Mining Closed Sequential Patterns in Large Datasets

Presenter: Ildar Nurgaliev
Lab: Dainfos

Main idea

Instead of mining the complete set of frequent subsequences we mine frequent *closed subsequences*

Benefits

- can mine really long sequences
- produce significantly less number of discovered frequent sequences

Preliminary Concepts
Sequence

- items: $I = \{i_1, i_2, ..., i_m\}$
- itemset (t_i): $t_i \subseteq I$
- sequence (ordered list): $s = \langle t_1, t_2, ..., t_m \rangle$
- size |s|: number of itemsets in s
- length l(s): $l(s) = \sum_{i=1}^{n} |t_i|$

Preliminary Concepts

α sub-sequence of β OR β super-sequence of α (contains)

- $\alpha = \langle \alpha_1, \alpha_2, ..., \alpha_m \rangle$
- $\beta = \langle \beta_1, \beta_2, ..., \beta_m \rangle$
- $\alpha \sqsubseteq \beta$ (if $\alpha \neq \beta$, written as $\alpha \sqsubset \beta$)
- iff $\exists i_1, i_2, ..., i_m$, such that
  $1 \leq i_1 < i_2 < ... < i_m \leq n$ and
  $\alpha_1 \subseteq \beta_i, \alpha_2 \subseteq \beta_{i_2}, ..., \alpha_m \subseteq \beta_{i_m}$
- β absorbs α: if β contains α and their *support* are the same

Preliminary Concepts

Support

- $D = \{s_1, s_2, ..., s_n\}$: sequence database
- each s associated with *id* (id of s_i is i)
- $|D|$: number of s in D
- $support(\alpha)$: number of s in D which contain α
  $support(\alpha) = |\{s | s \in D \text{ and } \alpha \sqsubseteq s\}|$
- min_sup: minimum support threshold

Preliminary Concepts

Frequent sequential pattern (FS) and closed FS (CS)

- FS: includes all s of *support(s)* $\leq$ *min_sup*
- $CS = \{\alpha | \alpha \in\ FS\ and\ \nexists \beta \in$ FS
  such that $\alpha \sqsubseteq \beta$ *and* $support(\alpha) = support(\beta)\}$
- *closed sequence mining*: find CS above *min_sup*
- database containment relation $D \sqsubseteq D'$:
  if $\exists$ an injective function $f : D \rightarrow D'$, s.t.
  $\forall s \in D, s \sqsubseteq f(s)$

Preliminary Concepts

Item extension

- Given: $s = \langle t_1, ..., t_m \rangle$ and item α
- $s \diamond \alpha$: concatenation (I-Step or S-Step)
- $s \diamond_i \alpha = \langle t_1, ..., t_m \cup \{\alpha\} \rangle$ if $\forall k \in r_m, k < \alpha$
  Example: $\langle (\alpha e) \rangle$ is I-Step extension of $\langle (\alpha) \rangle$
- $s \diamond_s \alpha = \langle t_1, ..., t_m, \{\alpha\} \rangle$
  Example: $\langle (\alpha)(c) \rangle$ is S-Step extension of $\langle (\alpha) \rangle$

Preliminary Concepts

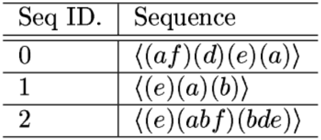
Sequence extension

- Given: $s = \langle t_1, ..., t_m \rangle$ and $p = \langle t'_1, ..., t'_n \rangle$
- $s \diamond p$: concatenation (itemset-extension or sequence-extension)
- $s \diamond_i p = \langle t_1, ..., t_m \cup t'_1, ..., t'_n \rangle$ if $\forall k \in t_m, j \in t'_1, k < j$
- $s \diamond_s p = \langle t_1, ..., t_m, t'_1, ..., t'_n \rangle$
- $s' = p \diamond s$: p - prefix and s - suffix of s'
  Example: $\langle (e)(\alpha) \rangle$ is prefix of $\langle (e)(abf)(bde) \rangle$ and $\langle (bf)(bde) \rangle$ is its suffix

Preliminary Concepts

s-projected database (*physical projection* and *pseudo projection*)

- $D_s = \{p | s' \in D, s' = r \diamond p$ s.t. r is minimum prefix containing s $(s \sqsubseteq r$ and $\nexists r', s \sqsubseteq r' \sqsubset r)\}$
  p can be empty



Seq ID.	Sequence
0	$\langle(af)(d)(e)(a)\rangle$
1	$\langle(e)(a)(b)\rangle$
2	$\langle(e)(abf)(bde)\rangle$

Example

- $D_{\langle(\alpha f)\rangle} = \{\langle(d)(e)(\alpha)\rangle, \langle(bde)\rangle\}$
- $D_{\langle(e)(\alpha)\rangle} = \{\$, \langle(b)\rangle, \langle(\_bf)(bde)\rangle\}$

Lexicographic Sequence Tree

Set Lexicographic Order

- Let $t = \{i_1, i_2, ..., i_k\}, t' = \{j_1, j_2, ..., j_l\}$, where $i_1 \leq ... \leq i_k$ and $j_1 \leq ... \leq j_l$
- $t < t'$ iff *either* of the following is true:
  1. $0 \leq h \leq min\{k, l\}$, we have $i_r = j_r$ for $r < h$, and $i_h < j_h$
  2. $k < l$, and $i_1 = j_1, i_2 = j_2, ..., i_k = j_k$

Example: $(a, f) < (b, f), (a, b) < (a, b, c)$ and $(a, b, c) < (b, c)$

Lexicographic Sequence Tree

Sequence Lexicographic Order

i. if $s' = s \diamond p$, then $s < s'$
ii. if $s = \alpha \diamond_i p$ and $s' = \alpha \diamond_s p'$, no matter what is order relation between p and p' is, $s < s'$
iii. if $s = \alpha \diamond_i p$ and $s' = \alpha \diamond_i p'$, $p < p'$ indicated $s < s'$
iv. $s = \alpha \diamond_s p$ and $s' = \alpha \diamond_s p'$, $p < p'$ indicates $s < s'$

Example: $\langle(a, b)\rangle < \langle(a, b)(a)\rangle$; $\langle(a, b)\rangle < \langle(a)(a)\rangle$

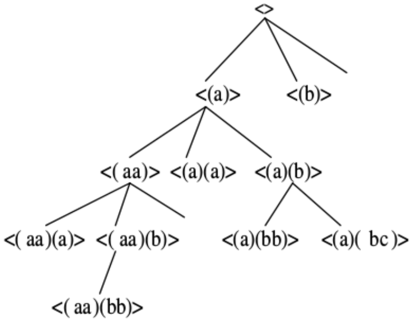
Lexicographic Sequence Tree

Lexicographic Sequence Tree construction

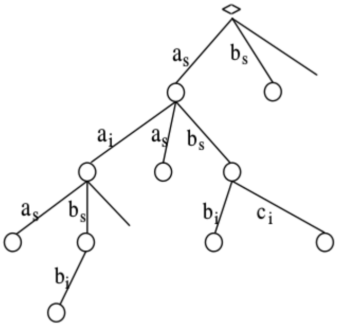
1. each node in the tree corresponds to a sequence, and the root is a *null* sequence;
2. if a parent node corresponds to a sequence s, its child is either an itemset-extension of s, or a sequence-extension of s;
3. the left sibling is less than the right sibling in sequence lexicographic order.

Lexicographic Sequence Tree

Lexicographic Sequence Tree and Prefix Search Tree



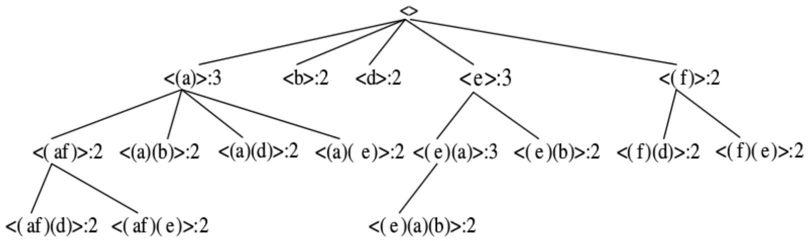
(a) lexicographic sequence tree



(b) prefix search tree

Lexicographic Sequence Tree

Example



Lexicographic Sequence Tree with min_sup = 2

Seq ID.	Sequence
0	$\langle (af)(d)(e)(a) \rangle$
1	$\langle (e)(a)(b) \rangle$
2	$\langle (e)(abf)(bde) \rangle$

Search Space Pruning and Prefix Sequence Lattice

LEMMA 1 (Common Prefix)

LEMMA 1. *Given a subsequence s, and its projected database D_s, if $\exists\alpha$, α is a common prefix for all the sequences with the same extension type (either itemset or sequence - extension) in D_s, then $\forall\beta$, if $s \diamond \beta$ is closed, α must be a prefix of β. That means $\forall\beta \sqsubset \alpha$, we need not search $s \diamond \beta$ and its descendants except the branch of $s \diamond \alpha$.*

Example: $D_s = \{\langle(d)(e)(af)\rangle, \langle(d)(e)(fg)\rangle\}$, all the sequences in D_s share a common prefix $\alpha = \langle(d)(e)\rangle$, so any sequence with prefix s but not $s \diamond \langle(d)(e)\rangle$ must not be closed. So we can "jump" to the branch $s \diamond \alpha$.

Search Space Pruning and Prefix Sequence Lattice

LEMMA 2 (Partial Order)

LEMMA 2. *Given a sequence s, and its projected database D_s, if among all the sequences in D_s, and item α does always occur before an item β (either in the same itemset for all sequences in D_s or in a different itemset, but not both), then $D_{s\diamond\alpha\diamond\beta} = D_{s\diamond\beta}$. Therefore, $\forall\gamma, s \diamond \beta \diamond \gamma$ is not closed. We need not search any sequence in the branch of $s \diamond \beta$.*

Search Space Pruning and Prefix Sequence Lattice

Theorem 1 (Equivalence of Projected Databases)

- $\mathcal{I}(D) = \sum_{i=1}^{n} l(s_i)$: total number items in D

Theorem 1: Given 2 sequences, s, s', $s \sqsubseteq s'$, then

$$D_s = D_{s'} \Leftrightarrow \mathcal{I}(D_s) = \mathcal{I}(D_{s'})$$

Example: Consider D-sample on 15 slide.

- $D_{\langle(af)\rangle} = D_{\langle(f)\rangle} = \{\langle(d)(e)\rangle, \langle(de)\rangle\}$, and
- $\mathcal{I}(D_{\langle(af)\rangle}) = \mathcal{I}(D_{\langle(f)\rangle}) = 4$.

Based on Theorem 1, the following search space pruning can be achieved.

Search Space Pruning and Prefix Sequence Lattice

Proof of Theorem 1

- $D_s = D_{s'} \rightarrow \mathcal{I}(D_s) = \mathcal{I}(D_{s'})$ (obvious);
- Since $s \sqsubseteq s'$, then $D_{s'} \sqsubseteq D_s$ and $\mathcal{I}(D_{s'}) \leq \mathcal{I}(D_s)$;
- *The equality* between $\mathcal{I}(D_{s'})$ and $\mathcal{I}(D_s)$ holds only if $\forall \gamma \in D_{s'}$, $\gamma \in D_s$, and vice versa. **Therefore**, $D_s = D_{s'}$.

Search Space Pruning and Prefix Sequence Lattice

LEMMA 3 (Early Termination by Equivalence)

LEMMA 3. *Given 2 sequences, $s \sqsubseteq s'$ and also $\mathcal{I}(D_s) = \mathcal{I}(D_{s'})$, then $\forall \gamma$, $support(s \diamond \gamma) = support(s' \diamond \gamma)$.*

Example: Consider D-sample on 15 slide.

- $\mathcal{I}(D_{\langle(af)\rangle}) = \mathcal{I}(D_{\langle(f)\rangle})$;
- both $\langle((af)(d))\rangle$ and $\langle(af)(e)\rangle$ are frequent;

We can conclude that the support of $\langle(af)(d)\rangle$ and $\langle(f)(d)\rangle$, $\langle(af)(e)\rangle$ and $\langle(f)(e)\rangle$ are the same without knowing the support of $\langle(f)(e)\rangle$ and $\langle(f)(d)\rangle$.

Search Space Pruning and Prefix Sequence Lattice

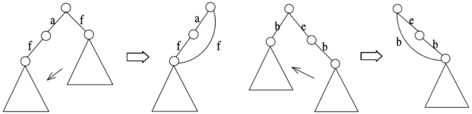
Projected database closed set (LS)

- $LS = \{s | support(s) \geq min\_sup\}$ and $\nexists'$, s.t $s \sqsubseteq s'$ and $\mathcal{I}(D_s) = \mathcal{I}(D_{s'})$;
- $CS \subseteq LS \subseteq FS$: instead of mining CS directly, CloSpan algorithm first produces the complete set of LS
- then non-closed sequence elimination is applied in LS to generate CS based of Lemma 3.

Search Space Pruning and Prefix Sequence Lattice

Corollary 1 (Backward Sub-Pattern)

Corollary 1. *If a sequence s < s' and s $\sqsupset s'$, the condition of $\mathcal{I}(D_s) = \mathcal{I}(D_{s'})$ is sufficient to stop searching any descendant of s' in the prefix searching tree.*



(a) backward sub-pattern (b) backward super-pattern

s' is *backward sub-pattern* of s if $s < s'$ and $s \sqsupset s'$ (s' is discovered after s)

Example: $\mathcal{I}(D_{\langle(f)\rangle}) = \mathcal{I}(D_{\langle(af)\rangle}) \to D_{\langle(f)\rangle} = D_{\langle(af)\rangle}$

Search Space Pruning and Prefix Sequence Lattice

Corollary 2 (Backward Super-Pattern)

Corollary 2. *If a sequence* $s < s'$ *and* $s \sqsupset s'$, *if the condition of* $\mathcal{D}_{\int} = \mathcal{I}(D_{s'})$ *holds, it is sufficient to translating the descendants of* s *to* s' *instead of searching any descendant of* s' *in the prefix search tree.*

Example: the same logic as in the previous example.

CloSpan: Design and Implementation

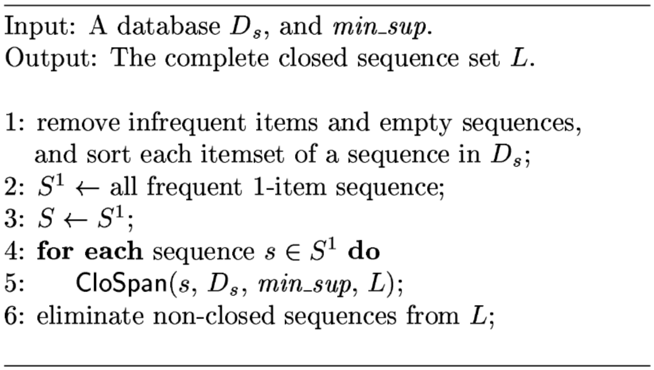
2 main steps

CloSpan divides mining process into 2 stages.

1. Generated the *LS* set, a superset of closed frequent sequences, and stores it in a prefix sequence lattice;
2. it does post-pruning to eliminate non-closed sequences.

CloSpan: Design and Implementation

Algorithm1: ClosedMining(D, min_sup, L)

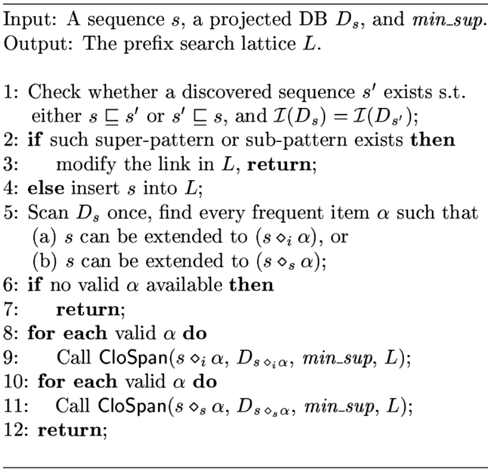


```
Input: A database $D_s$, and $min\_sup$.
Output: The complete closed sequence set $L$.

1: remove infrequent items and empty sequences,
   and sort each itemset of a sequence in $D_s$;
2: $S^1 \leftarrow$ all frequent 1-item sequence;
3: $S \leftarrow S^1$;
4: for each sequence $s \in S^1$ do
5:       CloSpan($s$, $D_s$, $min\_sup$, $L$);
6: eliminate non-closed sequences from $L$;
```

CloSpan: Design and Implementation

Algorithm2: CloSpan(s, D_s, min_sup, L)



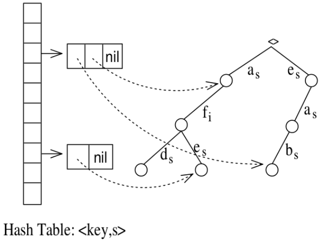
Input: A sequence s, a projected DB D_s, and min_sup.
Output: The prefix search lattice L.

```
1: Check whether a discovered sequence $s'$ exists s.t.
   either $s \sqsubseteq s'$ or $s' \sqsubseteq s$, and $\mathcal{I}(D_s) = \mathcal{I}(D_{s'})$;
2: **if** such super-pattern or sub-pattern exists **then**
3:     modify the link in $L$, **return**;
4: **else** insert $s$ into $L$;
5: Scan $D_s$ once, find every frequent item $\alpha$ such that
   (a) $s$ can be extended to $(s \diamond_i \alpha)$, or
   (b) $s$ can be extended to $(s \diamond_s \alpha)$;
6: **if** no valid $\alpha$ available **then**
7:     **return**;
8: **for each** valid $\alpha$ **do**
9:     Call CloSpan($s \diamond_i \alpha$, $D_{s \diamond_i \alpha}$, $min\_sup$, $L$);
10: **for each** valid $\alpha$ **do**
11:     Call CloSpan($s \diamond_s \alpha$, $D_{s \diamond_s \alpha}$, $min\_sup$, $L$);
12: **return**;
```

CloSpan: Design and Implementation

Algorithm2: CloSpan

- Hash index on the size of projected database in order to speed up check on Therem 1 (1-4 lines of CloSpan);
- if $\mathcal{I}(D_{s'}) = \mathcal{I}(D_s)$ then;
- if $s \sqsubseteq s'$, then we do not add $\langle \mathcal{I}(D_s), s \rangle$;
- if $s' \sqsubseteq s$, then we replace $\langle \mathcal{I}(D_{s'}), s' \rangle$ with $\langle \mathcal{I}(D_s), s \rangle$.

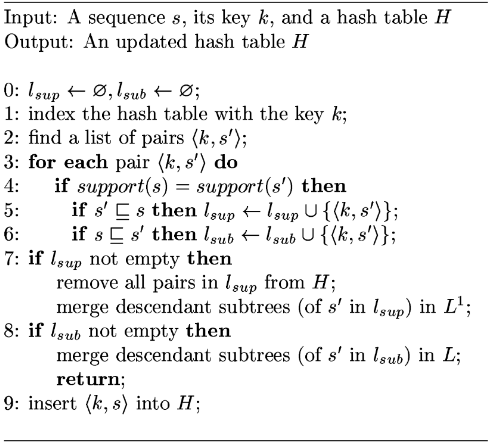


$$\langle \mathcal{I}(D_s), s \rangle$$

CloSpan: Design and Implementation

Algorithm3: checkProjectedDBSize(s, k, H)

Corresponds to line 1-4 in Algorithm 2.



```
Input: A sequence $s$, its key $k$, and a hash table $H$
Output: An updated hash table $H$

0: $l_{sup} \leftarrow \varnothing, l_{sub} \leftarrow \varnothing$;
1: index the hash table with the key $k$;
2: find a list of pairs $\langle k, s' \rangle$;
3: **for each** pair $\langle k, s' \rangle$ **do**
4:     **if** $support(s) = support(s')$ **then**
5:         **if** $s' \sqsubseteq s$ **then** $l_{sup} \leftarrow l_{sup} \cup \{\langle k, s' \rangle\}$;
6:         **if** $s \sqsubseteq s'$ **then** $l_{sub} \leftarrow l_{sub} \cup \{\langle k, s' \rangle\}$;
7: **if** $l_{sup}$ not empty **then**
       remove all pairs in $l_{sup}$ from $H$;
       merge descendant subtrees (of $s'$ in $l_{sup}$) in $L^1$;
8: **if** $l_{sub}$ not empty **then**
       merge descendant subtrees (of $s'$ in $l_{sub}$) in $L$;
       **return**;
9: insert $\langle k, s \rangle$ into $H$;
```

CloSpan: Design and Implementation

Algorithm3: hash function algorithm)

- Database size range from 0 to $\mathcal{I}(D)$, so if the values of $\mathcal{I}(D_s)$ are dense in a small range, performance degrade
- $r_i = random(s_i)$ (assign sequence)
- $\mathcal{L}(D_s) = \mathcal{I}(D_s) + \sum_{j=1}^{m} \sum_{k=i_j+1}^{n} l(s_k)$
- if $s \sqsupseteq s'$, $\mathcal{L}(D_s) = \mathcal{L}(D_{s'}) \leftrightarrow \mathcal{I}(D_s) = \mathcal{I}(D_{s'})$