Mining Trees

CS 145

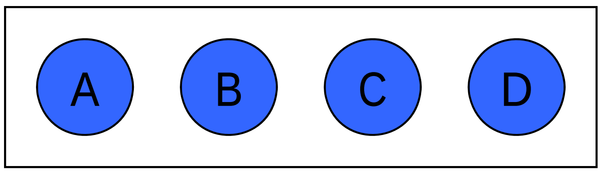
Fall 2015

Mining Complex Patterns

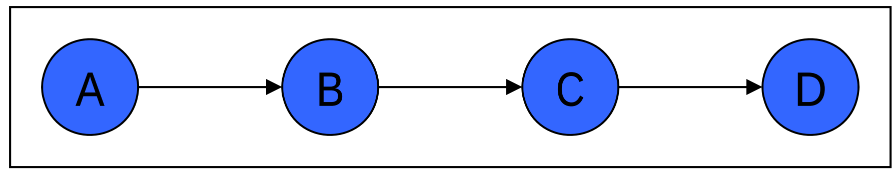
- Common Pattern Mining Tasks:
  - **Itemsets** (transactional, unordered data)
  - **Sequences** (temporal/positional: text, bioseqs)
  - **Tree patterns** (semi-structured/XML data, web mining)
  - **Graph patterns** (protein structure, web data, social network)

Example Pattern Types

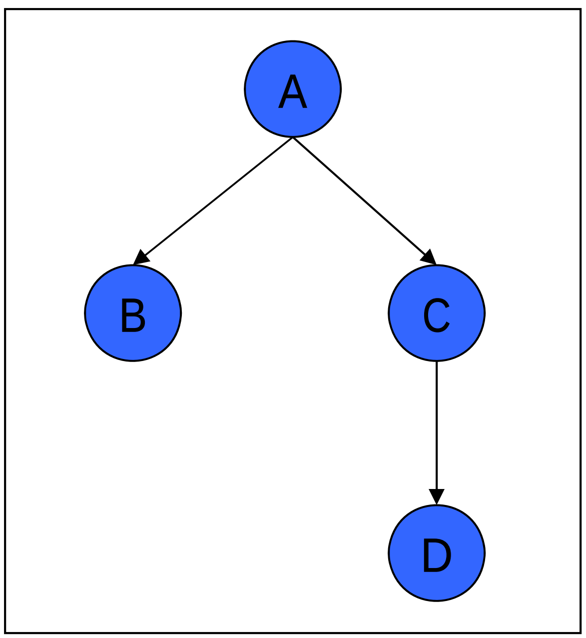
Itemset



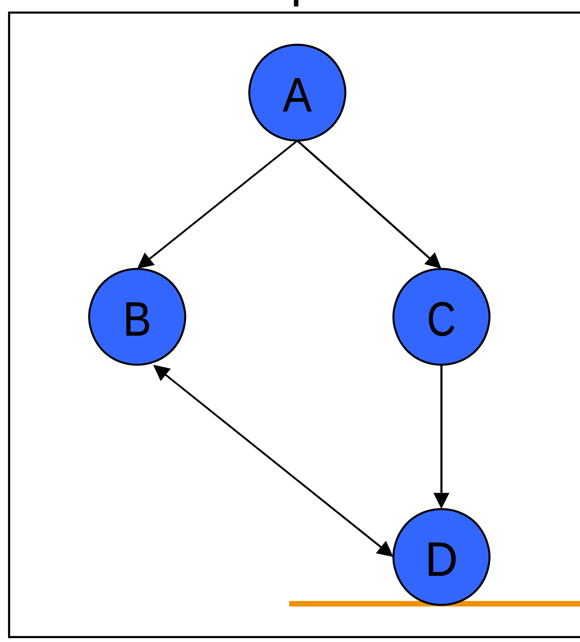
Sequence



Tree



Graph



- Can add attributes
  - To nodes
  - To edges
- **Attributes**
  - Labels
  - Type (directed or undirected)
  - Set-valued

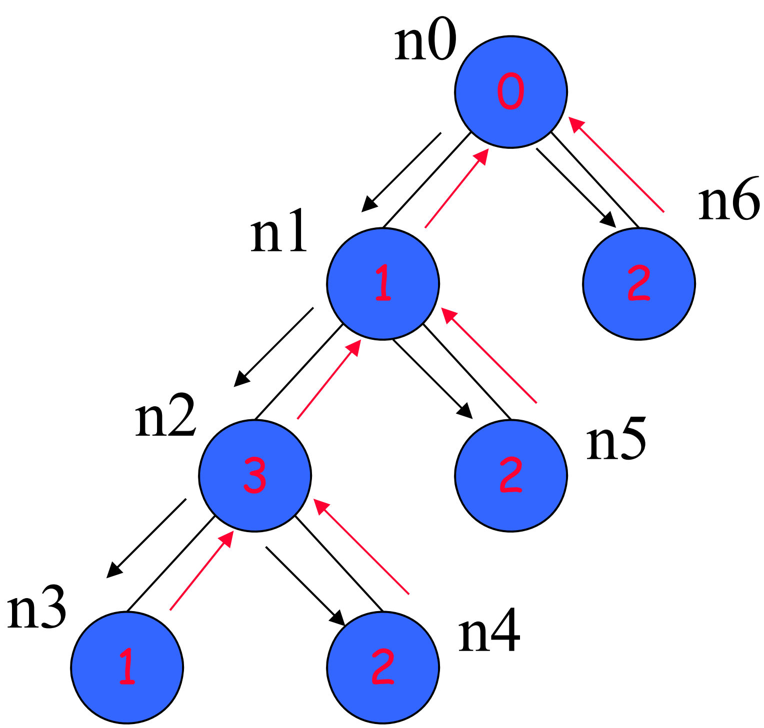
Induced vs Embedded Sub-trees

- Induced Sub-trees: $S = (V_s, E_s)$ is a sub-tree of $T = (V,E)$ if and only if
  - $V_s \subseteq V$
  - $e = (n_x, n_y) \in E_s$ iff $(n_x, n_y) \in E$ (n_x directly connected to n_y)
- Embedded Sub-trees: $S = (V_s, E_s)$ is a sub-tree of $T = (V,E)$ if and only if
  - $V_s \subseteq V$
  - $e = (n_x, n_y) \in E_s$ iff $n_x \leq_l n_y$ in T (n_x connected to n_y)
- An induced sub-tree is a special case of embedded sub-tree.
- We say S *occurs* in T and T *contains* S if S is an embedded sub-tree of T
- If S has *k* nodes, we call it a *k*-sub-tree

Mining Frequent Trees

- Support: the *support* of a subtree in a database of trees, is the number of trees containing the subtree.
- A subtree is frequent if its support is at least the minimum support.
- TreeMiner: Given a database of trees (a forest) and a minimum support, find all frequent subtrees.

String Representation of Trees



0 1 3 1 -1 2 -1 -1 2 -1 -1 2 -1

With N nodes, M branches, F max fanout

Adjacency Matrix requires: N(F+1) space

Adjacency List requires: 4N-2 space

Tree requires (node, child, sibling): 3N space

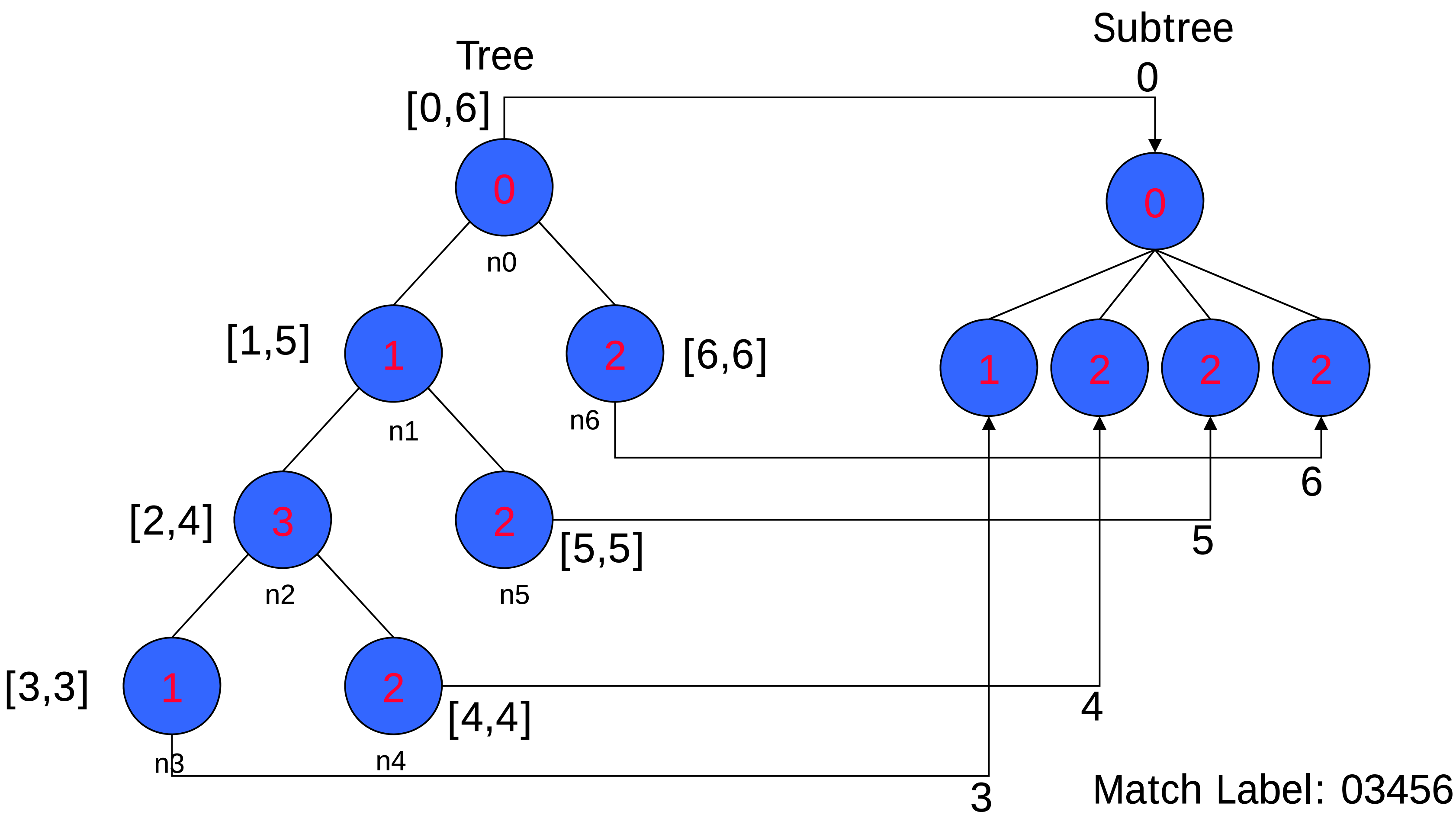
String representation requires: 2N-1 space

Tree: String Representation

- Like an itemset
- -1 as the backtrack item
- Assuming only labels on nodes
- For trees labels on edges can be treated as labels on nodes:

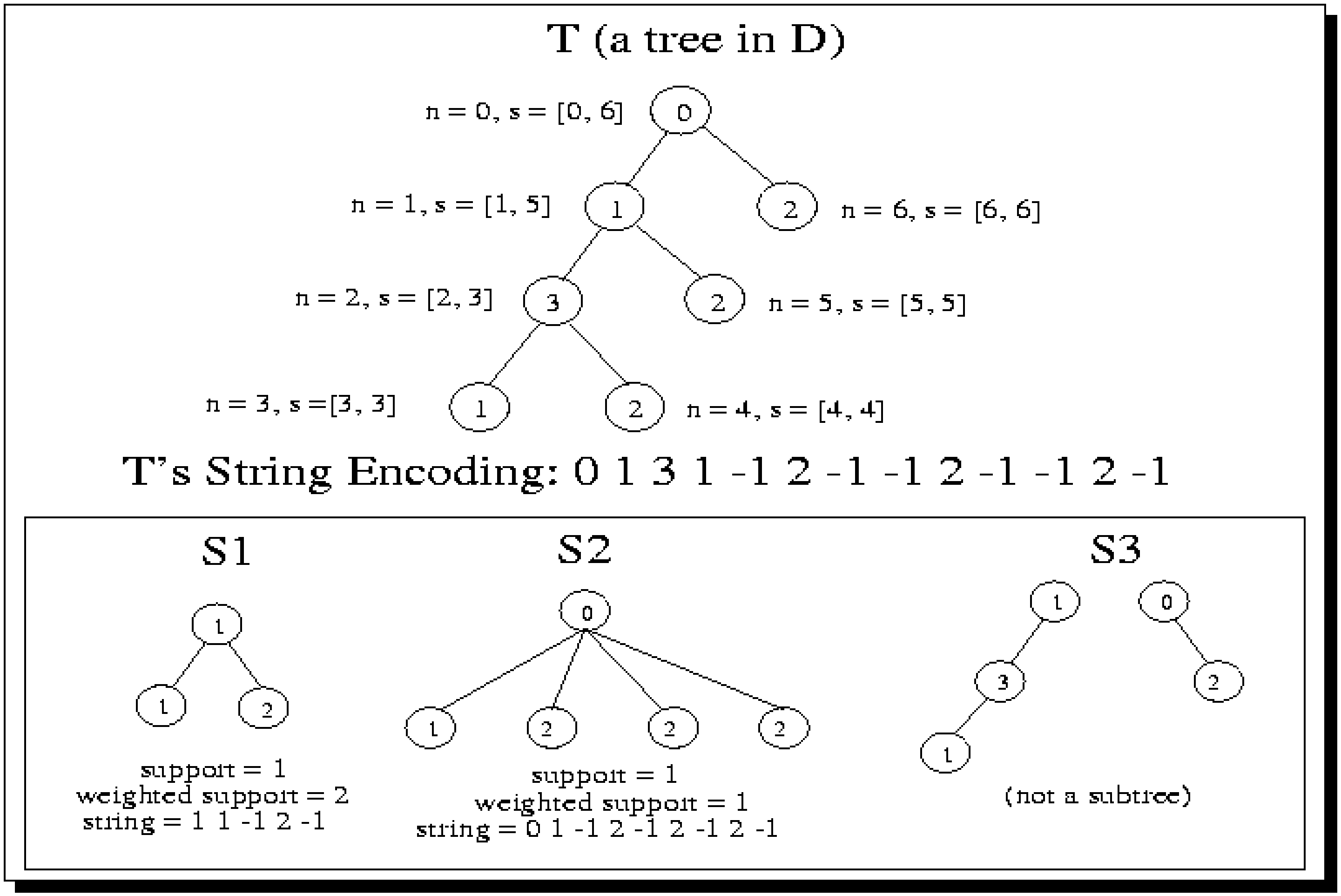
  edge-label+node-label = new label!

Match labels



vector < id, match label, scope >

An example


T (a tree in D)
n = 0, s = [0, 6]
0
n = 1, s = [1, 5]
1
2
n = 6, s = [6, 6]
n = 2, s = [2, 3]
3
2
n = 5, s = [5, 5]
n = 3, s =[3, 3]
1
2
n = 4, s = [4, 4]
T's String Encoding: 0 1 3 1 -1 2 -1 -1 2 -1 -1 2 -1
S1
S2
S3
support = 1
weighted support = 2
string = 1 1 -1 2 -1
support = 1
weighted support = 1
string = 0 1 -1 2 -1 2 -1 2 -1
(not a subtree)

Generic Mining Algorithms

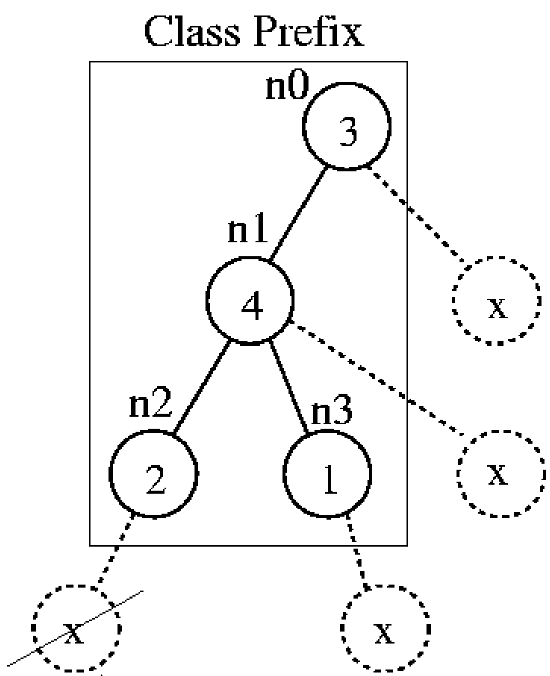
- Horizontal pattern matching based
- Vertical intersection based
- BFS or DFS

Candidate Generation & Support Counting

- Candidate Generation
  - Extend by a node or an edge
  - Avoid duplicates as far as possible

Trees: Systematic Candidate Generation

Two subtrees are in the same class iff they share a common prefix string *P* up to the (k-1)th node



Not valid position: Prefix 3 4 2 x

Equivalence Class

Prefix String: 3 4 2 −1 1

Element List: (label, attached to position)

(x, 0) // attached to n0: 3 4 2 −1 1 −1 −1 x −1

(x, 1) // attached to n1: 3 4 2 −1 1 −1 x −1 −1

(x, 3) // attached to n3: 3 4 2 −1 1 x −1 −1 −1

A valid element *x* attached to only the nodes lying on the path from root to **rightmost leaf** in prefix *P*

Candidate generation

- Given an equivalence class of k-subtrees, how do we generate candidate (k+1)-subtrees?

- Main idea: consider each ordered pair of elements in the class for extension, including self extension
  - Sort elements by node label and position

Class extension

Let P be a prefix class with encoding P, and let (x, i) and (y, j) denote any two elements in the class. Let Px denote the class representing extensions of element (x, i). Define a join operator $\otimes$ on the two elements, denoted $(x, i)\otimes(y, j)$, as follows:

case I – $(i = j)$:

1. If $P \mathrel{!=} \emptyset$, add (y, j) and $(y, \boldsymbol{ni})$ to class $[Px]$, where ni is the depth-first number for node (x, i) in tree Px.
2. If $P = \emptyset$, add $(y, j + 1)$ to $[Px]$.

case II – $(i > j)$: add (y, j) to class $[Px]$.

case III – $(i < j)$: no new candidate is possible in this case.