

# Mining Structures of Documents

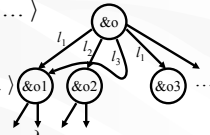
CPS 296.1  
Topics in Database Systems

## Overview

- Wang and Liu. "Discovering Typical Structures of Documents: A Road Map Approach." *SIGIR*, 1998
- Motivation: query/browsing tool, overview/summary, indexes, views, clustering, discovering access patterns, ...
- What is a "structure"??
  - Tree expression
- What is a "typical" structure?
  - Found in more than *MINSUP* documents
- How to discover typical structures?
  - Just like Apriori
  - Start from simple tree expressions to build more complex ones
  - Subexpressions of a typical tree expression must be typical

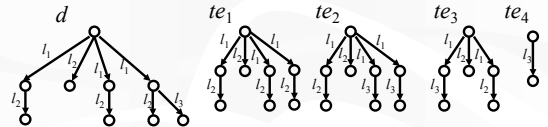
## Data model

- OEM, but with list/bag instead of set
  - List example:  $\text{val}(\&o) = \langle l_1:\&o1, l_2:\&o2, l_3:\&o1, l_1:\&o3, \dots \rangle$ 
    - Order matters
    - Not equivalent to  $\langle l_1:\&o1, l_3:\&o1, l_2:\&o2, l_1:\&o3, \dots \rangle$
  - Bag example:  $\text{val}(\&o) = \{ l_1:\&o1, l_2:\&o2, l_3:\&o1, l_1:\&o3, \dots \}$ 
    - Order matters not
    - Equivalent to  $\{ l_1:\&o1, l_3:\&o1, l_2:\&o2, l_1:\&o3, \dots \}$
- We will ignore the case of bag in our discussion



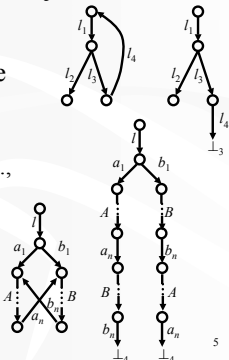
## Tree expression

- A structure that may occur in a document
  - Order of outgoing edges matters
  - Number of duplicates matters
  - Every path from the root in the tree expression must match a path from the root in the document



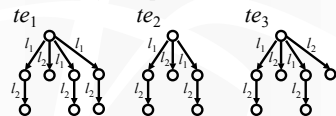
## Dealing with cycles

- Use an alias  $\perp_i$  to "point" back to the  $i$ -th node above the current one
  - A smart naming scheme— better than explicit id's (e.g., easy composition)
- Limitation: not a good representation for cycles with multiple entrances



## Comparing tree expressions

- Some tree expressions are weaker than others
  - If  $te$  is weaker than  $te'$ , then a document that supports  $te'$  must support  $te$  as well
  - "Weaker than" is defined recursively on the structure of the trees being compared



- $te_2$  is weaker than both  $te_1$  and  $te_3$
- $te_1$  and  $te_3$  are incomparable

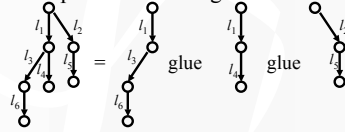
## Discovering frequent tree expressions

- Just like discovering frequent itemsets
  - Transaction = document
  - Itemset = tree expression
  - Transaction contains itemset = document supports tree expression
  - Itemset is in positive border = tree expression is frequent, and not weaker than any other frequent tree expression
- Question: item = ?
  - Or, from another perspective: How to “grow” tree expressions?

7

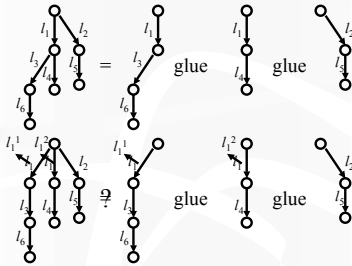
## Item $\approx$ path expression

- A path expression represents a root-to-leaf path in a tree expression
- A tree expression with  $k$  leaves can be constructed (represented) by “gluing” a sequence of path expressions, one for each of the  $k$  leaves
  - Order matters
  - Common prefixes are merged



8

## Dealing with duplicates



- Use superscripts to allow duplicate labels

9

## Algorithm sketch

- Pass 1: Scan all documents to identify  $L_1$ , the set of frequent tree expressions with 1 leaf
  - That is, the set of frequent path expressions
- ...
- Pass  $k$ 
  - Generate  $C_k$ , the set of candidate tree expressions with  $k$  leaves, from  $L_{k-1}$ , the set of frequent tree expressions with  $k-1$  leaves
    - Join and prune
  - Scan all documents to count each tree expression in  $C_k$ , and determine  $L_k$
  - Stop if  $L_k = \emptyset$
- ...

10

## Candidate generation: join

- Recall that a tree expression with  $k$  leaves can be represented by a sequence of  $k$  path expressions
- Given
  - $p_1 p_2 \dots p_{k-2} p_{k-1} \in L_{k-1}$
  - $p_1 p_2 \dots p_{k-2} p_k \in L_{k-1}$
- Generate  $p_1 p_2 \dots p_{k-2} p_{k-1} p_k$  in  $C_k$ 
  - Termed “extending  $p_{k-1}$  by  $p_k$ ” in the paper

11

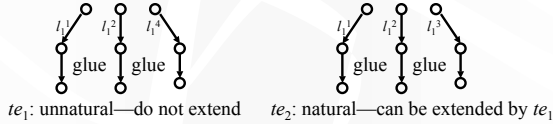
## Candidate pruning

- Join alone does not enforce the Apriori property
- For each  $p_1 p_2 \dots p_{k-2} p_{k-1} p_k \in C_k$ 
  - $p_1 p_2 \dots p_{k-2} p_{k-1}$  and  $p_1 p_2 \dots p_{k-2} p_k$  are frequent by construction
  - But we still should check other subsequence of length  $k-1$ ; if any such subsequence is not frequent, prune  $p_1 p_2 \dots p_{k-2} p_{k-1} p_k$  from  $C_k$
- Surprisingly, this strategy is not in the paper

12

## Other pruning strategies (strategy 1)

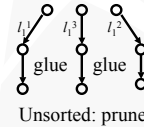
- Given  $p_1 p_2 \dots p_{k-2} p_{k-1}$  and  $p_1 p_2 \dots p_{k-2} p_k$  in  $L_{k-1}$
- Do not extend  $p_{k-1}$  by  $p_k$  if the superscripts in  $p_1 p_2 \dots p_{k-2} p_{k-1}$  do not occur in natural order (sorted and has no gap)
  - However, it is okay to have a gap between  $p_{k-2}$  and  $p_k$ , or between  $p_{k-1}$  and  $p_k$



13

## Other pruning strategies (strategy 2)

- For each  $p_1 p_2 \dots p_{k-1} p_k \in C_k$ , prune it if the superscripts for the some label do not occur in sorted order
  - Because no matter how we use this tree expression (to extend others or to be extended by others) these superscript will remain unsorted



14

## Summary

- First attempt at applying frequent itemset mining techniques to mining document structure
  - Mapping to the frequent itemset mining problem is fairly straightforward
- Patterns considered are restrictive
  - All paths start from the root
  - Cycles are not handled well
- Including subscripts in patterns really complicates things
  - two pruning strategies to deal with the complexity
  - Better idea: extend the notion of join instead?
- Repeated path expression matching is inefficient
  - How about building an index (like FP-tree)?

15

## End-semester logistics

- Course project
  - In-class presentation: Thursday, May 2, 2pm – 5pm
    - Talk: 20 – 25 minutes; Q&A: 5 – 10 minutes
    - Slides/demos encouraged
  - Report due Thursday, May 2, 11:59pm
- Grading
  - Check CourseInfo for possible recording errors
    - Deadline for requesting a correction: May 2, 11:59pm
  - Final grades will be assigned on May 4
- Office hours during reading period
  - Regular office hours + class meeting time, or by appointment

16