Data Mining
Association Rules: Advanced Concepts and Algorithms

Lecture Notes for Chapter 7

Introduction to Data Mining
by
Tan, Steinbach, Kumar

(modified by Predrag Radivojac, 2018)
### Continuous and Categorical Attributes

**Example of Association Rule:**

\[
\{\text{Number of Pages } \in [5, 10) \land (\text{Browser} = \text{Firefox})\} \rightarrow \{\text{Buy} = \text{No}\}
\]
Handling Categorical Attributes

- Transform categorical attribute into asymmetric binary variables

- Introduce a new “item” for each distinct attribute-value pair
  - Example: replace Browser Type attribute with
    - Browser Type = Internet Explorer
    - Browser Type = Firefox
    - Browser Type = Chrome
Handling Categorical Attributes

Potential Issues

- What if attribute has many possible values
  - Example: attribute country has more than 200 possible values
  - Many of the attribute values may have very low support
    - *Potential solution*: Aggregate the low-support attribute values

- What if distribution of attribute values is highly skewed
  - Example: 95% of the visitors have $\text{Buy} = \text{No}$
  - Most of the items will be associated with ($\text{Buy} = \text{No}$) item
    - *Potential solution*: drop the highly frequent items

- Transaction width increases $\Rightarrow$ more computation
Handling Continuous Attributes

- Different kinds of rules:
  - $\text{Age} \in [21, 35) \land \text{Salary} \in [70k, 120k) \rightarrow \text{Buy}$
  - $\text{Salary} \in [70k, 120k) \land \text{Buy} \rightarrow \text{Age: } \mu=28, \sigma=4$

- Different methods:
  - Discretization-based
  - Statistics-based
  - Non-discretization based
    - $\text{minApriori}$
Handling Continuous Attributes

- Use discretization
- Unsupervised:
  - Equal-width binning
  - Equal-depth binning
  - Clustering
- Supervised:

<table>
<thead>
<tr>
<th>Class</th>
<th>(v_1)</th>
<th>(v_2)</th>
<th>(v_3)</th>
<th>(v_4)</th>
<th>(v_5)</th>
<th>(v_6)</th>
<th>(v_7)</th>
<th>(v_8)</th>
<th>(v_9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anomalous</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>10</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Normal</td>
<td>150</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>150</td>
<td>100</td>
</tr>
</tbody>
</table>

Attribute values, \(v\)

[Diagram showing log-log plot with duration on the x-axis and count on the y-axis.]

- bin\(_1\)
- bin\(_2\)
- bin\(_3\)
Discretization Issues

● Size of the discretized intervals affect support & confidence

{Refund = No, (Income = $51,250)} → {Cheat = No}
{Refund = No, (60K ≤ Income ≤ 80K)} → {Cheat = No}
{Refund = No, (0K ≤ Income ≤ 1B)} → {Cheat = No}

– If intervals too small
  ◆ may not have enough support

– If intervals too large
  ◆ may not have enough confidence

● Potential solution: use all possible intervals
Discretization Issues

- **Execution time**
  - If intervals contain $n$ values, there are on average $O(n^2)$ possible ranges

- **Too many rules**
  - \{Refund = No, (Income = $51,250)\} \rightarrow \{Cheat = No\}
  - \{Refund = No, (51K \leq Income \leq 52K)\} \rightarrow \{Cheat = No\}
  - \{Refund = No, (50K \leq Income \leq 60K)\} \rightarrow \{Cheat = No\}
Approach by Srikant & Agrawal

- Preprocess the data
  - Discretize attribute using equi-depth partitioning
    - Use *partial completeness measure* to determine number of partitions
    - Merge adjacent intervals as long as support is less than max-support

- Apply existing association rule mining algorithms

- Determine interesting rules in the output
Approach by Srikant & Agrawal

- Discretization will lose information

Use *partial completeness measure* to determine how much information is lost

C: frequent itemsets obtained by considering all ranges of attribute values
P: frequent itemsets obtained by considering all ranges over the partitions

P is *K-complete* w.r.t C if $P \subseteq C$, and $\forall X \in C$, $\exists X' \in P$ such that:

1. $X'$ is a generalization of $X$ and $\text{support}(X') \leq K \times \text{support}(X)$  \hspace{1cm} (K \geq 1)
2. $\forall Y \subseteq X$, $\exists Y' \subseteq X'$ such that $\text{support}(Y') \leq K \times \text{support}(Y)$

Given $K$ (*partial completeness level*), can determine number of intervals (N)
Interestingness Measure

{Refund = No, (Income = $51,250)} → {Cheat = No}
{Refund = No, (51K ≤ Income ≤ 52K)} → {Cheat = No}
{Refund = No, (50K ≤ Income ≤ 60K)} → {Cheat = No}

- Given an itemset: $Z = \{z_1, z_2, \ldots, z_k\}$ and its generalization $Z' = \{z'_1, z'_2, \ldots, z'_k\}$

  $P(Z)$: support of $Z$
  $E_{Z'}(Z)$: expected support of $Z$ based on $Z'$

  $$E_{Z'}(Z) = \frac{P(z'_1)}{P(z'_1')} \times \frac{P(z'_2)}{P(z'_2')} \times \cdots \times \frac{P(z'_k)}{P(z'_k')} \times P(Z')$$

- $Z$ is $R$-interesting w.r.t. $Z'$ if $P(Z) \geq R \times E_{Z'}(Z)$
Interestingness Measure

- For S: \(X \rightarrow Y\), and its generalization S': \(X' \rightarrow Y'\)
  - \(P(Y|X)\): confidence of \(X \rightarrow Y\)
  - \(P(Y'|X')\): confidence of \(X' \rightarrow Y'\)
  - \(E_{S'}(Y|X)\): expected support of Z based on Z'

\[
E(Y \mid X) = \frac{P(y_1)}{P(y'_1)} \times \frac{P(y_2)}{P(y'_2)} \times \ldots \times \frac{P(y_k)}{P(y'_k)} \times P(Y' \mid X')
\]

- Rule S is R-interesting w.r.t its ancestor rule S' if
  - Support, \(P(S) \geq R \times E_{S'}(S)\) or
  - Confidence, \(P(Y|X) \geq R \times E_{S'}(Y|X)\)
Statistics-based Methods

- Example:
  \[ \text{Browser} = \text{Firefox} \land \text{Buy} = \text{Yes} \rightarrow \text{Age: } \mu = 23 \]

- Rule consequent consists of a continuous variable, characterized by their statistics
  - mean, median, standard deviation, etc.

- Approach:
  - Withhold the target variable from the rest of the data
  - Apply existing frequent itemset generation on the rest of the data
  - For each frequent itemset, compute the descriptive statistics for the corresponding target variable
    - Frequent itemset becomes a rule by introducing the target variable as rule consequent
  - Apply statistical test to determine interestingness of the rule
Statistics-based Methods

- How to determine whether an association rule interesting?
  - Compare the statistics for segment of population covered by the rule vs segment of population not covered by the rule:
    \[ A \implies B: \mu \text{ versus } \bar{A} \implies B: \mu' \]

- Statistical hypothesis testing:
  - Null hypothesis: H0: \( \mu' = \mu + \Delta \)
  - Alternative hypothesis: H1: \( \mu' > \mu + \Delta \)
  - \( Z \) has zero mean and variance 1 under null hypothesis
Example:

\[ r: \text{Browser} = \text{Firefox} \land \text{Buy} = \text{Yes} \rightarrow \text{Age: } \mu = 23 \]

- Rule is interesting if difference between \( \mu \) and \( \mu' \) is greater than 5 years (\( \Delta = 5 \))
- For \( r \), suppose \( n_1 = 50, s_1 = 3.5 \)
- For \( r' \) (complement): \( n_2 = 250, s_2 = 6.5 \)

\[
Z = \frac{\mu' - \mu - \Delta}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} = \frac{30 - 23 - 5}{\sqrt{\frac{3.5^2}{50} + \frac{6.5^2}{250}}} = 3.11
\]

- For 1-sided test at 95% confidence level, critical Z-value for rejecting null hypothesis is 1.64.
- Since \( Z \) is greater than 1.64, \( r \) is an interesting rule.
Min-Apriori (Han et al.)

Document-term matrix:

<table>
<thead>
<tr>
<th>TID</th>
<th>W1</th>
<th>W2</th>
<th>W3</th>
<th>W4</th>
<th>W5</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>D3</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Example:

$W_1$ and $W_2$ tend to appear together in the same document.
Min-Apriori

- Data contains only continuous attributes of the same "type"
  - e.g., frequency of words in a document

<table>
<thead>
<tr>
<th>TID</th>
<th>W1</th>
<th>W2</th>
<th>W3</th>
<th>W4</th>
<th>W5</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>D3</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

- Potential solution:
  - Convert into 0/1 matrix and then apply existing algorithms
    - lose word frequency information
  - Discretization does not apply as users want association among words not ranges of words
Min-Apriori

- How to determine the support of a word?
  - If we simply sum up its frequency, support count will be greater than total number of documents!
    - Normalize the word vectors – e.g., using L₁ norm
    - Each word has a support equals to 1.0

<table>
<thead>
<tr>
<th>TID</th>
<th>W1</th>
<th>W2</th>
<th>W3</th>
<th>W4</th>
<th>W5</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>D3</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>W1</th>
<th>W2</th>
<th>W3</th>
<th>W4</th>
<th>W5</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>0.40</td>
<td>0.33</td>
<td>0.00</td>
<td>0.00</td>
<td>0.17</td>
</tr>
<tr>
<td>D2</td>
<td>0.00</td>
<td>0.00</td>
<td>0.33</td>
<td>1.00</td>
<td>0.33</td>
</tr>
<tr>
<td>D3</td>
<td>0.40</td>
<td>0.50</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>D4</td>
<td>0.00</td>
<td>0.00</td>
<td>0.33</td>
<td>0.00</td>
<td>0.17</td>
</tr>
<tr>
<td>D5</td>
<td>0.20</td>
<td>0.17</td>
<td>0.33</td>
<td>0.00</td>
<td>0.33</td>
</tr>
</tbody>
</table>

Normalize
Min-Apriori

- New definition of support:

\[
\text{sup}(C) = \sum_{i \in T} \min_{j \in C} D(i, j)
\]

Example:

\[
\text{Sup}(W_1, W_2, W_3) = 0 + 0 + 0 + 0 + 0.17 = 0.17
\]
Anti-monotone property of Support

Example:

\[ \text{Sup}(W_1) = 0.4 + 0 + 0.4 + 0 + 0.2 = 1 \]
\[ \text{Sup}(W_1, W_2) = 0.33 + 0 + 0.4 + 0 + 0.17 = 0.9 \]
\[ \text{Sup}(W_1, W_2, W_3) = 0 + 0 + 0 + 0 + 0.17 = 0.17 \]
Multi-level Association Rules

Food
- Bread
  - Wheat
  - White
  - Skim
    - 2%
      - Foremost
        - Kemps
  - Milk

Electronics
- Computers
  - Desktop
  - Laptop
  - Accessory
- Home
  - TV
  - DVD
  - Printer
  - Scanner
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

  are indicative of association between milk and bread
Multi-level Association Rules

- How do support and confidence vary as we traverse the concept hierarchy?
  - If $X$ is the parent item for both $X_1$ and $X_2$, then
    \[ \sigma(X) \geq \sigma(X_1) + \sigma(X_2) \]
  
  - If \( \sigma(X_1 \cup Y_1) \geq \text{minsupt} \), and \( X \) is parent of \( X_1 \), \( Y \) is parent of \( Y_1 \), then
    \[ \sigma(X \cup Y_1) \geq \text{minsupt}, \sigma(X_1 \cup Y) \geq \text{minsupt} \]
    \[ \sigma(X \cup Y) \geq \text{minsupt} \]
  
  - If \( \text{conf}(X_1 \Rightarrow Y_1) \geq \text{minconf} \), then
    \[ \text{conf}(X_1 \Rightarrow Y) \geq \text{minconf} \]
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, too many frequent patterns involving items from the higher levels
- Increased dimensionality of the data
Multi-level Association Rules

- **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 will increase dramatically because we need to perform more passes over the data
  - May miss some potentially interesting cross-level association patterns
Sequential Patterns

Sequence Database:

<table>
<thead>
<tr>
<th>Object</th>
<th>Timestamp</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>2, 3, 5</td>
</tr>
<tr>
<td>A</td>
<td>20</td>
<td>6, 1</td>
</tr>
<tr>
<td>A</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>4, 5, 6</td>
</tr>
<tr>
<td>B</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>21</td>
<td>7, 8, 1, 2</td>
</tr>
<tr>
<td>B</td>
<td>28</td>
<td>1, 6</td>
</tr>
<tr>
<td>C</td>
<td>14</td>
<td>1, 8, 7</td>
</tr>
</tbody>
</table>

Timeline

Object A:

Object B:

Object C:
# Examples of Sequence Data

<table>
<thead>
<tr>
<th>Sequence 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>A set of items bought by a customer at time $t$</td>
<td>Books, diary products, CDs, etc</td>
</tr>
<tr>
<td>Web Data</td>
<td>Browsing activity of a particular Web visitor</td>
<td>A collection of files viewed by a Web visitor after a single mouse click</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>Events triggered by a sensor at time $t$</td>
<td>Types of alarms generated by sensors</td>
</tr>
<tr>
<td>Genome sequences</td>
<td>DNA sequence of a particular species</td>
<td>An element of the DNA sequence</td>
<td>Bases A,T,G,C</td>
</tr>
</tbody>
</table>

![Diagram](image)
Formal Definition of a Sequence

- A sequence is an ordered list of elements (transactions)

\[ s = < e_1, e_2, e_3, ... > \]

- Each element contains a collection of events (items)

\[ e_i = \{i_1, i_2, ..., i_k\} \]

- 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)
Examples of Sequences

● Web sequence:

< {Homepage} {Electronics} {Digital Cameras} {Canon Digital Camera} {Shopping Cart} {Order Confirmation} {Return to Shopping} >

● Sequence of initiating events causing the nuclear accident at 3-mile Island:

< {clogged resin} {outlet valve closure} {loss of feedwater} {condenser polisher outlet valve shut} {booster pumps trip} {main waterpump trips} {main turbine trips} {reactor pressure increases} >

● Sequence of books checked out at a library:

< {Fellowship of the Ring} {The Two Towers} {Return of the King} >
Formal Definition of a Subsequence

- A sequence \(<a_1 \ a_2 \ldots \ a_n>\) is contained in another sequence \(<b_1 \ b_2 \ldots \ b_m>\) \((m \geq n)\) if there exist integers \(i_1 < i_2 < \ldots < i_n\) such that \(a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_n \subseteq b_{i_n}\)

<table>
<thead>
<tr>
<th>Data sequence</th>
<th>Subsequence</th>
<th>Contained?</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;{2,4} {3,5,6} {8}&gt;)</td>
<td>(&lt;{2} {3,5}&gt;)</td>
<td></td>
</tr>
<tr>
<td>(&lt;{1,2} {3,4}&gt;)</td>
<td>(&lt;{1} {2}&gt;)</td>
<td></td>
</tr>
<tr>
<td>(&lt;{2,4} {2,4} {2,5}&gt;)</td>
<td>(&lt;{2} {4}&gt;)</td>
<td></td>
</tr>
</tbody>
</table>

- The \textit{support} of a subsequence \(w\) is defined as the fraction of data sequences that contain \(w\)

- A \textit{sequential pattern} is a frequent subsequence (i.e., a subsequence whose support is \(\geq \text{minsup}\))
Formal Definition of a Subsequence

- A sequence \(<a_1 \ a_2 \ldots \ a_n>\) is contained in another sequence \(<b_1 \ b_2 \ldots \ b_m>\) (\(m \geq n\)) if there exist integers \(i_1 < i_2 < \ldots < i_n\) such that \(a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_n \subseteq b_{i_n}\)

<table>
<thead>
<tr>
<th>Data sequence</th>
<th>Subsequence</th>
<th>Contained?</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;{2,4} {3,5,6} {8}&gt;)</td>
<td>(&lt;{2} {3,5}&gt;)</td>
<td>Yes</td>
</tr>
<tr>
<td>(&lt;{1,2} {3,4}&gt;)</td>
<td>(&lt;{1} {2}&gt;)</td>
<td>No</td>
</tr>
<tr>
<td>(&lt;{2,4} {2,4} {2,5}&gt;)</td>
<td>(&lt;{2} {4}&gt;)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- 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, \( \textit{minsup} \)

- Task:
  - Find all subsequences with support \( \geq \textit{minsup} \)
Sequential Pattern Mining: Challenge

- Given a sequence: `<{a b} {c d e} {f} {g h i}>`
  - Examples of subsequences:
    `<{a} {c d} {f} {g} >, <{c d e}>, <{b} {g}>`, etc.

- How many k-subsequences can be extracted from a given n-sequence?

  `<{a b} {c d e} {f} {g h i}>`  \( n = 9 \)

  \( k = 4 \):

  \[
  \begin{array}{cccccccccc}
  & & & & & & & & Y & \\
  & & & & & & & Y & Y & \\
  & & & & & & Y & & & \\
  & & & & & & & & & Y \\
  \end{array}
  \]

  Answer:

  \[
  \binom{n}{k} = \binom{9}{4} = 126
  \]
Sequential Pattern Mining: Example

Minsup = 50%

Examples of Frequent Subsequences:

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

• 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_1\}>, <\{i_1\} \{i_2\}>, \ldots, <\{i_n\} \{i_n\}>

• Candidate 3-subsequences:
  $<\{i_1, i_2, i_3\}>, <\{i_1, i_2, i_4\}>, \ldots, <\{i_1, i_2\} \{i_1\}>, <\{i_1, i_2\} \{i_2\}>, \ldots,$
  $<\{i_1\} \{i_1, i_2\}>, <\{i_1\} \{i_1, i_3\}>, \ldots, <\{i_1\} \{i_1\} \{i_1\}>, <\{i_1\} \{i_1\} \{i_2\}>, \ldots$
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 \(\text{minsup}\)
Candidate Generation

- **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):**
  - A frequent \((k - 1)\)-sequence \(w_1\) is merged with another frequent \((k - 1)\)-sequence \(w_2\) 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 sequences**
  \[w_1 = \{1\} \{2 3\} \{4\}\] and \[w_2 = \{2 3\} \{4 5\}\]
  will produce the candidate sequence \[\{1\} \{2 3\} \{4 5\}\] because the last two events in \(w_2\) (4 and 5) belong to the same element

- **Merging sequences**
  \[w_1 = \{1\} \{2 3\} \{4\}\] and \[w_2 = \{2 3\} \{4\} \{5\}\]
  will produce the candidate sequence \[\{1\} \{2 3\} \{4\} \{5\}\] because the last two events in \(w_2\) (4 and 5) do not belong to the same element

- **We do not have to merge these sequences**
  \[w_1 = \{1\} \{2 6\} \{4\}\] and \[w_2 = \{1\} \{2\} \{4 5\}\]
  to produce the candidate \[\{1\} \{2 6\} \{4 5\}\] because if the latter is a viable candidate, then it can be obtained by merging \(w_1\) with \[\{1\} \{2 6\} \{5\}\]
GSP Example

Frequent 3-sequences

Candidate Generation

Candidate Pruning
Timing Constraints (I)

\[
\begin{array}{cccc}
\{A\} & \{B\} & \{C\} & \{D\} \{E\} \\
\hline
\leq x_g & > n_g & \leq m_s \\
\end{array}
\]

\(x_g: \text{max-gap}\)

\(n_g: \text{min-gap}\)

\(m_s: \text{maximum span}\)

\(x_g = 2, n_g = 0, m_s = 4\)

<table>
<thead>
<tr>
<th>Data sequence</th>
<th>Subsequence</th>
<th>Contain?</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; {2,4} {3,5,6} {4,7} {4,5} {8} &gt;</td>
<td>&lt; {6} {5} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {1} {2} {3} {4} {5} &gt;</td>
<td>&lt; {1} {4} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {1} {2,3} {3,4} {4,5} &gt;</td>
<td>&lt; {2} {3} {5} &gt;</td>
<td></td>
</tr>
<tr>
<td>&lt; {1,2} {3} {2,3} {3,4} {2,4} {4,5} &gt;</td>
<td>&lt; {1,2} {5} &gt;</td>
<td></td>
</tr>
</tbody>
</table>
Timing Constraints (I)

- **x_g**: max-gap
- **n_g**: min-gap
- **m_s**: maximum span

\[
\{A, B\} \quad \{C\} \quad \{D, E\} \\
<= x_g \quad > n_g \\
<= m_s
\]

\( x_g = 2, \quad n_g = 0, \quad m_s = 4 \)

<table>
<thead>
<tr>
<th>Data sequence</th>
<th>Subsequence</th>
<th>Contain?</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; {2,4} {3,5,6} {4,7} {4,5} {8} &gt;</td>
<td>&lt; {6} {5} &gt;</td>
<td>Yes</td>
</tr>
<tr>
<td>&lt; {1} {2} {3} {4} {5} &gt;</td>
<td>&lt; {1} {4} &gt;</td>
<td>No</td>
</tr>
<tr>
<td>&lt; {1} {2,3} {3,4} {4,5} &gt;</td>
<td>&lt; {2} {3} {5} &gt;</td>
<td>Yes</td>
</tr>
<tr>
<td>&lt; {1,2} {3} {2,3} {3,4} {2,4} {4,5} &gt;</td>
<td>&lt; {1,2} {5} &gt;</td>
<td>No</td>
</tr>
</tbody>
</table>
Mining Sequential Patterns with Timing Constraints

Approach 1:

- Mine sequential patterns without timing constraints
- Post-process the discovered patterns

Approach 2:

- Modify GSP to directly prune candidates that violate timing constraints

Question:

- Does 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>1,2,4</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>2,3</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1,2</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>2,3,4</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1,2</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>2,3,4</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2,4,5</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>4,5</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>1,3</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>2,4,5</td>
</tr>
</tbody>
</table>

Suppose:

\[ x_g = 1 \text{ (max-gap)} \]
\[ n_g = 0 \text{ (min-gap)} \]
\[ m_s = 5 \text{ (maximum span)} \]
\[ \text{mins}up = 60\% \]

\(<\{2\} \{5\}> \text{ support } = 40\% \]
\[ \text{but} \]
\(<\{2\} \{3\} \{5\}> \text{ support } = 60\% \]

Problem exists because of max-gap constraint
No such problem if max-gap is infinite
**Contiguous Subsequences**

- **s** is a contiguous subsequence of
  \[ w = \langle e_1 \rangle \langle e_2 \rangle \ldots \langle e_k \rangle \]
  if any of the following conditions hold:
  1. **s** is obtained from **w** by deleting an item from either **e₁** or **eₖ**
  2. **s** is obtained from **w** by deleting an item from any element **eᵢ** that contains at least 2 items
  3. **s** is a contiguous subsequence of **s’** and **s’** is a contiguous subsequence of **w** (recursive definition)

**Examples:** \( s = \langle \{1\} \{2\} \rangle \)
- is a contiguous subsequence of \( \langle \{1\} \{2\} \{3\} \rangle, \langle \{1\} \{2\} \{3\} \rangle, \text{ and } \langle \{3\} \{4\} \{1\} \{2\} \{2\} \{3\} \{4\} \rangle \)
- is not a contiguous subsequence of \( \langle \{1\} \{3\} \{2\} \rangle \) and \( \langle \{2\} \{1\} \{3\} \{2\} \rangle \)
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)

\[
\begin{array}{c}
\{A\ B\} & \{C\} & \{D\ E\} \\
\leq x_g & > n_g & \leq ws \\
\leq m_s
\end{array}
\]

- \(x_g\): max-gap
- \(n_g\): min-gap
- \(ws\): window size
- \(m_s\): maximum span

\[x_g = 2,\ n_g = 0,\ ws = 1,\ m_s = 5\]

<table>
<thead>
<tr>
<th>Data sequence</th>
<th>Subsequence</th>
<th>Contain?</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;\ {2,4}\ {3,5,6} {4,7} {4,6} {8}&gt;)</td>
<td>(&lt;\ {3}\ {5} &gt;)</td>
<td>No</td>
</tr>
<tr>
<td>(&lt;\ {1}\ {2} {3} {4} {5}&gt;)</td>
<td>(&lt;\ {1,2}\ {3} &gt;)</td>
<td>Yes</td>
</tr>
<tr>
<td>(&lt;\ {1,2}\ {2,3} {3,4} {4,5}&gt;)</td>
<td>(&lt;\ {1,2}\ {3,4} &gt;)</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Modified Support Counting Step

- Given a candidate pattern: \(<\{a, c\}>\)
  - Any data sequences that contain

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

  will contribute to the support count of candidate 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: <E1> <E3>
General Support Counting Schemes

Object's Timeline

Sequence: (p) (q)

<table>
<thead>
<tr>
<th>Method</th>
<th>Support Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>COBJ</td>
<td>1</td>
</tr>
<tr>
<td>CWIN</td>
<td>6</td>
</tr>
<tr>
<td>CMINWIN</td>
<td>4</td>
</tr>
<tr>
<td>CDIST_O</td>
<td>8</td>
</tr>
<tr>
<td>CDIST</td>
<td>5</td>
</tr>
</tbody>
</table>

Assume:

- $x_g = 2$ (max-gap)
- $n_g = 0$ (min-gap)
- $ws = 0$ (window size)
- $m_s = 2$ (maximum span)
Frequent Subgraph Mining

- Extend association rule mining to finding frequent subgraphs
- Useful for web mining, computational chemistry, bioinformatics, spatial data sets, etc.
Graph Definitions

(a) Labeled Graph
(b) Subgraph
(c) Induced Subgraph
Representing Transactions as Graphs

- Each transaction is a clique of items

<table>
<thead>
<tr>
<th>Transaction Id</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{A, B, C, D}</td>
</tr>
<tr>
<td>2</td>
<td>{A, B, E}</td>
</tr>
<tr>
<td>3</td>
<td>{B, C}</td>
</tr>
<tr>
<td>4</td>
<td>{A, B, D, E}</td>
</tr>
<tr>
<td>5</td>
<td>{B, C, D}</td>
</tr>
</tbody>
</table>
Representing Graphs as Transactions

G1

G2

G3

<table>
<thead>
<tr>
<th></th>
<th>(a,b,p)</th>
<th>(a,b,q)</th>
<th>(a,b,r)</th>
<th>(b,c,p)</th>
<th>(b,c,q)</th>
<th>(b,c,r)</th>
<th>...</th>
<th>(d,e,r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>G2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>G3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>G3</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Challenges

● Node may contain duplicate labels
● Support and confidence
  – How to define them?
● Additional constraints imposed by pattern structure
  – Support and confidence are not the only constraints
  – Assumption: frequent subgraphs must be connected
● Apriori-like approach:
  – Use frequent k-subgraphs to generate frequent (k+1) subgraphs
  ◆ What is k?
Challenges...

- Support:
  - number of graphs that contain a particular subgraph

- Apriori principle still holds

- Level-wise (Apriori-like) approach:
  - Vertex growing:
    - $k$ is the number of vertices
  - Edge growing:
    - $k$ is the number of edges
Vertex Growing

\[ G_1 \]
\[
\begin{pmatrix}
0 & p & p & q \\
p & 0 & r & 0 \\
p & r & 0 & 0 \\
q & 0 & 0 & 0
\end{pmatrix}
\]

\[ G_2 \]
\[
\begin{pmatrix}
0 & p & p & 0 \\
p & 0 & r & 0 \\
p & r & 0 & r \\
0 & 0 & r & 0
\end{pmatrix}
\]

\[ G_3 = \text{join}(G_1, G_2) \]
\[
\begin{pmatrix}
0 & p & p & 0 & q \\
p & 0 & r & 0 & 0 \\
p & r & 0 & r & 0 \\
0 & 0 & r & 0 & 0 \\
q & 0 & 0 & 0 & 0
\end{pmatrix}
\]
Edge Growing

G1 + G2 → G3 = join(G1, G2)
Apriori-like Algorithm

- Find frequent 1-subgraphs

- Repeat
  - Candidate generation
    - Use frequent \((k - 1)\)-subgraphs to generate candidate \(k\)-subgraph
  - Candidate pruning
    - Prune candidate subgraphs that contain infrequent \((k - 1)\)-subgraphs
  - Support counting
    - Count the support of each remaining candidate
  - Eliminate candidate \(k\)-subgraphs that are infrequent

In practice, it is not as easy. There are many other issues.
Example: Dataset

G1

G2

G3

G4

<table>
<thead>
<tr>
<th></th>
<th>(a,b,p)</th>
<th>(a,b,q)</th>
<th>(a,b,r)</th>
<th>(b,c,p)</th>
<th>(b,c,q)</th>
<th>(b,c,r)</th>
<th>...</th>
<th>(d,e,r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>G2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>G3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>G4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>
Example

Minimum support count = 2

k=1
Frequent Subgraphs

k=2
Frequent Subgraphs

k=3
Candidate Subgraphs

(Pruned candidate)
Candidate Generation

- In Apriori:
  - Merging two frequent $k$-itemsets will produce a candidate $(k + 1)$-itemset

- In frequent subgraph mining (vertex/edge growing)
  - Merging two frequent $k$-subgraphs may produce more than one candidate $(k + 1)$-subgraph
Multiplicty of Candidates (Vertex Growing)

\[
\begin{align*}
M_{G1} &= \begin{pmatrix}
0 & p & p & q \\
p & 0 & r & 0 \\
p & r & 0 & 0 \\
q & 0 & 0 & 0 \\
\end{pmatrix} \\
M_{G2} &= \begin{pmatrix}
0 & p & p & 0 \\
p & 0 & r & 0 \\
p & r & 0 & r \\
0 & 0 & r & 0 \\
\end{pmatrix} \\
M_{G3} &= \begin{pmatrix}
0 & p & p & 0 & q \\
p & 0 & r & 0 & 0 \\
p & r & 0 & r & 0 \\
0 & 0 & r & 0 & ? \\
q & 0 & 0 & ? & 0 \\
\end{pmatrix}
\end{align*}
\]

G1 + G2 = G3 = join(G1,G2)
Multiplicity of Candidates (Edge growing)

- Case 1: identical vertex labels
Core: The (k-1) subgraph that is common between the joint graphs
Multiplicity of Candidates (Edge growing)

- Case 3: Core multiplicity
\textbf{Adjacency Matrix Representation}

\begin{itemize}
  \item The same graph can be represented in many ways
\end{itemize}

\begin{table}
\centering
\begin{tabular}{|c|cccccccc|}
\hline
 & \textbf{A(1)} & \textbf{A(2)} & \textbf{A(3)} & \textbf{A(4)} & \textbf{B(5)} & \textbf{B(6)} & \textbf{B(7)} & \textbf{B(8)} \\
\hline
\textbf{A(1)} & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
\textbf{A(2)} & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
\textbf{A(3)} & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
\textbf{A(4)} & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
\textbf{B(5)} & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
\textbf{B(6)} & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
\textbf{B(7)} & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
\textbf{B(8)} & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
\hline
\end{tabular}
\end{table}
Graph Isomorphism

A graph is isomorphic if it is topologically equivalent to another graph.
Graph Isomorphism

- Test for graph isomorphism is needed:
  - During candidate generation step, to determine whether a candidate has been generated
  - During candidate pruning step, to check whether its \((k - 1)\)-subgraphs are frequent
  - During candidate counting, to check whether a candidate is contained within another graph
Graph Isomorphism

- Use canonical labeling to handle isomorphism
  - Map each graph into an ordered string representation (known as its code) such that two isomorphic graphs will be mapped to the same canonical encoding
  - Example:
    - Lexicographically largest adjacency matrix

```
Graph 1:
{0 0 1 0}
{0 0 1 1}
{1 1 0 1}
{0 1 1 0}

String: 0010001111010110

Graph 2:
{0 1 1 1}
{1 0 1 0}
{1 1 0 0}
{1 0 0 0}

Canonical: 011101011001000
```