T-61.6020
PrefixSpan

Ari Nevalainen

ajnevala@cc.hut.fi
OUTLINE

- PROBLEM.
- METHOD.
- EXAMPLE
- ALGORITHM.
- SOME RESULTS.
- CONCLUSIONS.
- REFERENCES.
PROBLEM

- Sequential pattern mining with subsequences as patterns.
- A sequence database
  - `<a(abc)(ac)d(cf)>`
  - `<(ad)c(bc)(ae)>`
  - `<(ef)(ab)(df)cb>`
  - `<eg(af)cbc>`
- Inside a subsequence (...) items are listed alphabetically. An item can occur at most once in an subsequence.
- Subsequence with one item is written without brackets.
PrefixSpan is like APriori but, uses prefix-projection method to reduce a candidate generation.

- $a_i, b_j$ are items.
- $\alpha_i, \beta_j$ are itemsets.
- $\alpha, \beta$ are sequences of itemsets.
- $\alpha = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ and $\beta = \langle \beta_1, \beta_2, \ldots, \beta_m \rangle$. 
METHOD, subsequence

- $\alpha$ is subsequence of $\beta$, $\alpha \subseteq \beta$ if, 
  $\exists \ 1 \leq j_1 < \ldots < j_n \leq m$, such that 
  $\alpha_2 \subseteq \beta_{j_1}, \alpha_2 \subseteq \beta_{j_2}, \ldots, \alpha_n \subseteq \beta_{j_n}$.

- $\langle a(ab)c \subseteq a(abc)da(ac) \rangle$. 
METHOD, prefix, postfix

- \( \alpha = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle \) and \( \beta = \langle \beta_1, \beta_2, \ldots, \beta_m \rangle \).

- \( \alpha \) is prefix of \( \beta \) if \( \alpha_1 = \beta_1, \ldots, \alpha_{m-1} = \beta_{m-1}, \alpha_m \subseteq \beta_m \) and items in \( \beta_m - \alpha_m \) are alphabetically after those in \( \alpha_m \).

- \( a(ab)c \) is prefix of \( a(ab)(acd)d(ab) \).

- Sequence after prefix is postfix. \( (_d)d(ab) \) is postfix in \( a(ab)(acd)d(ab) \) after \( a(ab)c \).
METHOD, project

- Given $\beta \subseteq \alpha$, $\gamma \subseteq \alpha$, $\gamma$ is $\beta$-project of $\alpha$ if $\beta$ is prefix of $\gamma$ and there is no longer subsequence in $\alpha$ so that $\beta$ is its prefix.

- $c$-project of $a(ab)(cd)cd(ab)$ is $(\_d)cd(ab)$
Two types of last items, \( y \) and \( (_y) \).

- With \( y \): \( xyz \ldots \rightarrow z \ldots \)
- With \( y \): \( (xyz) \ldots \rightarrow (_z) \ldots \)
- With \( (_y) \): \( (vyz) \ldots \rightarrow (_z) \ldots \)
- With \( (_y) \): \( (\_xyz) \ldots \rightarrow (_z) \ldots \)

Where \( x, z \) can be zero, one or more items and \( v \) one or more items.
Method, next item from y-project.

- Two types of last items, $y$ and $(\_y)$.
- Every item that can be joined with $y, (\_y)$
- Items from b-project, $<(_c)(abc)d(cf)>$
  $(\_c),a,b,c,(\_c),d,c,f$
- Items from (_b)-project, $<(_c)(abc)d(cf)>$
  $(\_c),a,b,c,d,c,f$
Method, next item from y-project.

- Two types of last items, $y$ and ($_y$).
- With $y$: $xyz \rightarrow x_i, y, z_i$
- With $y$: $(xyz) \rightarrow x_i, y, z_i, (_zi)$
- With $y$: ($_xyz) \rightarrow (_xi), (_y), (_zi)$.
- with ($_y$): $xyz \rightarrow x_i, y, z_i$.
- with ($_y$): $(xyz) \rightarrow x_i, y, z_i$.
- with ($_y$): ($_zi) \rightarrow (_zi)$.
- Where $x, z$ can be zero, one or more items.
Method, joining items.

- Do projecting and scanning recursively until items can not be scanned any more.
- Join x with every item found from x-project and sequences from next round.

- $x, y \rightarrow xy$.
- $x, (_y) \rightarrow (xy)$.
- $(_x), y \rightarrow (_x)y$.
- $(_x), (_y) \rightarrow (_xy)$. 
EXAMPLE

- Finding sequential patterns starting with \{ab\}, and with support 2, from the database below.

- \langle a(abc)(ac)d(cf)\rangle
  \langle (ad)c(bc)(ae)\rangle
  \langle (ef)(ab)(df)cb\rangle
  \langle eg(af)cbc\rangle
EXAMPLE, cont

- \(<a(abc)(ac)d(cf)><(ad)c(bc)(ae)><(ef)(ab)(df)cb><eg(af)cbc>\)

- First we scan all items which have support at least 2. a,b,c,d,e,f

- a-project:
  \(<(abc)(ac)d(cf)><(_d)c(bc)(ae)><(_b)(df)cb><(_f)cbc>\)
EXAMPLE, cont

- a-project:
  \(<(abc)(ac)d(cf)>\)
  \(<(\_d)c(bc)(ae)>\)
  \(<(\_b)(df)cb>\)
  \(<(\_f)cbc>\)

- Items with support at least 2: \{a, b, c, d, f, (\_b)\}

- Items with support less than 2: \{e, (\_e), (\_c), (\_d), (\_f)\}

- Pruned a-project:
  \(<(abc)(ac)d(cf)>\)
  \(<c(bc)a>\)
  \(<(\_b)(df)cb>\)
  \(<cbc>\)
EXAMPLE, cont

- From a-project:
  - <(abc)(ac)d(cf)>
  - <c(bc)a>
  - <(_b)(df)cb>
  - <cbc>

- b-project:
  - <(_c)(ac)d(cf)>
  - <(_c)a>
  - <c>
EXAMPLE, cont

- **b-project:**
  
  \(<(_c)(ac)d(cf)>\)
  \(<(_c)a>\)
  \(<c>\)

- Items with support at least 2: \{(_c),a,c\}

- Items with support less than 2: \{d,f\}

- Pruned b-project:
  
  \(<(_c)(ac)c>\)
  \(<(_c)a>\)
  \(<c>\)
EXAMPLE, cont

- From b-project:
  \((\_c)(ac)c\)  
  \((\_c)a\)  
  \(c\)  
  \(c\)

- \((\_c)\)-project:
  \((ac)c\)  
  \(a\)
EXAMPLE, cont

- (\_c)-project:
  \((ac)c\)
  <a>

- Items with support at least 2: \{a\}
- Items with support less than 2: \{c\}
- Pruned (\_c)-project:
  <a>
  <a>

- a-project:
  <>
EXAMPLE, cont

- From b-project:
  \(((c)(ac)c)\)
  \(((c)a)\)
  \((c)\)

- a-project:
  \(((c)c)\)
  Only one sequence. We can not get anything with support 2.

- c-project:
  \((c)\)
  Only one sequence. We can not get anything with support 2.
EXAMPLE, items form a tree.

From the tree we get sequential patterns starting with ab:

a ab a(bc) a(bc)a aba abc
ALGORITHM

- Find from database all items which have at least support s. Add them to iList.
- call prefixSpan(Database,iList,s)
- program prefixSpan(Database,iList,s)
  for all items x in iList:
    Form x-project;
    Find supported Items;
    Prune x-project;
    If x-project has more than one sequence:
      nextLevel=prefixspan(x-project,Items,s);
      IList=join(x,Items+nextLevel);
  Return IList;
### SOME RESULTS, small support.

```
abzfghkilmqruvx
abcdefgijklmnopqrstuvwxyz
fghijklmnopqrstuvwxyz
klmnopqrstuvwxyz
pqrstuvwxyz

<table>
<thead>
<tr>
<th>Support</th>
<th>APriori</th>
<th>PrefixSpan</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0.001 s</td>
<td>0.003 s</td>
</tr>
<tr>
<td>5</td>
<td>0.009 s</td>
<td>0.146 s</td>
</tr>
<tr>
<td>4</td>
<td>0.748 s</td>
<td>0.160 s</td>
</tr>
<tr>
<td>3</td>
<td>362 s</td>
<td>3.435 s</td>
</tr>
<tr>
<td>2</td>
<td>&gt; 5000 s</td>
<td>131 s</td>
</tr>
</tbody>
</table>
```
SOME RESULTS, scalability.

- Good scalability:
CONCLUSIONS.

- PrefixSpan is faster than FreeSpan and GSP, when support value is small.
- PrefixSpan-2 use pseudo projections and is faster than prefixSpan.
- Pseudo projections:
  From \(<a(abc)(ac)d(cf)\>
  a-project \(<(abc)(ac)d(cf)\>
  ab-project \(<(_c)(ac)d(cf)\>
  Has lot of redundancy.
REFERENCES.