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 writen without brackets.

METHOD, definition

- PrefixSpan is like APriori but, uses prefix-projection method to reduce a candidate generation.
  - a_i, b_j are items.
  - α_i, β_j are itemsets.
  - α, β 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

- α is subsequence of β, $\alpha \subseteq \beta$ if, $\exists\ 1 \leq j_1 < \ldots < j_n \leq m$, such that $\alpha_2 \subseteq \beta_{j1}, \alpha_2 \subseteq \beta_{j2}, \ldots, \alpha_n \subseteq \beta_{jn}$.

- $\langle a(ab)c \subseteq a(abc)da(ac)\rangle$.

METHOD, prefix, postfix

- $\alpha = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ and $\beta = \langle \beta_1, \beta_2, \ldots, \beta_m \rangle$.
- α is prefix of β 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 alphabtically after those in α_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 $\beta \subseteq \alpha$, $\gamma \subseteq \alpha$, γ is β-project of α if β is prefix of γ and there is no longer subsequence in α so that β is its prefix.
- c-project of $a(ab)(cd)cd(ab)$ is $(\_d)cd(ab)$

METHOD, y-project.

- Two types of last items, y and $(\_y)$.
- With y: $xyz\ldots \longrightarrow z\ldots$
- With y: $(xyz)\ldots \longrightarrow (\_z)\ldots$
- With $(\_y)$: $(vyz)\ldots \longrightarrow (\_z)\ldots$
- With $(\_y)$: $(\_xyz)\ldots \longrightarrow (\_z)\ldots$
- 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: $xyz \longrightarrow x_i, y, z_i$
- With y: $(xyz) \longrightarrow x_i, y, z_i, (\_z_i)$
- With y: $(\_xyz) \longrightarrow (\_x_i), (\_y), (\_z_i)$.
- with $(\_y)$: $xyz \longrightarrow x_i, y, z_i$.
- with $(\_y)$: $(xyz) \longrightarrow x_i, y, z_i$.
- with $(\_y)$: $(\_z) \longrightarrow (\_z_i)$.
- 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.
- $x, y \longrightarrow xy$.
- $x, (\_y) \longrightarrow (xy)$.
- $(\_x), y \longrightarrow (\_x)y$.
- $(\_x), (\_y) \longrightarrow (\_xy)$.

EXAMPLE

- Finding sequential patterns startting with {ab}, and with support 2, from the database below.
- <a(abc)(ac)d(cf)>
  <(ad)c(bc)(ae)>
  <(ef)(ab)(df)cb>
  <eg(af)cbc>

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

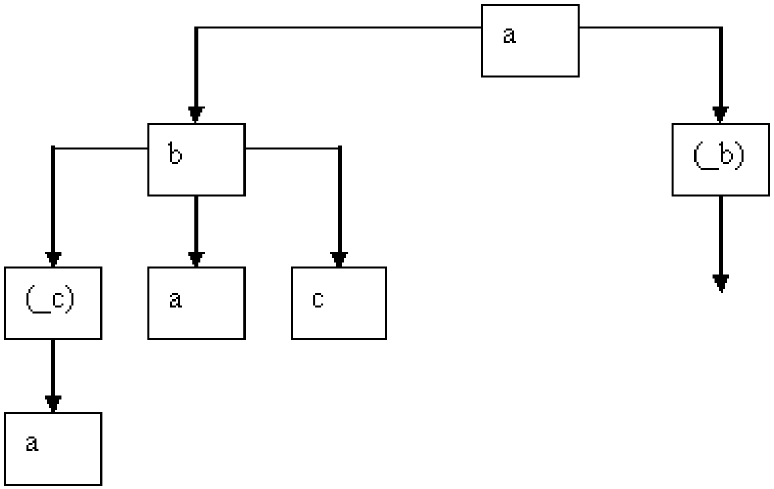
EXAMPLE,cont

- (_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 startting 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:
        nextLewel=prefixspan(x-project,Items,s);
    IList=join(x,Items+nextLewel);
Return IList;
```

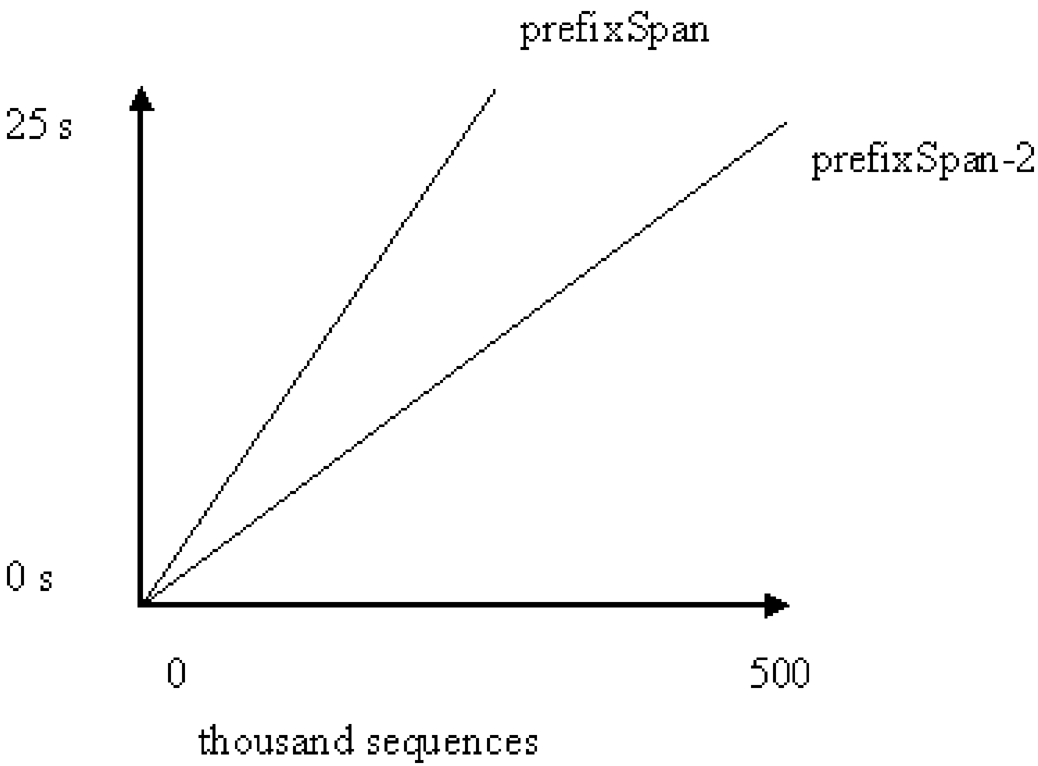
SOME RESULTS, small support.

a b z f g h k l m p q r u v x
a b c d e f g h i j k l m n o p q r s t u v x y z
f g h i j k l m n o p q r s t u v x y z
k l m n o p q r s t u v x y z
p q r s t u v x y z
u v x y z

Support	APriori	PrefixSpan
6	0.001 s	0.003 s
5	0.009 s	0.146 s
4	0.748 s	0.160 s
3	362 s	3.435 s
2	> 5000 s	131 s

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.

- . Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal and M-C. Hsu. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In Proceedings of the 17th international Conference on Data Engineering (April 02 - 06, 2001). ICDE '01. IEEE Computer Society, Washington, DC. http://www-sal.cs.uiuc.edu/ hanj/pdf/span01.pdf