Efficient Mining of Closed Patterns with Tough Constraints

Jianyong Wang
Digital Technology Center
University of Minnesota
Outline

• Motivation
  - Why is frequent pattern mining so fundamental in data mining?

• Recent progress in pattern discovery
  - Closed/Maximal frequent pattern mining
  - Constrained pattern discovery

• The BAMBOO algorithm
  - LPCLOSEST
  - Search space pruning
  - Further optimizations

• Experimental results
  - Comparison with LPMiner, CFP-tree and CLOSET+
  - Scalability test

• Conclusion
Part _ Recent progress in pattern discovery
- A survey

• Motivation
• Recent progress in pattern discovery
  - Closed/Maximal frequent pattern mining
  - Constrained pattern discovery
• The limitations with the current solutions
What is a frequent pattern?

- Pattern (set of items, sequence, etc.) that occurs together frequently in a database [AIS93]
  
  » Given a support threshold, \( min_{sup} \), an itemset \( X \) is frequent if \( Sup(X) \geq min_{sup} \)

- Finding regularities in data
  
  » What products are often purchased together? — beer and diapers?!
  
  » What are the subsequent purchases after buying a PC?
  
  » ……
Why is frequent pattern mining so fundamental in data mining

- Foundation for several essential data mining tasks
  - Association, correlation, causality analysis
  - Association based classification and clustering
    » [Liu98] Integrating Classification and Association Rule Mining. KDD98.
    » [Li01] CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules. ICDM01.
    » [Yin03] CPAR: Classification based on Predictive Associative Rules. SDM’03
Problem with the frequent pattern mining algorithm

- **Observation**
  - A lot of existing frequent itemset mining algorithms
    » Apriori, FP-growth, OP, PPmine, AFOPT, Inverted matrix, …

- **Problem**
  » Too many frequent patterns if the support threshold is low

- **Popular solutions**
  » Closed (or Maximal) frequent patterns
  » Constrained frequent pattern mining
Maximal frequent itemset Mining

• Mining maximally frequent itemsets
  - Maximally frequent itemset \(X\)
    » No superset of \(X\) is frequent
    » E.g., the set of frequent itemsets \(=\{a:5, b:6, c:4, ab:4, bc:3, ac:4, abc:2\}\), then only itemset abc is maximally frequent.
  - More concise result set and more efficient algorithm
  - But may lose information
    » We cannot get the exact support of each frequent itemset
Closed itemset Mining

- Mining frequent closed itemsets
  - Closed itemset $Y$
    » There exists no itemset $Y'$, such that $Sup(Y) = Sup(Y')$ and $Y' \supseteq Y$ hold.

- Typical frequent closed itemset mining algorithms
  » A-Close, CLOSET, MAFIA, CHARM, CFP-tree, CARPENTER, CLOSET+

- A bunch of different mining strategies/techniques
  » Search order, data representation, data compression, search space pruning, pattern closure checking schemes
Closed itemset Mining Strategies

• Search order

- Breadth-first search vs. depth-first search

  » Depth-first search is more efficient than breadth-first search for mining long patterns
Closed itemset Mining Strategies

• Data representation

  - Horizontal vs. vertical data formats

    » Need further performance study to compare these two schemes in terms of scalability, runtime and space usage efficiency

|   | I1 | I2 | I3 | I4 | I5 | ...
|---|----|----|----|----|----|---
| T1 | 1  | 1  | 1  | 0  | 1  | ...
| T2 | 0  | 1  | 1  | 1  | 0  | ...
| T3 | 1  | 0  | 1  | 0  | 1  | ...
| T4 | 1  | 0  | 1  | 1  | 1  | ...
| ...| ...| ...| ...| ...| ...| ...
Closed itemset Mining Strategies

- Data compression technique
  - FP-tree
  - diffset: Differences in the tids of a candidate pattern from its parent pattern

<table>
<thead>
<tr>
<th>tid</th>
<th>itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>f, c, a, m, p</td>
</tr>
<tr>
<td>20</td>
<td>f, c, a, m, p</td>
</tr>
<tr>
<td>30</td>
<td>f, c, a, b, m</td>
</tr>
<tr>
<td>40</td>
<td>f, b</td>
</tr>
<tr>
<td>50</td>
<td>c, b, p</td>
</tr>
</tbody>
</table>

Database

FP-tree
Closed itemset Mining Strategies

• Existing search space pruning methods
  - Item merging
    » If every transaction containing itemset $X$ also contains itemset $Y$ but not any proper superset of $Y$, then $X \cup Y$ forms a frequent closed itemset and there’s no need to search any itemset containing $X$ but no $Y$
  - Sub-itemset pruning
    » If prefix itemset $X$ is a proper subset of an already found frequent closed itemset $Y$ and $\text{sup}(X) = \text{sup}(Y)$, prefix $X$ can be safely pruned from the search space
Closed itemset Mining Strategies

- Database projection methods
  - E.g., CLOSET+ adopts two projection methods
    - Bottom-up physical projection
    - Top-down pseudo projection

<table>
<thead>
<tr>
<th>tid</th>
<th>item list</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>f, c, a, m, p</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>f, c, a, m, p</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>f, c, a, b, m</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>f, b</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>c, b, p</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tid</th>
<th>item list</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>f, c, a, m</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>f, c, a, m, p</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>f, c, a, b, m</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>f, b</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>c, b, p</td>
<td></td>
</tr>
</tbody>
</table>
Closed itemset Mining Strategies

• Subset checking techniques
  - Used to check whether a pattern is closed or not
  - Index structure
    » CHARM: Sum of transaction IDs
    » CLOSET+
      (1) 2-level hash indexed result tree structure for dense datasets
      (2) Pseudo projection based upward-checking for sparse datasets
Two-level hash-indexed result tree

- Compressed result tree structure
- Search space shrinking for subset checking
  - If itemset $S_c$ can be absorbed by another already mined itemset $S_a$, they have the following relationships:
    1) $\text{sup}(S_c) = \text{sup}(S_a)$
    2) $\text{length}(S_c) < \text{length}(S_a)$
    3) $\forall i, i \in S_c \Rightarrow i \in S_a$
  - Measures to enhance the checking
    » Two-level hash indices – support and itemID
    » Record length information in each result tree node
Two-level hash-indexed result tree
Pseudo-projection based upward checking

- Result-tree may consume much memory for sparse datasets
- Subset checking without maintenance of history itemsets
  - For a certain prefix $X$, as long as we can find any item which (1) appears in each prefix path w.r.t. prefix $X$, and (2) does not belong to $X$, any itemset with prefix $X$ will be non-closed, otherwise, if there’s no such item, the union of $X$ and the complete set of its locally frequent items with support $\text{sup}(X)$ will form a closed itemset.
Pseudo-projection based upward checking

- E.g., Prefix $X='c:4'$

<table>
<thead>
<tr>
<th>f</th>
<th>c</th>
<th>a</th>
<th>b</th>
<th>m</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

- E.g., Prefix $X='am:3'$

![Diagram](image)
Constrained frequent itemset Mining

• Anti-monotone constraint $P$
  - If $A \subseteq B$ then $P(B) \Rightarrow P(A)$ or $\neg P(A) \Rightarrow \neg P(B)$
    » E.g., $\text{Support (S)} \geq \text{min\_sup}$

• Monotone constraint $Q$
  - If $A \subseteq B$ then $Q(A) \Rightarrow Q(B)$ or $\neg Q(B) \Rightarrow \neg Q(A)$
    » E.g., $\text{Support (S)} \leq \text{max\_sup}$
Constrained frequent itemset Mining

• Convertible anti-monotone constraint $P$
  - If there is an order _ according to which $S_1$ is a prefix of $S_2$, then $P(S_2) \Rightarrow P(S_1)$ or $\neg P(S_1) \Rightarrow \neg P(S_2)$
    » E.g., $\text{avg\_price } (S) \geq c$ and descending order

• Convertible monotone constraint $Q$
  - If there is an order _ according to which $S_1$ is a prefix of $S_2$, then $Q(S_1) \Rightarrow Q(S_2)$ or $\neg Q(S_2) \Rightarrow \neg Q(S_1)$
    » E.g., $\text{avg\_price } (S) \geq c$ and ascending order
Dualminer: A dual-pruning algorithm for itemsets with constraints [Bucila02]

Support(a) < min_sup
All its supersets can be pruned

Support(cd) > max_sup
All its subsets can be pruned
Limitations with these solutions

• Closed or Constrained pattern mining are useful in
  - Shrinking the result set
  - Improving the efficiency

• Cannot handle some tough constraints
  - Useful in mining interesting patterns, e.g.,
    » A tough constraint is not an anti-monotone, monotone constraint, or convertible constraint.

• Can we push tough constraints into closed itemset mining?
  - E.g., length-decreasing support constraint
Length-decreasing support constraint

• Definition
  - Given a database $TDB$, function $f(x)$ is a length-decreasing support constraint w.r.t. $TDB$, if:
    $\sup(Y) \geq f(|Y|)$, where $|Y|$ is the length of itemset $Y$
  - An itemset $Y$ is valid (or frequent) if:
    $|TDB| \geq f(x) \geq f(x+1) \geq 1$
Some typical length-decreasing support constraint
Part II: Closed itemset mining with length-decreasing support constraint

- The BAMBOO algorithm
  - LPCLOSET
  - Search space pruning
  - Further optimizations

- Experimental results
  - Comparison with LPMiner, CFP-tree and CLOSET+
  - Scalability test
Running example

- \( f_{list} = \langle f:4, c:4, a:3, b:3, m:3, p:3, i:1 \rangle \),
- The length-decreasing support constraint \( f(x) \):
  \[
  f(x) = \begin{cases} 
  4, & \text{if } x \leq 3 \\
  3, & \text{if } 4 \leq x \leq 5 \\
  2, & \text{if } x \geq 6 
  \end{cases}
  \]

Table 1  a transaction database TDB

<table>
<thead>
<tr>
<th>tid</th>
<th>itemset</th>
<th>ordered item list</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>a, c, f, m, p</td>
<td>f, c, a, m, p</td>
</tr>
<tr>
<td>20</td>
<td>a, c, d, f, i, m, p</td>
<td>f, c, a, m, p, i</td>
</tr>
<tr>
<td>30</td>
<td>a, b, c, f, m</td>
<td>f, c, a, b, m</td>
</tr>
<tr>
<td>40</td>
<td>b, f</td>
<td>f, b</td>
</tr>
<tr>
<td>50</td>
<td>b, c, p</td>
<td>c, b, p</td>
</tr>
</tbody>
</table>
LPCLOSET

- FP-tree structure
- Bottom-up divide-and-conquer
- Search space pruning
- Closure checking scheme
- Simply integrating the length-decreasing support constraint
• **FP-tree representation**

![FP-tree diagram](image)

<table>
<thead>
<tr>
<th>Header table H</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>m</td>
<td>3</td>
</tr>
<tr>
<td>p</td>
<td>3</td>
</tr>
</tbody>
</table>
• Bottom-up divide-and-conquer
Search space pruning

- Item-merging
  » Given a prefix $P$, all its locally frequent items with the same support as $P$ can be safely merged with $P$ to form a new prefix
  » E.g., $P = p:3$ with local item set \{c:3, f:2, a:2, m:2\}

new prefix $P' = pc:3$ with local item set \{f:2, a:2, m:2\}
Search space pruning

- Sub-itemset pruning
  - Given a prefix $P$, if it is a sub-itemset of another already mined closed itemset with the same support, prefix $P$ can be safely pruned
  - E.g., $a : 3 \subseteq fcam : 3$

 Prefix $a : 3$ can be pruned
LPCCLOSET

- Closure checking scheme
  - Result tree with sum of transaction IDs as index
LPCLOSEST

• If $\text{sup}(P) \geq f(|P|)$ and $P$ is closed, output $P$ as a closed itemset satisfying the length-decreasing support constraint

• Result-tree pruning
  – No need to store a prefix itemset which cannot pass the checking of the length decreasing support constraint in the result tree
  – Implication
    » Check support constraint prior to pattern closure checking
Search space pruning based on the length-decreasing support constraint

- Previous methods adopted by LPMiner
  1) Transaction pruning
  2) Node pruning
  3) Path pruning

- Smallest Valid Extension (or SVE) property
  » Given an itemset $P$, $SVE(P) = \min(l | f(l) \leq \text{sup}(P))$
  » E.g., if $P=b:3$, $SVE(P)=4$

$$f(x) = \begin{cases} 
  4, & \text{if } x \leq 3 \\
  3, & \text{if } 4 \leq x \leq 5 \\
  2, & \text{if } x \geq 6 
\end{cases}$$
Deeply pruning

- **Invalid Item**
  
  » *Given a prefix* $P$, its projected database $TDB|_P$, *and any item* $x$, we use $\text{COUNT}^x[i]$ to record the total number of occurrences of item $x$ in transactions of $TDB|_P$ no shorter than $i$.

  » *If* $\forall i$, $\text{COUNT}^x[i] < f(i+|P|)$, *item* $x$ *is called invalid and can be safely pruned from* $TDB|_P$.

  » *E.g., in our running example,* $\text{COUNT}^b[i]=3$ ($1 \leq i \leq 2$), $\text{COUNT}^b[i]=2$ ($i=3$), $\text{COUNT}^b[i]=1$ ($4 \leq i \leq 5$), and $\text{COUNT}^b[i]=0$ ($i \geq 6$), *item* $b$ *is invalid.*

$$f(x) = \begin{cases} 
4, & \text{if } x \leq 3 \\
3, & \text{if } 4 \leq x \leq 5 \\
2, & \text{if } x \geq 6 
\end{cases}$$
• **Deeply pruning**
  
  - **Unpromising prefix**
    
    » Given a prefix $P$, its projected database $TDB|_P$, we use $\text{COUNT}^P[i]$ to record the total number of transactions in $TDB|_P$ with a length no shorter than $i$.
    
    » Prefix $P$ is called an unpromising prefix, if $\forall i, \text{COUNT}^P[i] < f(i+|P|)$
    
    » E.g., for prefix $p:3$, its projected database $TDB|_{p:3} = \{<fcam:2>, <cb:1>\}$, we have: $\text{COUNT}^{p:3}[i]=3$ ($1 \leq i \leq 2$), $\text{COUNT}^{p:3}[i]=2$ ($3 \leq i \leq 4$). Prefix $p:3$ is an unpromising prefix and can be pruned.

\[
f(x) = \begin{cases} 
4, & \text{if } x \leq 3 \\
3, & \text{if } 4 \leq x \leq 5 \\
2, & \text{if } x \geq 6 
\end{cases}
\]
Further optimization

- **SVE-based enhancement**
  - *Do we need to count all the projected transactions upon checking whether a prefix $P$ is promising or not?*
  - *No, the transactions with a length shorter than $SVE(P)$ can be ignored!*

- **Binning-based enhancement**
  - *If the maximal transaction length is $max_l$, we need to maintain a total number of $max_l$ counts in order to check whether an item is invalid or not*
  - *Manipulating a non-trivial memory is costly, can we relax a little the memory usage?*
BAMBOO algorithm

• Further optimization
  – Binning-based enhancement
    » We can maintain m counts, where 1 ≤ m ≤ max_l, denoted as COUNT\textsuperscript{x}[1..m], corresponding to length l₁, l₂, …, and lₘ, that is, COUNT\textsuperscript{x}[i] records the number of transactions no shorter than lᵢ in which item x appears.
    » Item x is called a relaxed invalid item if the following holds: COUNT\textsuperscript{x}[m] < f(max_l) and COUNT\textsuperscript{x}[i] < f(lᵢ₊₁)
    » Relaxed invalid items can be safely removed from mining
BAMBOO algorithm

• Use item merging and relaxed invalid item pruning methods to prune unpromising items
• Use transaction pruning method to prune some unpromising transactions
• Build FP-tree
• Mine FP-tree in a bottom-up divide-and-conquer manner
  – Apply unpromising prefix pruning, result-tree pruning and sub-itemset pruning methods to prune the search space
  – If an itemset is closed and pass the length-decreasing support constraint, output it as a valid pattern
• Stop when all the items in the global header table have been mined
Experimental results

- Comparison with LPMiner

Connect dataset
Experimental results

- Comparison with CFP-tree and CLOSET+

**Connect dataset**
Experimental results

- Comparison with CFP-tree and CLOSET+

Gazelle dataset
Experimental results

- Scalability and effectiveness of the pruning methods

T10I4D100k dataset
Conclusions

- How to push deeply the length-decreasing support into closed itemset mining?
  - Tough constraint
  - Downward-closure property cannot hold

- BAMBOO solution
  - Search space pruning
    » unpromising prefix pruning
    » invalid item pruning
  - Further optimization techniques
    » SVE and binning based enhancement
  - Much better performance than LPMiner and CLOSET+
That’s it,
thanks for your attention!