FP-growth

- Challenges of Frequent Pattern Mining
- Improving Apriori
- Fp-growth
  - Fp-tree
  - Mining frequent patterns with FP-tree
- Visualization of Association Rules
Challenges of Frequent Pattern Mining

Challenges
- Multiple scans of transaction database
- Huge number of candidates
- Tedious workload of support counting for candidates

Improving Apriori: general ideas
- Reduce passes of transaction database scans
- **Shrink number of candidates**
- Facilitate support counting of candidates

---

Transaction Database

<table>
<thead>
<tr>
<th>TID</th>
<th>List of item IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>I1, I2, I5</td>
</tr>
<tr>
<td>T200</td>
<td>I2, I4</td>
</tr>
<tr>
<td>T300</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T400</td>
<td>I1, I2, I4</td>
</tr>
<tr>
<td>T500</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T600</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T700</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T800</td>
<td>I1, I2, I3, I5</td>
</tr>
<tr>
<td>T900</td>
<td>I1, I2, I3</td>
</tr>
</tbody>
</table>
Association Rule Mining

- Find all frequent itemsets
- Generate strong association rules from the frequent itemsets
- Apriori algorithm is mining frequent itemsets for Boolean associations rules
Improving Apriori

- Reduce passes of transaction database scans
- **Shrink number of candidates**
- Facilitate support counting of candidates
- Use constraints

### The Apriori Algorithm — Example

**Database D**

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1 3 4</td>
</tr>
<tr>
<td>200</td>
<td>2 3 5</td>
</tr>
<tr>
<td>300</td>
<td>1 2 3 5</td>
</tr>
<tr>
<td>400</td>
<td>2 5</td>
</tr>
</tbody>
</table>

**Scan D**

- **C₁**
  - \{1\}: 2
  - \{2\}: 3
  - \{3\}: 3
  - \{4\}: 1
  - \{5\}: 3

- **L₁**
  - \{1\}: 2
  - \{2\}: 3
  - \{3\}: 3
  - \{5\}: 3

**Scan D**

- **C₂**
  - \{1 2\}: 1
  - \{1 3\}: 2
  - \{1 5\}: 1
  - \{2 3\}: 2
  - \{2 5\}: 3
  - \{3 5\}: 2

- **L₂**
  - \{1 2\}
  - \{1 3\}
  - \{1 5\}
  - \{2 3\}
  - \{2 5\}
  - \{3 5\}

**Scan D**

- **C₃**
  - \{2 3 5\}

- **L₃**
  - \{2 3 5\}: 2

**C₃**

- \{2 3 5\}
Apriori + Constraint

Database D

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1 3 4</td>
</tr>
<tr>
<td>200</td>
<td>2 3 5</td>
</tr>
<tr>
<td>300</td>
<td>1 2 3 5</td>
</tr>
<tr>
<td>400</td>
<td>2 5</td>
</tr>
</tbody>
</table>

Scan D

C₁

itemset sup.

{1} 2
{2} 3
{3} 3
{4} 1
{5} 3

L₁

itemset sup.

{1} 2
{2} 3
{3} 3
{4} 3
{5} 3

Scan D

C₂

itemset sup.

{1 2} 1
{1 3} 2
{1 5} 1
{2 3} 2
{2 5} 3
{3 5} 2

Constraint:
Sum{S.price} < 5

Scan D

C₃

itemset sup.

{2 3 5} 2

Push an Anti-monotone Constraint

Deep

Database D

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1 3 4</td>
</tr>
<tr>
<td>200</td>
<td>2 3 5</td>
</tr>
<tr>
<td>300</td>
<td>1 2 3 5</td>
</tr>
<tr>
<td>400</td>
<td>2 5</td>
</tr>
</tbody>
</table>

Scan D

C₁

itemset sup.

{1} 2
{2} 3
{3} 3
{4} 1
{5} 3

L₁

itemset sup.

{1} 2
{2} 3
{3} 3
{4} 3
{5} 3

Scan D

C₂

itemset sup.

{1 2} 1
{1 3} 2
{1 5} 1
{2 3} 2
{2 5} 3
{3 5} 2

Constraint:
Sum{S.price} < 5

Scan D

C₃

itemset sup.

{2 3 5} 2
Hash-based technique

- The basic idea in hash coding is to determine the address of the stored item as some simple arithmetic function content.
- Map onto a subspace of allocated addresses using a hash function.
- Assume the allocated address range from $b$ to $n+b-1$, the hashing function may take $h = (a \mod n) + b$.
- In order to create a good pseudorandom number, $n$ ought to be prime.

- Two different keywords may have equal hash addresses.
- Partition the memory into buckets, and to address each bucket.
  - One address is mapped into one bucket.
When scanning each transition in the database to generate frequent 1-itemsets, we can generate all the 2-itemsets for each transition and hash them into different buckets of the hash table.

We use $h=a \mod n$, $a$ address, $n <$ the size of $C_2$.

A 2-itemset whose bucket count in the hash table is below the support threshold cannot be frequent, and should be removed from the candidate set.
Transaction reduction

- A transaction which does not contain frequent k-itemsets should be removed from the database for further scans

Partitioning

- First scan:
  - Subdivide the transactions of database D into n non overlapping partitions
  - If the minimum support in D is \( \text{min\_sup} \), then the minimum support for a partition is \( \text{min\_sup} \times \text{number of transactions in that partition} \)
  - Local frequent items are determined
  - A local frequent item may not be a frequent item in D
- Second scan:
  - Frequent items are determined from the local frequent items
Partitioning

First scan:
- Subdivide the transactions of database D into n non overlapping partitions
- If the minimum support in D is min_sup, then the minimum support for a partition is

\[ \text{min}_\text{sup} \times \frac{\text{number of transactions in D}}{\text{number of transactions in that partition}} \]

- Local frequent items are determined
- A local frequent item may not be a frequent item in D

Second scan:
- Frequent items are determined from the local frequent items

Sampling

- Pick a random sample S of D
- Search for local frequent items in S
  - Use a lower support threshold
  - Determine frequent items from the local frequent items
  - Frequent items of D may be missed

- For completeness a second scan is done
Is Apriori fast enough?

- Basics of Apriori algorithm
  - Use frequent (k-1)-itemsets to generate k-itemsets candidates
  - Scan the databases to determine frequent k-itemsets

- It is costly to handle a huge number of candidate sets

- If there are $10^4$ frequent 1-itemsets, the Apriori algorithm will need to generate more than $10^7$ 2-itemsets and test their frequencies
To discover a 100-itemset

2^{100}-1 candidates have to be generated

\[ 2^{100}-1 = 1.27 \times 10^{30} \]

(Do you know how big this number is?)

\[ \ldots \]

- \( 7 \times 10^{27} \approx \) number of atoms of a person
- \( 6 \times 10^{49} \approx \) number of atoms of the earth
- \( 10^{78} \approx \) number of atoms of the universe

---

**Bottleneck of Apriori**

- Mining long patterns needs many passes of scanning and generates lots of candidates
- Bottleneck: \texttt{candidate-generation-and-test}

- Can we avoid \texttt{candidate generation}?
- May some new data structure help?
Mining Frequent Patterns Without *Candidate* Generation

- Grow long patterns from short ones using local frequent items
  - “abc” is a frequent pattern
  - Get all transactions having “abc”: DB|abc
  - “d” is a local frequent item in DB|abc → abcd is a frequent pattern

Construct FP-tree from a Transaction Database

<table>
<thead>
<tr>
<th>TID</th>
<th>Items bought (ordered) frequent items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{f, a, c, d, g, i, m, p}</td>
</tr>
<tr>
<td>200</td>
<td>{a, b, c, f, l, m, o}</td>
</tr>
<tr>
<td>300</td>
<td>{b, f, h, j, o, w}</td>
</tr>
<tr>
<td>400</td>
<td>{b, c, k, s, p}</td>
</tr>
<tr>
<td>500</td>
<td>{a, f, c, e, l, p, m, n}</td>
</tr>
</tbody>
</table>

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Sort frequent items in frequency descending order, f-list
3. Scan DB again, construct FP-tree

Header Table

<table>
<thead>
<tr>
<th>Item</th>
<th>Frequency</th>
<th>Head</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>4</td>
<td>c:1</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

F-list=f-c-a-b-m-p
Benefits of the FP-tree Structure

- Completeness
  - Preserve complete information for frequent pattern mining
  - Never break a long pattern of any transaction

- Compactness
  - Reduce irrelevant info—infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not count node-links and the count field)
  - There exists examples of databases, where compression ratio could be over 100

The size of the FP-trees bounded by the overall occurrences of the frequent items in the database

The height of the tree is bound by the maximal number of frequent items in a transaction
Partition Patterns and Databases

- Frequent patterns can be partitioned into subsets according to f-list
  
  \[ f\text{-list}=f-c-a-b-m-p \]
  
  Patterns containing p
  Patterns having m but no p
  ... Patterns having c but no a nor b, m, p
  Pattern f

- Completeness and non-redundency

Find Patterns Having p From p-conditional Database

- Starting at the frequent item header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item p
- Accumulate all of transformed prefix paths of item p to form p’s conditional pattern base

<table>
<thead>
<tr>
<th>Item</th>
<th>Frequency</th>
<th>Head</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Conditional pattern bases

<table>
<thead>
<tr>
<th>Item</th>
<th>Cond. Pattern Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>f:3</td>
</tr>
<tr>
<td>a</td>
<td>fc:3</td>
</tr>
<tr>
<td>b</td>
<td>fca:1, f:1, c:1</td>
</tr>
<tr>
<td>m</td>
<td>fca:2, fcab:1</td>
</tr>
<tr>
<td>p</td>
<td>fcam:2, cb:1</td>
</tr>
</tbody>
</table>
From Conditional Pattern-bases to Conditional FP-trees

- For each pattern-base
  - Accumulate the count for each item in the base
  - Construct the FP-tree for the frequent items of the pattern base

**m-conditional pattern base:**
- fca:2, fcab:1

**m-conditional FP-tree**

All frequent patterns relate to m
- m, f, c, a, m, fc, fc, fc, fc

Recursion: Mining Each Conditional FP-tree

Cond. pattern base of “am”: (fc:3)
- f:3
  - c:3
    - a:3

Cond. pattern base of “cm”: (f:3)
- f:3
  - c:3

Cond. pattern base of “cam”: (f:3)
- f:3

Recursion: Mining Each Conditional FP-tree
<table>
<thead>
<tr>
<th>item</th>
<th>conditional pattern base</th>
<th>conditional FP-tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>{(fcam:2), (cb:1)}</td>
<td>{(c:3)}</td>
</tr>
<tr>
<td>m</td>
<td>{(fca:2), (fcab:1)}</td>
<td>{(f:3, c:3, a:3)}</td>
</tr>
<tr>
<td>b</td>
<td>{(fca:1), (f:1), (c:1)}</td>
<td>leer</td>
</tr>
<tr>
<td>a</td>
<td>{(fc:3)}</td>
<td>{(f:3, c:3)}</td>
</tr>
<tr>
<td>c</td>
<td>{(f:3)}</td>
<td>{(f:3)}</td>
</tr>
<tr>
<td>f</td>
<td>leer</td>
<td>leer</td>
</tr>
</tbody>
</table>

**Mining Frequent Patterns With FP-trees**

- **Idea:** Frequent pattern growth
  - Recursively grow frequent patterns by pattern and database partition
- **Method**
  - For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting FP-tree is empty, or it contains only one path—single path will generate all the combinations of its sub-paths, each of which is a frequent pattern
Experiments: FP-Growth vs. Apriori

Data set T25I20D10K

<table>
<thead>
<tr>
<th>Item</th>
<th>conditional pattern base</th>
<th>conditional FP-tree</th>
<th>frequent patterns generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>(I2: 1: 1)</td>
<td>(I2: 2)</td>
<td>I2 15: 2, I1 15: 2, I2 11 15: 2</td>
</tr>
<tr>
<td>13</td>
<td>(I2: 1: 1, I2: 1)</td>
<td>(I2: 2)</td>
<td>I2 14: 2</td>
</tr>
<tr>
<td>11</td>
<td>(I2: 4)</td>
<td>(I2: 4)</td>
<td>I2 11: 4</td>
</tr>
</tbody>
</table>
Advantage when support decrease

No prove

- advantage is shown by experiments with artificial data

Advantages of FP-Growth

- Divide-and-conquer:
  - decompose both the mining task and DB according to the frequent patterns obtained so far
  - leads to focused search of smaller databases

- Other factors
  - no candidate generation, no candidate test
  - compressed database: FP-tree structure
  - no repeated scan of entire database
  - basic ops—counting local freq items and building sub FP-tree, no pattern search and matching
Visualization of Association Rules: Plane Graph

Visualization of Association Rules: Rule Graph
Challenges of Frequent Pattern Mining
Improving Apriori
Fp-growth
- Fp-tree
- Mining frequent patterns with FP-tree
Visualization of Association Rules
- Clustering
- k-means, EM