposed novel structure is substantially more efficient than PrefixSpan with physical-projection and pseudo-projection, especially for dense and highly similar sequence databases.**

Index Terms—**sequential pattern mining; dense database; highly similar sequence; long sequence; prefix frequent items**

I. Introduction

Sequential pattern mining discovers frequent subsequences as patterns in a sequence database, and the subsequences whose occurrence frequency in the set of sequences is no less than minimum support threshold (called min_sup). It is an important data mining problem with broad applications [1-4], including the analysis of customer purchase patterns or Web access patterns[5,6], the analysis of sequencing or time related processes such as scientific experiments, natural disasters, and disease treatments [7,8], the analysis of DNA sequences [9-12] and so on.

Many previous studies have contributed to the efficient mining of long sequence. Algorithms SPAM [13] and LAPIN [14] with sequence-extended sequence and itemset-extended sequence, FP-growth [15] with FP-tree and PrefixSpan [1] with projection-based are efficient for mining long sequence. Some studies contributed to mining highly similar sequence, such as SeqBDD [16] with binary decision diagram.

In this paper, we present an efficient method of dense and highly similar sequential pattern mining called PFI-PrefixSpan (Prefix Frequent Items based PrefixSpan). It is based on the Prefix-Frequent-Items Graph (PFI-Graph) which is used to assist in early pruning and avoid duplicated projections. FPI-Graph is a directed acyclic graph and presents the prefix frequent items of other items in sequential patterns. This novel algorithm can reduce the scale of projected databases and the time of building projected databases through adding the pruning steps and reducing the scanning of certain specific sequential patterns production.

The rest of this paper is organized as follows: Section 2 reviews PrefixSpan algorithm. Section 3 discusses the novel structure: PFI-Graph and the algorithm PFI-PrefixSpan. Section 4 shows the experimental results of sequential pattern mining. Finally, the conclusion is provided in Section 5.

II. PrefixSpan Algorithm

The key advantage of PrefixSpan, an algorithm that examines the prefix subsequences and projects only their corresponding suffix subsequences into projected databases, is that it does not generate any candidates and only counts the frequency of local items. It utilizes a divide-and-conquer framework by creating subsets of sequential patterns that can be further divided when necessary [19].

TABLE I.
A SEQUENCE DATABASE

Sequence id	Sequence
10	(1)(1 2 3)(1 3)(4)(3)
20	(1 4)(3)(2 3)(1 5)
30	(5)(1 2)(4)(3)(2)
40	(5)(1)(3)(2)(3)

Project supported by Fundamental Research Funds for the Central Universities (No. 2013YJS032), Scientific Research Funds for the Ningxia Universities (No. NGY2013094), and Beijing Natural Science Foundation (No. 4142042).

TABLE II.
PROJECTED DATABASE AND SEQUENTIAL PATTERNS

prefix	projected database <sequenceid: sequence>	pseudoprojected database <sequenceid: index_position>	pseudoprojected database <sequenceid, index_elements>	sequential patterns
1	10: (1 2 3)(1 3)(4)(3), 10: (_2 3)(1 3)(4)(3), 10: (_3)(4)(3), 20: (_4)(3)(2 3) 20: (1 5), 20: (_5), 30: (_2)(4)(3)(2), 40: (3)(2)(3)	10: 1, 2, 5 20: 1, 6 30: 2 40: 2	10: 0, 1, 2 20: 0, 3 30: 1 40: 1	(1), (1 2), (1 2)(3), (1 2)(4), (1)(2), (1)(2 3), (1)(3), (1)(1), (1)(4), (1 2)(4)(3), (1)(2 3)(1), (1)(2)(3), (1)(2)(1), (1)(4)(3), (1)(3)(2),(1)(3)(3), (1)(3)(1)
2	10: (_3)(1 3)(4)(3), 20: (_3)(1 5), 30: (4)(3)(2), 40: (3)	10: 3 20: 4 30: 3 40: 4	10: 1 20: 2 30: 1, 4 40: 3	(2), (2 3), (2 3)(1), (2)(3), (2)(1), (2)(4), (2)(4)(3)
3	10: (1 3)(4)(3), 10: (4)(3), 20: (2 3)(1 5), 20: (1 5), 30 :(2), 40: (2)(3)	10: 4, 6 20: 3, 5 30: 5 40: 3	10: 1, 2 20: 1, 2 30: 3 40: 2	(3), (3)(2), (3)(3), (3)(1)
4	10: (3), 20: (3)(2 3)(1 5), 30: (3)(2)	10: 7 20: 2 30: 4	10: 3 20: 0 30: 2	(4), (4)(2), (4)(3), (4)(3)(2)
5	30: (1 2)(4)(3)(2), 40: (1)(3)(2)(3)	30: 1 40: 1	30: 0 40: 0	(5), (5)(2), (5)(3), (5)(1), (5)(2)(3), (5)(3)(2), (5)(1)(2), (5)(1)(3), (5)(1)(3)(2)

The major consuming of PrefixSpan is database projection, and the technique to reduce the size of projected databases is pseudo projection [1]. The idea is outlined as follows: instead of performing physical projection, one can register the index of the corresponding sequence and the starting position of the projected suffix in the sequence. Pseudo projection reduces the consuming of projection substantially when the projected database can fit in main memory.

Instead of registering the index of the starting position of the projected suffix in the sequence, we register the index of the transactions (elements or events) in the sequence. Our experiment has shown that it is faster than the former method in finding the position. The two ways to register the index are shown in Table 2, column 3 and 4.

For example, suppose the sequence database *S* is given in Table 1 and *min_sup*=50% (0.5). The projection databases and the sequential patterns are shown in Table 2. There are 53 patterns, including 4 length-1 patterns, 25 length-2 patterns, 18 length-3 patterns and 2 length-4 patterns. The first column is physical projected database, whose two elements are sequence_id and suffix sequence. For example, in the first row 10: (1 2 3)(1 3)(4)(3), the sequence_id is 10, and the suffix projected sequence of prefix 1 (the first item 1 in sequence_id 10 in initial database) is (1 2 3)(1 3)(4)(3). The second row 10: (_2 3)(1 3)(4)(3) is the suffix projected sequence of prefix 1, which is the second item 1 in sequence_id 10 in initial database and the first item in second transaction. The second column is pseudo projected database, whose two elements are sequence_id and index_position. The index_position is the index of the starting position of the projected suffix in the sequence. For example, there are 3 physical suffix projected sequences of prefix 1 and sequence_id 10, and the start position in sequence_id are 1(position of the first item 1 is 0), 2(position of the second item 1 is 1) and 5(position of the third item 1 is 4). The value 10: 0, 1, 2 in column 3 means that the index of the transaction of the projected suffix sequence 10 are 0 (including the first item 1), 1(including the second item 1) and 2(including the third item 1). The fourth column is the sequential patterns of different prefixes.

III. NOVEL ALGORITHM

Although the efficiency of PrefixSpan algorithm is high, it still can be further improved in some respects. PrefixSpan algorithm constructs a projected database for each frequent pattern, and therefore there are a large number of projections when the frequent patterns are huge. We find that PrefixSpan algorithm may generate duplicated projections in the process of mining [18]. In order to reduce the size of projected databases and reduce memory consuming, Pei presented the pseudo projection [1], but the counts of projected is still huge. Therefore, we find some measures to reduce the reconstruction of the projection database, aiming to reduce the runtime and memory usage.

After finding a local frequent item, the PrefixSpan algorithm constructs a sequential pattern and a projected database. But when the size of projected database is lower than minimum support, it is useless to construct the projected database. Therefore, before the projected database is created, we should test its size at first.

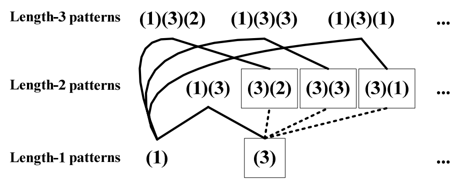

Figure 1. Some candidates and sequential patterns with prefix (1) and (3)

We also find some replicated projected database, as shown in Table 2 and Figure 1. For example, the sequential patterns which begin with prefix (1) and the second item is (3) are: (1)(3), (1)(3)(2), (1)(3)(3), (1)(3)(1). Whereas the sequential patterns begin with prefix (3) are: (3), (3)(2), (3)(3), (3)(1). Therefore, the proceedings to find patterns (1)**(3)**, (1)**(3)(2)**, (1)**(3)(3)**, (1)**(3)(1)** are duplicated work to find patterns **(3)**, **(3)(2)**, **(3)(3)**, **(3)(1)**. We call the item (1) is the Prefix Frequent Items (PFI) of item (3), denoted as *PFI*(3)={1}. We also find that the sequential patterns beginning with prefix (5) and the second item (3) are duplicated of the patterns beginning with (5) and the second item (1). For example, (5)**(3)**, (5)**(3)(2)**, and (5)(1)**(3)**, (5)(1)**(3)(2)**. Because finding all the local Prefix Frequent Items of all items consume much time, we just consider the PFI of length-1 patterns.

Definition 1 (PFI). Given a length-1 pattern α, β_i is a frequent item that appears in the prefix of α in some sequences. The *counts*(β_i) is the number of sequences in which β_i appears before α. If *counts*(β_i)=*support*(α), then β_i is one element of Prefix Frequent Items of α, denoted as *PFI*(α)={β_1,…,β_i,…, β_k}.

Definition 2 (CountsPFI). The number of elements in *PFI*(*α*) is called *CountsPFI*(*α*). The sum of all *CountsPFI*(α_i) is denoted as *CountsPFI.*

The major cost of PrefixSpan is the construction of projected databases. We give two ways to improve: (1) before the projected databases are constructed, adding the pruning step. Do not scan projected database when the projection sequence number is less than *min_sup*; (2) do not generate and scan the projected databases to some specific sequential patterns. For example, given a pattern *α*, if *PFI*(*α*)={$\beta_1,\ldots,\beta_i,\ldots,\ \beta_k$} is not null, then do not generate and scan the projected datasets when the prefixes belong to prefix set {(β_1)(*α*), ..., (β_k)(*α*)}. Generate the sequential patterns with regards to prefix (*α*), and at the same time generate the patterns with regards to prefix set {(β_1)(*α*), ..., (β_k)(*α*)}. For example, we know that item (1) belong to *PFI*(3), then we do not build projected database with regards to prefix (1)(3). We generate the patterns with regards to prefix (1)(3) until patterns with regards to prefix (3) generated, by adding item (1) to the beginning of patterns (3)(…) to generate **(1)**(3)(…). Therefore, our works aim to find out the *PFI*(α_k) of all 1-length pattern α_k.

We can see that, large *CountsPFI* means more duplicated works in generating patterns. Therefore, we can reduce runtime and memory usage by avoiding these works.

Definition 3 (FromPattern). If *β* belongs to Prefix Frequent Items of *α,* then *β* is a *FromPattern* of *α*, that is *β*∈*FromPattern*(*α*). If *γ* belongs to Prefix Frequent Items of *β*, then (*γ*)(*β*) is a *FromPattern* of *α*, that is (*γ*)(*β*)∈*FromPattern*(*α*). And so on, until no new Prefix Frequent Items are found. The *FromPattern* of *α* is denoted as *FromPattern*(*α*)={(*β*), (*γ*)(*β*), (*ξ*)(*γ*)(*β*), …}.

Definition 4 (ToPattern). If *α* belongs to Prefix Frequent Items of *β,* then *β* is the *ToPattern* of *α*, so *β*∈*ToPattern*(*α*), denoted as *ToPattern*(*α*)={(*β*), …}.

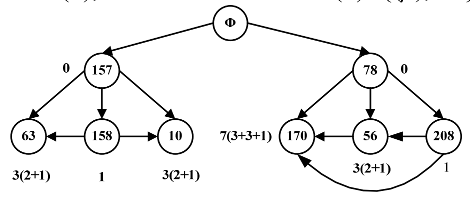

Figure 2. The PFI-Graph of data set new_orleans

PFI-Graph is a directed acyclic graph, and it records *CountsPFI*(*α*) as weight of each node *α*. Node in the graph is an item in sequential pattern which has *FromPattern* or *ToPattern* , as shown in Figure 2. The PFI-Graph of dataset new_orleans is shown in Figure 2, and the root of the graph is *Φ*. The weight of a node (denoted as *α*) is the *CountsPFI*(*α*), while its children nodes are items in *ToPattern*(*α*), and its parent nodes are items in *FromPattern*(*α*).

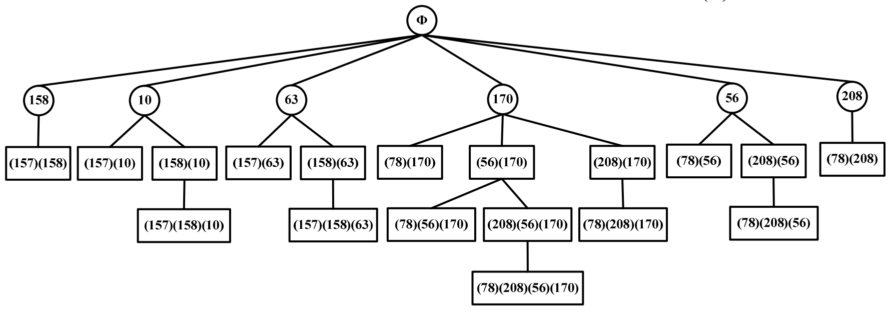

Figure 3. The FromPattern tree on data new_orleans

Take node (63) in Figure 2 as an example. The *PFI* of item (63) are (157) and (158), that is *PFI*(63)= {(157), (158)}. The weight or *CountsPFI*(63) of node (63) is 3. Here,3 is understood as 2+1, while 2 refers to the item number in *PFI*(63) and 1 refers to the item number in *PFI*(158). Meanwhile, (158) is the parent of (63). There are 3 links pointing to node (63) in the graph, and they are 157-->63, 158-->63 and 157-->158-->63. So the *FromPattern*(63)={(157), (158), (157)(158)} and *ToPattern*(63)=*null*. Node (56) has FromPattern and ToPattern, the *FromPattern*(56)={(78), (208)} and *ToPattern*(56)={(170)}. The (170) is the most complex node in Figure 2, and *PFI*(170)={(78), (56), (208)}. There are 7 links pointing to node (170) in the graph: 7 is equal to 3+3+1, the first value 3 refers to the item number in *PFI*(170), the second value 3 refers to the item number in *PFI*(56) and value 1 refers to the item number in *PFI*(208). Seven links pointing to (170) are: 157-->170, 56-->170, 208-->170, 157-->56 -->170, 157-->208-->170, 208-->56-->=70, and 157-->208 -->56-->170. It is clear that the *FromPattern*(170)= {(157), (56), (208), (157)(56), (157)(208), (208)(56), (157)(208)(56)} and *ToPattern* (170)=*null*. Therefore, when we scan the projected databases of prefix (170) and generate some sequential patterns (170)(α_i), we can generate some more patterns, such as:

(157)(170)(α_i), (56)(170)(α_i), (208)(170)(α_i),
(157)(56)(170)(α_i),
(157)(208)(170)(α_i),
(208)(56)(170)(α_i),
(157)(208)(56)(170)(α_i).

Referring to the Figure 2, we can build a FromPattern tree on data new_orleans, as shown in Figure 3. The root node is *Φ*. Recursively, if (*α*) is a node in the tree, then its children are all nodes (*α'*) such that (*α'*)=(*β*)(*α*) and *β*∈*FromPattern*(*α*).

Based on the concepts of PFI, FromPattern and ToPattern, algorithm Prefix Frequent Items Prefix- Span (PFI-PrefixSpan) can be described as follows. There are two works to do in the novel algorithm, the first one is to build the PFI-Graph, in order to find all the

FromPattern(α) and ToPattern(α) ($\alpha\in$PFI-Graph). The second one is to generate sequential patterns.

```
Algorithm PFI-PrefixSpan (PrefixSpan based on prefix frequent items)
Input: A sequence database $S$, and the minimum support threshold $min\_sup$.
Output: The complete set of sequential patterns
Method 1: Call PFI-PrefixSpan ($S$).
    The parameters $S$ is the sequence database $S$.
Steps:
    1. Scan $S$ once, find all length-1 patterns $\alpha_1, \ldots, \alpha_N$.
    2. Scan $S$ again, find all $PFI(\alpha_i)$ ($i$=1,…,$N$) and build PFI-Graph $G$.
    3. For each $\alpha_i$
       (a) Refer to $G$, find out $ToPattern(\alpha_i)$ ($i$=1,…,$N$) and $FromPattern(\alpha_i)$ ($i$=1,…,$N$).
       (b) Call $PrefixSpan(\alpha_i, l, S|\alpha_i, ToPattern(\alpha_i), FromPattern(\alpha_i))$.
Method 2: Call $PrefixSpan(\alpha, l, S|\alpha, ToPattern, FromPattern)$.
    The parameters are (a) $\alpha$ is a sequential pattern; (b) $l$ is the length of $\alpha$; and (c) $S|\alpha$ is the $\alpha$-projected database; (d) when $l$=1, ToPattern is the $ToPattern(\alpha)$, otherwise ToPattern is null. (e) FromPattern is the $FromPattern(\alpha)$.
Steps:
    1. Scan $S|\alpha$ once, find each frequent item, $b$.
    2. For each frequent item $b$,
       (a) append $b$ to $\alpha$ to from a sequential pattern $\alpha'$, and output $a'$;
       (b) If $FromPattern(\alpha)\neq null$.
           For each $\beta_i\in FromPattern(\alpha)$, append $\beta_i$ to $\alpha'$ to from sequential patterns $\alpha_i''$, and output $\alpha_i''$;
       (c) If $b\in ToPattern(\alpha)$, then stop, go to next $b$.
    3. For each $\alpha'$, construct $\alpha'$-projected database $S|\alpha'$, and call $PrefixSpan(a', l+1, S|a', null, FromPattern)$.
```

There are two problems in the algorithm PFI-PrefixSpan. The first one is if $\beta_i\in From\ Pattern(\alpha)$ and $\gamma_j\in FromPattern(\beta_i)$, then it is needed to add all γ_j to α_i''. It is a chain that γ_j-->β_i-->α', and so on, until there is no new *FromPattern* in the chain. We can get the chain from the PFI-Graph G. The second one is the new algorithm reduce the runtime and memory usage of constructing projected database than PrefixSpan, but adding the additional consuming of building and searching PFI-Graph.

IV. Performance Evaluation

In this chapter, we provide two experimental results. The first one is using some synthetic datasets to compare the performance of PrefixSpan with physical projection, PrefixSpan with pseudo projection and PFI-PrefixSpan. The purpose is to verify the performance of the second and third algorithms are better than the first one. But the PFI number in these synthetic datasets is small. Then we use three real data sets which have dense and highly similar sequences to compare pseudo projection PrefixSpan with PFI-PrefixSpan, whereas these data sets have big PFI number.

4.1 Test Environment and Data Sets

The experiments are performed on a 2.1 GHZ CUP with 2GB memory, and running on Win7. All the algorithms were coded in Java language. In our performance study, we use two kinds of data sets: synthetic data sets and real data sets. The synthetic data sets are generated by a data generator similar in spirit to the IBM data generator designed for testing sequential pattern mining algorithms. The convention for the data sets is as follows: C1N0.1T8S8 means that the data set contains 1000 sequences and the number of different items is 100. The average number of items in a transaction is 8 and the average number of transactions in sequence is 8.

Two real data sets new_york and new_orleans are from UCI[19], and the number of sequence and the number of average transactions in sequence are shown in Table 3. These files are the data underlying the Entree system.

TABLE III.
DATA SETS

dataset	#sequence	#transactions/sequence
new_york	1200	8
new_orleans	327	11

4.2 Experimental Results

Firstly, we conducte our experiments of three algorithms: (1) PrefixSpan with physical projection (abbreviated as Phy-PrefixSpan), (2) PrefixSpan with pseudo projection (abbreviated as Pseudo-PrefixSpan) and (3) PrefixSpan with Prefix-Frequent-Items (PFI-PrefixSpan) on two synthetic datasets.

The first test is on data set C1N0.1T8S8. The average transactions number in sequence is 8, and the different items is 100. The actual transactions number in sequence is 7, so the distinct item recurrence rate or density [14] m=average sequence length/different items =8/100=0.08 (or 0.07).

The memory usages of the three algorithms are shown in the Figure 4. The support thresholds are from 0.063 to 0.075, and the *CountsPFI* are 2 and 3. It makes clear distinction among three algorithms, and the memory usage of Pseudo-PrefixSpan is about 38% lower than Phy-PrefixSpan. Memory usage of PFI-PrefixSpan is significantly lower than the former two algorithms when *min_sup* is lower than 0.07. Figure 5 shows the processing time of the three algorithms at different support thresholds on data set C1N0.1T8S8. But the runtime of Pseudo-PrefixSpan and PFI-PrefixSpan is higher than Phy-PrefixSpan. The reason is that the time consumed for navigating from the pseudo location to physical location and build PFI-Graph is more than the time saved by using PFI-Graph. Figure 6 shows the distributions of frequent sequences, the length of frequent patterns is from 1 to 6 and length 3 and 4 patterns are the highest.

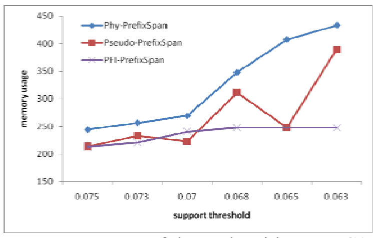

Figure 4. Memory usage of three algorithms on C1N0.1T8S8

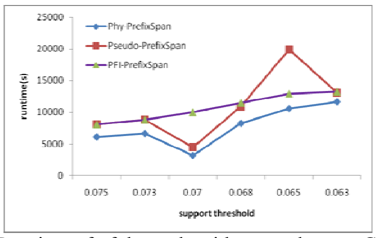

Figure 5. Runtime of of three algorithms on data set C1N0.1T8S8

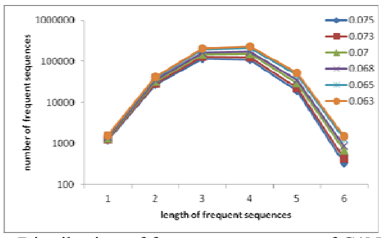

Figure 6. Distribution of frequent sequences of C1N0.1T8S8

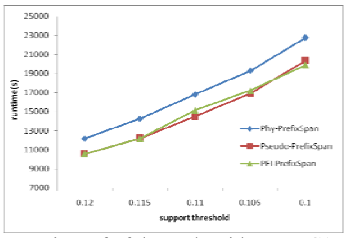

Figure 7. Runtime of of three algorithms on C1N0.1T8S10

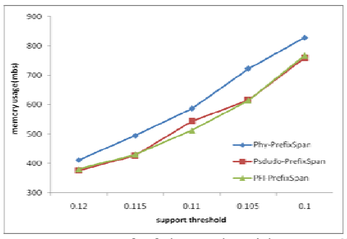

Figure 8. Memory usage of of three algorithms on C1N0.1T8S10

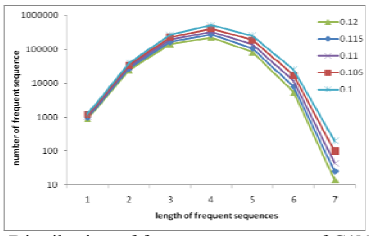

Figure 9. Distribution of frequent sequences of C1N0.1T8S10

The second test is on the data set C1N0.1T8S10, which contains 1000 sequences and the different items is 100 too. The average number of items in a transaction is 8 and the average number of transactions in a sequence is 10. The actual transactions number in sequence is 9, so the distinct item recurrence rate is 0.1 (or 0.09). It is denser than C1N0.1T8S8.

Figure 7 shows the runtime of the three algorithms at support thresholds from 0.1 to 0.12, and the *CountsPFI* is 4. On average, the runtime of Pseudo-PrefixSpan is about 16% lower than Phy- PrefixSpan, and PFI-PrefixSpan is about 20% lower than the Phy-PrefixSpan. The memory usage is shown in Figure 8, and both of PFI-PrefixSpan and Pseudo-PrefixSpan are about 12% lower than Phy-PrefixSpan. Figure 9 shows the distributions of frequent sequences. The length of frequent patterns is from 1 to 7, and length 4 pattern is the highest.

It can be seen from the experimental results that, the memory usage of Pseudo-PrefixSpan and PFI-PrefixSpan are significantly lower than Phy- PrefixSpan. But the performance of Pseudo-Prefix- Span and PFI-PrefixSpan is not very different. The reason is the small *CountsPFI* of synthetic data sets. We also tried other data sets, such as CXN0.03S8T20, CXN0.05S8T20, CXN0.03S8T30 and so on, but the *CountsPFI* is still less than 8, even when the sequence number meet the minimum support as low as 10.

TABLE IV.
SOME INFORMATION OF NEW_ORLEANS

min_sup	#patterns	#CountsPFI
0.13	246	7
0.12	279	7
0.11	366	9
0.1	452	11
0.09	536	11
0.08	677	11
0.07	949	14

Secondly, we make experiments of Pseudo-PrefixSpan and PFI-PrefixSpan on three real sequence databases, in order to verify that PrefixSpan with PFI-Graph performs better than PrefixSpan. Both of these algorithms use pseudo projection. The character of these data sets is highly similar items, whereas the *CountsPFI* is big.

The first test real data set is new_orleans, the number of sequence is 327 and the average number of transactions in a sequence is 11. There is one item of a transaction. As it is shown in Table 4, the *CountsPFI* is from 7 to 14 and ascending when the *min_sup* decreasing. Therefore, more duplicated work can be reduced with small *min_sup*.

Figures 10 to 12 show the runtime, memory usage and frequent sequence length of PrefixSpan and PFI-PrefixSpan on data set new_orleans. Figure 10 shows the runtime of two algorithms at different support thresholds, and the runtime of PFI- PrefixSpan is 13.1% lower than PrefixSpan. We can see that the distance between the two broken lines is almost steadily, therefore the advantages of the new algorithm is stable. The memory usage of two algorithms on new_orleans is shown in Figure 11. It is clear that the memory usage of PFI-PrefixSpan is reduced by 5.5% than PrefixSpan on average. When minimum support is 0.1, PFI-PrefixSpan reduces memory consumption by 10% than PrefixSpan. Figure 12 shows the distribution of frequent sequences of data set new_orleans, from which we can see that the length-3 pattern is the largest number of frequent pattern, and the 3, is about 27.3% (that is 3/11) of average sequence length 11.

From the Figures 10 to 12, we can assume that runtime and memory usage of PFI-PrefixSpan algorithm is lower than PrefixSpan when the minimum support is low. But if the minimum support is too low, the memory usage of PFI-PrefixSpan is bigger than PrefixSpan. For example, suppose the *min_sup* is 0.02, the number of sequences is 7, and the *CountsPFI* grows up to 34. Runtime of PFI-PrefixSpan is about 13% lower than the PrefixSpan. But the memory usage of the new algorithm is 6% higher than the old one. The reason is that the memory consumed to store and search PFI-Graph is increasing too much.

TABLE V.
SOME INFORMATION OF NEW_YORK

min_sup	#patterns	#CountsPFI
0.06	138	10
0.055	154	10
0.05	197	10
0.045	236	10
0.04	283	10
0.035	351	10
0.03	495	11

The second test is performed on the data set new_york. Sequence number of new_york is 1200, and average number of transactions in sequence is 8. Some information is shown in Table 5, the *min_sup* is from 0.03 to 0.06 and the *CountsPFI* values are 10 and 11. When *min_sup* is 0.11, the *CountsPFI* is 9. Therefore, the *CountsPFI* is around 10 when *min_sup* less than 0.1.

The runtime of PFI_PrefixSpan and PrefixSpan on data set new_york is shown in Figure 13. From this figure, we can see the average runtime of novel algorithm is about 7% lower than PrefixSpan. The distance between the two broken lines is almost steady. Figure 14 shows that when the *min_sup* are 0.04 and 0.035, the memory usage of PFI-PrefixSpan are 12% and 11.2% lower than PrefixSpan. On average, the memory usage of novel algorithm is 7% lower than PrefixSpan. Figure 15 shows the distribution of frequent sequences of data set new_york. We can see that the length 2 and 3 patterns are the largest number in all frequent patterns, and the length 2 is about 25% (that is 2/8) of average sequence length 8.

From the experiment results on synthetic data sets and real datasets we conclude that:

(1) PrefixSpan with pseudo projection algorithm uses lower memory usage than PrefixSpan with physical projection, especially in large sequence databases.

(2) Compared with Pseudo-PrefixSpan and Phy-PrefixSpan, novel algorithm PFI-PrefixSpan based on PFI-Graph can reduce the memory usage and the runtime.

(3) PFI-PrefixSpan applies to highly similar and dense database.

(4) It is difficult to build PFI-Graph in large sequence database, because the PFI is small and it only exists in very low minimum support. For example, when the *min_sup*=0.0005, the *CountsPFI* of data sets C10N0.1T2.5S10, C10N0.1T8S8, C10N0.1T8 S10, C10N0.05T8S8, C100N0.1T2.5S10, C100N0.1 T8S8 is 0. When the *min_sup* is 0.0001 and the number of sequence is 10, the *CountsPFI* of C100N0.1T2.5S10 and C100N0.1T8S8 is only 3.

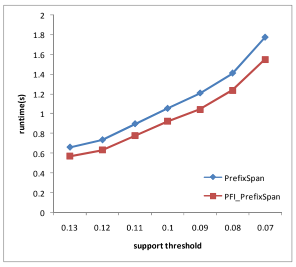

Figure 10. Performance of the algorithms on data set new_orleans

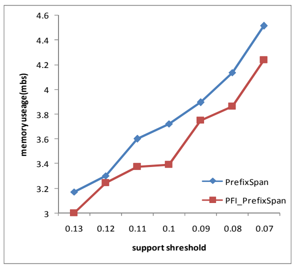

Figure 11. Memory usage comparison among the algorithms on data set new_orleans

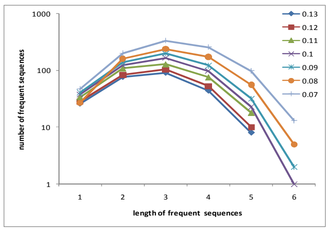

Figure 12. Distribution of frequent sequences of new_orleans

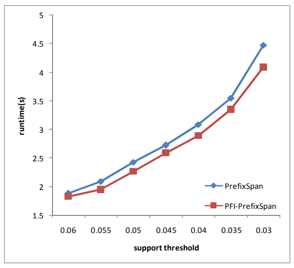

Figure 13. Performance of the algorithms on data set new_york

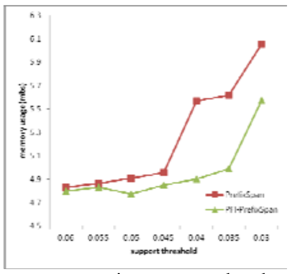

Figure 14. Memory usage comparison among the algorithms on data set new_york

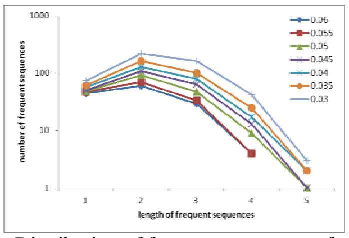

Figure 15. Distribution of frequent sequences of new_york

V. CONCLUSION

In order to improve the efficiency of sequential pattern mining on dense and highly similar sequence database, a novel algorithm PFI-PrefixSpan is provided in this paper. It is based on Prefix Frequent Items Graph (PFI-Graph) and it works quite well in the experiments. The memory usage and runtime of novel algorithm are about 25% and 20% less than PrefixSpan with physical projection (Phy-PrefixSpan) on average, respectively. It reduces about 10% runtime and 6% memory usage than PrefixSpan with pseudo projection (Pseudo-PrefixSpan) on given real datasets, respectively.

The novel algorithm is based on the Prefix Frequent Items (PFI) and PFI-Graph, and it checks weather the efforts for building a projected database is duplicated ones to others. So, the main idea of the novel algorithm is to avoid the duplicated projected database.

Some experiments in this paper verify that this algorithm is suitable for highly similar and dense sequence databases. But it is difficult to build PFI-Graph in large and sparse database.

REFERENCES

[1] Pei J and Wang J Y et al, “Mining sequential patterns by pattern-growth: The prefixspan approach,” IEEE Transactions On Knowledge And Data Engineering, 2004, 16(1):1-17.

[2] Huang T C K, “Knowledge gathering of fuzzy multi-time-interval sequential patterns. Information Sciences,” 2010, 180(1): 3316-3334.

[3] Padmaja P, Jyothi P N, and Bharagava M, “Recursive prefix suffix pattern detection approach for mining sequential patterns,” International Journal of Computer Applications, 2011, 29(3): 50-53.

[4] Yang S Y, Chao C M, and Chen P Z et al, “Incremental mining of across-streams sequential patterns in multiple data streams,” Journal of Computers, 2011, 6(3): 449-457.

[5] Ezeife C I, and Liu Y, “Fast incremental mining of web sequential patterns with PLWAP tree,” Data Ming and Knowledge Discovery, 2009, 19(2): 376-416.

[6] Vasumathi D, and Govardhan A, “BC-WASPT : Web access sequential pattern tree mining,” International Journal of Computer Science and Network Security, 2009, 9(6): 288-293.

[7] Sartipi K, and Safyallah H, “Dynamic knowledge extraction from software systems using sequential pattern mining,” International Journal of Software Engineering and Knowledge Engineering, 2009, 20(5): 1-22.

[8] Todaro M A, Kanneby T, Zotto M D, and Jondelius U. Phylogeny of Thaumastodermatidae (gastrotricha: macrodasyida) inferred from nuclear and mitochondrial. PlosOne, 2011, 6(3): 1-13.

[9] Alves R, Rodriguez-Baena D S, and Aguilar-Ruiz J S, “Gene association analysis: a survey of frequent pattern mining from gene expression data,” Briefings in Bioinformatics, 2010, 11(2): 210-224.

[10] Ahmed C F, Tanbeer S K, and Jeong B S, “A novel approach for mining high-utility sequential patterns in sequence databases,” ETRI Journal, 2010, 32(5): 676-686.

[11] Exarchos T P, Papaloukas C, and Lampros C, “Mining sequential patterns for protein fold recognition,” Journal of Biomedical Informatics, 2008, 41(1): 165-179.

[12] Liu H , Jiang Z and Fang X et al, “Generate gene expression profile from high-throughput sequencing data,” Frontiers of Thematics in China, 2011, 6(6): 1131-1145.

[13] Ayres J, Gehrke J, Yiu T, and Flannick J, “Sequential PAttern Mining using a bitmap representation,” In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '02, NewYork), 2002: 429-435.

[14] Yang Z, Wang Y, and Kitsuregawa M, “LAPIN: effective sequential pattern mining algorithms by last position induction for dense databases,” In: Proceeding of the 12th International Conference on Database Systems for Advanced Applications(Bangkok, Thailand, DASFAA 2007), Springer-Verlag, Berlin, Heidelberg, Vol. 4443, 2007: 1020-1023.

[15] El-Sayed M, Ruiz C, and Rundensteiner E A, “FS-Miner: Efficient and incremental mining of frequent sequence patterns in web logs,” In Proceedings of the 6th Annual ACM International Workshop on Web Information and Data Management(ACM, New York), 2004: 128-135.

[16] Elsa Loekito E, Bailey J, and Pei J, “A binary decision diagram based approach for mining frequent subsequences,” Knowledge And Information Systems, 2009, 24(2): 235-268.

[17] Han J W, Cheng H , Xin D, and Yan X F, “Frequent pattern mining: current status and future directions,” Data Mining and Knowledge Discovery, 2007, 15(1):55-86.

[18] Zhu J, “An efficient method of web sequential pattern mining based on session filter and transaction identification,” Journal of Networks, 2010, 5(9): 1017-1024.

[19] Frank, A., and Asuncion, A, UCI Machine Learning Repository, http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science, 2010

Meng Han is currently a candidate of Ph.D. student in Beijing Jiaotong University (Beijing, China), and also a lecturer in Beifang University of Nationalities (Yinchuan, China). Her research interests include data mining and machine learning.

Zhihai Wang received his PhD in Computer Science from Hefei University of Technology in 1998. He is now a professor in School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China. He has published dozens of papers in international conferences and journals. His research interest includes data mining and artificial intelligence.

Jidong Yuan is currently a candidate of Ph.D. student in Beijing Jiaotong University (Beijing, China). His research interests include machine learning and data mining.