Sequential Pattern Mining

- Given
  - a set of sequences, where each sequence consists of a list of elements and each element consists of set of items
  - user-specified min_support threshold

<table>
<thead>
<tr>
<th>id</th>
<th>Sequence</th>
<th>Length</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>20</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>30</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

Solution – 53 frequent subsequences

- Find all the frequent subsequences, i.e. the subsequences whose occurrence frequency in the set of sequences is no less than min_support

\[
\begin{array}{|c|c|}
\hline
\text{id} & \text{Sequence} \\
\hline
10 & <a(abc)(ac)d(cf)> \\
20 & <a(abc)(ac)d(cf)> \\
30 & <a(abc)(ac)d(cf)> = <a(cba)(ac)d(cf)> \\
40 & <eg(af)dcb> \\
\hline
\end{array}
\]

\[
\text{min_support} = 2
\]
Subsequence vs. super sequence

- Given two sequences $\alpha = \langle a_1 a_2 \ldots a_n \rangle$ and $\beta = \langle b_1 b_2 \ldots b_m \rangle$
- $\alpha$ is called a subsequence of $\beta$, denoted as $\alpha \subseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \ldots < j_n \leq m$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \ldots, a_n \subseteq b_{j_n}$
- $\beta$ is a super sequence of $\alpha$

$\beta = \langle a(abc)(ac)d(cf) \rangle$
$\alpha_1 = \langle aa(ac)d(c) \rangle$
$\alpha_2 = \langle (ac)(ac)d(cf) \rangle$
$\alpha_3 = \langle ac \rangle$
$\alpha_4 = \langle df(cf) \rangle$
$\alpha_5 = \langle (cf)d \rangle$
$\alpha_6 = \langle (abc)dcf \rangle$

Sequence Support Count

- A sequence database is a set of tuples $\langle \text{id}, s \rangle$
- A tuple $\langle \text{id}, s \rangle$ is said to contain a sequence $\alpha$, if $\alpha$ is a subsequence of $s$, i.e., $\alpha \subseteq s$
- The support of a sequence $\alpha$ is the number of tuples containing $\alpha$

<table>
<thead>
<tr>
<th>Id</th>
<th>Sequence</th>
<th>$\alpha$</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>$\langle a(abc)(ac)d(cf) \rangle$</td>
<td>$\alpha_1$</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>$\langle (ef)(ab)(df)cb \rangle$</td>
<td>$\alpha_2$</td>
<td>4</td>
</tr>
<tr>
<td>30</td>
<td>$\langle (ad)c(bc)(ae) \rangle$</td>
<td>$\alpha_3$</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>$\langle eg(af)cbc \rangle$</td>
<td>$\alpha_4$</td>
<td>30</td>
</tr>
</tbody>
</table>

Strategies

- Apriori-property based
  - AprioriSome (1995)
  - AprioriAll (1995)
  - DynamicSome (1995)
  - GSP (1996)

- Regular expression constraints
  - SPIRIT (1999)

- Data projection based
  - FreeSpan (2000)

Outline

- Mining Sequential Patterns
  - Problem statement
  - Definitions & examples
  - Strategies
  - PrefixSpan algorithm
    - Motivation
    - Definitions & examples
    - Algorithm
    - Example
    - Performance study
- Conclusions
Motivation and Background

- Shortcomings of Apriori-like approaches
  - Potentially huge set of candidate sequences
  - Multiple scans of databases
  - Difficulties at mining long sequential patterns

- FreeSpan (Frequent pattern-projected Sequential pattern mining) – pattern growth method
  - General idea is to use frequent items to recursively project sequence databases into a smaller projected databases and grow subsequence fragments in each projected database

- PrefixSpan (Prefix-projected Sequential pattern mining)
  - Less projections and quickly shrinking sequences

Prefix

- Given two sequences \( \alpha=\langle a_1a_2...a_n \rangle \) and \( \beta=\langle b_1b_2...b_m \rangle \), \( m \leq n \)
- Sequence \( \beta \) is called a prefix of \( \alpha \) if and only if:
  - \( b_i = a_i \) for \( i \leq m-1 \);
  - \( b_m \subseteq a_m \);
  - All the items in \( (a_m - b_m) \) are alphabetically after those in \( b_m \)

\[ \alpha = \langle a(abc)(ac)d(cf) \rangle \]
\[ \beta = \langle a(abc)a \rangle \]

Projection

- Given sequences \( \alpha \) and \( \beta \), such that \( \beta \) is a subsequence of \( \alpha \).
- A subsequence \( \alpha' \) of sequence \( \alpha \) is called a projection of \( \alpha \) w.r.t. \( \beta \) prefix if and only if
  - \( \alpha' \) has prefix \( \beta \);
  - There exist no proper super-sequence \( \alpha'' \) of \( \alpha' \) such that \( \alpha'' \) is a subsequence of \( \alpha \) and also has prefix \( \beta \)

\[ \alpha = \langle a(abc)(ac)d(cf) \rangle \]
\[ \beta = \langle (bc)a \rangle \]
\[ \alpha' = \langle (bc)(ac)d(cf) \rangle \]

Postfix

- Let \( \alpha' = \langle a_1a_2...a_n \rangle \) be the projection of \( \alpha \) w.r.t. prefix \( \beta = \langle a_1a_2...a_{m-1}a'_m \rangle \) (\( m \leq n \))
- Sequence \( \gamma = \langle a''_{m+1}...a_n \rangle \) is called the postfix of \( \alpha \) w.r.t. prefix \( \beta \), denoted as \( \gamma = \alpha / \beta \), where \( a''_m = (a_m - a'_m) \)
- We also denote \( \alpha = \beta \cdot \gamma \)

\[ \alpha' = \langle a(abc)(ac)d(cf) \rangle \]
\[ \beta = \langle a(abc)a \rangle \]
\[ \gamma = \langle _c)d(cf) \rangle \]
PrefixSpan – Algorithm

- **Input**: A sequence database $S$, and the minimum support threshold $\text{min\_sup}$
- **Output**: The complete set of sequential patterns
- **Method**: Call PrefixSpan($<>$, 0, $S$)
- **Subroutine** PrefixSpan($\alpha$, $l$, $S|\alpha$)
- **Parameters**:
  - $\alpha$: sequential pattern,
  - $l$: the length of $\alpha$;
  - $S|\alpha$: the $\alpha$-projected database, if $\alpha \neq <>$; otherwise; the sequence database $S$.

PrefixSpan – Algorithm (2)

**Method**
1. Scan $S|\alpha$ once, find the set of frequent items $b$ such that:
   a) $b$ can be assembled to the last element of $\alpha$ to form a sequential pattern; or
   b) $<b>$ can be appended to $\alpha$ to form a sequential pattern.
2. For each frequent item $b$, append it to $\alpha$ to form a sequential pattern $\alpha'$, and output $\alpha'$;
3. For each $\alpha'$, construct $\alpha'$-projected database $S|\alpha'$, and call PrefixSpan($\alpha'$, $l+1$, $S|\alpha'$).

PrefixSpan – Example

1. Find length-1 sequential patterns
   - $min\_support = 2$

2. Divide search space

PrefixSpan – Example (2)

3. Find subsets of sequential patterns
PrefixSpan - characteristics

- No candidate sequence needs to be generated by PrefixSpan
- Projected databases keep shrinking
- The major cost of PrefixSpan is the construction of projected databases

**How to reduce this cost?**

Different projection methods

- Bi-level projection
  - reduces the number and the size of projected databases
- Pseudo-Projection
  - reduces the cost of projection when projected database can be held in main memory

Bi-level Projection

- Scan to get 1-length sequences
- Construct a triangular matrix instead of projected databases for each length-1 patterns

<table>
<thead>
<tr>
<th>id</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a(b(c)de)(cf)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;a(d)(b(c)(ae)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;a(f)(ab)(df)c&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;a(ef)(abc)&gt;</td>
</tr>
</tbody>
</table>

**min_support = 2**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>(e)</th>
<th>(f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

ALL length-2 sequential pattern

Support(<ac>) = 4
Support(<ca>) = 2
Support(<(ac)>) = 1
Support(<cc>) = 3

Bi-level projection (2)

- For each length-2 sequential pattern α, construct the α-projected database and find the frequent items
- Construct corresponding S-matrix

Bi-level projection (3) - optimization

- "Do we need to include every item in a postfix in the projected databases?"
- **NO!** Item pruning in projected database by 3-way Apriori checking

<i>ab</i> <i>abc</i> <i>a(bc)></i>

<table>
<thead>
<tr>
<th>α</th>
<th>b</th>
<th>c</th>
<th>(c)</th>
<th>d</th>
<th>(d)</th>
<th>e</th>
<th>(e)</th>
<th>f</th>
<th>(f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<i>aca</i> is not frequent

Any super-sequence of it can never be a sequential pattern

<i>a(bc)>-projected database

<i>a(bd)> is not frequent

To construct <a(bc)>-projected database, sequence <a(bcdef)> should be projected to <(e)df> instead of <(e)def>
Pseudo-Projection

- **Observation**: postfixes of a sequence often appear repeatedly in recursive projected databases
- **Method**: instead of constructing physical projection by collecting all the postfixes, we can use pointers referring to the sequences in the database as a pseudo-projection
- Every projection consists of two pieces of information: pointer to the sequence in database and offset to the postfix in the sequence

```
\begin{tabular}{|c|c|c|}
\hline
Pointer & Offset & Postfix \\
\hline
s1 & 2 & \(\langle a\rangle\langle bc\rangle\langle ac\rangle\langle d\rangle\langle cf\rangle\rangle\) \\
\hline
s1 & 5 & \(\langle ac\rangle\langle d\rangle\langle cf\rangle\rangle\) \\
\hline
s1 & 6 & \(\langle _c\rangle\langle d\rangle\langle cf\rangle\rangle\) \\
\hline
\end{tabular}
```

Experimental Results

- **Environment**: 233MHz Pentium PC, 128 MB RAM, Windows NT, Visual C++ 6.0
- Reported test on synthetic data set: C10T8S8I8:
  - 1000 items
  - 10000 sequences
  - Average number of items within elements: 8
  - Average number of elements in a sequence: 8
- Competitors:
  - GSP
  - FreeSpan
  - PrefixSpan-1 (level-by-level projection)
  - PrefixSpan-2 (bi-level projection)

Runtime vs. support threshold

I/O costs vs. threshold and scalability
Outline

- Mining Sequential Patterns
  - Problem statement
  - Definitions & examples
  - Strategies
- PrefixSpan algorithm
  - Motivation
  - Definitions & examples
  - Algorithm
  - Example
  - Performance study
- Conclusions

Conclusions

- PrefixSpan
  - Efficient pattern growth method
  - Outperforms both GSP and FreeSpan
  - Explores prefix-projection in sequential pattern mining
  - Mines the complete set of patterns but reduces the effort of candidate subsequence generation
  - Prefix-projection reduces the size of projected database and leads to efficient processing
  - Bi-level projection and pseudo-projection may improve mining efficiency

References

- Pei J., Han J., Mortazavi-Asl J., Pinto H., Chen Q., Dayal U., Hsu M., PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth, 17th International Conference on Data Engineering (ICDE), April 2001
- Srikant R., Agrawal R., Mining sequential pattern: Generalizations and performance improvements, Proceedings 5th Int. /conf. Extending Database Technology (EDBT’96), pp. 3-17, 1996

THANK YOU !!!

Any Questions?