PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth *

Jian Pei Jiawei Han Behzad Mortazavi-Asl Helen Pinto
Intelligent Database Systems Research Lab. School of Computing Science, Simon Fraser University
Burnaby, B.C., Canada V5A 1S6 E-mail: {peijian, han, mortazav, hlpinto}@cs.sfu.ca

Qiming Chen Umeshwar Dayal Mei-Chun Hsu
Hewlett-Packard Labs. Palo Alto, California 94303-0969 U.S.A.
E-mail: {qchen, dayal, mchsu}@hpl.hp.com

Abstract

Sequential pattern mining is an important data mining problem with broad applications. It is challenging since one may need to examine a combinatorially explosive number of possible subsequence patterns. Most of the previously developed sequential pattern mining methods follow the methodology of Apriori *which may substantially reduce the number of combinations to be examined. However,* Apriori *still encounters problems when a sequence database is large and/or when sequential patterns to be mined are numerous and/or long.*

In this paper, we propose a novel sequential pattern mining method, called PrefixSpan *(i.e.,* **Prefix***-projected* **S***equential* **pa***ttern* **n** *mining), which explores prefix-projection in sequential pattern mining.* PrefixSpan *mines the complete set of patterns but greatly reduces the efforts of candidate subsequence generation. Moreover, prefix-projection substantially reduces the size of projected databases and leads to efficient processing. Our performance study shows that* PrefixSpan *outperforms both the* Apriori*-based* GSP *algorithm and another recently proposed method,* FreeSpan*, in mining large sequence databases.*

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 analyses of customer purchase behavior, Web access patterns, scientific experiments, disease treatments, natural disasters, DNA sequences, and so on.

*The work was supported in part by the Natural Sciences and Engineering Research Council of Canada (grant NSERC-A3723), the Networks of Centres of Excellence of Canada (grant NCE/IRIS-3), and the Hewlett-Packard Lab, U.S.A.

The sequential pattern mining problem was first introduced by Agrawal and Srikant in [2]: *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 of the frequent subsequences, i.e., the subsequences whose occurrence frequency in the set of sequences is no less than min_support.*

Many studies have contributed to the efficient mining of sequential patterns or other frequent patterns in time-related data, e.g., [2, 11, 9, 10, 3, 8, 5, 4]. Almost all of the previously proposed methods for mining sequential patterns and other time-related frequent patterns are Apriori-like, i.e., based on the Apriori property proposed in association mining [1], which states the fact that *any super-pattern of a nonfrequent pattern cannot be frequent*.

Based on this heuristic, a typical Apriori-like method such as GSP [11] adopts a multiple-pass, candidate-generation-and-test approach in sequential pattern mining. This is outlined as follows. The first scan finds all of the frequent items which 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*. Each candidate sequence contains one more item than a seed sequential pattern, where each element in the pattern may contain one 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 of the candidates whose support in the database is no less than min_support form the set of the newly found sequential patterns. This set then becomes the seed set for the next pass. The algorithm terminates when no new sequential pattern is found in a pass, or no candidate sequence can be generated.

Similar to the analysis of Apriori frequent pattern min-

ing method in [7], one can observe that the Apriori-like sequential pattern mining method, though reduces search space, bears three nontrivial, inherent costs which are independent of detailed implementation techniques.

- **Potentially huge set of candidate sequences**. Since the set of candidate sequences includes all the possible permutations of the elements and repetition of items in a sequence, the Apriori-based method may generate a really large set of candidate sequences even for a moderate seed set. For example, if there are 1000 frequent sequences of length-1, such as $\langle a_1 \rangle$, $\langle a_2 \rangle$, ..., $\langle a_{1000} \rangle$, an Apriori-like algorithm will generate $1000 \times 1000 + \frac{1000 \times 999}{2} = 1,499,500$ candidate sequences, where the first term is derived from the set $\langle a_1 a_1 \rangle$, $\langle a_1 a_2 \rangle$, ..., $\langle a_1 a_{1000} \rangle$, $\langle a_2 a_1 \rangle$, $\langle a_2 a_2 \rangle$, ..., $\langle a_{1000} a_{1000} \rangle$, and the second term is derived from the set $\langle (a_1 a_2) \rangle$, $\langle (a_1 a_3) \rangle$, ..., $\langle (a_{999} a_{1000}) \rangle$.
- **Multiple scans of databases.** Since the length of each candidate sequence grows by one at each database scan, to find a sequential pattern $\{(abc)(abc)(abc)(abc)(abc)\}$, the Apriori-based method must scan the database at least 15 times.
- **Difficulties at mining long sequential patterns.** A long sequential pattern must grow from a combination of short ones, but the number of such candidate sequences is exponential to the length of the sequential patterns to be mined. For example, suppose there is only a single sequence of length 100, $\langle a_1 a_2 \ldots a_{100} \rangle$, in the database, and the min_support threshold is 1 (i.e., every occurring pattern is frequent), to (re-)derive this length-100 sequential pattern, the Apriori-based method has to generate 100 length-1 candidate sequences, $100 \times 100 + \frac{100 \times 99}{2} = 14,950$ length-2 candidate sequences, $\binom{100}{3} = 161,700$ length-3 candidate sequences[1], ... Obviously, the total number of candidate sequences to be generated is greater than $\Sigma_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^{30}$.

In many applications, it is not unusual that one may encounter a large number of sequential patterns and long sequences, such as in DNA analysis or stock sequence analysis. Therefore, it is important to re-examine the sequential pattern mining problem to explore more efficient and scalable methods.

Based on our analysis, both the thrust and the bottleneck of an Apriori-based sequential pattern mining method come from its step-wise candidate sequence generation and test. Can we develop a method which may absorb the spirit of Apriori but avoid or substantially reduce the expensive candidate generation and test?

[1]Notice that Apriori does cut a substantial amount of search space. Otherwise, the number of length-3 candidate sequences would have been $100 \times 100 \times 100 + 100 \times 100 \times 99 + \frac{100 \times 99 \times 98}{3 \times 2} = 2,151,700$.

With this motivation, we first examined whether the FP-tree structure [7], recently proposed in frequent pattern mining, can be used for mining sequential patterns. The FP-tree structure explores maximal sharing of common prefix paths in the tree construction by reordering the items in transactions. However, the items (or subsequences) containing different orderings cannot be reordered or collapsed in sequential pattern mining. Thus the FP-tree structures so generated will be huge and cannot benefit mining.

As a subsequent study, we developed a sequential mining method [6], called FreeSpan (i.e., **Fre**qu**e**nt pattern-projected **S**equential **pa**ttern mining). Its general idea is to use frequent items to recursively project sequence databases into a set of smaller projected databases and grow subsequence fragments in each projected database. This process partitions both the data and the set of frequent patterns to be tested, and confines each test being conducted to the corresponding smaller projected database. Our performance study shows that FreeSpan mines the complete set of patterns and is efficient and runs considerably faster than the Apriori-based GSP algorithm. However, since a subsequence may be generated by any substring combination in a sequence, projection in FreeSpan has to keep the whole sequence in the original database without length reduction. Moreover, since the growth of a subsequence is explored at any split point in a candidate sequence, it is costly.

In this study, we develop a novel sequential pattern mining method, called PrefixSpan (i.e., **Prefix**-projected **S**equential **pa**ttern mining). Its general idea is to examine only the prefix subsequences and project only their corresponding postfix subsequences into projected databases. In each projected database, sequential patterns are grown by exploring only local frequent patterns. To further improve mining efficiency, two kinds of database projections are explored: *level-by-level projection* and *bi-level projection*. Moreover, a main-memory-based pseudo-projection technique is developed for saving the cost of projection and speeding up processing when the projected (sub)-database and its associated pseudo-projection processing structure can fit in main memory. Our performance study shows that bi-level projection has better performance when the database is large, and pseudo-projection speeds up the processing substantially when the projected databases can fit in memory. PrefixSpan mines the complete set of patterns and is efficient and runs considerably faster than both Apriori-based GSP algorithm and FreeSpan.

The remaining of the paper is organized as follows. In Section 2, we define the sequential pattern mining problem and illustrate the ideas of our previously developed pattern growth method FreeSpan. The PrefixSpan method is developed in Section 3. The experimental and performance results are presented in Section 4. In Section 5, we discuss its relationships with related works. We summarize our study and point out some research issues in Section 6.

2 Problem Definition and FreeSpan

In this section, we first define the problem of sequential pattern mining, and then illustrate our recently proposed method, FreeSpan, using an example.

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of all **items**. An **itemset** is a subset of items. A **sequence** is an ordered list of itemsets. A sequence s is denoted by $\langle s_1 s_2 \cdots s_l \rangle$, where s_j is an itemset, i.e., $s_j \subseteq I$ for $1 \leq j \leq l$. s_j is also called an **element** of the sequence, and denoted as $(x_1 x_2 \cdots x_m)$, where x_k is an item, i.e., $x_k \in I$ for $1 \leq k \leq m$. For brevity, the brackets are omitted if an element has only one item. That is, 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 $\alpha = \langle a_1 a_2 \cdots a_n \rangle$ is called a **subsequence** of another sequence $\beta = \langle b_1 b_2 \cdots b_m \rangle$ and β a **super sequence** of α, denoted as $\alpha \sqsubseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \cdots < j_n \leq m$ such that $a_1 \subseteq b_{j_1}$, $a_2 \subseteq b_{j_2}, \ldots, a_n \subseteq b_{j_n}$.

A **sequence database** S is a set of tuples $\langle sid, s \rangle$, where sid is a **sequence_id** and s is a sequence. A tuple $\langle sid, s \rangle$ is said to *contain* a sequence α, if α is a subsequence of s, i.e., $\alpha \sqsubseteq s$. The support of a sequence α in a sequence database S is the number of tuples in the database containing α, i.e., $support_S(\alpha) = |\{\langle sid, s \rangle | (\langle sid, s \rangle \in S) \wedge (\alpha \sqsubseteq s)\}|$. It can be denoted as $support(\alpha)$ if the sequence database is clear from the context. Given a positive integer ξ as the **support threshold**, a sequence α is called a (frequent) **sequential pattern** in sequence database S if the sequence is contained by at least ξ tuples in the database, i.e., $support_S(\alpha) \geq \xi$. A sequential pattern with length l is called an l-**pattern**.

Example 1 (Running example) Let our running database be *sequence database* S given in Table 1 and *min_support* = 2. The set of *items* in the database is $\{a, b, c, d, e, f, g\}$.

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

A *sequence* $\langle a(abc)(ac)d(cf) \rangle$ has five *elements*: (a), (abc), (ac), (d) and (cf), where items a and c appear more than once respectively in different elements. It is also a 9-*sequence* since there are 9 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 $\langle a(abc)(ac)d(cf) \rangle$ contributes only one to the *support* of $\langle a \rangle$. Also, sequence $\langle a(bc)df \rangle$ is a *subsequence* of $\langle a(abc)(ac)d(cf) \rangle$. Since both sequences 10 and 30 *contain* subsequence $s = \langle (ab)c \rangle$, s is a *sequential pattern* of length 3 (i.e., 3-*pattern*).

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

In Section 1, we outlined the Apriori-like method GSP [11]. To improve the performance of sequential pattern mining, a FreeSpan algorithm is developed in our recent study [6]. Its major ideas are illustrated in the following example.

Example 2 (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. Frequent items are listed in support descending order (in the form of $item : support$) as below,

$$\text{f\_list} = a : 4, b : 4, c : 4, d : 3, e : 3, f : 3$$

According to f_list, the complete set of sequential patterns in S can be divided into 6 disjoint subsets: (1) the ones containing only item a, (2) the ones containing item b but containing no items after b in f_list, (3) the ones containing item c but no items after c in f_list, and so on, and finally, (6) the ones containing item f.

The subsets of sequential patterns can be mined by constructing *projected databases*. Infrequent items, such as g in this example, are removed from construction of projected databases. The mining process is detailed as follows.

- **Finding sequential patterns containing only item** a**.** By scanning sequence database once, the only two sequential patterns containing only item a, $\langle a \rangle$ and $\langle aa \rangle$, are found.
- **Finding sequential patterns containing item** b **but no item after** b **in** f_list**.** This can be achieved by constructing the $\{b\}$-*projected database*. For a sequence α in S containing item b, a subsequence α' is derived by removing from α all items after b in f_list. α' is inserted into $\{b\}$-projected database. Thus, $\{b\}$-projected database contains four sequences: $\langle a(ab)a \rangle$, $\langle aba \rangle$, $\langle (ab)b \rangle$ and $\langle ab \rangle$. By scanning the projected database once more, all sequential patterns containing item b but no item after b in f_list are found. They are $\langle b \rangle$, $\langle ab \rangle$, $\langle ba \rangle$, $\langle (ab) \rangle$.
- **Finding other subsets of sequential patterns.** Other subsets of sequential patterns can be found similarly, by constructing corresponding projected databases and mining them recursively.

Note that $\{b\}$-, $\{c\}$-, ..., $\{f\}$-projected databases are constructed simultaneously during one scan of the original

sequence database. All sequential patterns containing only item a are also found in this pass.

This process is performed recursively on projected-databases. Since FreeSpan projects a large sequence database recursively into a set of small projected sequence databases based on the currently mined frequent sets, the subsequent mining is confined to each projected database relevant to a smaller set of candidates. Thus, FreeSpan is more efficient than GSP.

The major cost of FreeSpan is to deal with 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 is the same as the original sequence database, except for the removal of infrequent item g. 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 PrefixSpan: Mining Sequential Patterns by Prefix Projections

In this section, we introduce a new pattern-growth method for mining sequential patterns, called PrefixSpan. Its major idea is that, instead of projecting sequence databases by considering all the possible occurrences of frequent subsequences, the projection is based only on frequent prefixes because any frequent subsequence can always be found by growing a frequent prefix. In Section 3.1, the PrefixSpan idea and the mining process are illustrated with an example. The algorithm PrefixSpan is then presented and justified in Section 3.2. To further improve its efficiency, two optimizations are proposed in Section 3.3 and Section 3.4, respectively.

3.1 Mining sequential patterns by prefix projections: An example

Since items within an element of a sequence can be listed in any order, without loss of generality, we assume they are listed in alphabetical order. For example, the sequence in S with Sequence_id 10 in our running example is listed as $\langle a(abc)(ac)d(cf)\rangle$ in stead of $\langle a(bac)(ca)d(fc)\rangle$. With such a convention, the expression of a sequence is unique.

Definition 1 (Prefix, projection, and postfix) Suppose all the items in an element are listed alphabetically. Given a sequence $\alpha = \langle e_1 e_2 \cdots e_n\rangle$, a sequence $\beta = \langle e'_1, e'_2 \cdots e'_m\rangle$ $(m \leq n)$ 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 items in $(e_m - e'_m)$ are alphabetically after those in e'_m.

Given sequences α and β such that β is a subsequence of α, i.e., $\beta \sqsubseteq \alpha$. A subsequence α' of sequence α (i.e., $\alpha' \sqsubseteq \alpha$) is called **a projection** of α w.r.t. prefix β if and only if (1) α' has prefix β and (2) there exists no proper super-sequence α'' of α' (i.e., $\alpha' \sqsubseteq \alpha''$ but $\alpha' \neq \alpha''$) such that α'' is a subsequence of α and also has prefix β.

Let $\alpha' = \langle e_1 e_2 \cdots e_n\rangle$ be the projection of α w.r.t. prefix $\beta = \langle e_1 e_2 \cdots e_{m-1} e'_m\rangle$ $(m \leq n)$. Sequence $\gamma = \langle e''_m e_{m+1} \cdots e_n\rangle$ is called the **postfix** of α w.r.t. prefix β, denoted as $\gamma = \alpha/\beta$, where $e''_m = (e_m - e'_m)$.[2] We also denote $\alpha = \beta \cdot \gamma$.

If β is not a subsequence of α, both projection and postfix of α w.r.t. β are empty.

For example, $\langle a\rangle$, $\langle aa\rangle$, $\langle a(ab)\rangle$ and $\langle a(abc)\rangle$ are *prefixes* of sequence $\langle a(abc)(ac)d(cf)\rangle$, but neither $\langle ab\rangle$ nor $\langle a(bc)\rangle$ is considered as a prefix. $\langle (abc)(ac)d(cf)\rangle$ is the *postfix* of the same sequence w.r.t. prefix $\langle a\rangle$, $\langle (\_bc)(ac)d(cf)\rangle$ is the *postfix* w.r.t. prefix $\langle aa\rangle$, and $\langle (\_c)(ac)d(cf)\rangle$ is the *postfix* w.r.t. prefix $\langle ab\rangle$.

Example 3 (PrefixSpan) For the same sequence database S in Table 1 with $min\_sup = 2$, sequential patterns in S can be mined by a prefix-projection method in the following steps.

Step 1: Find length-1 sequential patterns. Scan S once to find all frequent items in sequences. Each of these frequent items is a length-1 sequential pattern. They are $\langle a\rangle : 4$, $\langle b\rangle : 4$, $\langle c\rangle : 4$, $\langle d\rangle : 3$, $\langle e\rangle : 3$, and $\langle f\rangle : 3$, where $\langle pattern\rangle : count$ represents the pattern and its associated support count.

Step 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 having prefix $\langle a\rangle$; ...; and (6) the ones having prefix $\langle f\rangle$.

Step 3: Find subsets of sequential patterns. The subsets of sequential patterns can be mined by constructing corresponding *projected databases* and mine 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.

First, let us find sequential patterns having prefix $\langle a\rangle$. Only the sequences containing $\langle a\rangle$ should be collected. Moreover, in a sequence containing $\langle a\rangle$, only the subsequence prefixed with the first occurrence of $\langle a\rangle$ should be considered. For example, in sequence $\langle (ef)(ab)(df)cb\rangle$, only the subsequence $\langle (\_b)(df)cb\rangle$ should be considered for mining sequential patterns having prefix $\langle a\rangle$. Notice that $(\_b)$ means that the last element in the prefix, which is a, together with b, form one element. As another example, only the subsequence $\langle (abc)(ac)d(cf)\rangle$ of sequence $\langle a(abc)(ac)d(cf)\rangle$ should be considered.

Sequences in S containing $\langle a\rangle$ are projected w.r.t. $\langle a\rangle$ to form the $\langle a\rangle$*-projected database*, which consists of four

[2] If e''_m is not empty, the postfix is also denoted as $\langle (\_$ items in $e''_m) e_{m+1} \cdots e_n\rangle$.

Prefix	Projected (postfix) 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

postfix sequences: $\langle (abc)(ac)d(cf) \rangle$, $\langle (\_d)c(bc)(ae) \rangle$, $\langle (\_b)(df)cb \rangle$ and $\langle (\_f)cbc \rangle$. By scanning $\langle a \rangle$-projected database once, all the length-2 sequential patterns having prefix $\langle a \rangle$ can be found. They are: $\langle aa \rangle : 2$, $\langle ab \rangle : 4$, $\langle (ab) \rangle : 2$, $\langle ac \rangle : 4$, $\langle ad \rangle : 2$, and $\langle af \rangle : 2$.

Recursively, all sequential having patterns prefix $\langle a \rangle$ can be partitioned into 6 subsets: (1) those having prefix $\langle aa \rangle$, (2) those having prefix $\langle ab \rangle$, ..., and finally, (6) those having prefix $\langle af \rangle$. These subsets can be mined by constructing respective projected databases and mining each recursively as follows.

The $\langle aa \rangle$-projected database consists of only one non-empty (postfix) subsequences having prefix $\langle aa \rangle$: $\langle (\_bc)(ac)d(cf) \rangle$. Since there is no hope to generate any frequent subsequence from a single sequence, the processing of $\langle aa \rangle$-projected database terminates.

The $\langle ab \rangle$-projected database consists of three postfix sequences: $\langle (\_c)(ac)d(cf) \rangle$, $\langle (\_c)a \rangle$, and $\langle c \rangle$. Recursively mining $\langle ab \rangle$-projected database returns four sequential patterns: $\langle (\_c) \rangle$, $\langle (\_c)a \rangle$, $\langle a \rangle$, and $\langle c \rangle$ (i.e., $\langle a(bc) \rangle$, $\langle a(bc)a \rangle$, $\langle aba \rangle$, and $\langle abc \rangle$.)

$\langle (ab) \rangle$ projected database contains only two sequences: $\langle (\_c)(ac)d(cf) \rangle$ and $\langle (df)cb \rangle$, which leads to the finding of the following sequential patterns having prefix $\langle (ab) \rangle$: $\langle c \rangle$, $\langle d \rangle$, $\langle f \rangle$, and $\langle dc \rangle$.

The $\langle ac \rangle$-, $\langle ad \rangle$- and $\langle af \rangle$- projected databases can be constructed and recursively mined similarly. The sequential patterns found are shown in Table 2.

Similarly, we can find sequential patterns having prefix $\langle b \rangle$, $\langle c \rangle$, $\langle d \rangle$, $\langle e \rangle$ and $\langle f \rangle$, respectively, by constructing $\langle b \rangle$-, $\langle c \rangle$- $\langle d \rangle$-, $\langle e \rangle$- and $\langle f \rangle$-projected databases and mining them respectively. The projected databases as well as the sequential patterns found are shown in Table 2.

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.

3.2 PrefixSpan: Algorithm and correctness

Now, let us justify the correctness and completeness of the mining process in Section 3.1.

Based on the concept of prefix, we have the following lemma on the completeness of partitioning the sequential pattern mining problem.

Lemma 3.1 (Problem partitioning) *Let α be a length-l ($l \geq 0$) sequential pattern and $\{\beta_1, \beta_2, \ldots, \beta_m\}$ be the set of all length-$(l+1)$ sequential patterns having prefix α. The complete set of sequential patterns having prefix α, except for α itself, can be divided into m disjoint subsets. The j^{th} subset ($1 \leq j \leq m$) is the set of sequential patterns having prefix β_j. Here, we regard $\emptyset$ as a default sequential pattern for every sequence database.*

Based on Lemma 3.1, PrefixSpan partitions the problem 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, PrefixSpan constructs the corresponding projected databases.

Definition 2 (Projected database) Let α be a sequential pattern in sequence database S. The **α-projected database**, denoted as $S|_\alpha$, is the collection of postfixes of sequences in S w.r.t. prefix α.

To collect counts in projected databases, we have the following definition.

Definition 3 (Support count in projected database) Let α be a sequential pattern in sequence database S, and β be a sequence having 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 $\beta \sqsubseteq \alpha \cdot \gamma$.

Please note that, in general, $support_{S|_\alpha}(\beta) \leq support_{S|_\alpha}(\beta/\alpha)$. For example, $support_S(\langle (ad) \rangle) = 1$ holds in our running example. However, $\langle (ad) \rangle / \langle a \rangle = \langle d \rangle$ and $support_{S|_{\langle a \rangle}}(\langle d \rangle) = 3$.

We have the following lemma on projected databases.

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

1. $S|_\beta = (S|_\alpha)|_\beta$;

2. *for any sequence γ having prefix α, $support_S(\gamma) = support_{S|_\alpha}(\gamma)$; and*
3. *The size of α-projected database cannot exceed that of S.*

Based on the above reasoning, we have the algorithm of PrefixSpan as follows.

Algorithm 1 (PrefixSpan**)**

Input: A sequence database S, and the minimum support threshold min_sup

Output: The complete set of sequential patterns

Method: Call PrefixSpan$(\langle\rangle, 0, S)$.

Subroutine PrefixSpan$(\alpha, l, S|_\alpha)$

Parameters: α: a sequential pattern; l: the length of α; $S|_\alpha$: the α-projected database, if $\alpha \neq \langle\rangle$; otherwise, the sequence database S.

Method:

1. Scan $S|_\alpha$ once, find the set of frequent items b such that
   (a) b can be assembled to the last element of α to form a sequential pattern; or
   (b) $\langle b\rangle$ 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 α', and output α';
3. For each α', construct α'-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 Apriori-like algorithms, PrefixSpan only grows longer sequential patterns from the shorter frequent ones. It does not generate nor test 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 postfix 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 will become quite small when prefix grows; and (2) projection only takes the postfix 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 postfix) and thus the shrinking factor is much 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 are a good number of sequential patterns, the cost is non-trivial. In Section 3.3 and Section 3.4, interesting techniques are developed, which dramatically reduces the number of projected databases.

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

3.3 Scaling up pattern growth by bi-level projection

As analyzed before, the major cost of PrefixSpan is to construct projected databases. If the number and/or the size of projected databases can be reduced, the performance of sequential pattern mining can be improved substantially. In this section, a bi-level projection scheme is proposed to reduce the number and the size of projected databases.

Before introducing the method, let us examine the following example.

Example 4 Let us re-examine mining sequential patterns in sequence database S in Table 1. The first step is the same: Scan S to find the length-1 sequential patterns: $\langle a\rangle$, $\langle b\rangle$, $\langle c\rangle$, $\langle d\rangle$, $\langle e\rangle$ and $\langle f\rangle$.

At the second step, instead of constructing projected databases for each length-1 sequential pattern, we construct a 6×6 lower triangular matrix M, as shown in Table 3.

a	2					
b	(4, 2, 2)	1				
c	(4, 2, 1)	(3, 3, 2)	3			
d	(2, 1, 1)	(2, 2, 0)	(1, 3, 0)	0		
e	(1, 2, 1)	(1, 2, 0)	(1, 2, 0)	(1, 1, 0)	0	
f	(2, 1, 1)	(2, 2, 0)	(1, 2, 1)	(1, 1, 1)	(2, 0, 1)	1
	a	b	c	d	e	f

Table 3. The *S-matrix.*

The matrix M registers the supports of all the length-2 sequences which are assembled using length-1 sequential patterns. A cell at the diagonal line has one counter. For example, $M[c, c] = 3$ indicates sequence $\langle cc\rangle$ appears in three sequences in S. Other cells have three

counters respectively. For example, $M[a, c] = (4, 2, 1)$ means $support_S(\langle ac \rangle) = 4$, $support_S(\langle ca \rangle) = 2$ and $support_S(\langle (ac) \rangle) = 1$. Since the information in cell $M[c, a]$ is symmetric to that in $M[a, c]$, a triangle matrix is sufficient. This matrix is called an *S-matrix*.

By scanning sequence database S the second time, the *S-matrix* can be filled up, as shown in Table 3. All the length-2 sequential patterns can be identified from the matrix immediately.

For each length-2 sequential pattern α, construct α-projected database. For example, $\langle ab \rangle$ is identified as a length-2 sequential pattern by *S-matrix*. The $\langle ab \rangle$-projected database contains three sequences: $\langle (\_c)(ac)(cf) \rangle$, $\langle (\_c)a \rangle$, and $\langle c \rangle$. By scanning it once, three frequent items are found: $\langle a \rangle$, $\langle c \rangle$ and $\langle (\_c) \rangle$. Then, a 3×3 *S-matrix* for $\langle ab \rangle$-projected database is constructed, as shown in Table 4.

a	0		
c	(1, 0, 1)	1	
$(\_c)$	($\emptyset$, 2, $\emptyset$)	($\emptyset$, 1, $\emptyset$)	$\emptyset$
	a	c	$(\_c)$

Table 4. The *S-matrix* in $\langle ab \rangle$-projected database.

Since there is only one cell with support 2, only one length-2 pattern $\langle (\_c)a \rangle$ can be generated and no further projection is needed. Notice that $\emptyset$ means that it is not possible to generate such a pattern. So, we do not need to look at the database.

To mine the complete set of sequential patterns, other projected databases for length-2 sequential patterns should be constructed. It can be checked that such a *bi-level projection* method produces the exactly same set of sequential patterns as shown in Example 3. However, in Example 3, to find the complete set of 53 sequential patterns, 53 projected databases are constructed. In this example, only projected databases for length-2 sequential patterns are needed. In total, only 22 projected databases are constructed by bi-level projection.

Now, let us justify the mining process by bi-level projection.

Definition 4 (*S-matrix*, or sequence-match matrix) Let α be a length-l sequential pattern, and α'_1, α'_2, ..., α'_m be all of length-$(l + 1)$ sequential patterns having prefix α within α-projected database. The *S-matrix* of α-projected database, denoted as $M[\alpha'_i, \alpha'_j]$ $(1 \leq i \leq j \leq m)$, is defined as follows.

1. $M[\alpha'_i, \alpha'_i]$ contains one counter. If the last element of α'_i has only one item x, i.e. $\alpha'_i = \langle \alpha x \rangle$, the counter registers the support of sequence $\langle \alpha'_i x \rangle$ (i.e., $\langle \alpha x x \rangle$) in α-projected database. Otherwise, the counter is set to $\emptyset$;
2. $M[\alpha'_i, \alpha'_j]$ $(1 \leq i < j \leq m)$ is in the form of (A, B, C), where A, B and C are three counters.
   - If the last element in α'_j has only one item x, i.e. $\alpha'_j = \langle \alpha x \rangle$, counter A registers the support of sequence $\langle \alpha'_i x \rangle$ in α-projected database. Otherwise, counter A is set to $\emptyset$;
   - If the last element in α'_i has only one item y, i.e. $\alpha'_i = \langle \alpha y \rangle$, counter B registers the support of sequence $\langle \alpha'_j y \rangle$ in α-projected database. Otherwise, counter B is set to $\emptyset$;
   - If the last elements in α'_i and α'_j have the same number of items, counter C registers the support of sequence α'' in α-projected database, where sequence α'' is α'_i but inserting into the last element of α'_i the item in the last element of α'_j but not in that of α'_i. Otherwise, counter C is set to $\emptyset$.

Lemma 3.3 *Given a length-l sequential pattern α.*

1. *The S-matrix can be filled up after two scans of α-projected database; and*
2. *A length-$(l + 2)$ sequence β having prefix α is a sequential pattern if and only if the S-matrix in α-projected database says so.*

Lemma 3.3 ensures the correctness of bi-level projection. The next question becomes "*do we need to include every item in a postfix in the projected databases?*"

Let us consider the $\langle ac \rangle$-projected database in Example 4. The *S-matrix* in Table 3 tells that $\langle ad \rangle$ is a sequential pattern but $\langle cd \rangle$ is not. According to the Apriori property [1], $\langle acd \rangle$ and any super-sequence of it can never be a sequential pattern. So, based on the matrix, we can exclude item d from $\langle ac \rangle$-projected database. This is the *3-way* Apriori *checking* to prune items for the efficient construction of projected databases. The principle is stated as follows.

Optimization 1 (Item pruning in projected database by 3-way Apriori **checking)** The 3-way Apriori checking should be employed to prune items in the construction of projected databases. To construct the α-projected database, where α is a length-l sequential pattern, let e be the last element of α and α' be the prefix of α such that $\alpha = \alpha' \cdot e$.

- If $\alpha' \cdot (x)$ is not frequent, then item x can be excluded from projection.[3]
- Let e' be formed by substituting any item in e by x. If $\alpha' \cdot e'$ is not frequent, then item x can be excluded

[3]For example, suppose $\langle ac \rangle$ is not frequent. Item c can be excluded from construction of $\langle ab \rangle$-projected database.

from the first element of postfixes if that element is a superset of e.[4]

This optimization applies the 3-way Apriori checking to reduce projected databases further. Only fragments of sequences necessary to grow longer patterns are projected.

3.4 Pseudo-Projection

The major cost of PrefixSpan is projection, i.e., forming projected databases recursively. Here, we propose a *pseudo-projection* technique which reduces the cost of projection substantially when a projected database can be held in main memory.

By examining a set of projected databases, one can observe that postfixes of a sequence often appear repeatedly in recursive projected databases. In Example 3, sequence $\langle a(abc)(ac)d(cf)\rangle$ has postfixes $\langle(abc)(ac)d(cf)\rangle$ and $\langle(\_c)(ac)d(cf)\rangle$ as projections in $\langle a\rangle$- and $\langle ab\rangle$-projected databases, respectively. They are redundant pieces of sequences. If the sequence database/projected database can be held in main memory, such redundancy can be avoided by pseudo-projection.

The method goes as follows. When the database can be held in main memory, instead of constructing a *physical* projection by collecting all the postfixes, one can use pointers referring to the sequences in the database as a *pseudo-projection*. Every projection consists of two pieces of information: *pointer* to the sequence in database and *offset* of the postfix in the sequence.

For example, suppose the sequence database S in Table 1 can be held in main memory. When constructing $\langle a\rangle$-projected database, the projection of sequence $s_1 = \langle a(abc)(ac)d(cf)\rangle$ consists two pieces: a *pointer* to s_1 and *offset* set to 2. The offset indicates that the projection starts from position 2 in the sequence, i.e., postfix $(abc)(ac)d$. Similarly, the projection of s_1 in $\langle ab\rangle$-projected database contains a pointer to s_1 and offset set to 4, indicating the postfix starts from item c in s_1.

Pseudo-projection avoids physically copying postfixes. Thus, it is efficient in terms of both running time and space. However, it is not efficient if the pseudo-projection is used for disk-based accessing since random access disk space is very costly. Based on this observation, PrefixSpan always pursues pseudo-projection once the projected databases can be held in main memory. Our experimental results show that such an integrated solution, disk-based bi-level projection for disk-based processing and pseudo-projection when data can fit into main memory, is always the clear winner in performance.

[4]For example, suppose $\langle a(bd)\rangle$ is not frequent. To construct $\langle a(bc)\rangle$-projected database, sequence $\langle a(bcde)df\rangle$ should be projected to $\langle(\_e)df\rangle$. The first d can be omitted. Please note that we must include the second d. Otherwise, we may fail to find pattern $\langle a(bc)d\rangle$ and those having it as a prefix.

4 Experimental Results and Performance Study

In this section, we report our experimental results on the performance of PrefixSpan in comparison with GSP and FreeSpan. It shows that PrefixSpan outperforms other previously proposed methods and is efficient and scalable for mining sequential patterns in large databases.

All the experiments are performed on a 233MHz Pentium PC machine with 128 megabytes main memory, running Microsoft Windows/NT. All the methods are implemented using Microsoft Visual C++ 6.0.

We compare performance of four methods as follows.

- GSP. The GSP algorithm was implemented as described in [11].
- FreeSpan. As reported in [6], FreeSpan with alternative level projection is more efficient than FreeSpan with level-by-level projection. In this paper, FreeSpan with alternative level projection is used.
- PrefixSpan-1. PrefixSpan-1 is PrefixSpan with level-by-level projection, as described in Section 3.2.
- PrefixSpan-2. PrefixSpan-2 is PrefixSpan with bi-level projection, as described in Section 3.3.

The synthetic datasets we used for our experiments were generated using standard procedure described in [2]. The same data generator has been used in most studies on sequential pattern mining, such as [11, 6]. We refer readers to [2] for more details on the generation of data sets.

We test the four methods on various datasets. The results are consistent. Limited by space, we report here only the results on dataset $C10T8S8I8$. In this data set, the number of items is set to $1,000$, and there are $10,000$ sequences in the data set. The average number of items within elements is set to 8 (denoted as $T8$). The average number of elements in a sequence is set to 8 (denoted as $S8$). There are a good number of long sequential patterns in it at low support thresholds.

The experimental results of scalability with support threshold are shown in Figure 1. When the support threshold is high, there are only a limited number of sequential patterns, and the length of patterns is short, the four methods are close in terms of runtime. However, as the support threshold decreases, the gaps become clear. Both FreeSpan and PrefixSpan win GSP. PrefixSpan methods are more efficient and more scalable than FreeSpan, too. Since the gaps among FreeSpan and GSP are clear, we focus on performance of various PrefixSpan techniques in the remaining of this section.

As shown in Figure 1, the performance curves of PrefixSpan-1 and PrefixSpan-2 are close when sup-

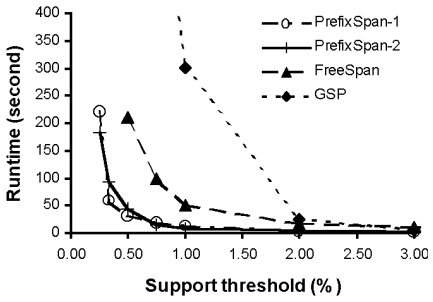

Figure 1. PrefixSpan, FreeSpan and GSP on data set $C10T8S8I8$.

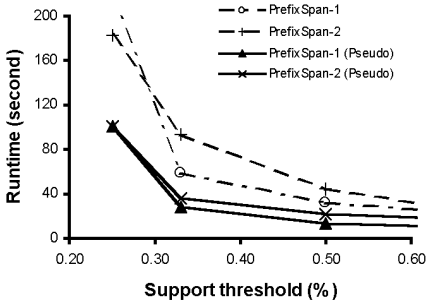

Figure 2. PrefixSpan and PrefixSpan (pseudo-proj) on data set $C10T8S8I8$.

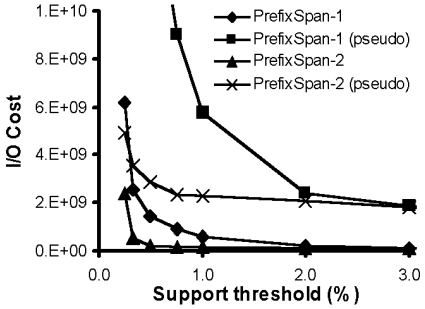

Figure 3. PrefixSpan and PrefixSpan (pseudo-proj) on large data set $C1kT8S8I8$.

port threshold is not low. When the support threshold is low, since there are many sequential patterns, PrefixSpan-1 requires a major effort to generate projected databases. Bi-level projection can leverage the problem efficiently. As can be seen from Figure 2, the increase of runtime for PrefixSpan-2 is moderate even when the support threshold is pretty low.

Figure 2 also shows that using pseudo-projections for the projected databases that can be held in main memory improves efficiency of PrefixSpan further. As can be seen from the figure, the performance of level-by-level and bi-level pseudo-projections are close. Bi-level one catches up with level-by-level one when support threshold is very low. When the saving of less projected databases overcomes the cost of for mining and filling the *S-matrix*, bi-level projection wins. That verifies our analysis of level-by-level and bi-level projection.

Since pseudo-projection improves performance when the projected database can be held in main memory, a related question becomes: "can such a method be extended to disk-based processing?" That is, instead of doing physical projection and saving the projected databases in hard disk, should we make the projected database in the form of disk address and offset? To explore such an alternative, we pursue a simulation test as follows.

Let each sequential read, i.e., reading bytes in a data file from the beginning to the end, cost 1 unit of I/O. Let each random read, i.e., reading data according to its offset in the file, cost 1.5 unit of I/O. Also, suppose a write operation cost 1.5 I/O. Figure 3 shows the I/O costs of PrefixSpan-1 and PrefixSpan-2 as well as of their pseudo-projection variations over data set $C1kT8S8I8$ (where $C1k$ means 1 million sequences in the data set). PrefixSpan-1 and PrefixSpan-2 win their pseudo-projection variations clearly. It can also be observed that bi-level projection wins level-by-level projection as the support threshold becomes low. The huge number of random reads in disk-based pseudo-projections is the performance killer when the database is too big to fit into main memory.

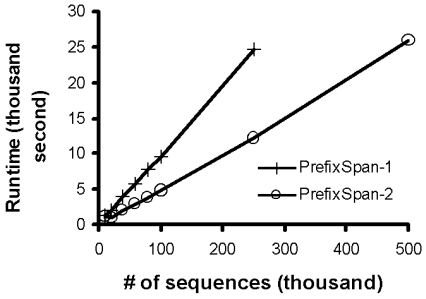

Figure 4. Scalability of PrefixSpan.

Figure 4 shows the scalability of PrefixSpan-1 and PrefixSpan-2 with respect to the number of sequences. Both methods are linearly scalable. Since the support threshold is set to 0.20%, PrefixSpan-2 performs better.

In summary, our performance study shows that PrefixSpan is more efficient and scalable than FreeSpan and GSP, whereas FreeSpan is faster than GSP when the support threshold is low, and there are many long patterns. Since PrefixSpan-2 uses bi-level projection to dramatically reduce the number of projections, it is more efficient than PrefixSpan-1 in large databases with low support threshold. Once the projected databases can be held in main memory, pseudo-projection always leads to the most efficient solution. The experimental results are consistent with our theoretical analysis.

5 Discussions

As supported by our analysis and performance study, both PrefixSpan and FreeSpan are faster than GSP, and PrefixSpan is also faster than FreeSpan. Here, we summarize the factors contributing to the efficiency of PrefixSpan, FreeSpan and GSP as follows.

- **Both PrefixSpan and FreeSpan are pattern-growth methods, their searches are more focused and thus efficient.** Pattern-growth methods try to grow longer patterns from shorter ones. Accordingly, they divide the search space and focus only on the subspace potentially supporting further pattern growth at a time. Thus, their search spaces are focused and are confined by projected databases. A projected database for a sequential pattern α contains all and only the necessary information for mining sequential patterns that can be grown from α. As mining proceeds to long sequential patterns, projected databases become smaller and smaller. In contrast, GSP always searches in the original database. Many irrelevant sequences have to be scanned and checked, which adds to the unnecessarily heavy cost.
- **Prefix-projected pattern growth is more elegant than frequent pattern-guided projection.** Comparing with frequent pattern-guided projection, employed in FreeSpan, prefix-projected pattern growth is more progressive. Even in the worst case, PrefixSpan still guarantees that projected databases keep shrinking and only takes care postfixes. When mining in dense databases, FreeSpan cannot gain much from projections, whereas PrefixSpan can cut both the length and the number of sequences in projected databases dramatically.
- **The Apriori property is integrated in bi-level projection PrefixSpan.** The *Apriori* property is the essence of the Apriori-like methods. Bi-level projection in PrefixSpan applies the *Apriori* property in the pruning of projected databases. Based on this property, bi-level projection explores the 3-way checking to determine whether a sequential pattern can potentially lead to a longer pattern and which items should be used to assemble longer patterns. Only fruitful portions of the sequences are projected into the new databases. Furthermore, 3-way checking is efficient since only corresponding cells in S*-matrix* are checked, while no further assembling is needed.

6 Conclusions

In this paper, we have developed a novel, scalable, and efficient sequential mining method, called PrefixSpan. Its general idea is to examine only the prefix subsequences and project only their corresponding postfix subsequences into projected databases. In each projected database, sequential patterns are grown by exploring only local frequent patterns. To further improve mining efficiency, two kinds of database projections are explored: *level-by-level projection* and *bi-level projection*, and an optimization technique which explores pseudo-projection is developed. Our systematic performance study shows that PrefixSpan mines the complete set of patterns and is efficient and runs considerably faster than both Apriori-based GSP algorithm and FreeSpan. Among different variations of PrefixSpan, bi-level projection has better performance at disk-based processing, and pseudo-projection has the best performance when the projected sequence database can fit in main memory.

PrefixSpan represents a new and promising methodology at efficient mining of sequential patterns in large databases. It is interesting to extend it towards mining sequential patterns with time constraints, time windows and/or taxonomy, and other kinds of time-related knowledge. Also, it is important to explore how to further develop such a pattern growth-based sequential pattern mining methodology for effectively mining DNA databases.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, Sept. 1994.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering (ICDE'95)*, pages 3–14, Taipei, Taiwan, Mar. 1995.

[3] C. Bettini, X. S. Wang, and S. Jajodia. Mining temporal relationships with multiple granularities in time sequences. *Data Engineering Bulletin*, 21:32–38, 1998.

[4] M. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. In *Proc. 1999 Int. Conf. Very Large Data Bases (VLDB'99)*, pages 223–234, Edinburgh, UK, Sept. 1999.

[5] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proc. 1999 Int. Conf. Data Engineering (ICDE'99)*, pages 106–115, Sydney, Australia, Apr. 1999.

[6] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. Freespan: Frequent pattern-projected sequential pattern mining. In *Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00)*, pages 355–359, Boston, MA, Aug. 2000.

[7] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, pages 1–12, Dallas, TX, May 2000.

[8] H. Lu, J. Han, and L. Feng. Stock movement and n-dimensional inter-transaction association rules. In *Proc. 1998 SIGMOD Workshop Research Issues on Data Mining and Knowledge Discovery (DMKD'98)*, pages 12:1–12:7, Seattle, WA, June 1998.

[9] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.

[10] B. Özden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proc. 1998 Int. Conf. Data Engineering (ICDE'98)*, pages 412–421, Orlando, FL, Feb. 1998.

[11] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. 5th Int. Conf. Extending Database Technology (EDBT'96)*, pages 3–17, Avignon, France, Mar. 1996.