Handling a Concept Hierarchy

Multi-level Association Rules

• Why should we incorporate concept hierarchy?
  - Rules at lower levels may not have enough support to appear in any frequent itemsets
  - Rules at lower levels of the hierarchy are overly specific
    • e.g., skim milk ⇒ white bread,
    2% milk ⇒ wheat bread,
    skim milk ⇒ wheat bread, etc.
    are indicative of an association between milk and bread

• How do support and confidence vary as we traverse the concept hierarchy?
  - If X is the parent item for both X1 and X2, then
    \[ \sigma(X) \leq \sigma(X1) + \sigma(X2) \]
  - If \( \sigma(X1 \cup Y1) \geq \text{minsup} \), and X is parent of X1, Y is parent of Y1
    then \( \sigma(X \cup Y) \geq \text{minsup} \)
    \( \sigma(X) \geq \text{minsup} \)
    \( \sigma(X1 \cup Y) \geq \text{minsup} \)
  - If \( \text{conf}(X1 \Rightarrow Y1) \geq \text{minconf} \),
    then \( \text{conf}(X1 \Rightarrow Y) \geq \text{minconf} \)

Multi-level Association Rules

• Approach 1:
  - Extend current association rule formulation by augmenting each transaction with higher level items
    • Original Transaction:
    (skim milk, wheat bread)
    • Augmented Transaction:
    (skim milk, wheat bread, milk, bread, food)
  - Issues:
    • Items that reside at higher levels have much higher support counts
    • If support threshold is low, there are too many frequent patterns involving items from the higher levels
    • Increased dimensionality of the data

• Approach 2:
  - Generate frequent patterns at highest level first
  - Then, generate frequent patterns at the next highest level, and so on...
  - Issues:
    • I/O requirements increase dramatically because we need to perform more passes over the data
    • May miss some potentially interesting cross-level association patterns
Mining Sequential Patterns

Examples of Sequence Data

<table>
<thead>
<tr>
<th>Database</th>
<th>Sequence</th>
<th>Element (Transaction)</th>
<th>Event (Item)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Purchase history of a given customer</td>
<td></td>
<td>Books, diary products, CDs, etc</td>
</tr>
<tr>
<td>Web data</td>
<td>Browsing activity of a particular Web visitor</td>
<td></td>
<td>Home page, index page, contact info, etc</td>
</tr>
<tr>
<td>Event data</td>
<td>History of events generated by a given sensor</td>
<td></td>
<td>Types of alarms generated by sensors</td>
</tr>
<tr>
<td>Genome sequences</td>
<td>DNA sequence of a particular species</td>
<td></td>
<td>Bases A, T, G, C</td>
</tr>
</tbody>
</table>

Examples of Sequences

- Web sequence:
  `< (Homepage) (Electronics) (Digital Cameras) (Canon Digital Camera) (Shopping Cart) (Order Confirmation) (Return to Shopping) >`

- Sequence of books checked out at a library (or films rented at a video store):
  `< (Fellowship of the Ring) (The Two Towers, Return of the King) >`

Formal Definition of a Sequence

- A sequence is an ordered list of elements (transactions)
  \( S = \langle e_1, e_2, e_3, \ldots \rangle \)
  - Each element contains a collection of events (items)
    \( e_i = \{i_1, i_2, \ldots, i_k\} \)
  - Each element is attributed to a specific time or location
- Length of a sequence, \(|S|\), is given by the number of elements of the sequence
- A \( k \)-sequence is a sequence that contains \( k \) events (items)

Formal Definition of a Subsequence

- A sequence \( \langle a_1, a_2, \ldots, a_n \rangle \) is contained in another sequence \( \langle b_1, b_2, \ldots, b_m \rangle \) (\( m \geq n \)) if there exist integers \( i_1 < i_2 < \ldots < i_k \) such that \( a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_n \subseteq b_{i_n} \)
- The support of a subsequence \( w \) is defined as the fraction of data sequences that contain \( w \)
- A sequential pattern is a frequent subsequence (i.e., a subsequence whose support is \( \geq \text{minsup} \))
**Sequential Pattern Mining: Definition**

- **Given:**
  - a database of sequences
  - a user-specified minimum support threshold, \( \text{minsup} \)

- **Task:**
  - Find all subsequences with support \( \geq \text{minsup} \)

**Sequential Pattern Mining: Challenge**

- Given a sequence: \(<\{a \ b\} \ {c \ d \ e}\> \{f \ g \ h \ i\}>\)
  - Examples of subsequences: \(<\{a\}>\), \(<\{c \ d \ e\}>\), \(<\{b \} \ {g \} \ >\), etc.
  - How many \( k \)-subsequences can be extracted from a given \( n \)-sequence?

\[
\binom{n}{k} = \frac{n!}{k!(n-k)!}
\]

**Sequential Pattern Mining: Example**

- **Minsup = 50%**

**Examples of Frequent Subsequences:**

- \(<\{1,2\}>\) s=60%
- \(<\{2,3\}>\) s=60%
- \(<\{2,4\}>\) s=80%
- \(<\{3,4\}>\) s=80%
- \(<\{1\}>\) s=80%
- \(<\{2\}>\) s=60%
- \(<\{1,2\}>\) s=60%
- \(<\{1,3\}>\) s=60%
- \(<\{1,2,3\}>\) s=60%
- \(<\{1,2,4\}>\) s=60%
- \(<\{1,2,5\}>\) s=60%
- \(<\{1,3,4\}>\) s=60%
- \(<\{1,3,5\}>\) s=60%
- \(<\{3,4,5\}>\) s=60%

**Extracting Sequential Patterns: Brute-force**

- **Given n events:** \( i_1, i_2, i_3, \ldots, i_n \)
  - **Candidate 1-subsequences:** \(<\{i_1\}>\), \(<\{i_2\}>\), \(<\{i_3\}>\), \ldots, \(<\{i_n\}>\)
  - **Candidate 2-subsequences:** \(<\{i_1, i_2\}>\), \(<\{i_1, i_3\}>\), \ldots, \(<\{i_1, i_n\}>\), \(<\{i_2, i_3\}>\), \ldots, \(<\{i_2, i_n\}>\), \ldots, \(<\{i_{n-1}, i_n\}>\)
  - **Candidate 3-subsequences:** \(<\{i_1, i_2, i_3\}>\), \(<\{i_1, i_2, i_4\}>\), \ldots, \(<\{i_1, i_2, i_n\}>\), \(<\{i_1, i_3, i_4\}>\), \ldots, \(<\{i_1, i_3, i_n\}>\), \ldots, \(<\{i_1, i_n-1, i_n\}>\), \ldots, \(<\{i_{n-1}, i_n-1, i_n\}>\), \ldots

**Candidate Generation Algorithm**

- **Base case (k=2):**
  - Merging two frequent 1-sequences \(<\{i_1\}>\) and \(<\{i_2\}>\) will produce two candidate 2-sequences: \(<\{i_1\} \ {i_2}\>\) and \(<\{i_1 \ i_2\}>\)

- **General case (k=2):**
  - Two frequent (k-1)-sequences \(w_1\) and \(w_2\) are merged together to produce a candidate k-sequence if the subsequence obtained by removing the first event in \(w_1\) is the same as the subsequence obtained by removing the last event in \(w_2\):
  - The resulting candidate after merging is given by the sequence \(w_1\) extended with the last event of \(w_2\).
  - If the last two events in \(w_2\) belong to the same element, then the last event in \(w_2\) becomes part of the last element in \(w_1\).
  - Otherwise, the last event in \(w_2\) becomes a separate element appended to the end of \(w_1\).

**Candidate Generation Examples**

- Merging the sequences \(w_1 = \{1\} \ {2} \ {3} \ {4}\) and \(w_2 = \{2\} \ {3} \ {4}\) will produce the candidate sequence \(<\{1\} \ {2} \ {3} \ {4}\>) because the last two events in \(w_1\) and \(w_2\) belong to the same element.

- Merging the sequences \(w_1 = \{1\} \ {2} \ {3} \ {4}\) and \(w_2 = \{2\} \ {3} \ {4}\) will produce the candidate sequence \(<\{1\} \ {2} \ {3} \ {4}\>) because the last two events in \(w_1\) and \(w_2\) do not belong to the same element.

- We do not have to merge the sequences \(w_1 = \{1\} \ {2} \ {3} \ {4}\) and \(w_2 = \{2\} \ {4}\) to produce the candidate \(<\{1\} \ {2} \ {4}\>) because if the latter is a viable candidate, then it can be obtained by merging \(w_1\) with \(<\{1\} \ {2} \ {3}\>).
Generalized Sequential Pattern (GSP)

Step 1:
- Make the first pass over the sequence database D to yield all the 1-element frequent sequences

Step 2:
- Repeat until no new frequent sequences are found
  - Candidate Generation:
    - Merge pairs of frequent subsequences found in the (k-1)th pass to generate candidate sequences that contain k items
  - Candidate Pruning:
    - Prune candidate k-sequences that contain infrequent (k-1)-subsequences
  - Support Counting:
    - Make a new pass over the sequence database D to find the support for these candidate sequences
  - Candidate Elimination:
    - Eliminate candidate k-sequences whose actual support is less than minsup

GSP Example

Frequent 3-sequences

<table>
<thead>
<tr>
<th>Candidate Generation</th>
<th>Candidate Pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; {1} {2} {3} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {1} {2} {5} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {1} {5} {3} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {2} {3} {4} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {2} {5} {3} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {3} {4} {5} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {5} {3} {4} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {1} {2} {5} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {1} {2} {5} {3} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {1} {5} {3} {4} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {2} {3} {4} {5} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {2} {5} {3} {4} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {1} {2} {5} {3} &gt;</td>
<td></td>
</tr>
</tbody>
</table>

Timing Constraints (I)

\[ x_g = 2, n_g = 0, m_s = 4 \]

Mining Sequential Patterns with Timing Constraints

- Approach 1:
  - Mine sequential patterns without timing constraints
  - Postprocess the discovered patterns

- Approach 2:
  - Modify GSP to directly prune candidates that violate timing constraints
  - Question:
    - Does the Apriori principle still hold?

Apriori Principle for Sequence Data

<table>
<thead>
<tr>
<th>Object</th>
<th>Timestamp</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>2,4</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>3,5</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1,2</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3,4</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>2,3,4</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>3,4,5</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>3,4</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>1,2</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>2,4,5</td>
</tr>
</tbody>
</table>

Contiguous Subsequences

\( s \) is a contiguous subsequence of \( w = \langle e_1, e_2, \ldots, e_k \rangle \)
if any of the following conditions hold:
1. \( s \) is obtained from \( w \) by deleting an item from either \( e_i \) or \( e_j \)
2. \( s \) is obtained from \( w \) by deleting an item from any element \( e_i \) that contains 2 or more items
3. \( s \) is a contiguous subsequence of \( s' \) and \( s' \) is a contiguous subsequence of \( w \) (recursive definition)

Examples:
- \( s = \langle 1 \rangle \) (2) > is a contiguous subsequence of \( \langle 1 \rangle \) (2) 3 >, \( \langle 1 \rangle \) (2) (3) >, and \( \langle 3 \rangle \) (4) (1) (2) (3) (4) >
- is not a contiguous subsequence of \( \langle 1 \rangle \) (3) (2) > and \( \langle 2 \rangle \) (1) (3) (2) >
Modified Candidate Pruning Step

- Without maxgap constraint:
  - A candidate k-sequence is pruned if at least one of its (k-1)-subsequences is infrequent

- With maxgap constraint:
  - A candidate k-sequence is pruned if at least one of its contiguous (k-1)-subsequences is infrequent

Timing Constraints (II)

- $x_g$: max-gap
- $n_g$: min-gap
- $ws$: window size
- $m_s$: maximum span

<table>
<thead>
<tr>
<th>Data sequence</th>
<th>Subsequence</th>
<th>Contain?</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;1,2,3,4&gt;)</td>
<td>(&lt;1,2)</td>
<td>Yes</td>
</tr>
<tr>
<td>(&lt;1,2,3,4&gt;)</td>
<td>(&lt;3,4)</td>
<td>Yes</td>
</tr>
<tr>
<td>(&lt;1,2,3,4&gt;)</td>
<td>(&lt;1,2)</td>
<td>Yes</td>
</tr>
<tr>
<td>(&lt;1,2,3,4&gt;)</td>
<td>(&lt;2,3,4)</td>
<td>Yes</td>
</tr>
<tr>
<td>(&lt;1,2,3,4&gt;)</td>
<td>(&lt;3,4)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Modified Support Counting Step

Given a candidate pattern: \(<a, c>\>

Any data sequences that contain

- \(<...a...c...>\>
- \(<...a...>...c...>\) (where \(\text{time}(c) - \text{time}(a) \leq ws\))
- \(<...c...>...a...>\) (where \(\text{time}(a) - \text{time}(c) \leq ws\))

will contribute to the support count of the pattern

Other Formulation

- In some domains, we may have only one very long time series
  - Example:
    - monitoring network traffic events for attacks
    - monitoring telecommunication alarm signals
  - Goal is to find frequent sequences of events in the time series
    - This problem is also known as frequent episode mining

Pattern: \(<E_1, E_3>\>