MINING IN FREQUENT PATTERN USING EFFICIENT PATTERN-GROWTH METHODS

Dr.K.Krishnamoorthy 1, T.Praveenkumar 2
1. Professor, Sudharsan Engineering College, Pudukkottai-622501, Tamilnadu
e-mail: kkr_510@yahoo.co.in
2. Assistant Professor Sudharsan Engineering College, Pudukkottai-622501, Tamilnadu
e-mail: tpraveenkumarsec@gmail.com

Abstract

Mining frequent patterns from large databases plays an essential role in many data mining tasks and has broad applications. Most of the previously proposed methods adopt apriori-like candidate-generation-and-test approaches. However, those methods may encounter serious challenges when mining datasets with prolific patterns and/or long patterns.

In this work, we develop a class of novel and efficient pattern-growth methods for mining various frequent patterns from large databases. Pattern-growth methods adopt a divide- and-conquer approach to decompose both the mining tasks and the databases. Then, they use a pattern fragment growth method to avoid the costly candidate-generation-and-test processing completely. Moreover, effective data structures are proposed to compress crucial information about frequent patterns and avoid expensive, repeated database scans. A comprehensive performance study shows that pattern-growth methods, \textit{FP-growth} and \textit{H-mine}, are efficient and scalable. They are faster than some recently reported new frequent pattern mining methods.

Interestingly, pattern growth methods are not only efficient, but also effective. With pattern growth methods, many interesting patterns can also be mined efficiently, such as patterns with some tough non-anti-monotonic constraints and sequential patterns. These techniques have strong implications to many other data mining tasks.

Keywords: Frequent patterns, Prolific Patterns, Pattern Growth, FP-Growth, H-Mine.

Introduction

“The universe is full of magical things patiently waiting for our wits to grow sharper.” \cite{1} Data mining is to find valid, novel, potentially useful, and ultimately understandable patterns in data. In general, there are many kinds of patterns (knowledge) that can be discovered from data. For example, association rules can be mined for market basket analysis, classification rules can be found for accurate classifiers, clusters and outliers can be identified for customer relation management.

Frequent pattern mining plays an essential role in many data mining tasks, such as mining association rules, correlations, causality, sequential patterns, episodes, multidimensional patterns, max-patterns, partial periodicity, and emerging patterns. Frequent pattern mining techniques can also be extended to solve many other problems, such as iceberg-cube computation and...
classification. Thus, effective and efficient frequent pattern mining is an important and interesting research problem.

**Contribution**

In this paper, we study the problem of efficient and effective frequent pattern mining, as well as some of its extensions and applications. In particular, we make the following contributions.

- We systematically develop a pattern-growth method for frequent pattern mining. A novel algorithm, FP-growth, is proposed for efficiently mining frequent patterns from large dense datasets. Furthermore, to achieve efficient frequent pattern mining in various situations, we design H-mine, which is highly scalable and space preserving for very large databases.

- As an inherent problem, frequent pattern mining may return too many patterns.

Constraint-based data mining is an important approach to solve the problem of effective data mining. We study the problem of constraint-based frequent pattern mining using pattern-growth methods. Our study shows that pattern-growth methods can push constraints deeper into the mining process, even including such constraints using aggregate $AVG()$ and $SUM()$, which other methods cannot handle.

- We extend the pattern-growth method to allow the mining of sequential patterns. Our study shows that pattern-growth methods are more efficient in mining large sequence databases. Interesting techniques are developed to solve the sequential pattern mining problem effectively.

**The frequent pattern mining problem**

The frequent pattern mining problem was first introduced by R. Agrawal, et al. in [AIS93]

Let $I = \{i_1, \ldots, im\}$ be a set of items. An itemset $X \subseteq I$ is a subset of items. Hereafter, we write itemsets as $X = i j_1 \cdot \cdot \cdot i j_n$, i.e. omitting set brackets. Particularly, an itemset with $l$ items is called an $l$-itemset.

A transaction $T = (tid, X)$ is a tuple where $tid$ is a transaction-id and $X$ is an itemset. A transaction $T = (tid, X)$ is said to contain itemset $Y$ if $Y \subseteq X$.

A transaction database $TDB$ is a set of transactions. The support of an itemset $X$ in transaction database $TDB$, denoted as $supTDB(X)$ or $sup(X)$, is the number of transactions in $TDB$ containing $X$.

**Problem statement.**

Given a user-specified support threshold $min sup$, $X$ is called a frequent itemset or frequent pattern if $sup(X) \geq min sup$. The problem of mining frequent itemsets is to find the complete set of frequent itemsets in a transaction database $TDB$ with respect to a given support threshold $min sup$.

Association rules can be derived from frequent patterns. $X \Rightarrow Y$ holds in the transaction database $TDB$ with confidence $c$, where $c = \frac{sup(X \cup Y)}{sup(X)}$.

Given a transaction database $TDB$,
a support threshold $\min sup$ and a confidence threshold $\min conf$, the problem of association rule mining is to find the complete set of association rules that have support and confidence no less than the user-specified thresholds, respectively.

Association rule mining can be divided into two steps. First, frequent patterns with respect to support threshold $\min sup$ are mined. Second, association rules are generated with respect to confidence threshold $\min conf$. As shown in many studies (e.g., [AS94]), the first step, mining frequent patterns, is significantly more costly in terms of time than the rule generation step.

As we shall see later, frequent pattern mining is not only used in association rule mining. Instead, frequent pattern mining is the basis for many data mining tasks, such as sequential pattern mining and associative classification. It also has broad applications, such as basket data analysis, cross-marketing, catalog design, sale campaign analysis, web log (click stream) analysis, etc.

**Apriori Heuristic and Algorithm**

To achieve efficient mining frequent patterns, an anti-monotonic property of frequent itemsets, called the Apriori heuristic, was identified in [AS94].

**Theorem 2.1 (Apriori)** Any superset of an infrequent itemset cannot be frequent. In other words, every subset of a frequent itemset must be frequent. Proof. To prove the theorem, we only need to show $sup(X) \leq sup(Y)$ if $X \supseteq Y$.

Given a transaction database $TDB$. Let $X$ and $Y$ be two itemsets such that $X \supseteq Y$. For each transaction $T$ containing itemset $X$, $T$ also contains $Y$, which is a subset of $X$. Thus, we have $sup(X) \leq sup(Y)$.

The Apriori heuristic can prune candidates dramatically. Based on this property, a fast frequent itemset mining algorithm, called Apriori, was developed. Apriori finds the complete set of frequent itemsets as follows.

1. Scan $TDB$ once to find frequent items, i.e. items appearing in at least 3 transactions.

They are $a, b, c, f, m, p$. Each of these six items forms a length-1 frequent itemset. Let $L_1$.

2. The set of length-2 candidates, denoted as $C_2$, is generated from $L_1$. Here, we use the Apriori heuristic to prune the candidates. Only those candidates that consist of frequent subsets can be potentially frequent. An itemset $xy \in C_2$ if and only if $x, y \in L_1$. Thus, $C_2 = \{ab, ac, \ldots, ap, bc, \ldots, mp\}$.

3. Scan $TDB$ once more to count the support of each itemset in $C_2$. The itemsets in $C_2$ passing the support threshold form the length-2 frequent itemsets, $L_2$. In this example, $L_2$ contains itemsets $ac, af, am, cf, cm$, and $fm$.

4. Then, we form the set of length-3 candidates. Only those length-3 itemsets for which every length-2 sub-itemset is in $L_2$ are qualified as candidates. For example, $acf$ is a length-3 candidate since $ac, af$ and $cf$ are all in $L_2$.

One scan of $TDB$ identifies the subset of length-3 candidates passing the support threshold and form the
set \( L_3 \) of length-3 frequent itemsets. A similar process goes on until no candidate can be derived or no candidate is frequent.

One can verify that the above process eventually finds the complete set of frequent itemsets in the database \( TDB \).

The Apriori algorithm is presented as follows.

Algorithm 1 (Apriori)

Input: transaction database \( TDB \) and support threshold \( \min sup \)

Output: the complete set of frequent patterns in \( TDB \) with respect to support threshold \( \min sup \)

Method:

1. scan transaction database \( TDB \) once to find \( L_1 \), the set of frequent 1-itemsets;

2. for \((k = 2; L_{k-1} = \emptyset, k + +) \) do

   (a) generate \( C_k \), the set of length-\( k \) candidates. A \( k \)-itemset \( X \) is in \( C_k \) if and only if every length-(\( k - 1 \)) subset of \( X \) is in \( L_{k-1} \);

   (b) if \( C_k = \emptyset \) then go to Step 3;

   (c) scan transaction database \( TDB \) once to count the support for every itemset in \( C_k \);

   (d) \( L_k = \{ X | (X \in C_k) \land (sup(X) \geq \min sup) \} \);

Information from transaction databases is essential for mining frequent patterns. Therefore, if we can extract the concise information for frequent pattern mining and store it into a compact structure, then it may facilitate frequent pattern mining. Motivated by this thinking, in this section, we develop a compact data structure, called \( FP-tree \), to store complete but no redundant information for frequent pattern mining.

Example. Let the transaction database, \( TDB \), be the first two columns of Table 3.1 (same as the transaction database used in Example 2.1), and the minimum support threshold be 3 (i.e., \( \min sup = 3 \)).

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
<th>(Ordered)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>( f, a, c, d )</td>
<td>( f, c, a, m, p )</td>
</tr>
<tr>
<td>200</td>
<td>( a, b, c )</td>
<td>( f, c, a, b, m )</td>
</tr>
<tr>
<td>300</td>
<td>( b, f )</td>
<td>( f )</td>
</tr>
<tr>
<td>400</td>
<td>( b, c )</td>
<td>( c )</td>
</tr>
<tr>
<td>500</td>
<td>( a, f, c, e )</td>
<td>( f, c, a, m, p )</td>
</tr>
</tbody>
</table>

A compact data structure can be designed based on the following observations:

1. Since only the frequent items will play a role in the frequent-pattern mining, it is necessary to perform one scan of transaction database \( TDB \) to identify the set of frequent items (with frequency count obtained as a by-product).

2. If the set of frequent items of each transaction can be stored in some compact structure, it may be possible to avoid repeatedly scanning the original transaction database.
3. If multiple transactions share a set of frequent items, it may be possible to merge the shared sets with the number of occurrences registered as count. It is easy to check whether two sets are identical if the frequent items in all of the transactions are listed according to a fixed order.

1. A scan of TDB derives a list of frequent items, \( h(f:4), (c:4), (a:3), (b:3), (m:3), (p:3)i \) (the number after “:” indicates the support), in which items are ordered in frequency-descending order. (In the case that two or more items have exactly same support count, they are sorted alphabetically.) This ordering is important since each path of a tree will follow this order.

2. Then, the root of a tree is created and labelled with “null”. The FP-tree is constructed as follows by scanning the transaction database TDB the second time.

(a) The scan of the first transaction leads to the construction of the first branch of the tree: \( h(f:1), (c:1), (a:1), (m:1), (p:1)i \). Notice that the frequent items in the transaction are listed according to the order in the list of frequent items.

(b) For the second transaction, since its (ordered) frequent item list \( h(f:4), c, a, b, m \) shares a common prefix \( h(f:3), c, a \) with the existing path \( h(f:2), c, a, m \), 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 \( (b:1) \).

(c) For the third transaction, since its frequent item list \( h(f:4), b \) shares only the node \( h(f:3) \) with the \( f \)-prefix subtree, \( f \)’s count is incremented by 1, and a new node \( (b:1) \) is created and linked as a child of \( (f:3) \).

(d) The scan of the fourth transaction leads to the construction of the second branch of the tree, \( h(c:1), (b:1), (p:1)i \).

(e) For the last transaction, since its frequent item list \( h(f:4), c, a, m, p \) is identical to the first one, the path is shared with the count of each node along the path incremented by 1.

To facilitate tree traversal, an item header table is built in which each item points to its first occurrence in the tree via a node-link. Nodes with the same item-name are linked in sequence via such node-link. After scanning all the transactions, the tree, together with the associated node-links, are shown in Figure 3.1.

Based on this example, a frequent-pattern tree can be designed as follows.

**Definition.** *(FP-tree)* A frequent-pattern tree (or FP-tree in short) is a tree structure defined below.

1. It consists of one root labeled as “null”, a set of item-prefix subtrees as the children of the root, and a frequent-item-header table.

2. Each node in the item-prefix subtree consists of three fields: item-name, count, and node-link, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this
node, and node-link links to the next node in the FP-tree carrying the same item-name, or null if there is none.

3. Each entry in the frequent-item-header table consists of two fields,—(1) item-name and (2) head of node-link (a pointer pointing to the first node in the FP-tree carrying the item-name).

Based on this definition, we have the following FP-tree construction algorithm.

Algorithm (FP-tree construction)

Input: A transaction database $\mathcal{D}$ and a minimum support threshold $\text{min sup}$.

Output: FP-tree, the frequent-pattern tree of $\mathcal{D}$.

Method: The FP-tree is constructed as follows.

1. Scan the transaction database $\mathcal{D}$ once. Collect $F$, the set of frequent items, and the support of each frequent item. Sort $F$ in support-descending order as $FList$, the list of frequent items.

2. Create the root of an FP-tree, $T$, and label it as “null”. For each transaction $t$ in

Select the frequent items in transaction $t$ and sort them according to the order of $FList$. Let the sorted frequent-item list in $t$ be $[p|P]$, where $p$ is the first element and $P$ is the remaining list. Call $insert$ tree($[p|P], T$).

The function $insert$ tree($[p|P], T$) is performed as follows. If $T$ has a child $N$ such that $N.item-name = p.item-name$, then increment $N$’s count by 1; else create a new node $N$, with count initialized to 1, parent link linked to $T$, and node-link linked to the nodes with the same item-name via the node-link structure. If $P$ is nonempty, call $insert$ tree($P, N$) recursively.

Analysis. The FP-tree construction takes exactly two scans of the transaction database:

1. The first scan collects the set of frequent items; and

2. The second constructs the FP-tree.

The cost of inserting a transaction $t$ into the FP-tree is $O(|f\text{req}(t)|)$, where $f\text{req}(t)$ is the set of frequent items in $t$. In next section, we will show that the FP-tree contains complete information for frequent-pattern mining.

**H-mine: Scalable Space-preserving Mining**

we developed $FP$-growth, a pattern-growth method for frequent pattern mining. Although $FP$-growth is more efficient than Apriori in many cases, it may still encounter some difficulties in some cases, as illustrated below.

- **Huge space is required to serve the mining.** $FP$-growth avoids candidate generation by compressing the transaction database into an FP-tree and pursuing partition-based mining recursively. However, if the database is huge and sparse, the FP-tree will be large
and the space requirement for recursion is a challenge.

- **Real databases contain all the cases.** Real data sets can be sparse and/or dense in different applications. For example, in telecommunication data analysis, calling patterns for home users vs. business users could be very different: some are frequent and dense (e.g., to family members and close friends), but some are huge and sparse. Similar situations arise for market basket analysis, census data analysis, classification and predictive modelling, etc. It is hard to select an appropriate mining method on the fly if no algorithm fits all.

- **Large applications need more scalability.** Many existing methods are efficient when the data set is not very large. Otherwise, their core data structures (such as FP-tree) or the intermediate results (e.g., the set of candidates in Apriori or the recursively generated conditional databases in FP-growth) may not fit in main memory and easily cause thrashing.

This poses a new challenge: “Can we work out a better method which is (1) efficient in all occasions (dense vs. sparse, huge vs. memory-based data sets), and (2) space requirement is small, even for very large databases?” We propose a new data structure, H-struct, and a new mining method, H-mine, to overcome these difficulties. One major feature of H-mine is that it is space-preserving, meaning (1) H-mine is moderate in memory usage; (2) it can fully utilize all available main memory space if necessary; and (3) it performs well even with very small main memory. We make the following progress.

1. A memory-based, efficient pattern-growth algorithm, \( H\text{-}mine(Mem) \), is proposed for mining frequent patterns for the data sets that can fit in (main) memory. A simple, memory-based hyperstructure, \( H\text{-}struct \), is designed for fast mining.

2. We show that, theoretically, \( H\text{-}mine(Mem) \) has polynomial space usage and is thus more space efficient than FP-growth and TreeProjection when mining sparse data sets, and also more efficient than Apriori-based methods which generate a large number of candidates. Experimental results show that \( H\text{-}mine(Mem) \) has exactly predictable space overhead and, in many cases, it is faster than memory-based Apriori and FP-growth with very limited space usage.

3. Based on \( H\text{-}mine(Mem) \), we propose \( H\text{-}mine \), a scalable algorithm for mining large databases by first partitioning the database, mining each partition in memory using \( H\text{-}mine(Mem) \), and then consolidating global frequent patterns.

4. For dense data sets, \( H\text{-}mine \) is integrated with FP-growth dynamically by detecting the swapping condition and constructing FP-trees for efficient mining.

5. Such efforts ensure that \( H\text{-}mine \) is scalable in both large and medium sized databases and in both sparse and dense data sets. Our comprehensive performance study confirms that \( H\text{-}mine \) is highly scalable and is faster than
Apriori and FP-growth in all occasions.

**Pattern-growth Sequential Pattern Mining**

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

Recent works have also extended the scope from mining sequential patterns to mining partial periodic patterns. Ö zden, et al. [ORS98] introduce cyclic association rules which are essentially partial periodic patterns with perfect periodicity in the sense that each pattern reoccurs in every cycle, with 100% confidence. Han, et al. [HDI99] developed a frequent pattern mining method for mining partial periodicity patterns which are frequent maximal patterns where each pattern appears in a fixed period with a fixed set of offsets, and with sufficient support.

Almost all of the above proposed methods for mining sequential patterns and other time-related frequent patterns are Apriori-like, i.e., based on the Apriori heuristic, which states the fact that any super-pattern of an infrequent pattern cannot be frequent, and a candidate generation-and-test paradigm proposed in association mining [AS94].

A typical Apriori-like sequential pattern mining method, such as GSP [SA96b], adopts a multiple-pass, candidate generation-and-test approach, 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 item or multiple items. The number of items in a sequence is called the length of the sequence. So, all the candidate sequences in a pass will have the same length. The scan of the database in one pass finds the support for each candidate sequence. All the candidates 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 when no candidate sequence can be generated.

The Apriori-like sequential pattern mining method, though reduces search space, bears three nontrivial, inherent costs which are independent of detailed implementation techniques.

- **A huge set of candidate sequences could be generated in a large sequence database.** Since the set of candidate sequences includes all the possible permutations of the elements and repetition of items in a sequence, the Apriori-based method may generate a very large set of candidate sequences even for a moderate seed set.

- **Many database scans in mining.** Since the length of each candidate sequence...
grows by one at each database scan, to find a sequential pattern \( (abc)(abc)(abc)(abc) \), the Apriori-based method must scan the database at least 15 times. This bears some nontrivial cost.

- The Apriori-based method encounters difficulty when mining long sequential patterns.

This is because a long sequential pattern must grow up from a huge number of short sequential patterns, but the number of such candidate sequences is exponential to the length of the sequential patterns to be mined.

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 stepwise candidate sequence generation and test. Given the success of pattern-growth methods in frequent pattern mining, can we develop a pattern-growth method for sequential pattern mining which absorbs the spirit of Apriori but avoid or substantially reduce the expensive candidate generation and test? We systematically develop pattern-growth methods for mining sequential patterns efficiently. The new methods are non-Apriori and apply a divide-and-conquer, pattern-growth principle. The general idea is that sequence databases are recursively projected into a set of smaller projected databases and sequential patterns are grown in each projected databases by exploring only local frequent fragments. Two pattern growth schemes, FreeSpan (for Frequent pattern-projected Sequential pattern mining) and PrefixSpan (for Prefix-projected Sequential pattern mining), are proposed. They mine the complete set of sequential patterns but greatly reduce the efforts of candidate subsequence generation. To further improve mining efficiency, three kinds of database projections: level-by-level projection, bi-level projection, and pseudo-projection, are explored. A comprehensive performance study shows that FreeSpan and PrefixSpan outperform Apriori-based GSP algorithm and an integrated PrefixSpan is the fastest one in mining large sequence databases.

Conclusion

As our world is now in its information era, a huge amount of data is accumulated every day. A real universal challenge is to find actionable knowledge from a large amount of data. Data mining is an emerging research direction to meet this challenge. Many kinds of knowledge (patterns) can be mined from various data. In this thesis, we focus on the problem of mining frequent patterns efficiently and effectively, and develop a new class of pattern-growth methods.

Bibliography


1. R. Agrawal, T. Imielinski, and A. Swami. Mining


15. J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceberg cubes with complex measures. In *Proc. 2001

17. 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.
