

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

# Preliminary Concepts

$\alpha$  sub-sequence of  $\beta$  OR  $\beta$  super-sequence of  $\alpha$  (contains)

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

# Preliminary Concepts

## Support

- $D = \{s_1, s_2, \dots, s_n\}$ : 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  $\alpha$   
 $support(\alpha) = |\{s | s \in D \text{ and } \alpha \sqsubseteq s\}|$
- $min\_sup$ : minimum support threshold

# Preliminary Concepts

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

- FS: includes all  $s$  of  $support(s) \leq min\_sup$
- $CS = \{\alpha | \alpha \in FS \text{ and } \nexists \beta \in FS$   
such that  $\alpha \sqsubseteq \beta \text{ and } support(\alpha) = support(\beta)\}$
- *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:  $s = \langle t_1, \dots, t_m \rangle$  and item  $\alpha$
- $s \diamond \alpha$ : concatenation (I-Step or S-Step)
- $s \diamond_i \alpha = \langle t_1, \dots, t_m \cup \{\alpha\} \rangle$  if  $\forall k \in r_m, k < \alpha$   
Example:  $\langle (\alpha e) \rangle$  is I-Step extension of  $\langle (\alpha) \rangle$
- $s \diamond_s \alpha = \langle t_1, \dots, t_m, \{\alpha\} \rangle$   
Example:  $\langle (\alpha)(c) \rangle$  is S-Step extension of  $\langle (\alpha) \rangle$



# Preliminary Concepts

## Sequence extension

- Given:  $s = \langle t_1, \dots, t_m \rangle$  and  $p = \langle t'_1, \dots, t'_n \rangle$
- $s \diamond p$ : concatenation (itemset-extension or sequence-extension)
- $s \diamond_i p = \langle t_1, \dots, t_m \cup t'_1, \dots, t'_n \rangle$  if  $\forall k \in t_m, j \in t'_1, k < j$
- $s \diamond_s p = \langle t_1, \dots, t_m, t'_1, \dots, t'_n \rangle$
- $s' = p \diamond s$ :  $p$  - 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)*

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

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

# Lexicographic Sequence Tree

## *Set Lexicographic Order*

- Let  $t = \{i_1, i_2, \dots, i_k\}$ ,  $t' = \{j_1, j_2, \dots, j_l\}$ , where  $i_1 \leq \dots \leq i_k$  and  $j_1 \leq \dots \leq j_l$
- $t < t'$  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  $i_1 = j_1, i_2 = j_2, \dots, i_k = j_k$

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

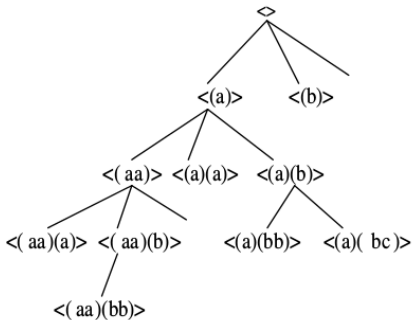
# Lexicographic Sequence Tree

## *Lexicographic Sequence Tree construction*

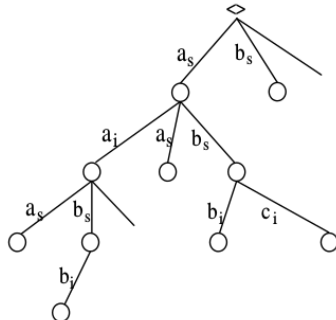
1. each node in the tree corresponds to a sequence, and the root is a *null* sequence;
2. if a parent node corresponds to a sequence  $s$ , its child is either an itemset-extension of  $s$ , or a sequence-extension of  $s$ ;
3. the left sibling is less than the right sibling in sequence lexicographic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

# Lexicographic Sequence Tree

## Lexicographic Sequence Tree and Prefix Search Tree



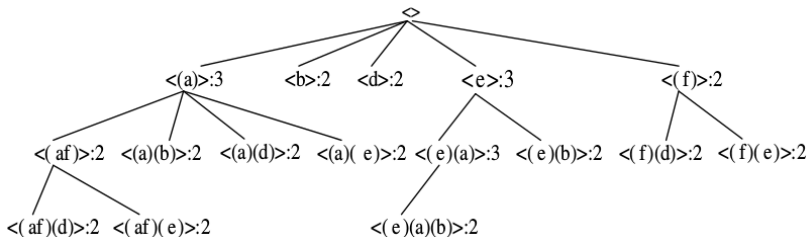
(a) lexicographic sequence tree



(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  $\exists \alpha$ ,  $\alpha$  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,  $\alpha$  must be a prefix of  $\beta$ . 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  $\alpha$  does always occur before an item  $\beta$  (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)

- $\mathcal{I}(D) = \sum_{i=1}^n I(s_i)$ : 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.

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

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  $\forall \gamma$ ,  $\text{support}(s \diamond \gamma) = \text{support}(s' \diamond \gamma)$ .*

**Example:** Consider D-sample on 15 slide.

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

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

# Search Space Pruning and Prefix Sequence Lattice

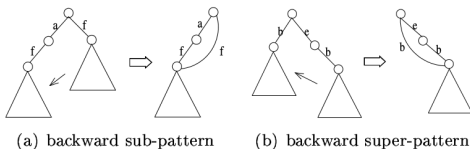
## Projected database closed set (LS)

- $LS = \{s | support(s) \geq min\_sup\}$  and  $\nexists s'$ , s.t.  $s \sqsubset s'$  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.*



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

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

# Search Space Pruning and Prefix Sequence Lattice

## Corollary 2 (Backward Super-Pattern)

Corollary 2. *If a sequence  $s < s'$  and  $s \sqsupseteq s'$ , if the condition of  $\mathcal{D}_f = \mathcal{I}(D_{s'})$  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

Algorithm 1: 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

## Algorithm 2: CloSpan( $s, D_s, \text{min\_sup}, L$ )

---

Input: A sequence  $s$ , a projected DB  $D_s$ , and  $\text{min\_sup}$ .

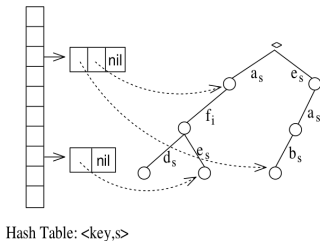
Output: The prefix search lattice  $L$ .

- 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( $s \diamond_i \alpha, D_{s \diamond_i \alpha}, \text{min\_sup}, L$ );
- 10: **for each** valid  $\alpha$  **do**
- 11:     Call CloSpan( $s \diamond_s \alpha, D_{s \diamond_s \alpha}, \text{min\_sup}, L$ );
- 12: **return**;

# CloSpan: Design and Implementation

## Algorithm : CloSpan

- Hash index on the size of projected database in order to speed up check on Theorem 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

## Algorithm 3: 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:  $l_{sup} \leftarrow \emptyset, l_{sub} \leftarrow \emptyset$ ;  
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

## Algorithm 3: 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;
- by Theorem 1 we could use necessary propositions of holding  $D_s = D_{s'}$  in a part of hash key;
- $\mathcal{L}(D_s) = \mathcal{I}(D_s) + \sum_{j=1}^m \sum_{k=i_j+1}^n l(s_k)$ ;
- if  $s \sqsubseteq s'$ ,  $\mathcal{L}(D_s) = \mathcal{L}(D_{s'}) \leftrightarrow \mathcal{I}(D_s) = \mathcal{I}(D_{s'})$ .

# Non-Closed Sequence Elimination

Check out for super sequence

- $support(s)$  as its Hash function
- find all the sequences with the same support of  $s$
- check whether there is a super-sequence containing  $s$ .
- if  $s \sqsubseteq s'$  and
$$support(s) = support(s') \rightarrow \mathcal{T}(D_s) = \mathcal{T}(D_{s'})$$
(corresponding sequences' id sum)
- that's why  $\mathcal{T}(D_s) = \mathcal{T}(D_{s'})$  could be used as a Hash function instead of support (more sparse)

# Conclusion

## *CloSpan*

- Solve closed sequential pattern mining problem;
- *CloSpan* outperforms *PrefixSpan* by more than one order of magnitude;
- capable of mining longer frequent sequences in a large data set with low  $\text{min\_sup}$ ;
- it does not modify the frequent pattern mining algorithm: it only defines the early termination condition of search branch;
- this method can be extended to other existing sequential pattern mining algorithms (SPADE, SPAM).