Chapter 8, Sequence Data Mining

Young-Rae Cho
Associate Professor
Department of Computer Science
Baylor University

Topics

- **Single Sequence Mining**
  - Frequent sequence pattern mining
  - Finding sub-sequences that frequently occur in a sequence

- **Sequence Dataset Mining**
  - Frequent sequence pattern mining
  - Finding sub-sequences that frequently occur among sequences
  - Sequence data clustering
  - Grouping similar sequences
  - Sequence data classification
    - Classifying a new sequence
Applications

- **Examples**
  - Customer shopping sequential patterns
    - e.g., First buy a computer, then a CD-ROM, and then a printer within 3 months
  - Stock market changes
  - Web-log patterns
  - Medical treatment records
  - Gene or protein sequences
  - Natural disaster records

- **Challenges**
  - Finding the complete set satisfying the minimum support threshold
  - Developing efficient and scalable algorithms
  - Incorporating various kinds of user-specific constraints

Problem Definition

- **Scope**
  - Transaction data → Sequential transaction data
  - Frequent itemset patterns → (Frequent) Sequential patterns

- **Definitions**
  - Sequence: an ordered list of items, e.g., $x = \langle a, c, f, e, c, d \rangle$
  - Sub-sequence of $x$: an ordered sequence of items from $x$
    - Not necessarily consecutive (different from a substring)
    - $\langle a, e, c \rangle$, $\langle c, f, d \rangle$, $\langle a, c \rangle$, $\langle c, e, c \rangle$, $\langle a, f, c, d \rangle$, $\langle e, f, e \rangle$
  - Frequent sequence pattern

<table>
<thead>
<tr>
<th>SID</th>
<th>sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;acadd&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;abcaed&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;efadfcbe&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;egafcbde&gt;</td>
</tr>
</tbody>
</table>

$\langle a, c, d \rangle$ is a frequent sequence pattern with 75% minimum support
**Extended Problem Definition**

- **Extended Definitions**
  - Consider “or” at a single position
  - **Sequence**: An ordered list of elements and each element is a set of items, e.g., \( x = (ab)c(df)(ef)d \)
  - **Sub-sequence of** \( x \)
    - \((ab)d <a c (ef)> <a f e> <(ab) e f> <b f d> <c d f>\)
  - **Frequent sequence pattern**

<table>
<thead>
<tr>
<th>SID</th>
<th>sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(&lt;a(abc)(ac)d(cf)&gt;)</td>
</tr>
<tr>
<td>20</td>
<td>(&lt;(ab)a(bc)(ae)&gt;)</td>
</tr>
<tr>
<td>30</td>
<td>(&lt;(ef)(ab)(df)cb&gt;)</td>
</tr>
<tr>
<td>40</td>
<td>(&lt;eg(af)cbc&gt;)</td>
</tr>
</tbody>
</table>

\((ab) c\) is a frequent sequence pattern with 75% minimum support

**Properties of Sequence Patterns**

- **Properties**
  - Anti-monotonic property \(\rightarrow\) Apriori algorithm
  - If a sequence \( S \) is not frequent, then none of the super-sequences of \( S \) are frequent

- **Example**
  - If \(<(ab)d>\) is infrequent,
    so do \(<(ab)de>\) and \(<(abc)d>\) and \(<a(ab)d>\)

<table>
<thead>
<tr>
<th>SID</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(&lt;a(abc)(ac)d(cf)&gt;)</td>
</tr>
<tr>
<td>20</td>
<td>(&lt;(ab)a(bc)(ae)&gt;)</td>
</tr>
<tr>
<td>30</td>
<td>(&lt;(ef)(ab)(df)cb&gt;)</td>
</tr>
<tr>
<td>40</td>
<td>(&lt;eg(af)cbc&gt;)</td>
</tr>
</tbody>
</table>

min\_sup = 75%
GSP Algorithm

- **GSP**
  - Generalized Sequential Pattern mining

- **Algorithm**
  1. Initially, find all frequent length-1 sequences ( = frequent 1-itemset )
  2. Generate candidate length-(k+1) sequences from frequent length-k sequences
  3. Count support for each candidate sequence to select frequent sequences
  4. Repeat (2) and (3) until no frequent sequence or no candidate is found

Length-1 Sequences

- **Example**
  - | SID | sequence |
  - |-----|---------|
  - | 10  | <a(abc)(ac)d(cf)> |
  - | 20  | <(ab)a(bc)(ae)> |
  - | 30  | <(ef)(ab)(df)cb> |
  - | 40  | <eg(af)cbc> |

  - min_sup = 75%

- **Candidate Length-1 Sequences**
  - <a>, <b>, <c>, <d>, <e>, <f>, <g>

- **Frequent Length-1 Sequences**
  - <a>, <b>, <c>, <e>, <f>
### Length-2 Sequences

#### Candidate & Frequent Length-2 Sequences

<table>
<thead>
<tr>
<th>SID</th>
<th>&lt;a&gt;</th>
<th>&lt;b&gt;</th>
<th>&lt;c&gt;</th>
<th>&lt;e&gt;</th>
<th>&lt;f&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a&gt;</td>
<td>&lt;aa&gt;</td>
<td>&lt;ab&gt;</td>
<td>&lt;ba&gt;</td>
<td>&lt;ac&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;b&gt;</td>
<td>&lt;bb&gt;</td>
<td>&lt;bc&gt;</td>
<td>&lt;cb&gt;</td>
<td>&lt;be&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;c&gt;</td>
<td>&lt;cc&gt;</td>
<td>&lt;ce&gt;</td>
<td>&lt;ec&gt;</td>
<td>&lt;cf&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;f&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Length-3 Sequences

#### Candidate & Frequent Length-3 Sequences

<table>
<thead>
<tr>
<th>SID</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ab)a(bc)(ae)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;ef(ab)(df)cb&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(af)cbc&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SID</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ab)a(bc)(ae)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;ef(ab)(df)cb&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(af)cbc&gt;</td>
</tr>
</tbody>
</table>
Summary of GSP Algorithm

- **Strength**
  - Apriori pruning

- **Weakness**
  - Generates a huge set of candidate sequences
  - Requires multiple scans of database
  - Inefficient for mining long sequential patterns

- **References**

PrefixSpan Algorithm

- **PrefixSpan**
  - Prefix-projected sequential pattern mining

- **Prefix**
  - Suppose all items in an element are listed alphabetically.
  - Given $x = <e_1, e_2, \ldots, e_m>$, $y = <e'_1, e'_2, \ldots, e'_m>$ $(m \leq n)$ is a prefix of $x$ if and only if (1) $e_i = e'_i$ for $(i \leq m-1)$ (2) $e'_m \subseteq e_m$ (3) all items in $(e_m - e'_m)$ are alphabetically after those in $e'_m$
  - $<a>$, $<aa>$, $<a(ab)>$ and $<a(abc)>$ are prefixes of $<a(abc)(ac)d(cf)>$

- **Main Idea**
  - Keep track of prefixes (instead of all candidate sequences) from the sequence database
  - Project their suffixes into projected databases
Projected Database

- **Definition**
  - A set of maximum suffixes of sub-sequences of a given prefix

- **Example**
  - <a>-projected database includes
    - <(abc)(ac)d(cf)>
    - <(_b)a(bc)(ae)>
    - <(_b)(df)cb>
    - <(_f)cbc>

<table>
<thead>
<tr>
<th>SID</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;(abc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ab)a(bc)(ae)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)cb&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(af)cbc&gt;</td>
</tr>
</tbody>
</table>

PrefixSpan Algorithm

- **Algorithm**
  1. Find all frequent length-1 sequences
     - <a>, <b>, <c>, <e>, <f>
  2. Divide search space
     - Sequences having prefix <a>
     - Sequences having prefix <b>
     - Sequences having prefix <c>
     - Sequences having prefix <e>
     - Sequences having prefix <f>
  3. Construct projected database for each search space
  4. Find subsets of sequential patterns recursively
PrefixSpan Algorithm

Example

<table>
<thead>
<tr>
<th>SID</th>
<th>sequence</th>
<th>min_sup = 75%</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ab)a(bc)(ae)&gt;</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)cb&gt;</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(af)cabc&gt;</td>
<td></td>
</tr>
</tbody>
</table>

pattern   | projected database |
---        |-------------------|
<a>       | <(abc)(ac)d(cf)>,  |
          | <(_b)a(bc)(ae)>,  |
          | <(_b)(df)c>cb,    |
          | <(_f)c>bc>        |
<b>       | <(_c)(ac)d(cf)>,   |
          | <a(bc)(ae)>,      |
          | <(df)c>cb, <c>    |
<c>       | <(ac)d(cf)>, <(ae)>, |
          | <b>, <bc>         |
<e>       | <(_f)(ab)(df)c>cb, |
          | <g(af)c>bc>       |
<f>       | <(ab)(df)c>cb, <abc> |

PrefixSpan Algorithm

Example (continued)

pattern   | projected database |
---        |-------------------|
<ab>      | <(_c)(ac)d(cf)>,   |
          | <(_c)(ae)>, <c>    |
<ac>      | <(ac)d(cf)>, <(ae)>, |
          | <b>, <bc>         |
<bc>      | <(_c)(ac)d(cf)>,   |
          | <a(bc)(ae)>, <(df)c> |

PrefixSpan Algorithm

- **Example (continued)**

<table>
<thead>
<tr>
<th>pattern</th>
<th>projected database</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ab&gt;</td>
<td>&lt;(_c)(ac)d(cf)&gt;, &lt;(_c)(ae)&gt;, &lt;c&gt;</td>
</tr>
<tr>
<td>&lt;ac&gt;</td>
<td>&lt;(ac)d(cf)&gt;, &lt;(ae)&gt;, &lt;b&gt;, &lt;bc&gt;</td>
</tr>
<tr>
<td>&lt;(ab)&gt;</td>
<td>&lt;(_c)(ac)d(cf)&gt;, &lt;a(bc)(ae)&gt;, &lt;(df)cb&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pattern</th>
<th>projected database</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;(ab)c&gt;</td>
<td>&lt;d(cf)&gt;, &lt;(ae)&gt;, &lt;b&gt;</td>
</tr>
</tbody>
</table>

Summary of PrefixSpan Algorithm

- **Strength**
  - Efficient (major cost is to construct projected databases)
  - Project databases keep shrinking rapidly

- **Weakness**
  - Searching frequency redundantly

- **References**
Constraint-Based Sequence Mining

- **Constraint Types**
  - Prefix anti-monotonic
    - If a sequence $s$ violates the constraint $C$, then so does any sequence having $s$ as a prefix
  - Prefix monotonic
    - If a sequence $s$ satisfies the constraint $C$, then so does any sequence having $s$ as a prefix

- **Application**
  - Push prefix anti-monotonic constraints for early pruning

- **Reference**
  - Pei, J., Han, J. and Wang, W, "Mining sequential patterns with constraints in large databases." *In Proceedings of CIKM (2002)*

Questions?

- Lecture Slides on the Course Website,
  "www.ecs.baylor.edu/faculty/cho/4352"