A Hybrid Approach to Mining Frequent Sequential Patterns

Erich Allen Peterson
Department of Applied Science
University of Arkansas at Little Rock
2801 S. University Ave.
Little Rock, AR 72204

Peiyi Tang
Department of Computer Science
University of Arkansas at Little Rock
2801 S. University Ave.
Little Rock, AR 72204

ABSTRACT

The mining of frequent sequential patterns has been a hot and well studied area—under the broad umbrella of research known as KDD (Knowledge Discovery and Data Mining)—for well over a decade. Yet researchers are still uncovering interesting problems, new algorithms, and ways to improve upon existing methods. In this paper, we marry state-of-the-art frequent sequential pattern mining algorithms (e.g., SPAM, FOF, PrefixSpan), data structures (e.g., aggregate tree, bitmap), and other tried-and-true methods for candidate generation (e.g., apriori), in an attempt to derive a new algorithm with the best qualities of the aforementioned algorithms. In this paper, we disseminate the new algorithm created, lessons learned, and future work to be done.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms

Algorithms, Design

Keywords

Projection Database, Frequent Patterns, Apriori, Pattern Growth

1. INTRODUCTION

Mining frequent sequential patterns has attracted a significant amount of research. Popularity in this area is primarily due to its large area of applicability, and that it is the first (and hardest) of a two step process in mining association rules. Ever since Agrawal and Srikant [1] introduced their unique approach and seminal work on the subject of itemset data mining, researchers have been able to use it as a springboard to solve the more general problem of mining frequent sequential patterns. In [1], the authors introduced the algorithms known as Apriori, AprioriTid, and AprioriHybrid (henceforth combined and referred to simply as apriori). Later, other researchers developed what are now state-of-the-art apriori-based algorithms such as GSP [7], SPAM [2], and SPADE [9]; whose central aim was to reduce the time and/or space complexities inherent in mining frequent sequential patterns (N.B. the time complexity of all known algorithms is exponential). Apriori-based algorithms usually use a candidate "generate-and-test" type of approach, which exploits the *downward closure* property: if frequent itemset α is not frequent, then any superset of α must not be frequent either.

Other algorithms have been proposed which do not follow the traditional apriori-based approach, but instead follow a pattern-growth based approach. Some of these algorithms include PrefixSpan [5] and FreeSpan [3]. Pattern-growth algorithms take a more incremental approach in generating possible frequent sequences, and use what might be called a divide-and-conquer approach. Pattern-growth algorithms make projections of the database in an attempt to reduce the search space.

Our motivation for this research, was an attempt to marry state-of-the-art frequent sequential pattern mining algorithms (e.g., SPAM, FOF, PrefixSpan), data structures (e.g., aggregate tree, bitmap), and other tried-and-true methods for candidate generation (e.g., apriori), in an attempt to derive a new algorithm with the best qualities of the aforementioned algorithms. For candidate generation we chose an apriori-based method, because of its seemingly optimal nature. Of course, we must extend this to frequent sequential pattern mining, which can be seen as mining frequent sequential itemsets. We also chose the FOF [6] algorithm for its overall pattern-growth style mining method and aggregate tree data structure similar to the FP-tree [4]. The FOF algorithm makes pseudo-projections of the database, by pointing to subtrees within an aggregate tree representing the sequence database. Lastly, SPAM's idea of using a bitmap to represent each itemset was used.

The rest of the paper is broken down as follows: Section 2 and Section 3 disseminates some key concepts and provides a problem statement for mining frequent sequential patterns; Section 4 presents the new algorithm; and finally, Section 5 discusses some future work and concludes the paper.

2. FREQUENT SEQUENTIAL PATTERN MINING

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items. An itemset α is a subset of I. A sequence s is an ordered list of itemsets that is denoted as $\langle s_1, s_2, \ldots, s_j\rangle$, where each s_k is an itemset

ACMSE '09 March 19-21, 2009, Clemson, SC, USA.

and $1 \leq k \leq j$ (j is called the length of the sequence). A sequence $A = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ is a subsequence of another sequence $B = \langle \beta_1, \beta_2, \ldots, \beta_m \rangle$, denoted as $A \sqsubseteq B$, iff $n \leq m$ and there exist integers $1 \leq k_1 < k_2 < \cdots < k_n \leq m$ such that $\alpha_1 \subseteq \beta_{k_1}, \alpha_2 \subseteq \beta_{k_2}, \cdots, \alpha_n \subseteq \beta_{k_n}$. A sequence database S contains a set of tuples $\langle sid, s \rangle$, where sid is a unique identifier for the sequence and s is a sequence. For example, (1) shows a sample database to be used throughout this paper (minus the *sids*)[1]:

$$S = \{\langle ac, ab, c \rangle, \langle ac, ab, a \rangle, \langle c, abc, c \rangle, \langle c, abc, a \rangle\} \qquad (1)$$

A sequential pattern is also a sequence. The *support* of sequential pattern p in database S is defined as:

$$Support_S(p) = |\{\langle sid, s \rangle \mid \langle sid, s \rangle \in S \wedge p \sqsubseteq s\}|\,.$$

Given a positive integer η between 0 and $|S|$, known as the support threshold, the sequential pattern p is considered frequent iff $Support_S(p) \geq \eta$. Therefore, given a sequence database S and support threshold η, the problem of mining frequent sequential patterns is to find all sequential patterns whose support in database S is greater than or equal to η.

3. PATTERN GROWTH AND APRIORI FUNDAMENTALS

All pattern-growth algorithms are based on the fundamental principle of conditional searching. In essence, pattern-growth algorithms find frequent sequential patterns by traversing the search space tree in the depth-first manner, and continuously creating projections of the original database in an attempt to reduce the size of the database. A formal framework for pattern-growth algorithms for mining frequent sequential patterns with singleton itemsets was presented in [8]. We briefly extend it for general sequential patterns as follows. Given an itemset α and a sequence s that supports $\langle \alpha \rangle$ (i.e., $\langle \alpha \rangle \sqsubseteq s$), the *$\alpha$-prefix* of s is the prefix of s, which includes the left most itemset in the sequence, to the *first* itemset that contains α inclusive; the first itemset that contains α is known as the *first-occurrence* of α. For example, the bc-prefix of sequence $\langle b, \underline{bc}d, ab \rangle$ is $\langle b, \underline{bc}d \rangle$ and itemset $\underline{bc}d$ is the first-occurrence of bc. The *α-projection* of s is what is left after the α-prefix is removed. In the example above, the bc-projection of sequence $\langle b, \underline{bc}d, ab \rangle$ is $\langle ab \rangle$. Note that if α is only contained in the last itemset of s, the α-prefix is s itself and the α-projection is the empty sequence $\langle \rangle$. For example, the a-projection of $\langle b, bcd, \underline{a}b \rangle$ is $\langle \rangle$. Given a sequence database S and an itemset $\alpha \subseteq I$, the *α-projection database* of S, denoted as S_α, is the multi-set of α-projections of the sequences in S that support $\langle \alpha \rangle$. That is,

$$S_a = \{\alpha\text{-projection of } s \mid \langle \alpha \rangle \sqsubseteq s \wedge s \in S\}$$

For example, the ac-projection database of the sequence database in (1) would be $S_\alpha = \{\langle ab, c \rangle, \langle ab, a \rangle, \langle c \rangle, \langle a \rangle\}$. One can prove that, given an itemset $\alpha \subseteq I$ and sequence database S, the support of sequential pattern p (including the empty pattern $\langle \rangle$) in the α-projection database S_α is equal to the support of pattern $\langle \alpha, p \rangle$[2] in the original database S. That, is

$$Support_{S_\alpha}(p) = Support_S(\langle \alpha, p \rangle).$$

Based on this, the set of all non-empty frequent sequential patterns can be found by calling the recursive function in Figure 1 with *Pattern-Grow*($\langle \rangle$, S, η) [3].

The algorithm in Figure 1, however, does not take advantage of the downward closure property of frequent sequential patterns. That is, if a sequential pattern p is not frequent in the database S (i.e., $Support_S(p) < \eta$), then any super-sequence $p' \sqsupseteq p$ will not be frequent either, because $Support_S(p') \leq Support_S(p) < \eta$. In particular, if single-itemset sequential pattern $\langle \alpha \rangle$ is not frequent in S, any $\langle \alpha' \rangle$ such that $\alpha' \supseteq \alpha$ is not frequent either, because $\langle \alpha \rangle \sqsubseteq \langle \alpha' \rangle$. In other words, if we know that $Support_S(\langle \alpha \rangle) < \eta$, we do not need to calculate $Support_S(\langle \alpha' \rangle)$. To prune away these infrequent single-itemset sequences $\langle \alpha' \rangle$, we extend the abstract algorithm in Figure 1 with apriori (downward closure) pruning as shown in Figure 2. The *Apriori-Gen* function called in line 7 roughly uses the same method of candidate generation and pruning as can be found in the original *apriori-gen* algorithm presented in [1]—with the exception of bitmaps being used to represent itemsets. The *Apriori-Gen* algorithm is summarized in Figure 4. The algorithm in Figure 2 can be regarded as the marriage of pattern-growth mining and apriori (pruning and candidate generation) techniques.

```
function Pattern-Grow(pattern q, database S, int η) {
  F ← ∅;
  for each α ∈ 𝒫(I) − {∅} do
    if Support_S(⟨α⟩) ≥ η then
      F ← F ∪ {⟨q, α⟩};
      Let S_α be the α-projection database of S;
      F ← F ∪ Pattern-Grow(⟨q, α⟩, S_α, η);
    endif
  endfor
  return F;
}
```

Figure 1: Abstract Pattern-Growth Mining Algorithm

4. HYBRIDMINE ALGORITHM

In this section, we present our new algorithm *HybridMine* and its constituent data structures. Our new algorithm is an implementation of the abstract *Pattern-Grow-Apriori* mining algorithm we formulated in Figure 2. In that abstract algorithm, the marriage of pattern-growth and apriori-based methods are manifest.

4.1 Data Structures

Our new algorithm uses an implementation of what is known as an aggregate tree (similar to the FOF algorithm[4]).

[1] All itemsets are represented by a string of the items in their lexicographical order. For example itemset $\{a, b, c\}$ is written as *abc*.

[2] If sequential pattern p is $\langle b_1, \cdots, b_n \rangle$, $\langle \alpha, p \rangle$ is the pattern with α as the first itemset followed by the sequence of itemsets of p (i.e., by $\langle \alpha, p \rangle$ we mean $\langle \alpha, b_1, \cdots, b_n \rangle$).

[3] Note that $\mathcal{P}(I)$ refers to the powerset of I. Also note that if sequential pattern q is $\langle b_1, \cdots, b_n \rangle$, $\langle q, \alpha \rangle$ is $\langle b_1, \cdots, b_n, \alpha \rangle$. If q is empty sequence $\langle \rangle$, $\langle q, \alpha \rangle$ is $\langle \alpha \rangle$.

[4] The original FOF algorithm was developed for a special case of frequent sequential pattern mining, in which each itemset in a sequence is a singleton. Thus, we extend the aggregate tree to store at each node an itemset.

```
function Pattern-Grow-Apriori(pattern $q$, database $S$, int $\eta$) {
   $F \leftarrow \emptyset$;
   $C_1 \leftarrow I$;
   for $i \leftarrow 1$ to $|I|$ do
      if $C_i \neq \emptyset$ do
         if $i > 1$ then
            $C_i \leftarrow$ Apriori-Gen($C_{i-1}$);
         endif
         $C'_i \leftarrow \emptyset$;
         for each $\alpha \in C_i$ do
            if $Support_S(\langle\alpha\rangle) \geq \eta$ then
               $F \leftarrow F \cup \{\langle q, \alpha\rangle\}$;
               Let $S_\alpha$ be the $\alpha$-projection database of $S$;
               $F \leftarrow F \cup$ Pattern-Grow-Apriori($\langle q, \alpha\rangle$ , $S_\alpha, \eta$);
               $C'_i \leftarrow C'_i \cup \{\alpha\}$;
            endif
         endfor
         $C_i \leftarrow C'_i$;
      endif
   endfor
   return $F$;
}
```

Figure 2: Abstract Pattern-Growth-Apriori Mining Algorithm

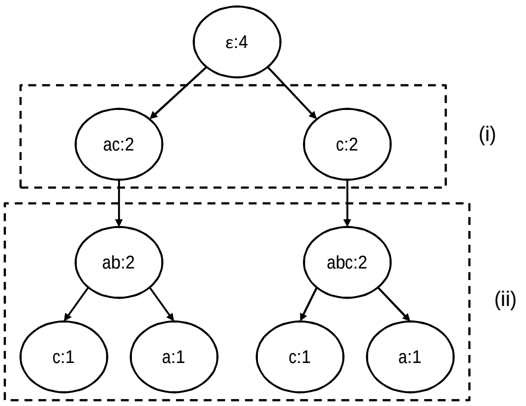

Figure 3: Aggregate Tree Example

```
function Apriori-Gen (itemsets $C_{i-1}$) {
   insert into $C_i$
   select $p.item_1, p.item_2, \ldots, p.item_{i-1}, q.item_{i-1}$
   from $C_{i-1}$ $p$ , $C_{i-1}$ $q$
   where $p.item_1 = q.item_1, \ldots p.item_{i-2} = q.item_{i-2}$,
         $p.item_{i-1} < q.item_{i-1}$;

   forall itemsets $c \in C_i$
      forall $(i-1)$ subsets $s$ of $c$ do
         if $(s \notin C_{i-1})$ then
            delete $c$ from $C_i$;
   return $C_i$;
}
```

Figure 4: Apriori-Gen

The aggregate tree of the original database can be constructed by entering each sequence in S into the initial tree, which has only one root node pointed to by a pointer *current-node*. For a sequence $A = \langle\alpha_1, \alpha_2, \ldots, \alpha_n\rangle$ the construction enters each α_i using the following algorithm. If *current-node* has a child with label α_i, increase the count of that child and make *current-node* point to that child. Otherwise, create a new child node with label α_i and count 1, and make *current-node* point to this new child node. Figure 3 shows the aggregate tree of the original example sequence database in (1). The label of each node is represented by a bitmap of n bits, one for each item in $I = \{i_1, i_2, \ldots, i_n\}$. If item i_j is resident within the itemset of the label, the j^{th} element of the bitmap is set to one.

The new algorithm also uses the FOF (First-Occurrence Forests) data structure to represent projection databases. A FOF is simply a list of the pointers to mutually exclusive nodes within the aggregate tree. Thus, collectively the FOF structure can be seen as representing a forest of aggregate trees. After the aggregate tree of the original database is built, an initial FOF structure it created, which only contains one pointer to the root to the original aggregate tree.

If a database S (be it the original or a projection) is represented by a forest of aggregate trees, the task of counting the support of $\langle\alpha\rangle$ for an itemset α and finding its projection, amounts to finding the so-called first-occurrences of the itemset in the forest. In particular, a node in an aggregate tree is a *first-occurrence* of itemset α if α is a subset of its label l, $\alpha \subseteq l$, and α is not a subset of any of its ancestor node's labels. For example, the aggregate tree in Figure 3 has two first-occurrences of c: ac:2 in the left subtree and c:2 in the right subtree. All the other nodes which contain c (i.e., c is a subset of) are not first-occurrences, because they are descendants of the first-occurrences of c. If the first-occurrences of an itemset α are found within the aggregate trees of S, the sum of the counts of those nodes effectively computes the support of $\langle\alpha\rangle$ within S (i.e., $Support_S(\langle\alpha\rangle)$). For example, the first-occurrences of the itemset c are shown in region (i) of Figure 3 (i.e., nodes ac:2 and c:2) and thus has a support of $2+2=4$. Moreover, the subtrees rooted at the children of all the first-occurrences of α form the forest of aggregate trees representing the α-projection of S (i.e., S_α). In the example in Figure 3, the c-projection is therefore the forest shown in region (ii).

Because itemsets are implemented as bitmap data structures, found in each node of the aggregate tree, we can find first-occurrences very easily using fast bit-wise operations. For example, to find out whether an arbitrary Bitmap $Map2$ supports another Bitmap $Map1$ (i.e., $Map1 \subseteq Map2$), we simply perform the following bit-wise operation/test:

$$\neg(\neg Map1 \vee (Map2 \wedge Map1)) \equiv 0$$

If the above statement is true, then $Map2$ does indeed support $Map1$, and a first-occurrence of $Map1$ has been found.

4.2 Hybrid Mining Algorithm

Aggregating all the algorithms and data structures disseminated previously, the new $HybridMine$ algorithm is presented in Figures 5 and 4[5].

Within the $HybridMine$ algorithm, both argument S and variable T are of type FOF (i.e., a list of pointers to first-occurrence aggregate trees). C_i $(i = 1, 2, \ldots)$ are the candi-

[5] The algorithm in Figure 4 is based on the original Apriori algorithm by Agrawal and Srikant, and maintains some of its SQL syntax for brevity.

```
function HybridMine(pattern $q$, database $S$, int $\eta$) {
$Q \leftarrow \emptyset$;
$C_1 \leftarrow I$;
for $i \leftarrow 1$ to $|I|$ do
    if (($i > 1$ AND $C_{i-1}$ is not empty) OR $i = 1$) then
        if $i > 1$ then
            $C_i \leftarrow \textit{Apriori-Gen}(C_{i-1})$;
        endif
        for $p \leftarrow 1$ to $C_i.Size()$ do
            Clear $T$.
11:         for each node $s$ in the FOF list $S$ do
12:             for each child $c$ of $s$
                    Find the set of first-occurrences $F$ of $C_{i,p}$
                        within the subtree rooted at $c$;
                    for each first-occurrence $f$ in $F$
                        Push the pointer to $f$ onto list $T$;
                        $T.Count = T.Count + f.Count$;
                    endfor
                endfor
            endfor
            if ($T.Count \geq \eta$) then
                $Q \leftarrow Q \cup \{\langle q, C_{i,p}\rangle\}$;
                $Q \leftarrow Q \cup HybridMine(\langle q, C_{i,p}\rangle, T, \eta)$;
            else
                Remove $C_{i,p}$ from $C_i$;
            endif
        endfor
    endif
endfor
return $Q$;
};
```

Figure 5: Hybrid Mining Algorithm

date sets of itemsets from I with i items. The *Apriori-Gen* algorithm shown in Figure 4 generates the candidate itemsets of size i from the itemsets of size $i-1$ in C_{i-1} using the apriori algorithm. $C_i.Size()$ refers to the number of itemsets in C_i (i.e. $|C_i|$). For each candidate itemset in C_i, denoted as $C_{i,p}$ ($1 \leq p \leq C_i.Size()$), the algorithm finds all the first occurrences of $C_{i,p}$ in database S. Remember all the aggregate trees representing projection database S are the subtrees rooted at the *children* of the nodes pointed by the FOF structure S. The two nested **for** loops at lines 11 and 12 are simply to find all the first-occurrences F of $C_{i,p}$. T is the FOF structure that is used to hold the first-occurrences found, by pointing to them. $T.Count$ is used to accumulate the counts (stored in $f.Count$) of all these first-occurrences. After all the first-occurrences are found and their counts accumulated, $T.Count$ is the support of $\langle C_{i,p}\rangle$ in database S. If $T.Count \geq \eta$, we know that $\langle C_{i,p}\rangle$ is frequent in S and the same function is called recursively using the projection database represented by T with the grown prefix pattern $\langle p, C_{i,p}\rangle$. Otherwise, $C_{i,p}$ is removed from C_i because it is infrequent.

5. CONCLUSIONS AND FUTURE WORK

We have presented a new frequent sequential pattern mining algorithm which combines the favorable attributes of several other state-of-the-art algorithms. Some preliminary experimental results show that the new algorithm would most likely outperform GSP and possibly FreeSpan. However, the algorithm would in all likely hood not outperform SPAM, PrefixSpan, or SPADE.

Our study of the algorithm points to one of the reasons for this: the apriori algorithm's large number of executions during the mining process (each time a frequent pattern is found, the apriori algorithm is called again). In a worst-case scenario, in which all sequential patterns are frequent and all sequences are of the same length, the number of candidates generated would be $\sum_{i=0}^{l}(2^{|I|}-1)^i$, where l is the length of the sequences. Of course, this is rarely the case, yet the number of apriori calls can be quite high when using real-world datasets. The aforementioned problem has enticed the formulation of new approaches to the problem. Some of those might include using a similar mechanism for generating new candidates as found in SPAM, which does not use an apriori approach, but rather grows each sequence incrementally. Another method found in SPAM is the passing of only frequently found itemsets to the next recursive call, thus allowing for the pruning of potentially many more infrequent sequences.

6. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th Internatioanl Conference on Very Large Data Bases*, pages 487–499, 1994.

[2] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 429–435, New York, NY, USA, 2002. ACM.

[3] 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: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 355–359, New York, NY, USA, 2000. ACM.

[4] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, 2000.

[5] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proceedings of the 17th International Conference on Data Engineering*, pages 215–224, Washington, DC, USA, 2001. IEEE Computer Society.

[6] E. A. Peterson and P. Tang. Mining frequent sequential patterns with first-occurrence forests. In *In Proceedings of the 46th ACM Southeastern Conference (ACMSE)*, 2008.

[7] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *EDBT '96: Proceedings of the 5th International Conference on Extending Database Technology*, pages 3–17, London, UK, 1996. Springer-Verlag.

[8] P. Tang, M. P. Turkia, and K. A. Gallivan. Mining web access patterns with first-occurrence linked WAP-trees. In *Proceedings of the 16th International Conference on Software Engineering and Data Engineering (SEDE'07)*, pages 247–252, Las Vegas, USA, July 2007.

[9] M. J. Zaki. Spade: an efficient algorithm for mining frequent sequences. In *Machine Learning Journal, special issue on Unsupervised Learning*, pages 31–60, 2001.