PrefixSpan: Mining Sequential Patterns by Prefix-Projected Pattern

Poonam Sharma[1] and Gudla.Balakrishna[2]

[1]Department of Computer Engineering, MLVTEC, Bhilwara, India
poonam_bhilwara@yahoo.co.in[1]
[2]Department of Computer Engineering Mewar University, Chittorgarh, India
balakrishnagudla@gmail.com

ABSTRACT

Sequential pattern mining discovers frequent subsequences as patterns in a sequence database. Most of the previously developed sequential pattern mining methods, such as GSP, explore a candidate generation-and-test approach [1] to reduce the number of candidates to be examined. However, this approach may not be efficient in mining large sequence databases having numerous patterns and/or long patterns. In this paper, we propose a projection-based, sequential pattern-growth approach for efficient mining of sequential patterns. In this approach, a sequence database is recursively projected into a set of smaller projected databases, and sequential patterns are grown in each projected database by exploring only locally frequent fragments. Based on an initial study of the pattern growth-based sequential pattern mining, FreeSpan, we propose a more efficient method, called PSP, which offers ordered growth and reduced projected databases technique is developed in PrefixSpan.

KEYWORDS

Sequential pattern, frequent pattern, candidate sequences, sequence database.

1. Introduction

Sequential pattern mining, which discovers frequent subsequences as patterns in a sequence database, is an important data mining problem with broad applications, including the analysis of customer purchase patterns or Web access patterns, the analysis of sequencing or time related processes such as scientific experiments, natural disasters, and disease treatments, the analysis of DNA sequences etc.

The sequential pattern mining problem[2] was first introduced by Agrawal and Srikant Given a set of sequences, where each sequence consists of a list of elements and each element consists of a set of items, and given a user-specified min_support threshold, sequential pattern mining is to find all frequent subsequences, i.e., the subsequence whose occurrence frequency in the set of sequences is no less than min_support Srikant and Agrawal generalized[7] their definition of sequential patterns in[2] to include time constraints, sliding time window, and user-defined taxonomy, and presented an a priori-based, improved algorithm GSP (i.e., generalized sequential patterns). Almost all of the proposed methods for mining sequential patterns and other time-related frequent patterns are a priori-like, i.e., based on the a priori principle, which states the fact that any super-pattern of an infrequent pattern cannot be frequent, and based on a

DOI : 10.5121/ijcses.2011.2408

candidate generation and test paradigm proposed in association mining[1]. A typical a priori-like sequential pattern mining method, such as GSP[7], adopts a multiple-pass, candidate generation-and-test approach outlined as follows: The first scan finds all of the frequent items that form the set of single item frequent sequences. Each subsequent pass starts with a seed set of sequential patterns, which is the set of sequential patterns found in the previous pass. This seed set is used to generate new potential patterns, called **candidate sequences**, based on the a priori principle. Each candidate sequence contains one more item than a seed sequential pattern, where each element in the pattern may contain one item or multiple items. The number of items in a sequence is called the length of the sequence. So, all the candidate sequences in a pass will have the same length. The scan of the database in one pass finds the support for each candidate sequence. All the candidates with support no less than min_support in the database form the set of the newly found sequential patterns. This set is then used as the seed set for the next pass. The algorithm terminates when no new sequential pattern is found in a pass, or when no candidate sequence can be generated.

The a priori-like sequential pattern mining method, though reducing search space, bears three nontrivial, inherent costs that are independent of detailed implementation techniques.

- A huge set of candidate sequences could be generated in a large sequence database. Since the set of candidate sequences includes all the possible permutations of the elements and repetition of items in a sequence, the priori-based method may generate a really large set of candidate sequences even for a moderate seed set. For example, two frequent sequences of length-1, (a) and (b), will generate five candidate sequences of length-2: (aa), (ab), (ba), (bb) and ((ab)), where ((ab)) represents that two events, a and b, happen in the same time slot. If there are 1,000 frequent sequences of length-1, such as (a_1), (a_2),…..(a_{1000}), an priori like algorithm will generate 1,000 * 1, 000 + 1000*999/2=1,499,500 candidate sequences. The cost of candidate sequence generation, test, and support counting is inherent to the priori-based method, no matter what technique is applied to optimize its detailed implementation.
- Multiple scans of databases in mining. The length of each candidate sequence grows by one at each database scan. In general, to find a sequential pattern of length l, the priori-based method must scan the database at least l times. This bears a nontrivial cost when long patterns exist.
- The priori-based method generates a combinatorial explosive number of candidates when mining long sequential patterns. A long sequential pattern contains a combinatorial explosive number of subsequences, and such subsequences must be generated and tested in the priori-based mining. Thus, the number of candidate sequences is exponential to the length of the sequential patterns to be mined. For example, let the database contain only one single sequence of length 100, (a_1……a_{100}), and the min_support threshold be 1 (i.e., every occurring pattern is frequent). To (re)derive this length-100 sequential pattern, the a priori-based method has to generate 100 length-1 candidate sequences (i.e., (a_1,a_2,….a_{100}) , 100* 100 + 100*99/2=14,950 length-2 candidate sequences, and so on.

In case of DNA analysis or stock sequence analysis, the databases often contain a large number of sequential patterns and many patterns are long. It is important to re-examine the sequential pattern mining problem to explore more efficient and scalable methods. In this paper, we explore a pattern-growth approach for efficient mining of sequential patterns in large sequence database. The approach adopts a divide-and conquers pattern-growth principle as follows: Sequence databases are recursively projected into a set of smaller projected databases based on the current sequential pattern(s), and sequential patterns are grown in each projected databases by exploring only locally frequent fragments.

Firstly we proposed a straightforward pattern growth method, FreeSpan (for Frequent pattern-projected Sequential pattern mining) [8], which reduce the efforts of candidate subsequence generation. In this paper, we introduce another and more efficient method, called PrefixSpan (for Prefix-projected Sequential pattern mining), which offers ordered growth and reduced

projected databases.

Sequence_id	Sequence
10	$\langle a(abc)(ac)d(cf)\rangle$
20	$\langle (ad)c(bc)(ae)\rangle$
30	$\langle (ef)(ab)(df)cb\rangle$
40	$\langle eg(af)cbc\rangle$

Table 1: A Sequence Database

2. Problem definition and the GSP algorithm

In this section, the problem of sequential pattern mining is defined, and the most representative a priori-based sequential pattern mining method, GSP[7], is illustrated using an example.

Problem Definition

Let I= $\{i_1,i_2,\ldots\ldots i_n\}$ be a set of all items. An itemset is a subset of items. A sequence is an ordered list of itemsets. A Sequence is denoted by $(s_1, s_2\ldots\ldots s_j)$, where s_j is an itemset.sj is also called an element of the sequence, and denoted as $(x_1, x2\ldots..x_m)$, where x_k is an item. For brevity, the brackets are omitted if an element has only one item, i.e., element (x) is written as x. An item can occur at most once in an element of a sequence, but can occur multiple times in different elements of a sequence. The number of instances of items in a sequence is called the length of the sequence. A sequence with length l is called an l-sequence. A sequence $\alpha= (a_1, a_2..... a_n)$ is called a subsequence of another sequence $\beta = (b_1, b_2\ldots.. b_m)$ and β a super sequence of α, denoted as $\alpha \subseteq \beta$, if there exist integers $1 \leq j_1< j_2 <\ldots\ldots\ldots<j_n \leq m$ such that $a_1 \subseteq b_{1j}$, $a_2 \subseteq b_{2j,}$ $a_n \subseteq b_{jn}$. A sequence database S is a set of tuples (s_{id}, s) , where s_{id} is a sequence_id and s a sequence. A tuple (s_{id}, s) is said to contain a sequence α, if α is a subsequence of s. The support of a sequence α in a sequence database S is the number of tuples in the database containing α, i.e.

$support_S(\alpha) = | \{ ((s_{id}, s) | (s_{id}, s) \in S) \cap (\alpha \subseteq s) \} |$

It can be denoted as support (α) if the sequence database is clear from the context. Given a positive integer min_support as the support threshold, a sequence α is called a sequential pattern in sequence database S if support (α) $\geq$ min_support. A sequential pattern with length l is called an l-pattern.

Example 1: Let our running sequence database be S given in Figure 1 and min_support = 2. The set of items in the database is {a, b, c, d, e, f, g}. A sequence (a(abc)(ac)d(cf)) has five elements: (a), (abc), (ac), (d), and (cf), where items a and c appears more than once, respectively, in different elements. It is a 9-sequence since there are nine instances appearing in that sequence. Item a happens three times in this sequence, so it contributes 3 to the length of the sequence. However the whole sequence (a(abc)(ac)d(cf)) contributes only 1 to the support of (a). Also, sequence (a(bc)df) is a subsequence of (a(abc)(ac)d(cf)). Since both sequences 10 and 30 contain subsequence s = ((ab)c), s is a sequential pattern of length 3 (i.e., 3-pattern).

Problem Statement: Given a sequence database and the min_support threshold, sequential pattern mining is to find the complete set of sequential patterns in the database.

Algorithm GSP

A typical sequential pattern mining method, GSP[7], mines sequential patterns by adopting a candidate subsequence generation-and-test approach, based on the a priori principle. The method is illustrated using the following example:

Example 2 (GSP): Given the database S and min_support in Example 1, GSP first scans S, collects the support for each item, and finds the set of frequent items, i.e., frequent length-1 subsequences (in the form of "item: support"):

(a): 4; (b) : 4; (c) : 3; (d) : 3; (e) : 3; (f) : 3; (g) : 1

By filtering the infrequent item g, we obtain the first seed set L_1 = { (a), (b), (c), (d), (e), (f)}, each member in the set representing a 1-element sequential pattern. Each subsequent pass starts with the seed set found in the previous pass and uses it to generate new potential sequential patterns, called candidate sequences. For L1, a set of 6 length-1 sequential patterns generates a set of 6*6+6*5/2 = 51 candidate sequences,

C_2 ={(aa),(ab),…..(af),(ba),(bb),…(ff), ((ab)), ((ac)),….((ef))}.

In the multiscan mining process the set of candidates is generated by a self-join of the sequential patterns found in the previous pass. In the kth pass, a sequence is a candidate only if each of its length (k-1) subsequences is a sequential pattern found at the (k-1)th pass. A new scan of the database collects the support for each candidate sequence and

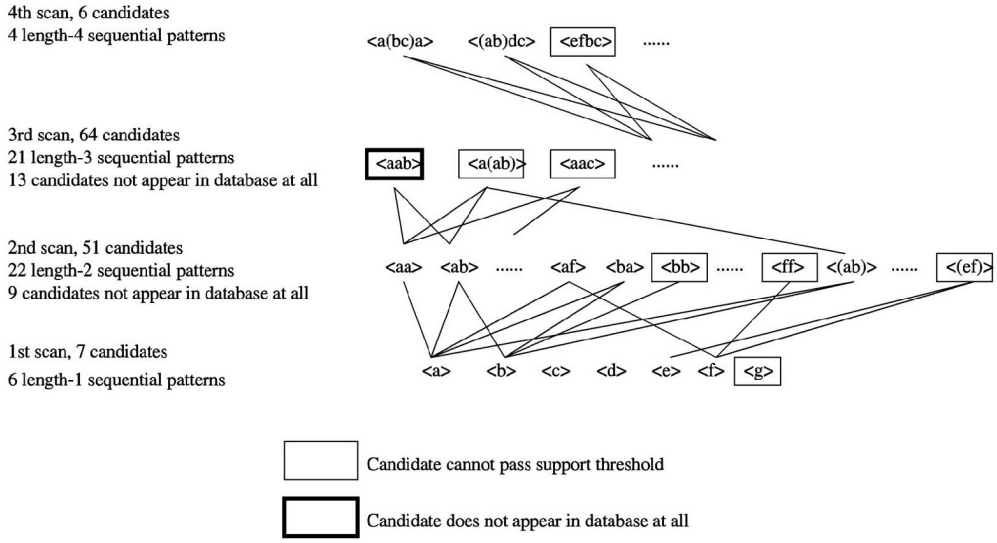

Figure 1: Candidates and sequential patterns in GSP

finds the new set of sequential patterns. This set becomes the seed for the next pass. The algorithm terminates when no sequential pattern is found in a pass, or when there is no candidate sequence generated. Clearly, the number of scans is at least the maximum length of sequential patterns. It needs one more scan if the sequential patterns obtained in the last scan still generate new candidates. GSP, though benefits from the a priori pruning, still generates a large number of candidates. In this example, 6 length-1 sequential patterns generate 51 length-2 candidates, 22 length-2 sequential patterns generate 64 length-3 candidates, etc. Some candidates generated by GSP may not appear in the database at all. For example, 13 out of 64 length-3 candidates do not appear in the database.

3. Mining sequential patterns by pattern Growth

As analyzed in Section 1 and Example 2, the GSP algorithm shares similar strengths and weaknesses as the a priori method. For frequent pattern mining, a frequent pattern growth method called FP-growth[9] has been developed for efficient mining of frequent patterns without candidate generation. The method uses a data structure called FP-tree to store compressed frequent patterns in transaction database and recursively mines the projected conditional FP-trees to achieve high performance.

We can mine sequential patterns by extension of the FP-tree structure but it cannot be so optimistic because it is easy to explore the sharing among a set of unordered items, but it is difficult to explore the sharing of common data structures among a set of ordered items. For example, a set of frequent itemsets {abc, cbad, ebadc, cadb} share the same tree branch {abcde} in the FP-tree. However, if they were a set of sequences, there is no common prefix subtree structure that can be shared among them because one cannot change the order of items to form sharable prefix subsequences. We can explore the spirit of FP-growth: divide the sequential patterns to be mined based on the subsequences obtained so far and project the sequence database based on the partition of such patterns. Such a methodology is called **sequential pattern mining** by pattern growth. The general idea is outlined as follows: Instead of repeatedly scanning the entire database and generating and testing large sets of candidate sequences, one can recursively project a sequence database into a set of smaller databases associated with the set of patterns mined so far and, then, mine locally frequent patterns in each projected database. We first outline a projection-based sequential pattern mining method, called FreeSpan[8] , and then systematically introduce an improved method PrefixSpan[10]. Both methods generate projected databases, but they differ at the criteria of database projection: FreeSpan creates projected databases based on the current set of frequent patterns without a particular ordering (i.e., growth direction), whereas PrefixSpan projects databases by growing frequent prefixes. Our study shows that, although both FreeSpan and PrefixSpan are efficient and scalable, PrefixSpan is substantially faster than FreeSpan in most sequence databases.

3.1. FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining

For a sequence $\alpha = (s_1 \ldots\ldots .. s_l)$, the itemset $s_1 U \ldots .. U s_l$ is called α's projected itemset. FreeSpan is based on the following property: If an itemset X is infrequent, any sequence whose projected itemset is a superset of X cannot be a sequential pattern. FreeSpan mines sequential patterns by partitioning the search space and projecting the sequence sub databases recursively based on the projected itemsets.

Let list = $(x_1 \ldots . x_n)$ be a list of all frequent items in sequence database S. Then, the complete set of sequential patterns in S can be divided into n disjoint subsets: 1) the set of sequential patterns containing only item x_1, 2) those containing item x_2 but no item in $(x_3, \ldots ; x_n)$, and so on. In general, the i_{th} subset $(1 \leq i \leq n)$ is the set of sequential patterns containing item x_i but no item in $(x_{i+1;} \ldots x_n)$. At the time of deriving p's projected database from DB, the set of frequent items X of DB is already known. Only those items in X will need to be projected into p's projected database. This effectively discards irrelevant information and keeps the size of the projected database minimal. By recursively doing so, one can mine the projected databases and generate the complete set of sequential patterns in the given partition without duplication. The details are illustrated in the following example:

Example (FreeSpan): Given the database S and min_support in Example 1, FreeSpan first scans S, collects the support for each item, and finds the set of frequent items. This step is similar to GSP. Frequent items are listed in support descending order (in the form of "item: support"), that is list = a : 4; b : 4; c : 4; d : 3; e : 3; f : 3. They form six length-one sequential patterns:

(a) : 4; (b) : 4; (c) : 4; (d) : 3; (e) : 3; (f) : 3.

According to the list, the complete set of sequential patterns in S can be divided into six disjoint subsets:

1. the ones containing only item a,
2. the ones containing item b but no item after b in list,
3. the ones containing item c but no item after c in list, and so on, and, finally,
4. the ones containing item f.

The sequential patterns related to the six partitioned subsets can be mined by constructing six projected databases (obtained by one additional scan of the original database). Infrequent items, such as g in this example, are removed from the projected databases. The process for mining each projected database is detailed as follows:

- Mining sequential patterns containing only item a. By mining the (a)-projected database: {(aaa), (aa), (a), (a)}, only one additional sequential pattern containing only item a, i.e., (aa): 2, is found.
- Mining sequential patterns containing item b but no item after b in the list. By mining the (b)-projected database: {(a(ab)a), (aba),((ab)b),(ab)}, four additional sequential patterns containing item b but no item after b in list are found. They are
  {(ab) : 4, (ba) : 2, ((ab)) : 2, (aba) : 2}.
- Mining sequential patterns containing item c but no item after c in the list. Mining the (c)-projected database: {(a(abc)(ac)c), (ac(bc)a), ((ab)cb), (acbc)}, proceeds as follows:
  One scan of the projected database generates the set of length-2 frequent sequences, which are {(ac) : 4, ((bc)) : 2, (bc) : 3, (cc) : 3, (ca) : 2, (cb) : 3}. One additional scan of the (c)-projected database generates all of its projected databases. The mining of the (ac)-projected database: {(a(abc)(ac)c), (ac(bc)a), ((ab)cb), (acbc)} generates the set of length-3 patterns as follows:
  {(acb) : 3, (acc) : 3, ((ab)c) : 2, (aca) : 2}.
  Four projected database will be generated from them. The mining of the first one, the (acb)-projected database: {(ac (bc) a), ((ab) cb), (acbc)} generates no length-4 pattern. The mining along this line terminates. On the one hand, the strength of FreeSpan is that it searches a smaller projected database than GSP in each subsequent database projection. This is because FreeSpan projects a large sequence database recursively into a set of small projected sequence databases based on the currently mined frequent item-patterns, and the subsequent mining is confined to each projected database relevant to a smaller set of candidates. On the other hand, the major overhead of FreeSpan is that it may have to generate many nontrivial projected databases. If a pattern appears in each sequence of a database, its projected database does not shrink (except for the removal of some infrequent items). For example, the {f}-projected database in this example contains three of the same sequences as that in the original sequence database, except for the removal of the infrequent item g in sequence 40. Moreover, since a length-k subsequence may grow at any position, the search for length-(k + 1) candidate sequence will need to check every possible combination, which is costly.

3.2 PrefixSpan: Prefix-Projected Sequential Patterns Mining

As we have seen, in FreeSpan algorithm, we have to pay high cost at handling projected databases. So, the next to be done is reduce the size of projected database and the cost of checking at every possible position of a potential candidate sequence. To avoid checking every possible combination of a potential candidate sequence, one can first fix the order of items within each element. Since items within an element of a sequence can be listed in any order, without loss of generality, one can assume that they are always listed alphabetically. For example, the sequence in S with Sequence_id 10 in our running example is listed as h(a(abc)(ac)d(cf)) instead of (a(bac)(ca)d(f c)). With such a convention, the expression of a sequence is unique. Then, we examine whether one can fix the order of item projection in the generation of a projected database. Intuitively, if one follows the order of the prefix of a sequence and projects only the suffix of a sequence, one can examine in an orderly manner all the possible subsequence and their associated projected database.

Definition 1 (Prefix). Suppose all the items within an element are listed alphabetically. Given a

sequence $\alpha = (e_1,e_2,\ldots.e_n)$ (where each e_i corresponds to a frequent element in S), a sequence $\beta = (e`_1, e`_2,\ldots.e`_m)$ ($m \leq n$) is called a Prefix of α if and only if : 1) $e`_i = e_i$ for ($i \leq m$ -1); 2) $e`_m \subseteq e_m$ and 3) all the frequent items in ($e_m$-$e`_m$) are alphabetically after those in $e`_m$. For example, (a), (aa), (a(ab)), and (a(abc)) are prefixes of sequence s s = (a(abc)(ac)d(cf)).

Definition 2 (Suffix): Given a sequence $\alpha = (e_1,e_2,\ldots.e_n)$ (where each e_i corresponds to a frequent element in S). Let $\beta = (e_1,e_2,\ldots.e_{m-1}e`_m)$ ($m \leq n$) be the prefix of α. Sequence $\gamma = (e``_m e_{m+1}\ldots e_n)$ is called the suffix of α with regards to prefix β, denoted as $\gamma=\alpha/\beta$,where $e^n_m = (e_m - e`_m)$. For example, for the sequence s =(a(abc)(ac)d(cf)), ((abc))(ac)d(cf)) is the suffix with regards to the prefix (a), (_bc)(ac)d(cf)) is the suffix with regards to the prefix (aa), and ((_c)(ac)d(cf)) is the suffix with regards to the prefix (a(ab)).

Lemma 3.1 Problem partitioning

1. Let $\{(x_1), (x_2), \ldots..,(x_n)\}$ be the complete set of length- 1 sequential patterns in a sequence database S. The complete set of sequential patterns in S can be divided into n disjoint subsets. The i_{th} subset ($1 \leq i \leq n$) is the set of sequential patterns with prefix (x_i).

2. Let α be a length-l sequential pattern and $\{\beta_1,\beta_2 \ldots..,\beta_m\}$ be the set of all length-(l + 1) sequential patterns with prefix α. The complete set of sequential patterns with prefix α, except for α itself, can be divided into m disjoint subsets. The jth subset ($1 \leq j \leq m$) is the set of sequential patterns prefixed with βj.

Proof: The first half is a special case where α = (). For a sequential pattern γ with prefix α, where α is of length l, the length-(l + 1) prefix of γ must be a sequential pattern, according to the a priori heuristic. Furthermore, the length-(l+ 1) prefix of γ is also with prefix α, according to the definition of prefix. Therefore, there exists some j ($1 \leq j \leq m$) such that βj is the length-(l + 1) prefix of γ. Thus, γ is in the jth subset. On the other hand, since the length-k prefix of a sequence γ is unique, γ belongs to only one determined subset. That is, the subsets are disjoint. So, we have the point that, the problem can be partitioned recursively. That is, each subset of sequential patterns can be further divided when necessary. This forms a divide-and conquer framework. To mine the subsets of sequential patterns, the corresponding projected databases can be constructed.

Definition 3 (Projected database) Let α be a sequential pattern in a sequence database S. The α-projected database, denoted as $S|_\alpha$, is the collection of suffixes of sequences in S with regards to prefix α. To collect counts in projected databases, we have the following definition:

Definition 4 (Support count in projected database) Let α be a sequential pattern in sequence database S, and β be a sequence with prefix α. The support count of β in α-projected database $S|_\alpha$, denoted as support $S|_\alpha(\beta)$, is the number of sequences γ in $S|_\alpha$ such that $\beta \subseteq \alpha.\gamma$.

prefix	projected (suffix) database	sequential patterns
$\langle a\rangle$	$\langle(abc)(ac)d(cf)\rangle$, $\langle(\_d)c(bc)(ae)\rangle$, $\langle(\_b)(df)cb\rangle$, $\langle(\_f)cbc\rangle$	$\langle a\rangle$, $\langle aa\rangle$, $\langle ab\rangle$, $\langle a(bc)\rangle$, $\langle a(bc)a\rangle$, $\langle aba\rangle$, $\langle abc\rangle$, $\langle(ab)\rangle$, $\langle(ab)c\rangle$, $\langle(ab)d\rangle$, $\langle(ab)f\rangle$, $\langle(ab)dc\rangle$, $\langle ac\rangle$, $\langle aca\rangle$, $\langle acb\rangle$, $\langle acc\rangle$, $\langle ad\rangle$, $\langle adc\rangle$, $\langle af\rangle$
$\langle b\rangle$	$\langle(\_c)(ac)d(cf)\rangle$, $\langle(\_c)(ae)\rangle$, $\langle(df)cb\rangle$, $\langle c\rangle$	$\langle b\rangle$, $\langle ba\rangle$, $\langle bc\rangle$, $\langle(bc)\rangle$, $\langle(bc)a\rangle$, $\langle bd\rangle$, $\langle bdc\rangle$, $\langle bf\rangle$
$\langle c\rangle$	$\langle(ac)d(cf)\rangle$, $\langle(bc)(ae)\rangle$, $\langle b\rangle$, $\langle bc\rangle$	$\langle c\rangle$, $\langle ca\rangle$, $\langle cb\rangle$, $\langle cc\rangle$
$\langle d\rangle$	$\langle(cf)\rangle$, $\langle c(bc)(ae)\rangle$, $\langle(\_f)cb\rangle$	$\langle d\rangle$, $\langle db\rangle$, $\langle dc\rangle$, $\langle dcb\rangle$
$\langle e\rangle$	$\langle(\_f)(ab)(df)cb\rangle$, $\langle(af)cbc\rangle$	$\langle e\rangle$, $\langle ea\rangle$, $\langle eab\rangle$, $\langle eac\rangle$, $\langle eacb\rangle$, $\langle eb\rangle$, $\langle ebc\rangle$, $\langle ec\rangle$, $\langle ecb\rangle$, $\langle ef\rangle$, $\langle efb\rangle$, $\langle efc\rangle$, $\langle efcb\rangle$.
$\langle f\rangle$	$\langle(ab)(df)cb\rangle$, $\langle cbc\rangle$	$\langle f\rangle$, $\langle fb\rangle$, $\langle fbc\rangle$, $\langle fc\rangle$, $\langle fcb\rangle$

TABLE 2: Projected Databases and Sequential Patterns

Lemma 3.2 (Projected database) Let α and β be two sequential patterns in a sequence database S such that α is a prefix of β.

1. $S|_\alpha = (S|_\alpha)_\beta$,
2. for any sequence γ with prefix α, $support_{S(\gamma)}$= support $S|_\alpha(\gamma)$, and
3. the size of α-projected database cannot exceed that of S.

Proof sketch: The first part of the lemma follows the fact that, for a sequence γ, the suffix of γ with regards to β, γ/β, equals to the sequence resulted from first doing projection of γ with regards to α, i.e., γ/α, and then doing projection γ/α with regards to β. That is γ/β = (γ/α)/β. The second part of the states that to collect support count of a sequence γ, only the sequences in the database sharing the same prefix should be considered. Furthermore, only those suffixes with the prefix being a super-sequence of γ should be counted. The third part of the on the size of a projected database. Obviously, the α-projected database can have the same number of sequences as S only if α appears in every sequence in S. Otherwise, only those sequences in S which are super-sequences of α appear in the α-projected database. So, the α-projected database cannot contain more sequences than S. For every sequence γ in S such that γ is a super-sequence of α, γ appears in the α-projected database in whole only if α is a prefix of γ. Otherwise, only a subsequence of γ appears in the α-projected database. Therefore, the size of α-projected database cannot exceed that of S.

Example 4 (**PrefixSpan**) for the same sequence database S in Figure 1 with min_support = 2, sequential patterns in S can be mined by a prefix-projection method in the following steps:

1. Find length-1 sequential patterns. Scan S once to find all the frequent items in sequences. Each of these frequent items is a length-1 sequential pattern. They are (a) : 4, (b) : 4, (c) : 4, (d) : 3, (e) : 3, and (f) : 3, where the notation "(pattern) : count" represents the pattern and its associated support count.
2. Divide search space. The complete set of sequential patterns can be partitioned into the following six subsets according to the six prefixes: 1) the ones with prefix (a), 2) the ones with prefix (b) . . . and 3) the ones with prefix (f).
3. Find subsets of sequential patterns. The subsets of sequential patterns can be mined by constructing the corresponding set of projected databases and mining each recursively. The projected databases as well as sequential patterns found in them are listed in Table 2, while the mining process is explained as follows:

a. Find sequential patterns with prefix (a). Only the sequences containing (a) should be collected. Moreover, in a sequence containing (a), only the subsequence prefixed with the first occurrence of (a) should be considered. For example, in sequence ((ef)(ab)(df)cb), only the subsequence ((_b)(df)cb) should be considered for mining sequential patterns prefixed with (a). Notice that (_ b) means that the last element in the prefix, which is a, together with b, form one element.

The sequences in S containing (a) are projected with regards to (a) to form the (a)-projected database, which consists of four suffix sequences: ((abc)(ac)d(cf)), ((_d)c(bc)(ae)), ((_ b)(df)cb), and ((_f)cbc). By scanning the (a)-projected database once, its locally frequent items are a : 2, b : 4, b : 2, c : 4, d : 2, and f : 2. Thus, all the length-2 sequential patterns prefixed with (a) are found, and they are: (aa) : 2, (ab) : 4, ((ab)) : 2, (ac) : 4, (ad) : 2, and (af) : 2. Recursively, all sequential patterns with prefix (a) can be partitioned into six subsets:

1) Those prefixed with (aa), 2) those with (ab) . . . and, finally, 3) those with (af). These subsets can be mined by constructing respective projected databases and mining each recursively as follows:

i. The (aa)-projected database consists of two nonempty (suffix) sub sequences prefixed with (aa) : f{((_bc)(ac)d(cf)), {((_e))}.Since there is no hope to generate any frequent subsequence from this projected database, the processing of the (aa)-projected database terminates.

ii. The (ab)-projected database consists of three suffix sequences: ((_ c)(ac)d(cf)), ((_c)a), and (c). Recursively mining the (ab)-projected database returns four sequential patterns: ((_c)),

((_c)a), (a), and (c) (i.e., (a(bc)), (a(bc)a), (aba), and (abc).) They form the complete set of sequential patterns prefixed with (ab).

.

iii. The ((ab))-projected database contains only two sequences: ((_c)(ac)d(cf)) and ((df)cb), which leads to the finding of the following sequential patterns prefixed with ((ab)): (c), (d), (f), and (dc).

iv. The (ac), (ad), and (af)-projected databases can be constructed and recursively mined similarly. The sequential patterns found are shown in Table 2.

b. Find sequential patterns with prefix (b), (c), (d), (e), and (f), respectively. This can be done by constructing the (b), (c), (d), (e), and (f)-projected databases and mining them, respectively. The projected databases as well as the sequential patterns found are shown in Tables 2.

4. The set of sequential patterns is the collection of patterns found in the above recursive mining process. One can verify that it returns exactly the same set of sequential patterns as what GSP and FreeSpan do. Based on the above discussion, the algorithm of PrefixSpan is presented as follows:

Algorithm 1 (PrefixSpan) Prefix-projected sequential pattern mining.
Input: A sequence database S, and the minimum support
Threshold min_support.
Output: The complete set of sequential patterns.
Method: Call PrefixSpan (<>, 0, S).
Subroutine: PrefixSpan (α, l, $S|_\alpha$)
The parameters are 1) α is a sequential pattern; 2) l is the length of α; and 3) $S|_\alpha$ is the α-projected database if $\alpha \neq$ ( ), otherwise, it is the sequence database S.

Method:
1. Scan $S|_\alpha$ once, find each frequent item, b, such that
(a) b can be assembled to the last element of α to form a sequential pattern; or
(b) (b) can be appended to α to form a sequential pattern.
2. For each frequent item b, append it to α to form a sequential pattern $\alpha`$ and output $\alpha`$.
3. For each $\alpha`$, construct $\alpha`$- projected database $S|_{\alpha`}$, and call PrefixSpan ($\alpha`$, l+1, $S|_{\alpha`}$).

Analysis: The correctness and completeness of the algorithm can be justified based on Lemma 3.1 and Lemma 3.2, as shown in Theorem 3.1, later. Here, we analyze the efficiency of the algorithm as follows:

- No candidate sequence needs to be generated by PrefixSpan. Unlike a priori-like algorithms, PrefixSpan only grows longer sequential patterns from the shorter frequent ones. It neither generates nor tests any candidate sequence nonexistent in a projected database. Comparing with GSP, which generates and tests a substantial number of candidate sequences, PrefixSpan searches a much smaller space.
- Projected databases keep shrinking. As indicated in Lemma 3.2, a projected database is smaller than the original one because only the suffix subsequences of a frequent prefix are projected into a projected database. In practice, the shrinking factors can be significant because 1) usually, only a small set of sequential patterns grow quite long in a sequence database and, thus, the number of sequences in a projected database usually reduces substantially when prefix grows; and 2) projection only takes the suffix portion with respect to a prefix. Notice that FreeSpan also employs the idea of projected databases. However, the projection there often takes the whole string (not just suffix) and, thus, the shrinking factor is less than that of PrefixSpan.
- The major cost of PrefixSpan is the construction of projected databases. In the worst case,

PrefixSpan constructs a projected database for every sequential pattern. If there exist a good number of sequential patterns, the cost is nontrivial.

Theorem 3.1 (PrefixSpan) A sequence α is a sequential pattern if and only if PrefixSpan says so.

Proof sketch (Direction if). A length-l sequence α ($l \geq 1$) is identified as a sequential pattern by PrefixSpan if and only if α is a sequential pattern in the projected database of its length-(l - 1) prefix α^-. If l = 1, the length-0 prefix of α is $\alpha^- = (\ )$ and the projected database is S itself. So, α is a sequential pattern in S. If l > 1, according to Lemma 3.2, $S|_{\alpha^-}$ is exactly the α^-- projected database, and $support_s(\alpha) = support_{S|\alpha^-}(\alpha)$. Therefore, if α is a sequential pattern in $S|_{\alpha^-}$, it is also a sequential pattern in S. By this, we show that a sequence α is a sequential pattern if PrefixSpan says so.

3.3 Pseudoprojection

The above analysis shows that the major cost of PrefixSpan is database projection, i.e., forming projected databases recursively. Usually, a large number of projected databases will be generated in sequential pattern mining. If the number and/or the size of projected databases can be reduced, the performance of sequential pattern mining can be further improved. One technique which may reduce the number and size of projected databases is Pseudoprojection. The idea is outlined as follows: Instead of performing physical projection, one can register the index (or identifier) of the corresponding sequence and the starting position of the projected suffix in the sequence. Then, a physical projection of a sequence is replaced by registering a sequence identifier and the projected position index point. Pseudoprojection reduces the cost of projection substantially when the projected database can fit in main memory.

Example 5 (Pseudoprojection): For the same sequence database S in Table 1 with min_support = 2, sequential patterns in S can be mined by Pseudoprojection method as follows:

Suppose the sequence database S in Table 1 can be held in main memory. Instead of constructing the (a)-projected database, one can represent the projected suffix sequences using pointer (sequence_id) and offset(s). For example, the projection of sequence s1 = (a(abc)d(ae)(cf)) with regard to the (a)-projection s1 which could be the string_id s_1 and 2) the offset(s), which should be a single integer, such as 2, if there is a single projection point; and a set of integers, such as {2,3, 6}, if there are multiple projection points. Each offset indicates at which position the projection starts in the sequence. The projected databases for prefixes (a), (b), (c), (d), (f), and (aa) are shown in Table 3, where $ indicates the prefix has an occurrence in the current sequence but its projected suffix is empty, whereas ¢ indicates that there is no occurrence of the prefix in the corresponding sequence. From Table 3, we can see that the pseudo projected database usually takes much less space than its corresponding physically projected one.

Pseudoprojection avoids physically copying suffixes. Thus, it is efficient in terms of both running time and space. However, it may not be efficient if the Pseudoprojection is used for disk-based accessing since random access disk space is costly. Based on this observation, the suggested approach is that if the original sequence database or the projected databases is too big to fit into main memory, the physical projection should be applied; however, the execution should be swapped to Pseudoprojection once the projected databases can fit in main memory.

Sequence_id	Sequence	$\langle a \rangle$	$\langle b \rangle$	$\langle c \rangle$	$\langle d \rangle$	$\langle f \rangle$	$\langle aa \rangle$	...
10	$\langle a(abc)(ac)d(cf) \rangle$	2, 3, 6	4	5, 7	8	\$	3, 6	...
20	$\langle (ad)c(bc)(ae) \rangle$	2	5	4, 6	3	$\emptyset$	7	...
30	$\langle (ef)(ab)(df)cb \rangle$	4	5	8	6	3, 7	$\emptyset$	...
40	$\langle eg(af)cbc \rangle$	4	6	6	$\emptyset$	5	$\emptyset$	...

TABLE 3: A Sequence Database and Some of Its Pseudoprojected Databases

4. CONCLUSION

In this paper an efficient pattern-growth method, PrefixSpan, is proposed and studied. PrefixSpan recursively projects a sequence database into a set of smaller projected sequence databases and grows sequential patterns in each projected database by exploring only locally frequent fragments. It mines the complete set of sequential patterns and substantially reduces the efforts of candidate subsequence generation. Since PrefixSpan explores ordered growth by prefix-ordered expansion, it results in less "growth points" and reduced projected databases in comparison with our previously proposed pattern-growth algorithm, FreeSpan. Furthermore, a Pseudoprojection technique is proposed for PrefixSpan to reduce the number of physical projected databases to be generated. A comprehensive performance study shows that PrefixSpan outperforms the priori-based GSP algorithm, FreeSpan, and SPADE in most cases, and PrefixSpan integrated with Pseudoprojection is the fastest among all the tested algorithms.
This mining methodology can be extended to mining sequential patterns with user-specified constraints. The high promise of the pattern-growth approach may lead to its further extension toward efficient mining of other kinds of frequent patterns, such as frequent substructures. Much work is needed to explore new applications of frequent pattern mining. For example, bioinformatics has raised a lot of challenging problems, and we believe frequent pattern mining may contribute a good deal to it with further research efforts.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. 1994 Int'l Conf. Very Large Data Bases (VLDB '94), pp. 487-499, Sept. 1994..

[2] R. Agrawal and R. Srikant, "Mining Sequential Patterns," Proc. 1995 Int'l Conf. Data Eng. (ICDE '95), pp. 3-14, Mar. 1995.

[3] R.J. Bayardo, "Efficiently Mining Long Patterns from Databases," Proc. 1998 ACM-SIGMOD Int'l Conf. Management of Data (SIGMOD '98), pp. 85-93, June 1998.

[4] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu, "FreeSpan: Frequent Patterrn- Projected Sequential Pat t e r n Mining," Proc. 2000 ACM SIGKDD Int'l Conf. Knowledge Discovery in Databases (KDD '00), pp. 355-359, Aug. 2000.

[5] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," Proc. 2000 ACM-SIGMOD Int'l Conf. Management of Data (SIGMOD '00), pp. 1-12, May 2000.

[6] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," Proc. Fifth Int'l Conf. Extending Database Technology (EDBT '96), pp. 3-17, Mar. 1996

[7] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," Proc. Fifth Int'l Conf. Extending Database Technology (EDBT '96), pp. 3-17, Mar. 1996.

[8] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu, "FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining," Proc. 2000 ACM SIGKDD Int'l Conf. Knowledge Discovery in Databases (KDD '00), pp. 355-359, Aug. 2000.

[9] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," Proc. 2000 ACM-SIGMOD Int'l Conf. Management of Data (SIGMOD '00), pp. 1-12, May 2000.

[10] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth," Proc. 2001 Int'l Conf. Data Eng. (ICDE '01), pp. 215-224, Apr. 2001.