Mining Frequent Patterns by Pattern-Growth: Methodology and Implications *

Jiawei Han
School of Computing Science
Simon Fraser University
Burnaby, B.C., Canada V5A 1S6
han@cs.sfu.ca

Jian Pei
School of Computing Science
Simon Fraser University
Burnaby, B.C., Canada V5A 1S6
peijian@cs.sfu.ca

ABSTRACT

Mining frequent patterns has been a focused topic in data mining research in recent years, with the development of numerous interesting algorithms for mining association, correlation, causality, sequential patterns, partial periodicity, constraint-based frequent pattern mining, associative classification, emerging patterns, etc. Most of the previous studies adopt an Apriori-like, candidate generation-and-test approach. However, based on our analysis, candidate generation and test may still be expensive, especially when encountering long and numerous patterns.

A new methodology, called **frequent pattern growth**, which mines frequent patterns without candidate generation, has been developed. The method adopts a divide-and-conquer philosophy to project and partition databases based on the currently discovered frequent patterns and grow such patterns to longer ones in the projected databases. Moreover, efficient data structures have been developed for effective database compression and fast in-memory traversal. Such a methodology may eliminate or substantially reduce the number of candidate sets to be generated and also reduce the size of the database to be iteratively examined, and, therefore, lead to high performance.

In this paper, we provide an overview of this approach and examine its methodology and implications for mining several kinds of frequent patterns, including association, frequent closed itemsets, max-patterns, sequential patterns, and constraint-based mining of frequent patterns. We show that *frequent pattern growth* is efficient at mining large databases and its further development may lead to scalable mining of many other kinds of patterns as well.

Keywords

Scalable data mining methods and algorithms, frequent patterns, associations, sequential patterns, constraint-based mining

1. INTRODUCTION

Since the introduction of association mining in [2], there have been many studies on efficient and scalable frequent pattern mining algorithms. A milestone in these studies is the development of an Apriori-based, level-wise mining method for associations [3; 19], which has sparked the development of various kinds of Apriori-like association mining algorithms, as well as its extensions to mining correlation [8], causality [27], sequential patterns [4], episodes [20], max-patterns [5], constraint-based mining [29; 21; 17; 12], associative classification [18], cyclic association rules [22], ratio rules [16], iceberg queries and iceberg cubes [10; 7], partial periodicity [13], emerging patterns [9], and many other patterns.

There is an important, common ground among all these methods developed: the use of an *anti-monotone* Apriori *property* of frequent patterns [3]: *if any length-k pattern is not frequent in the database, none of its length-$(k+1)$ super-patterns can be frequent.* This property leads to the powerful pruning of the set of itemsets to be examined in the search for longer frequent patterns based on the existing ones.

Besides applying the Apriori property, most of the developed methods adopt a level-wise, candidate generation-and-test approach, which scans the database multiple times (although there have been many techniques developed for reducing the number of database scans). The first scan finds all of the length-1 frequent patterns. The *k-th* (for $k > 1$) scan starts with a *seed set* of length-$(k-1)$ frequent patterns found in the previous pass and generates new potential length-k patterns, called *candidate patterns*. The *k-th* scan of the database finds the *support* of every length k candidate pattern. The candidates which pass the minimum support threshold are identified as frequent patterns and become the seed set for the next pass. The computation terminates when there is no frequent pattern found or there is no candidate pattern that can be generated in any pass.

The candidate generation approach achieves good performance by reducing the number of candidates to be generated. However, when the minimum support threshold is low or the length of the patterns to be generated is long, the candidate generation-based algorithm may still bear the following non-trivial costs, independent of detailed implementation techniques.

1. The number of candidates to be generated may still be huge, especially when the length of the patterns to be generated is long. For example, to generate one frequent pattern of length 100, such as $\{a_1, a_2, \ldots, a_{100}\}$, the number of candidates that has to be generated will

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

be at least $\Sigma_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^{30}$.

2. Each scan of the database examines the entire database against the whole set of current candidates, which is quite costly when the database is large and the number of candidates to be examined is numerous.

To overcome this difficulty, a new approach, called *frequent pattern growth*, has been developed, in a series of studies, such as [15; 25; 23; 14; 24; 26], which adopts a divide-and-conquer methodology and mines frequent patterns without candidate generation. The approach has several distinct features:

1. Instead of generating a large number of candidates, the method preserves (in some compressed forms) the essential groupings of the original data elements for mining. Then the analysis is focused on counting the frequency of the relevant data sets instead of candidate sets.
2. Instead of scanning the *entire* database to match against the *whole* corresponding set of candidates in each pass, the method partitions the data set to be examined as well as the set of patterns to be examined by database projection. Such a divide-and-conquer methodology substantially reduces the search space and leads to high performance.
3. With the growing capacity of main memory and the substantial reduction of database size by database projection as well as the space needed for manipulating large sets of candidates, a substantial portion of data can be put into main memory for mining. New data structures and methods, such as FP-tree and pseudo-projection (for mining sequential patterns), have been developed for data compression and pointer-based traversal. The performance studies have shown the effectiveness of such techniques.

A few pieces of work have contributed to the development of the frequent pattern-growth methodology, as illustrated below.

The TreeProjection method [1] proposes a database projection technique which explores the projected databases associated with different frequent itemsets. The FP-growth algorithm [15] performs database projection when the database size is huge and then constructs a compressed data structure, FP-tree, when the compressed tree can fit in main memory. The remaining mining will be focused on the recursively generated, projected FP-trees. Besides mining frequent itemsets, the FP-tree structure can be used for mining frequent closed itemsets, which is presented in the CLOSET algorithm [25].

The frequent pattern-growth methodology influences constraint-based mining of frequent itemsets as well. The constraint-pushing techniques developed for Apriori-based mining [21] can be applied to pattern growth mining. In addition, some complex kinds of constraints, such as convertible constraints, which cannot be pushed deep into the mining process by Apriori, can be done so with frequent pattern growth [24], due to the facts that (1) pattern growth only needs to examine part of the database (the projected one), and (2) data can be organized in a structured way to facilitate the controlled growth of frequent patterns.

Similar divide-and-conquer ideas but different projection techniques have been developed for mining sequential patterns, which are presented in two algorithms, FreeSpan [14] and PrefixSpan [26]. The performance study shows that both methods outperform the classical Apriori-based sequential pattern mining algorithm GSP [28], and PrefixSpan has considerably better performance than FreeSpan.

In this paper, we provide an overview of several recently developed frequent pattern growth mining methods and discuss their implications. The remaining of the paper is organized as follows. In Section 2, we examine the FP-growth method for mining frequent itemsets and also mention the CLOSET method for mining frequent closed itemsets. In Section 3, we look at the impact of FP-growth to constraint-based mining of frequent patterns and the handling of convertible constraints. In Section 4, we introduce two pattern-growth-based methods for mining sequential patterns: FreeSpan [14] and PrefixSpan [26]. In Section 5, we discuss the potential extensions of pattern-growth methods and conclude our study.

2. FP-GROWTH: PATTERN GROWTH FOR MINING FREQUENT ITEMSETS

As shown by many researchers [2; 3], mining frequent itemsets represents the core of mining association rules, correlations, and many other patterns.

Let a *transaction database* TDB consist of a set of transactions in the form of $T = (tid, X)$ where tid is a *transaction-id* and X an itemset (i.e., a set of items). A transaction T is said to *contain* itemset Y if and only if $Y \subseteq X$. The support of an itemset W in TDB, denoted as $sup(W)$, is the number of transactions in TDB containing W. Given a user-specified minimum support threshold, min_sup, W is frequent if and only if $sup(W) \geq min\_sup$. The problem of mining frequent itemsets is to *find the complete set of frequent itemsets in a transaction database* TDB *w.r.t. given support threshold* min_sup.

Here we examine how one can develop a pattern growth method, FP-growth [15], for efficient mining of frequent itemsets in large databases. FP-growth first performs a frequent item-based database projection when the database is large and then switches to main-memory-based mining by constructing a compact data structure, called FP-tree, and transforming mining database into mining this compact tree. We first show how FP-tree be constructed from a database.

Example 1. (FP-tree) Let the transaction database, DB, be (the first two columns of) Table 1 and the minimum support threshold be 3.

tid	Itemset	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

Table 1: The transaction database TDB.

First, a scan of DB derives a *list* of frequent items, $\langle (f:4), (c:4), (a:3), (b:3), (m:3), (p:3) \rangle$, (the number after ":" indicates the support), and with items ordered in frequency descending order. This ordering is important since

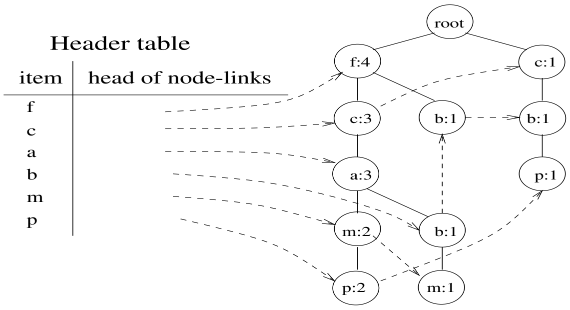

Figure 1: FP-tree for transaction database in Table 1.

each path of a tree will follow this order. For convenience of later discussions, the frequent items in each transaction are listed in this ordering in the rightmost column of Table 1.
Second, the root of a tree, labeled with "*null*" is created. Scan the DB the second time. The scan of the first transaction leads to the construction of the first branch of the tree: $\langle(f:1),(c:1),(a:1),(m:1),(p:1)\rangle$. Notice that the branch is not ordered in $\langle f,a,c,m,p\rangle$ as in the transaction but is ordered according to the order in the *list* of frequent items. For the second transaction, since its (ordered) frequent item list $\langle f,c,a,b,m\rangle$ shares a common prefix $\langle f,c,a\rangle$ with the existing path $\langle f,c,a,m,p\rangle$, the count of each node along the prefix is incremented by 1, and one new node $(b:1)$ is created and linked as a child of $(a:2)$ and another new node $(m:1)$ is created and linked as the child of $(b:1)$. Remaining transactions can be inserted similarly.
To facilitate tree traversal, an item header table is built, in which each item points, via a head of node-link, to its first occurrence in the tree. Nodes with the same item-name are linked in sequence via node-links. After scanning all transactions in DB, the tree with the associated node-links is shown in Figure 1. □

The FP-tree built in Example 2 has some nice properties as follows: (1) *FP-tree contains complete information of TDB w.r.t. frequent itemset mining*: every transaction in TDB is mapped onto one path in the FP-tree, and the frequent itemset information is completely stored in the tree; (2) *FP-tree is a highly compact structure*: since there are often a lot of sharing of frequent items among transactions, the size of the tree is usually much smaller than its original database; and (3) *there is a node-link property*: for every frequent item x, all transactions containing x can be obtained by following x's node-links starting from x's head in the FP-tree header table.
Based on this compact structure, FP-growth mines the complete set of frequent itemsets as follows.

Example 2. (FP-growth) Let us examine the mining process based on the constructed FP-tree (Figure 1).
According to the list of frequent items, the complete set of frequent itemsets can be divided into 6 subsets without overlap: (1) frequent itemsets having item p; (2) the ones having item m but no p; ... ; and (6) the one having only item f. FP-growth finds these subsets of frequent itemsets as follows.
Based on node-link property, we collect all the transactions that p participates by starting from p's head (in the header table) and following p's node-links.
Item p derives a frequent itemset $(p:3)$ and two paths in the FP-tree: $\langle f:4,c:3,a:3,m:2,p:2\rangle$ and $\langle c:1,b:1,p:1\rangle$. The first path indicates that string "(f,c,a,m,p)" appears twice in the database. Notice although string $\langle f,c,a\rangle$ appears three times and $\langle f\rangle$ itself appears even four times, they only appear twice *together* with p. Thus to study which strings appear together with p, only p's prefix path $\langle fcam:2\rangle$ counts. Similarly, the second path indicates string "(c,b,p)" appears once in the set of transactions in TDB, or p's prefix path is $\langle cb:1\rangle$. These two prefix paths of p, "$\{(fcam:2),(cb:1)\}$", form p's sub-database, which is called p's *conditional database* (i.e., the sub-database under the condition of p's existence). Construction of an FP-tree on this conditional sub-database (which is called p's conditional FP-tree) leads to only one branch $(c:3)$. Hence only one frequent itemset $(cp:3)$ is derived. The search for frequent itemsets having p terminates.
For item m, it derives a frequent itemset $(m:3)$ and two paths $\langle f:4,c:3,a:3,m:2\rangle$ and $\langle f:4,c:3,a:3,b:1,m:1\rangle$. Notice p appears together with m as well, however, there is no need to include p here in the analysis since any frequent itemsets involving p has been analyzed in the previous examination of p. Similar to the above analysis, m's conditional sub-database is, $\{(fca:2),(fcab:1)\}$. Constructing an FP-tree on it, we derive m's conditional FP-tree, $\langle fca:3\rangle$, a single frequent itemset path.
Since m's conditional FP-tree, $\langle fca:3\rangle$, has a single branch, instead of recursively constructing its conditional FP-trees, one can simply enumerate all the combinations of its components, i.e., $\{(a:3),(c:3),(f:3),(ca:3),(fa:3),(fca:3),(fc:3)\}$. Such simple pattern enumeration for single-path FP-trees has been proven truly useful at reducing mining efforts.

item	conditional sub-database	conditional FP-tree
p	$\{(fcam:2),(cb:1)\}$	$\{(c:3)\}\|p$
m	$\{(fca:2),(fcab:1)\}$	$\{(fca:3)\}\|m$
b	$\{(fca:1),(f:1),(c:1)\}$	$\emptyset$
a	$\{(fc:3)\}$	$\{(fc:3)\}\|a$
c	$\{(f:3)\}$	$\{(f:3)\}\|c$
f	$\emptyset$	$\emptyset$

Table 2: Conditional (sub)-databases and conditional FP-trees of frequent 1-itemsets

Similarly, the remaining frequent itemsets can be mined by constructing corresponding conditional sub-databases and perform mining on them, respectively. The conditional sub-databases and the conditional FP-trees generated are summarized in Table 2. □

When the database is too big to make its FP-tree fit in memory, the database can be projected into its conditional sub-databases (without constructing disk-based FP-trees). Two methods can be used for the projection of a database into its conditional sub-databases: *parallel projection* and *partition projection*. The former projects each transaction into all of its projected databases in one scan, whereas the latter projects each transaction only to its first projected database (according to the ordering of items). The former facilitates parallel processing but requires large disk space to store all of the projected databases, whereas the latter ensures that the additional disk space required is no more than the original database but it needs additional projections of its (projected) transactions to subsequent projected databases in the later processing.

After one or a few rounds of projections, the corresponding conditional FP-trees should be able to fit in memory. Then a memory-based FP-tree can be constructed for fast mining.

The FP-growth algorithm is presented in [15]. Its performance analysis shows that the FP-growth mining of both long and short frequent itemsets is efficient and scalable. It is about an order of magnitude faster than Apriori [3] and other candidate generation-based algorithms, and is also faster than TreeProjection, a projection-based algorithm proposed in [1].

In comparison with the candidate generation-based algorithms, FP-growth has the following advantages: (1) FP-tree is highly compact, usually substantially smaller than the original database, and thus saves the costly database scans in the subsequent mining process. (2) It avoids costly candidate sets generation and test by successively concatenating frequent 1-itemsets found in the (conditional) FP-trees: It never generates any combinations of new candidate sets which are not in the database because the itemset in any transaction is always encoded in the corresponding path of the FP-trees. In this context, the mining methodology is not Apriori-like (*restricted*) *generation-and-test* but *frequent pattern* (*fragment*) *growth only*. The major operations of mining are count accumulation and prefix path count adjustment, which are usually much less costly than candidate generation and itemset matching operations performed in most Apriori-like algorithms. (3) It applies a partitioning-based divide-and-conquer method which dramatically reduces the size of the subsequent conditional sub-databases and conditional FP-trees. Several other optimization techniques, including ordering of frequent items, and employing the least frequent events as suffix, also contribute to the efficiency of the method.

Besides mining frequent itemsets, an extension of the FP-growth method, called CLOSET [25], can be used to mine frequent closed itemsets and max-patterns, where a *frequent closed itemset* is a frequent itemset, c, where there is no proper superset of c sharing the same support count with c, and a *max-pattern* is a frequent pattern, p, such that any proper superpattern of p is not frequent. Max-patterns and frequent closed itemset can be used to reduce the number of frequent itemsets and association rules generated at association mining.

By frequent pattern growth, one can also mine closed frequent itemsets and max-patterns, using the FP-tree structure. Moreover, a single prefix-path compression technique can be developed for compressing FP-trees or conditional FP-trees that contain single prefix paths. This will further enhance the performance and reduce the efforts of redundancy checking at mining closed frequent itemsets and max-patterns.

3. PUSHING MORE CONSTRAINTS IN PATTERN-GROWTH MINING

Frequent pattern mining often generates a large number of frequent itemsets and rules, which reduces not only the efficiency but also the effectiveness of mining since users have to sift through a large number of mined rules to find useful ones.

Recent work has highlighted the importance of the paradigm of constraint-based mining: the user is allowed to express his focus in mining, by means of a rich class of constraints that capture application semantics. Besides allowing user exploration and control, the paradigm allows many of these constraints to be pushed deep inside mining, thus pruning the search space and achieving high performance.

Previous studies [21; 17; 6; 12] have identified three classes of constraints, *anti-monotone, monotone*, and *succinct*, which can be pushed deep in frequent itemset mining. While these cover a large class of useful constraints, many other useful and natural constraints remain. For example, consider the constraints $avg(S)\ \theta\ v$, and $sum(S)\ \theta\ v$ ($\theta \in \{\geq, \leq\}$). The first is neither anti-monotone, nor monotone, nor succinct. The second is anti-monotone when θ is $\leq$ and *all items have non-negative values*. But if S can contain items of arbitrary values, the constraint is rather like the first one. This means these constraints are hard to optimize.

With the development of frequent pattern growth method, databases can be projected and partitioned in an organized way, as well as the patterns to be searched for. Thus some constraints which are hard to optimize under the Apriori mining framework can be optimized with the frequent pattern growth method. Let's examine one example.

Example 3. Let Table 3 be our transaction database $\mathcal{T}$, with a set of items $I = \{a, b, c, d, e, f, g, h\}$. Let the support threshold be $min\_support = 2$. Also, let each item have an attribute *value* (such as *profit*), with the concrete value shown in Table 4.

Transaction ID	Items in transaction
10	a, b, c, d, f
20	b, c, d, f, g, h
30	a, c, d, e, f
40	c, e, f, g

Table 3: The transaction database $\mathcal{T}$ in Example 3

Item	a	b	c	d	e	f	g	h
Value	40	0	−20	10	−30	30	20	−10

Table 4: The profit of each item in Example 3.

The constraint $C_{avg} \equiv avg(S) \geq 25$ is not anti-monotone (nor monotone, nor succinct). For example, $avg(df) = (10 + 30)/2 < 25$, violates the constraint. However, upon adding one more item a, $avg(adf) = (40 + 10 + 30)/3 \geq 25$, adf satisfies C_{avg}. □

This example shows that a constraint like $avg(S) \geq v$ cannot be pushed deep into the Apriori mining algorithm because the subsets (supersets) of a valid itemset could well be invalid and vice versa. Let us examine how to push such a constraint deep into the mining process in the frequent pattern growth mining.

Example 4. With the same minimum support threshold over transaction database $\mathcal{T}$ in Table 3, one can list items in value descending order $\mathcal{R}$: $\langle a(40), f(30), g(20), d(10), b(0), h(-10), c(-20), e(-30)\rangle$.

A database can be then partitioned according to the ordered frequent items. With the frequent pattern growth mining, the constraint C can be pushed deep into the mining process, as shown in Figure 2.

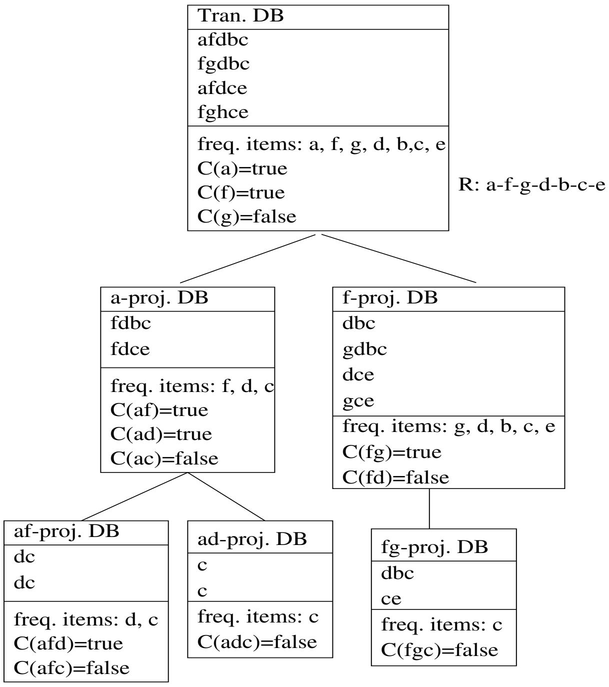

Figure 2: Mining frequent itemsets satisfying constraint $avg(S) \geq 25$.

By scanning $\mathcal{T}$ once, we find support count for every item. Since h appears in only one transaction, it is an infrequent item and is thus dropped without further consideration. The set of frequent 1-itemsets is $\langle a, f, g, d, b, c, e\rangle$, listed in order $\mathcal{R}$. Among them, only a and f satisfy the constraint. The fact that itemset g does not satisfy the constraint implies that none of any 1-itemsets after g in order $\mathcal{R}$ can satisfy the constraint avg. Similarly, itemsets having g, d, b, c or e as prefix cannot satisfy the constraint. Thus, the set of frequent itemsets satisfying the constraint can be partitioned into two subsets:

1. The ones having itemset a as a prefix w.r.t. $\mathcal{R}$, i.e., those containing item a; and
2. The ones having itemset f as a prefix w.r.t. $\mathcal{R}$, i.e., those containing item f but no a.

They form two projected databases [15] which can be mined with the constraint C pushed in. We examine the first one only since the second is similar.

Since a is a frequent itemset satisfying the constraint, the frequent itemsets having a as a proper prefix can be found in *a-projected database* (the subset of transactions containing a). It contains two transactions: $bcdf$ and $cdef$. Since items b and e is infrequent within this projected database, neither ab nor ae can be frequent. So, they are pruned. The frequent items in the a-projected database is f, d, c, listed in the order $\mathcal{R}$. Since ac does not satisfy the constraint, there is no need to create an ac-projected database.

To check what can be mined in the a-projected database with af and ad, as prefix, respectively, we need to construct the two projected databases and mine them. This process is similar to the mining of a-projected databases.

The *af-projected database* contains two frequent items d and c, and only afd satisfy the constraint. Moreover, since $afdc$ does not satisfies the constraint, the process in this branch is complete. Since afc violates the constraint, there is no need to construct afc-projected database. The *ad-projected database* contains one frequent item c, but adc does not satisfy the constraint. Therefore, the set of frequent itemsets satisfying the constraint and having a as prefix contains a, af, afd, and ad.

In summary, the complete set of frequent itemsets satisfying the constraint contains 6 itemsets: a, f, af, ad, afd, fg. The method with ordered itemsets and frequent pattern growth generates and tests only a small set of itemsets. □

This example shows that by proper ordering of itemsets, frequent pattern growth method may push some tough constraints (called *convertible constraints*) deeper than the Apriori methods. A systematic classification of such constraints and a study of how to push them into the mining process are in [23; 24].

4. PREFIXSPAN: MINING SEQUENTIAL PATTERNS BY PATTERN GROWTH

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.

A *sequence database* S is a set of tuples $\langle sid, s\rangle$, where sid is a *sequence_id* and s is a sequence (i.e., an ordered list of itemsets). 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. Given a positive integer ξ as the *support threshold*, a sequence α is called a *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*.

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.

Sequential pattern mining is more challenging than mining frequent itemsets. Sequences allow multiple occurrences of items and combination of items into itemsets, which may lead to a combination explosion. For example, using items a and b, there are only three possible itemsets: a, b and ab. However, even the length of sequences is limited to 3, there are 12 possible sequences: $\langle aaa\rangle$, $\langle aab\rangle$, ... , $\langle bbb\rangle$, $\langle (ab)a\rangle$, ... , $\langle b(ab)\rangle$.

Sequential patterns also have the *Apriori property*: *every non-empty sub-sequence of a sequential pattern is a sequential pattern.* A typical sequential pattern mining algorithm, GSP [28], is based on this *Apriori* property to reduce search space. However, the method bears similar non-trivial, inherent costs as to Apriori in mining frequent itemsets.

Following the similar philosophy of frequent pattern growth, two algorithms, FreeSpan [14] and PrefixSpan [26], are developed for pattern growth-based sequential pattern mining. FreeSpan mines sequential patterns by projecting the sequence database based on any frequent subsequences and growing subsequences in any position; whereas PrefixSpan does it by projecting the database based on only the frequent prefix subsequences and adding postfixes in the growth. Both methods find the complete set of sequential patterns but the latter is more efficient since it involves less database projections and less subsequence combinations to be examined. This analysis has also been verified by the performance results, and thus we examine only PrefixSpan using an example.

Example 5. (PrefixSpan) Suppose we want to mine sequential patterns in a sequence database S, shown in Table 5, with the support threshold set to 2. PrefixSpan works as follows.

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 5: A sequence database

First, we find length-1 sequential patterns by scanning S once. This derives the set of frequent items in sequences, i.e., the set of length-1 sequential patterns: $\{(\langle a\rangle : 4), (\langle b\rangle : 4), (\langle c\rangle : 4), (\langle d\rangle : 3), (\langle e\rangle : 3)$, and $(\langle f\rangle : 3)\}$.
Then, the search space can be partitioned into the following six subsets according to the six prefixes: (1) the ones with prefix $\langle a\rangle$; ... ; and (6) the ones with prefix $\langle f\rangle$. 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 6, and the mining process is explained as follows.
The sequential patterns with prefix $\langle a\rangle$ are mined in the (prefix) $\langle a\rangle$-projected database. It is the collection that contains only those subsequences prefixed with the first occurrence of $\langle a\rangle$. For example, in sequence $\langle (ef)(ab)(df)cb\rangle$, only the subsequence $\langle (\_b)(df)cb\rangle$ should count. Notice that $(\_b)$ means that the last element in the prefix, which is a, together with b, form one element (i.e., occurring together). Thus the $\langle a\rangle$*-projected database* consists of four 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 prefixed with $\langle a\rangle$ 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 patterns with prefix $\langle a\rangle$ can be partitioned into 6 subsets: (1) that prefixed with $\langle aa\rangle$, (2) that with $\langle ab\rangle$, ... , and finally, (6) that with $\langle af\rangle$. These subsets can be mined by constructing respective projected databases and mining each recursively.
For example, the $\langle aa\rangle$-projected database consists of only one non-empty (postfix) subsequences prefixed with $\langle aa\rangle$: $\langle (\_bc)(ac)d(cf)\rangle$. Since there is no hope to generate any frequent subsequence from a single sequence, the processing of $\langle aa\rangle$-projected database terminates. Similarly, 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 it 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 $\langle abc\rangle$.)
Using the same method, sequential patterns with prefix $\langle b\rangle$, $\langle c\rangle$, $\langle d\rangle$, $\langle e\rangle$ and $\langle f\rangle$, can be mined from the corresponding projected databases respectively. The projected databases as well as the sequential patterns found are shown in Table 6. □

The example shows that PrefixSpan examines only the prefix subsequences and projects only their corresponding postfix subsequences into projected databases, and in each projected database, sequential patterns are grown by exploring only local frequent patterns.
To further improve mining efficiency, two kinds of optimizations are explored [26]: (1) *pseudo-projection*, and (2) *bi-level projection*. Pseudo-projection is based on the following idea: 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. This 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 of disk space is very costly. Therefore, when the sequence database cannot be held in main memory, a *bi-level projection* method is explored, which projects databases not at every level but at every two levels. In comparison with *level-by-level projection*, bi-level projection reduces the cost of database projection and leads to improved performance when the database is huge and the support threshold is low.
A systematic performance study in [26] shows that PrefixSpan with these two optimizations is efficient and scalable. It mines the complete set of patterns and runs considerably faster than both Apriori-based GSP algorithm [28] and FreeSpan [14].

5. DISCUSSIONS AND CONCLUSIONS

We have presented a pattern-growth methodology for mining several kinds of frequent patterns in large databases. Our performance study shows that the algorithms derived from the pattern-growth methodology are more efficient and scalable than many other frequent pattern mining methods. According to our analysis, the high performance of the pattern-growth methodology is due to the following factors: (1) it adopts a divide-and-conquer strategy to project and partition a large database recursively into a set of progressively smaller ones, and the patterns to be searched for in each corresponding projected database are also reduced substantially; (2) it integrates disk-based database projection algorithms with main memory-based data structures and fast in-memory traversal algorithms, which can be well-tuned to achieve combined high performance by swapping disk-based algorithm into memory-based one when the projected and compressed data set can fit in memory; and (3) it makes good use of the Apriori property implicitly as well as other properties, such as the single tree-path property, but avoids generating a large number of candidates, which ensures each counting and testing is on the real data sets rather than on the potential candidate sets. These several techniques combined lead to high performance mining algorithms.

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 6: Projected databases and sequential patterns

There are many issues which still need to be studied in depth. One direction is to develop techniques to further improve the mining efficiency, such as materialization and incremental computation of FP-trees or projected databases, parallel and distributed pattern-growth mining, reduction of the cost of projection, and so on. Another direction is to expand the scope of pattern-growth mining towards mining more sophisticated patterns, such as mining multiple dimensional and/or multiple level associations or sequential patterns, mining correlations and causal structures, mining partial matching patterns (such as DNA sequence patterns which contain insertions, deletions, and mutations), mining partial periodicity, associative classification, constraint-based mining of sequential and other patterns [11], and many other tasks. Many of these mining tasks have been studied in the context of Apriori-based mining. However, a re-examination of these mining tasks in the framework of frequent pattern growth may lead to the development of more efficient mining algorithms as well as potentially new methods due to database partition and the structural nature of the pattern-growth approach.

6. REFERENCES

[1] R. Agarwal, C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent itemsets. In *Journal of Parallel and Distributed Computing*, 2000.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD'93*, pp. 207–216, Washington, DC, May 1993.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB'94*, pp. 487–499, Santiago, Chile, Sept. 1994.

[4] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE'95*, pp. 3–14, Taipei, Taiwan, Mar. 1995.

[5] R. J. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD'98*, pp. 85–93, Seattle, WA, June 1998.

[6] R. J. Bayardo, R. Agrawal, and D. Gunopulos. Constraint-based rule mining on large, dense data sets. In *ICDE'99*, Sydney, Australia, April 1999.

[7] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD'99*, pp. 359–370, Philadelphia, PA, June 1999.

[8] S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: Generalizing association rules to correlations. In *SIGMOD'97*, pp. 265–276, Tucson, Arizona, May 1997.

[9] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *KDD'99*, pp. 43–52, San Diego, CA, Aug. 1999.

[10] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In *VLDB'98*, pp. 299–310, New York, NY, Aug. 1998.

[11] M. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. In *VLDB'99*, pp. 223–234, Edinburgh, UK, Sept. 1999.

[12] G. Grahne, L. Lakshmanan, and X. Wang. Efficient mining of constrained correlated sets. In *ICDE'00*, pp. 512–521, San Diego, CA, Feb. 2000.

[13] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *ICDE'99*, pp. 106–115, Sydney, Australia, April 1999.

[14] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. FreeSpan: Frequent pattern-projected sequential pattern mining. In *KDD'00*, pp. 355–359, Boston, MA, Aug. 2000.

[15] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD'00*, pp. 1–12, Dallas, TX, May 2000.

[16] F. Korn, A. Labrinidis, Y. Kotidis, and C. Faloutsos. Ratio rules: A new paradigm for fast, quantifiable data mining. In *VLDB'98*, pp. 582–593, New York, NY, Aug. 1998.

[17] L. V. S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *SIGMOD'99*, pp. 157–168, Philadelphia, PA, June 1999.

[18] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *KDD'98*, pp. 80–86, New York, NY, Aug. 1998.

[19] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *KDD'94*, pp. 181–192, Seattle, WA, July 1994.

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

[21] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *SIGMOD'98*, pp. 13–24, Seattle, WA, June 1998.

[22] B. Özden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *ICDE'98*, pp. 412–421, Orlando, FL, Feb. 1998.

[23] J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In *KDD'00*, pp. 350–354, Boston, MA, Aug. 2000.

[24] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent itemsets with convertible constraints. In *ICDE'01*, Heidelberg, Germany, April 2001.

[25] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *DMKD'00*, pp. 11–20, Dallas, TX, May 2000.

[26] 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. In *ICDE'01*, Heidelberg, Germany, April 2001.

[27] C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. In *VLDB'98*, pp. 594–605, New York, NY, Aug. 1998.

[28] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *EDBT'96*, pp. 3–17, Avignon, France, Mar. 1996.

[29] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *KDD'97*, pp. 67–73, Newport Beach, CA, Aug. 1997.