**Data Mining Techniques:**

Frequent Patterns in Sets and Sequences

Mirek Riedewald

Some slides based on presentations by Han/Kamber and Tan/Steinbach/Kumar

---

### Frequent Pattern Mining Overview

- Basic Concepts and Challenges
- Efficient and Scalable Methods for Frequent Itemsets and Association Rules
- Pattern Interestingness Measures
- Sequence Mining

---

### What Is Frequent Pattern Analysis?

- **Find patterns** (itemset, sequence, structure, etc.) that occur frequently in a data set
- **First proposed for frequent itemsets and association rule mining**
- **Motivation:** Find inherent regularities in data
  - What products were often purchased together?
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to a new drug?
- **Applications**
  - Market basket analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, DNA sequence analysis

---

### Association Rule Mining

- **Given a set of transactions,** find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction

#### Market-Basket transactions

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>

#### Example of Association Rules

- (Diaper) → (Beer), (Milk, Bread) → (Eggs, Coke), (Beer, Bread) → (Milk),

Implication means co-occurrence, not causality!

---

### Definition: Frequent Itemset

- **Itemset**
  - A collection of one or more items
  - Example: \{Milk, Bread, Diaper\}
  - k-itemset: itemset that contains k items
- **Support count (\(\sigma\))**
  - Frequency of occurrence of an itemset
  - E.g., \(\sigma(\{Milk, Bread, Diaper\}) = 2\)
- **Support (\(s\))**
  - Fraction of transactions that contain an itemset
  - E.g., \(s(\{Milk, Bread, Diaper\}) = 2/5\)
- **Frequent Itemset**
  - An itemset whose support is greater than or equal to a \(\minsup\) threshold

---

### Definition: Association Rule

- **Association Rule** = implication expression of the form \(X \rightarrow Y\), where \(X\) and \(Y\) are itemsets
  - Ex.: \{Milk, Diaper\} → \{Beer\}
- **Rule Evaluation Metrics**
  - **Support (\(s\)) = \(P(X \cup Y)\)**
    - Estimated by fraction of transactions that contain both \(X\) and \(Y\)
  - **Confidence (\(c\)) = \(P(Y | X)\)**
    - Estimated by fraction of transactions that contain \(X\) and \(Y\) among all transactions containing \(X\)

---

### Definition: Market Basket transactions

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>

Example: \{Milk, Diaper\} → \{Beer\}

- \(s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|D|} = \frac{2}{5}\)
- \(c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3}\)
### Association Rule Mining Task

- Given a transaction database DB, find all rules having support ≥ minsup and confidence ≥ minconf.
- Brute-force approach:
  - List all possible association rules
  - Compute support and confidence for each rule
  - Remove rules that fail the minsup or minconf thresholds
  - Computationally prohibitive!

### Mining Association Rules

**Example rules:**

- \{Milk, Diaper\} → \{Beer\} (s=0.4, c=0.67)
- \{Milk, Beer\} → \{Diaper\} (s=0.4, c=1.0)
- \{Diaper, Beer\} → \{Milk\} (s=0.4, c=0.67)
- \{Beer\} → \{Milk, Diaper\} (s=0.4, c=0.67)
- \{Diaper\} → \{Milk, Beer\} (s=0.4, c=0.5)
- \{Milk\} → \{Diaper, Beer\} (s=0.4, c=0.5)

**Observations:**
- All the above rules are binary partitions of the same itemset \{Milk, Diaper, Beer\}
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

### Mining Association Rules

- Two-step approach:
  1. Frequent Itemset Generation
     - Generate all itemsets that have support ≥ minsup
  2. Rule Generation
     - Generate high-confidence rules from each frequent itemset, where each rule is a binary partitioning of the frequent itemset
     - Frequent itemset generation is still computationally expensive

### Frequent Itemset Generation

- Brute-force approach:
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the database
  - Match each transaction against every candidate
  - Complexity = O(N*M*w) => expensive since M=2^d

### Frequent Itemset Generation

- Given d items, there are 2^d possible candidate itemsets

### Computational Complexity

- Given d unique items, total number of itemsets = 2^d
- Total number of possible association rules?

\[
R = \sum_{k=2}^{d} \frac{d!}{k!(d-k)!} \sum_{j=0}^{k} \frac{(d-k)!}{j!(d-k-j)!}
\]

If d=6, R = 602 possible rules
Frequent Pattern Mining Overview

- Basic Concepts and Challenges
- Efficient and Scalable Methods for Frequent Itemsets and Association Rules
- Pattern Interestingness Measures
- Sequence Mining

Reducing Number of Candidates

- Apriori principle: If an itemset is frequent, then all of its subsets must also be frequent
- Apriori principle holds due to the following property of the support measure:
  \[ \forall X, Y : (X \subseteq Y) \Rightarrow \sigma(X) \geq \sigma(Y) \]
  - Support of an itemset never exceeds the support of its subsets
  - This is known as the anti-monotone property of support

Illustrating the Apriori Principle

Itemset Count

- \{Bread, Milk\} 3
- \{Bread, Beer\} 2
- \{Bread, Diaper\} 2
- \{Milk, Beer\} 3
- \{Milk, Diaper\} 3
- \{Beer, Diaper\} 3

Minimum Support = 3

If every subset is considered, \(4C_1 + 6C_2 + 6C_3 = 41\) with support-based pruning, \(6 + 6 + 1 = 13\)

Apriori Algorithm

- Generate \(L_1\) = frequent itemsets of length \(k=1\)
- Repeat until no new frequent itemsets are found
  - Generate \(C_{k+1}\), the length-(\(k+1\)) candidate itemsets, from \(L_k\)
  - Prune candidate itemsets in \(C_{k+1}\) containing subsets of length \(k\) that are not in \(L_k\) (and hence infrequent)
  - Count support of each remaining candidate by scanning DB; eliminate infrequent ones from \(C_{k+1}\)
  - \(L_{k+1}=C_{k+1} \setminus \{\text{infrequent}\} \), \(k = k + 1\)

Important Details of Apriori

- How to generate candidates?
  - Step 1: self-joining \(L_k\)
  - Step 2: pruning
- Example of Candidate-generation for \(L_3=\{\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{a,c,e\}, \{b,c,d\}\}\)
  - Self-joining \(L_3\)
    - \(\{a,b,c,d\}\) from \(\{a,b,c\}\) and \(\{a,b,d\}\)
    - \(\{a,c,d,e\}\) from \(\{a,c,d\}\) and \(\{a,c,e\}\)
  - Pruning:
    - \(\{a,c,d,e\}\) is removed because \(\{a,d,e\}\) is not in \(L_3\)
    - \(C_3=\{\{a,b,c,d\}\}\)
How to Generate Candidates?

• Step 1: self-joining $L_{k-1}$
  insert into $C_k$
  
  ```sql
  select p.item_1, p.item_2, ..., p.item_{k-1}, q.item_{k-2}
  from $L_{k-1}$ p, $L_{k-1}$ q
  where p.item_1 = q.item_1 AND ... AND p.item_{k-2} = q.item_{k-2}
  AND p.item_{k-1} < q.item_{k-1}
  ```

• Step 2: pruning
  – for all itemsets $c$ in $C_k$
    – for all $(k-1)$-subsets $s$ of $c$
      – if $s$ is not in $L_{k-1}$ then delete $c$ from $C_k$

How to Count Supports of Candidates?

• Why is counting supports of candidates a problem?
  – Total number of candidates can be very large
  – One transaction may contain many candidates

• Method:
  – Candidate itemsets stored in a hash-tree
  – Leaf node contains list of itemsets
  – Interior node contains a hash table
  – Subset function finds all candidates contained in a transaction

Generate Hash Tree

• Suppose we have 15 candidate itemsets of length 3:
  – $(1,4,5), (1,2,4), (4,5,7), (1,2,5), (1,3,6), (2,3,4), (5,6,7), (3,4,5), (3,5,6), (6,8,9), (3,6,7), (3,6,8)$

• We need:
  – Hash function
  – Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)

Subset Operation Using Hash Tree

**Hash Function**

1.47 3.69 2.58

**Transaction**

12356

**Hash Function**

1.47 3.69 2.58

**Transaction**

12356

**Hash Function**

1.47 3.69 2.58

**Transaction**

12356

Match transaction against 9 out of 15 candidates
Association Rule Generation

- Given a frequent itemset $L$, find all non-empty subsets $f \subseteq L$ such that $f \rightarrow L - f$ satisfies the minimum confidence requirement
  - If $\{A,B,C,D\}$ is a frequent itemset, candidate rules are:
    - $ABC \rightarrow D$, $ABD \rightarrow C$, $ACD \rightarrow B$, $BCD \rightarrow A$
    - $A \rightarrow BC$, $B \rightarrow AC$, $C \rightarrow BD$, $D \rightarrow AB$
  - If $|L| = k$, then there are $2^k - 2$ candidate association rules (ignoring $L \rightarrow \emptyset$ and $\emptyset \rightarrow L$)

Rule Generation

- How do we efficiently generate association rules from frequent itemsets?
  - In general, confidence does not have an anti-monotone property
    - $c(ABC \rightarrow D)$ can be larger or smaller than $c(AB \rightarrow D)$
  - But confidence of rules generated from the same itemset has an anti-monotone property
    - For $\{A,B,C,D\}$, $c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$
    - Confidence is anti-monotone w.r.t. number of items on the right-hand side of the rule

Rule Generation for Apriori Algorithm

- Candidate rule is generated by merging two rules that share the same prefix in the rule consequent
  - Join($CD \rightarrow AB$, $BD \rightarrow AC$) would produce the candidate rule $D \rightarrow ABC$
  - Prune rule $D \rightarrow ABC$ if its subset $AD \rightarrow BC$ does not have high confidence

Improving Apriori

- Challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- General ideas
  - Reduce passes of transaction database scans
  - Further shrink number of candidates
  - Facilitate support counting of candidates

Bottleneck of Frequent-Pattern Mining

- Apriori generates a very large number of candidates
  - $10^4$ frequent 1-itemsets can result in more than $10^7$ candidate 2-itemsets
  - Many candidates might have low support, or do not even exist in the database
- Apriori scans entire transaction database for every round of support counting
  - Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?
How to Avoid Candidate Generation

- Grow long patterns from short ones using local frequent items
  - Assume \{a,b,c\} is a frequent pattern in transaction database DB
  - Get all transactions containing \{a,b,c\}
    - Notation: DB|\{a,b,c\}
  - \{d\} is a local frequent item in DB|\{a,b,c\}, if and only if \{a,b,c,d\} is a frequent pattern in DB

Construct FP-tree from a Transaction Database

<table>
<thead>
<tr>
<th>TID</th>
<th>Items bought</th>
<th>(ordered) frequent items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{f, a, c, d, g, l, m, p}</td>
<td>{f, c, a, m, p}</td>
</tr>
</tbody>
</table>
| 200 | \{a, b, c, f, l, m, o\} | \{f, c, a, b, m\} | **min_support = 3**
| 300 | \{b, f, h, j, o, w\} | \{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\} |

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

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 larger than the original database
  - For some example DBs, compression ratio over 100
Partition Patterns and Databases

- Frequent patterns can be partitioned into subsets according to f-list
  - f-list = f-c-a-b-m-p
  - Patterns containing p
  - Patterns having m, but no p
  - Patterns having b, but neither m nor p
  - ...
  - Patterns having c, but neither a, b, m, nor p
  - Pattern f

- This partitioning is complete and non-redundant

Construct Conditional Pattern Base For Item X

- Conditional pattern base = set of prefix paths in FP-tree that co-occur with x
- Traverse FP-tree by following link of frequent item x in header table
- Accumulate paths with their frequency counts

Conditional pattern base of "am": f:3
Cond. pattern base of "cm": f:3
Cond. pattern base of "cam": f:3
Cond. pattern base of "fm": f:3

Recursion: Mining 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

FP-Tree Algorithm Summary

- 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 recursively on each newly created conditional FP-tree
  - Stop recursion when resulting FP-tree is empty
    - Optimization if tree contains only one path: single path will generate all the combinations of its sub-paths, each of which is a frequent pattern

FP-Growth vs. Apriori: Scalability With Support Threshold

Data set T25L20D10K

- FP-growth runtime
- Apriori runtime


**Why Is FP-Growth the Winner?**

- 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 operations: counting local frequent single items and building sub FP-tree
    - No pattern search and matching

**Factors Affecting Mining Cost**

- Choice of minimum support threshold
  - Lower support threshold \( \Rightarrow \) more frequent itemsets
- Dimensionality (number of items) of the data set
  - More space needed to store support count of each item
  - If number of frequent items also increases, both computation and I/O costs may increase
- Size of database
  - Each pass over DB is more expensive
- Average transaction width
  - May increase max. length of frequent itemsets and traversals of hash tree (more subsets supported by transaction)
- How can we further reduce some of these costs?

**Compact Representation of Frequent Itemsets**

- Some itemsets are redundant because they have identical support as their supersets
- Number of frequent itemsets \( \approx 3 \times \sum_{k=1}^{10} C_k \)
- Need a compact representation

**Maximal Frequent Itemset**

An itemset is maximal-frequent if none of its supersets is frequent

**Closed Itemset**

- A frequent itemset is closed if none of its supersets has the same support
  - Lossless compression of the set of all frequent itemsets

**Maximal vs Closed Frequent Itemsets**

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
<th>Support</th>
<th>TID</th>
<th>Items</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{A,B}</td>
<td>4</td>
<td>4</td>
<td>{A,C,D}</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>{B,C,D}</td>
<td>2</td>
<td>5</td>
<td>{D}</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>{A,B,C,D}</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \min_{sup} = 2 \]
Maximal vs Closed Frequent Itemsets

- Minimum support = 2
- # Closed = 9
- # Maximal = 4
- How to efficiently find maximal frequent itemsets? (similar for closed ones)
  - Naïve: first find all frequent itemsets, then remove non-maximal ones
  - Better: use maximality property for pruning
- Effectiveness depends on itemset generation strategy
- See book for details

Methods for Frequent Itemset Generation

- Traversal of itemset lattice
  - General-to-specific: Apriori
  - Specific-to-general: good for pruning for maximal frequent itemsets

Alternative Methods for Frequent Itemset Generation

- Traversal of itemset lattice
  - Equivalence Classes: search one class first, before moving on to the next one

Extension: Mining Multiple-Level Association Rules

- Items often form hierarchies
  - Most relevant pattern might only show at the right granularity
- Flexible support settings
  - Items at the lower level are expected to have lower support

- Uniform support
  - Level 1: min_sup = 5%
  - Level 2: min_sup = 1%

- Reduced support
  - Level 1: min_sup = 5%, support = 10%
  - Level 2: min_sup = 1%, support = 4%
### Extension: Mining Multi-Dimensional Associations

- Single-dimensional rules: one type of predicate
  - \( \text{buy}(X, \text{“milk”}) \rightarrow \text{buy}(X, \text{“bread”}) \)
- Multi-dimensional rules: \( \geq 2 \) types of predicates
  - Interdimensional association rules (no repeated predicates)
    - \( \text{age}(X, \text{“19-25”}) \land \text{occupation}(X, \text{“student”}) \rightarrow \text{buy}(X, \text{“coke”}) \)
  - Hybrid-dimensional association rules (repeated predicates)
    - \( \text{age}(X, \text{“19-25”}) \land \text{buy}(X, \text{“popcorn”}) \rightarrow \text{buy}(X, \text{“coke”}) \)
- See book for efficient mining algorithms

### Lift

- Ex.: 2000 transactions have bread and milk, 1000 have bread but no milk, 1750 have milk but no bread, 250 have neither
- Rule bread \( \rightarrow \) milk has support 0.4, confidence 0.67
- Does it mean that people who buy bread also tend to buy milk?
- Misleading: 75% of all people buy milk, while among bread purchasers only 67% do
- But bread \( \rightarrow \) no milk only has support 0.2, confidence 0.33
- Measure of dependent/correlated events: lift

\[
\text{lift}(A,B) = \frac{P(A \cup B)}{P(A)P(B)}
\]

\[
lift(A,B) = \frac{2000/5000}{5000/5000+3750/5000} = 0.89
\]

### Which Measure Is Best?

- Does it identify the right patterns?
- Does it result in an efficient mining algorithm?

### Lift vs. Other Correlation Measures

- Intuition: Are milk and coffee usually bought together?
  - \( m \lor c \lor m \lor c \)
  - \( m \land c \land \neg m \land c \land \neg m \land c \land \neg m \land c \land \neg m \land c \)

\[
\text{all}\_\text{conf}(A) = \frac{\text{sup}(A)}{\text{max}\_\text{rem}\_\text{sup}(A)}
\]

\[
\text{cosine}(A,B) = \frac{\text{P}(A \land B)}{\sqrt{\text{P}(A)\text{P}(B)}}
\]

### Frequent Pattern Mining Overview

- Basic Concepts and Challenges
- Efficient and Scalable Methods for Frequent Itemsets and Association Rules
- Pattern Interestingness Measures
- Sequence Mining

### Which Measure Is Best?

The Ps and Cs are various desirable properties, e.g., symmetry under variable permutation (O1), which we do not cover in this class. Take away message: no interestingness measure has all the desirable properties.
**Frequent Pattern Mining Overview**

- Basic Concepts and Challenges
- Efficient and Scalable Methods for Frequent Itemsets and Association Rules
- Pattern Interestingness Measures
- Sequence Mining

**Introduction**

- Sequence mining: relevant for transaction, time-series, and sequence databases
- Applications of sequential pattern mining
  - Customer shopping sequences: first buy computer, then peripheral device within 3 months
  - Medical treatments, natural disasters (e.g., earthquakes), science & engineering processes, stocks and markets
  - Telephone calling patterns, Weblog click streams
  - DNA sequences and gene structures

**What Is Sequential Pattern Mining?**

- Given a set of sequences, find all frequent subsequences

A sequence database

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;(abc)(ac)(cd)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ad)(bc)(ae)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)b&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;ef(af)(bc)&gt;</td>
</tr>
</tbody>
</table>

An element may contain a set of items. Items within an element are unordered and we list them alphabetically.

- A sequence: `<[ef](ab)][df]c>l` is a subsequence of `<[a](bc)[ac])[(df)]>l`

Given support threshold `min_sup` = 2, `<(ab)c>` is a sequential pattern.

**Challenges of Sequential Pattern Mining**

- Huge number of possible patterns
  - A mining algorithm should
    - find all patterns satisfying the minimum support threshold
    - be highly efficient and scalable
    - be able to incorporate user-specific constraints

**Apriori Property of Sequential Patterns**

- If a sequence `S` is not frequent, then none of the super-sequences of `S` is frequent
  - E.g., if `<hb>` is infrequent, then so are `<hab>` and `<(ah)b>`

<table>
<thead>
<tr>
<th>Seq. ID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;(bd)(bc)(ae)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(bf)(ce)(fg)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ah)(ef)(ab)&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;(be)(ce)j&gt;</td>
</tr>
<tr>
<td>50</td>
<td>&lt;(ad)(bc)(ade)&gt;</td>
</tr>
</tbody>
</table>

Given support threshold `min_sup` = 2, find all frequent subsequences

**GSP: Generalized Sequential Pattern Mining**

- Initially, every item in DB is a candidate of length `k=1`
- For each level (i.e., sequences of length `k`) do
  - Scan database to collect support count for each candidate sequence
  - Generate candidate length-`(k+1)` sequences from length-`k` frequent sequences
    - Join phase: sequences `s1` and `s2` join, if `s1` without its first item is identical to `s2` without its last item
    - Prune phase: delete candidates that contain a length-`k` subsequence that is not among the frequent ones
  - Repeat until no frequent sequence or no candidate can be found
  - Major strength: Candidate pruning by Apriori
Finding Length-1 Sequential Patterns
- Initial candidates: all singleton sequences
  - \(<a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>\)
- Scan database once, count support for candidates

<table>
<thead>
<tr>
<th>Seq_ID</th>
<th>Sequence</th>
<th>Sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;bd&gt;x(ac)&gt;</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>&lt;df&gt;xe(lg)&gt;</td>
<td>5</td>
</tr>
<tr>
<td>30</td>
<td>&lt;ah&gt;(bf)abf&gt;</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>&lt;be&gt;(ce)b(fg)&gt;</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>&lt;a(bd)bcb(ade)&gt;</td>
<td>1</td>
</tr>
</tbody>
</table>

\(\text{min} \_ \text{Sup} = 2\)

GSP: Generating Length-2 Candidates
51 length-2 Candidates

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Cand.</th>
<th>Sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;a(bd)bcb(ade)&gt;</td>
<td>&lt;a&gt;</td>
<td>3</td>
</tr>
<tr>
<td>&lt;b&gt;</td>
<td>&lt;a&gt;</td>
<td>5</td>
</tr>
<tr>
<td>&lt;c&gt;</td>
<td>&lt;a&gt;</td>
<td>4</td>
</tr>
<tr>
<td>&lt;d&gt;</td>
<td>&lt;a&gt;</td>
<td>3</td>
</tr>
<tr>
<td>&lt;e&gt;</td>
<td>&lt;a&gt;</td>
<td>3</td>
</tr>
<tr>
<td>&lt;f&gt;</td>
<td>&lt;a&gt;</td>
<td>2</td>
</tr>
<tr>
<td>&lt;g&gt;</td>
<td>&lt;a&gt;</td>
<td>1</td>
</tr>
<tr>
<td>&lt;h&gt;</td>
<td>&lt;a&gt;</td>
<td>1</td>
</tr>
</tbody>
</table>

Without Apriori property, \(8 \times 8 + 8 \times 7/2 = 92\) candidates

Apriori prunes 44.57% candidates

The GSP Mining Process
- Scan 5: 1 candidate, 1 length-5 seq. pattern
- Scan 4: 8 candidates, 6 length-4 seq. patterns
- Scan 3: 47 candidates, 19 length-3 seq. patterns, 20 candidates not in DB at all
- Scan 2: 51 candidates, 19 length-2 seq. patterns, 10 candidates not in DB at all
- Scan 1: 8 candidates, 6 length-1 seq. patterns \(\text{min} \_ \text{Sup} = 2\)

Candidate Generate-and-Test Drawbacks
- Huge set of candidate sequences generated
- Multiple Scans of entire database needed
  - Length of each candidate grows by one at each database scan

Prefix and Suffix (Projection)
- \(<a>, <aa>, <a(ab)>\) and \(<a(abc)>\) are prefixes of sequence \(<a(abc)(ac)d(cf)>\)
- Given sequence \(<a(abc)(ac)d(cf)>\), we have:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Suffix (Prefix-Based Projection)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;a&gt;</td>
<td>(&lt;a(bc)(ac)d(cf)&gt;)</td>
</tr>
<tr>
<td>&lt;aa&gt;</td>
<td>(&lt;a(ab)&gt;)</td>
</tr>
<tr>
<td>&lt;ab&gt;</td>
<td>(&lt;a(bc)(ac)d(cf)&gt;)</td>
</tr>
<tr>
<td>&lt;bc&gt;</td>
<td>(&lt;d(cf)&gt;)</td>
</tr>
<tr>
<td>&lt;(bc)&gt;</td>
<td>(&lt;(ac)d(cf)&gt;)</td>
</tr>
</tbody>
</table>

Mining Sequential Patterns by Prefix Projections
- Step 1: find length-1 frequent sequential patterns
  - \(<a>, <b>, <c>, <d>, <e>, <f>\)
- Step 2: divide search space. The complete set of sequential patterns can be partitioned into six subsets:
  - The ones having prefix \(<a>\);
  - The ones having prefix \(<b>\);
  - ...
  - The ones having prefix \(<f>\)
Finding Seq. Patterns with Prefix <a>

- Only need to consider projections w.r.t. <a>
  - <a>-projected database: <(abc)(ac)d(cf)>, <(_d)(bc)(ae)>, <(_b)(df)cb>, <(_f)cbc>
- Find all length-2 frequent seq. patterns having prefix <a>: <aa>, <ab>, <(ab)>, <ac>, <ad>, <af>
  - Further partition into those 6 subsets
    - Having prefix <aa>;
    - Having prefix <ab>;
    - Having prefix <(ab)>;
    - Having prefix <ac>;
    - Having prefix <ad>;
    - Having prefix <af>

<table>
<thead>
<tr>
<th>SID</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a&gt;&lt;b&gt;&lt;c&gt;&lt;d&gt;&lt;e&gt;&lt;f&gt;&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;a&gt;&lt;b&gt;&lt;c&gt;&lt;d&gt;&lt;e&gt;&lt;f&gt;&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;a&gt;&lt;b&gt;&lt;c&gt;&lt;d&gt;&lt;e&gt;&lt;f&gt;&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;a&gt;&lt;b&gt;&lt;c&gt;&lt;d&gt;&lt;e&gt;&lt;f&gt;&gt;</td>
</tr>
</tbody>
</table>

Efficiency of PrefixSpan

- No candidate sequence needs to be generated
- Projected databases keep shrinking
- Major cost of PrefixSpan: constructing projected databases
  - Can be improved by pseudo-projections

Pseudo-Projection

- Major cost of PrefixSpan: projection
  - Postfixes of sequences often appear repeatedly in recursive projected databases
  - When (projected) database can be held in memory, use pointers
    - Pointer to the sequence, offset of the postfix
  - Why is this a bad idea when the (projected) database does not fit in memory?

Pseudo-Projection vs. Physical Projection

- Pseudo-projection avoids physically copying postfixes
  - Efficient in running time and space when database can be held in main memory
- Not efficient when database cannot fit in main memory
  - Disk-based random access
- Suggested Approach:
  - Integration of physical and pseudo-projection
  - Swapping to pseudo-projection when the data set fits in memory

Performance on Data Set C10T8S8I8
Performance on Data Set Gazelle

Effect of Pseudo-Projection

Sequence Mining Variations
- Multidimensional and multilevel patterns
- Constraint-based mining of sequential patterns
- Periodicity analysis
- Mining biological sequences
  - Hot research area, major topic by itself
- All these not discussed in class; see book
- Some of my own research: finding relevant sequences in bursty data; see paper

Frequent-Pattern Mining: Summary
- Important task in data mining
- Scalable frequent pattern mining methods
  - Apriori (itemsets, candidate generation & test)
  - GSP (sequences, candidate generation & test)
  - Projection-based (FP-growth for itemsets, PrefixSpan for sequences)
- Mining a variety of rules and interesting patterns

Frequent-Pattern Mining: Research Problems
- Mining fault-tolerant frequent, sequential and structured patterns
  - Patterns allows limited faults (insertion, deletion, mutation)
- Mining truly interesting patterns
  - Surprising, novel, concise,...
- Application exploration
  - E.g., DNA sequence analysis and bio-pattern classification