Data Mining: Foundation, Techniques and Applications

Lesson 12: Mining and Searching Trees

Li Cuiping(李翠平)
School of Information
Renmin University of China

Anthony Tung(鄧锦浩)
School of Computing
National University of Singapore

Outline

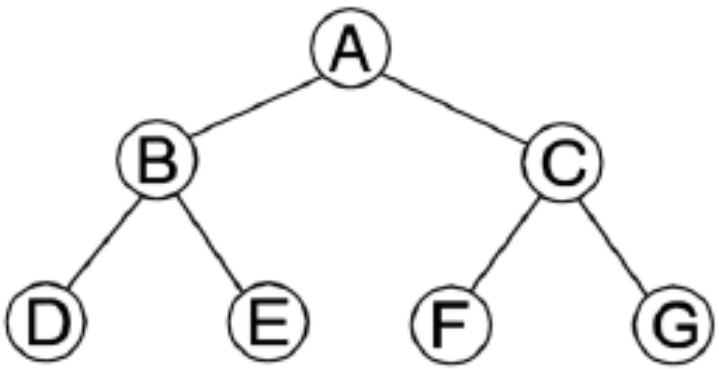
- Importance of Trees
- Distance between Trees
- Kernel methods for Trees
- Fast Distance Computations
- Mining Frequent Subtrees

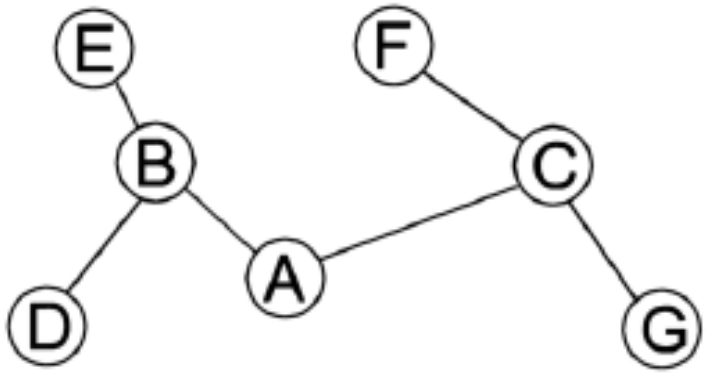
Importance of Trees

- Between sequences and graphs
- Equivalent to acyclic graph
- Represents hierarchal structures
- Examples
  - XML documents
  - Programs
  - RNA structure

Types of Trees

- Is there a root?
- Are the nodes labeled?
- Are the children of a node ordered?





Framework

- Many data mining problems requires the notion of a distance/similarity measure
  - Clustering
  - Classification (Nearest Neighbor, SVM)
- How to compute distance between two trees?
- How to quickly approximate the distance?
- Which structures occurs frequently in a database of trees?

Outline

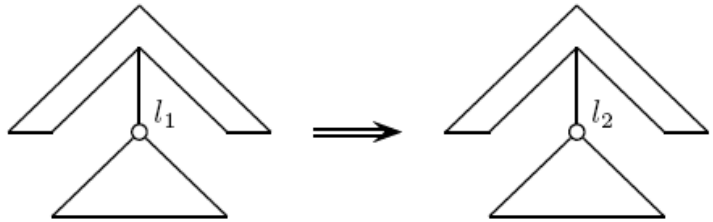
- Importance of Trees
- Distance between Trees
- Kernel methods for Trees
- Fast Distance Computations
- Mining Frequent Subtrees

Distance Measure

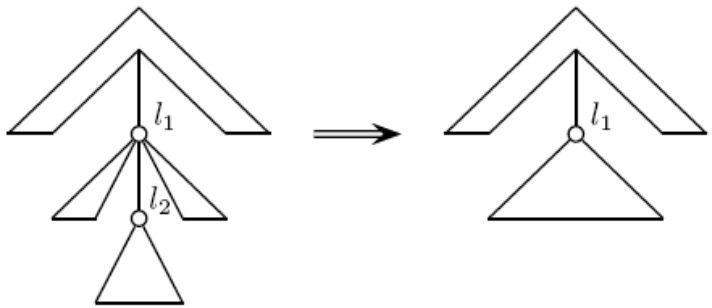
- Many ways to define distance
- Convert to standard types and adopt the distance metric there
- How many operations to transform one tree to another? (Edit distance)
- Inverse of similarity
  dist(S, T) = maxSim – sim(S,T)
- Relationship between different definitions?

Operations on Trees

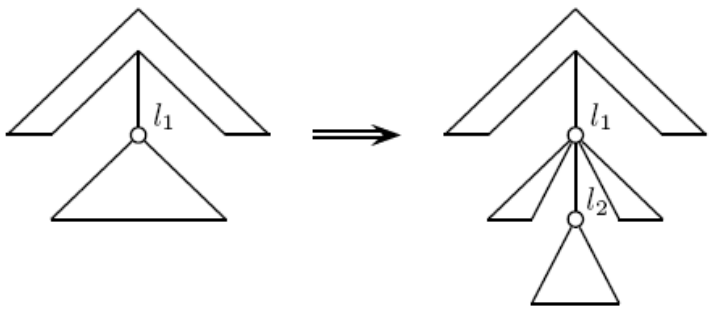
- Relabel



- Delete



- Insert

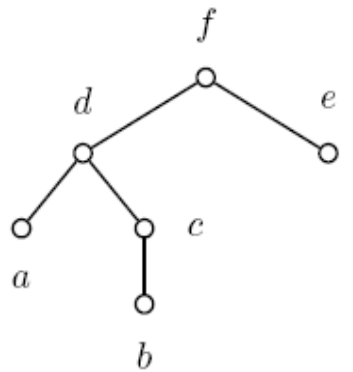


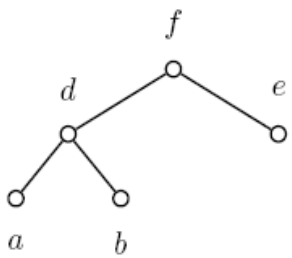
Remarks on Edit Distance

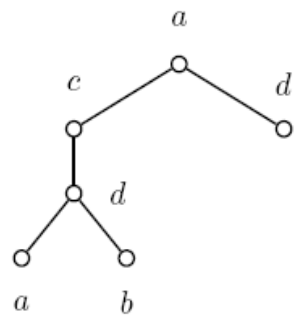
- Ordered trees are tractable
- Approach based on dynamic programming
- NP-hard for unordered trees
- Approach is to impose restrictions so that DP can be used

Edit Script

- Edit script(S, T): sequence of operations to transform S to T
- Example

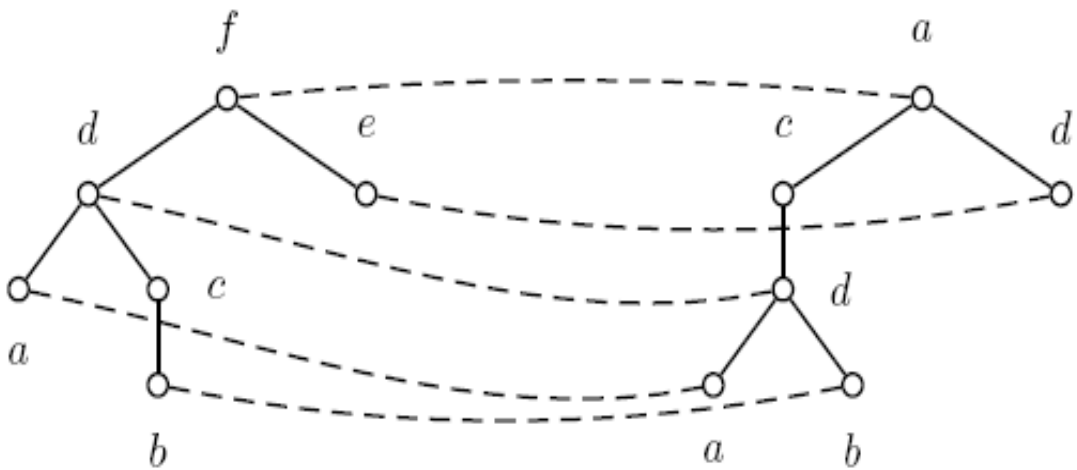
1. S= 

2. Delete 

3. Insert c
   Relabel f → a
   Relabel e → d


Edit Distance Mapping

- Edit distance mapping(S, T): alternative representation of edit operations
  - relabel: v → w
  - delete: v → $
  - insert: $ → w
- Mapping corresponding to the script

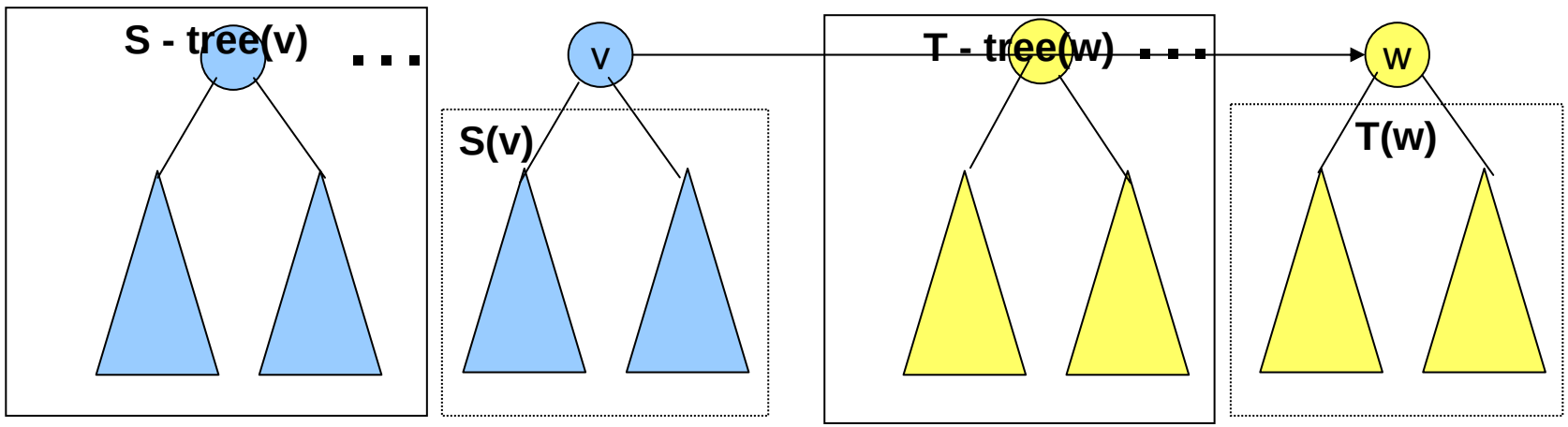


Edit Distance for Ordered Trees

- Generalize the problem to forests.
- $C(\varphi, \varphi) = 0$
- $C(S, \varphi) = C(S - v, \varphi) + cost(v \rightarrow \$)$
- $C(\varphi, T) = C(\varphi, T - w) + cost(\$ \rightarrow w)$
- $C(S, T)$ = minimum of
  1. $C(S - v, T) + cost(v \rightarrow \$)$ [deleting v]
  2. $C(S, T - w) + cost(\$ \rightarrow w)$ [inserting w]
  3. $C(S - tree(v), T - tree(w)) +$
     $C(S(v) - v, T(w)) + cost(v \rightarrow w)$[relabel $v \rightarrow w$]

Illustration of Case 3

- C(S – tree(v), T – tree(w)) + C(S(v), T(w)) + cost(v → w) [relabel v → w]



Algorithm Complexity

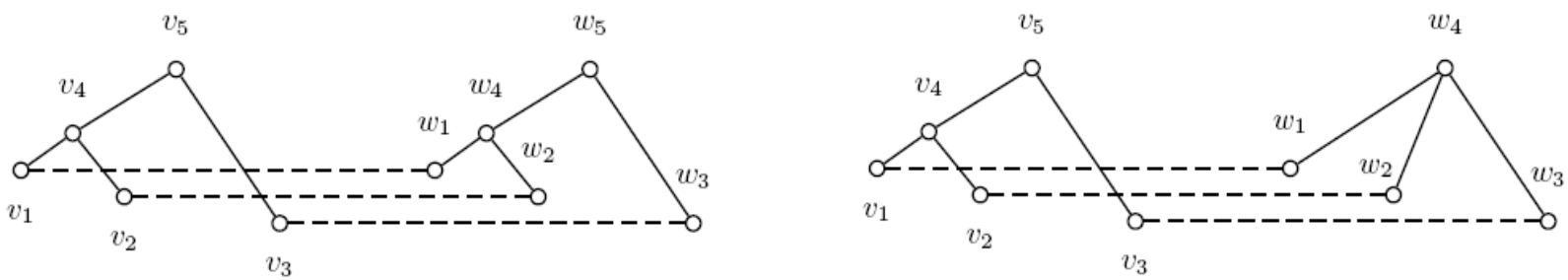
- Number of subproblems bounded by $O(|S|^2|T|^2)$
- Zhang and Shasha, 1989 showed that the number of relevant subproblems is $O(|S||T|\min(S_D, S_L)\ \min(T_D, T_L))$ and space is $O(|S||T|)$
- Further improvements, required decomposition of a rooted tree into disjoint paths

Decomposition into Paths

- Concept of heavy and light nodes/edges (Harel and Tarjan, 1984)
- Root is light, child with max size is heavy
- Removal of light edges partitions T into disjoint heavy paths
- Important property: light depth(v) ≤ log|T| + O(1)
- Complexity can be reduced to $O(|S|^2|T|\log|T|)$

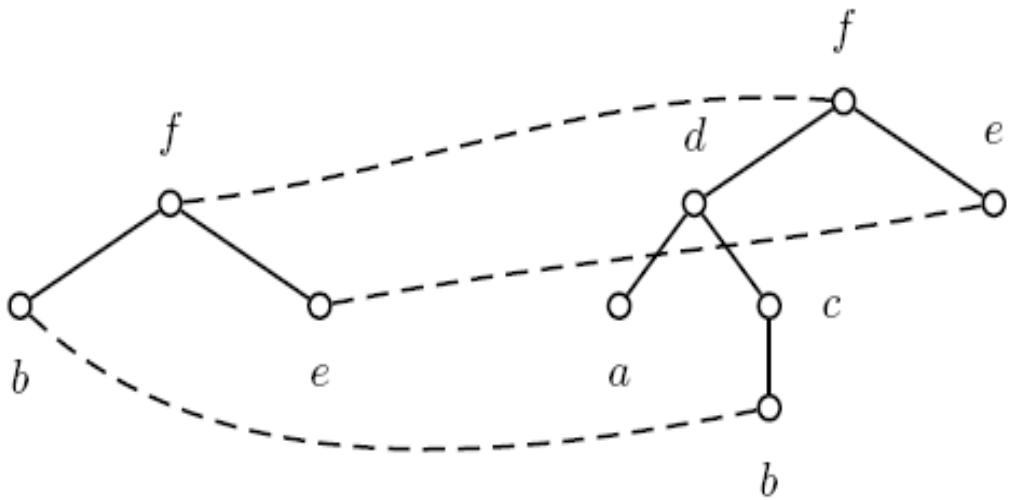
Unordered Edit Distance

- NP-hard
- Special cases (in P)
  - T is a sequence
  - Number of leaves in T is logarithmic
- Impose additional constraints
  - Disjoint subtrees map to disjoint subtrees



Tree Inclusion

- Is there a sequence of deletion operations on S which can transform it to T?
- Special case of edit distance which only allows deletions



Complexity of Tree Inclusion

- Ordered trees
  - Concept of embeddings (restriction of mappings)
  - O(|S||T|) using the algorithm of Kilpelainen and Mannila
- Unordered trees
  - NP-complete (what did you expect ?)
  - Special cases

Related Problems on Trees

- Tree Alignment (covered in the survey paper)
- Robinson-Fould's Distance for leaf labeled trees, where edge = bipartition of leaves
- Tree Pattern Matching
- Maximum Agreement Subtree
- Largest Common Subtree
- Smallest Common Supertree
- Many are generalizations of problems on strings

Summary of Tree Distance

- Edit distance
  - Concept of edit mapping
  - Dynamic programming for ordered trees
  - Constrained edit distance for unordered trees
- Tree inclusion
  - Special case of edit distance
  - Specialized algorithms are more efficient
  - Useful for determining embedded trees

Outline

- Importance of Trees
- Distance between Trees
- Kernel methods for Trees
- Fast Distance Computations
- Mining Frequent Subtrees

Kernels for Trees

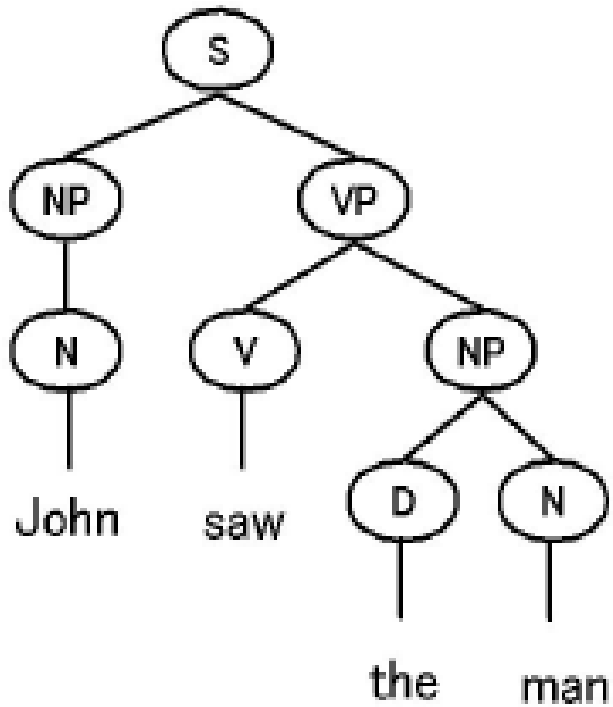
- SVM requires kernel methods
- Effective in high dimensional spaces
- A measure of the similarity between two trees

Vector Representation of a Tree

- $V_T = [\#subtree_1(T), \#subtree_2(T), \ldots]$
- $V_S \bullet V_T = \sum \#subtree_i(S) \times \#subtree_i(T)$
  $= \sum \sum C(n_S, n_T)$
- $C(n_S, n_T) = \sum \#subtree_i$ rooted at $n_S \times$
  $\#subtree_i$ rooted at n_T

Recurrence for Parse Trees

- No node shares a label with its siblings
- There is a one-to-one correspondence between two groups of children
- $C(n_S, n_T) = 0$ if n_S != n_T
- $C(n_S, n_T) = 1$ if n_S and n_T are leaves
- $C(n_S, n_T) = \prod (1 + C(n_S^i, n_T^i))$



Kernel for Labeled Ordered Trees

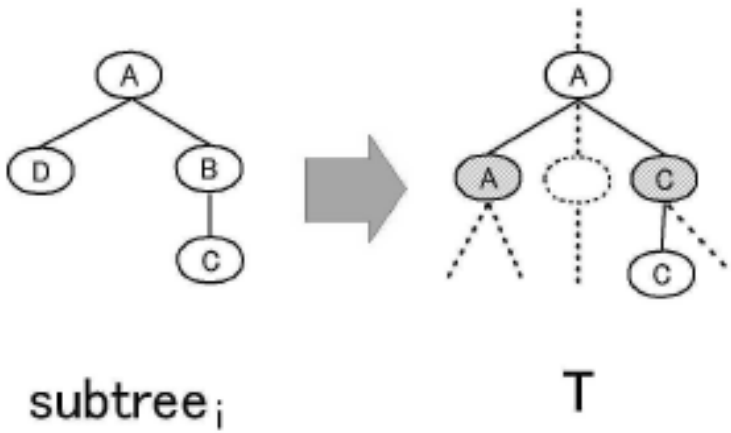
- No direct correspondence between children of a node
- Luckily nodes are ordered
- Use Dynamic Programming to consider all possible matchings where order of children is preserved

Recurrence for Ordered Trees

- Let $D(n_S, n_T, i, j)$ be $C(n_S, n_T)$ where we consider up to the ith child of n_S and the jth child of n_T
- $C(n_S, n_T) = D(n_S, n_T, \#child(n_S), \#child(n_T))$
- Since all matchings preserve the left-to-right ordering of the children, hence
- $$D(n_S, n_T, i, j) = D(n_S, n_T, i-1, j) + D(n_S, n_T, i, j-1) - D(n_S, n_T, i-1, j-1) + D(\mathbf{n}_S, n_T, i-1, j-1) \times C(n_S^i, n_T^j)$$

Extension: Label Mutations

- Define a mutation score function f(v|w), suppose f(A|A) = 1, f(A|D) = 0.5, f(C|B) = 0.8 and f(C|C) = 1
- Then the "count" of subtree$_i$ wrt to T is f(A|A) f(A|D) f(C|B) f(C|C) = 0.4

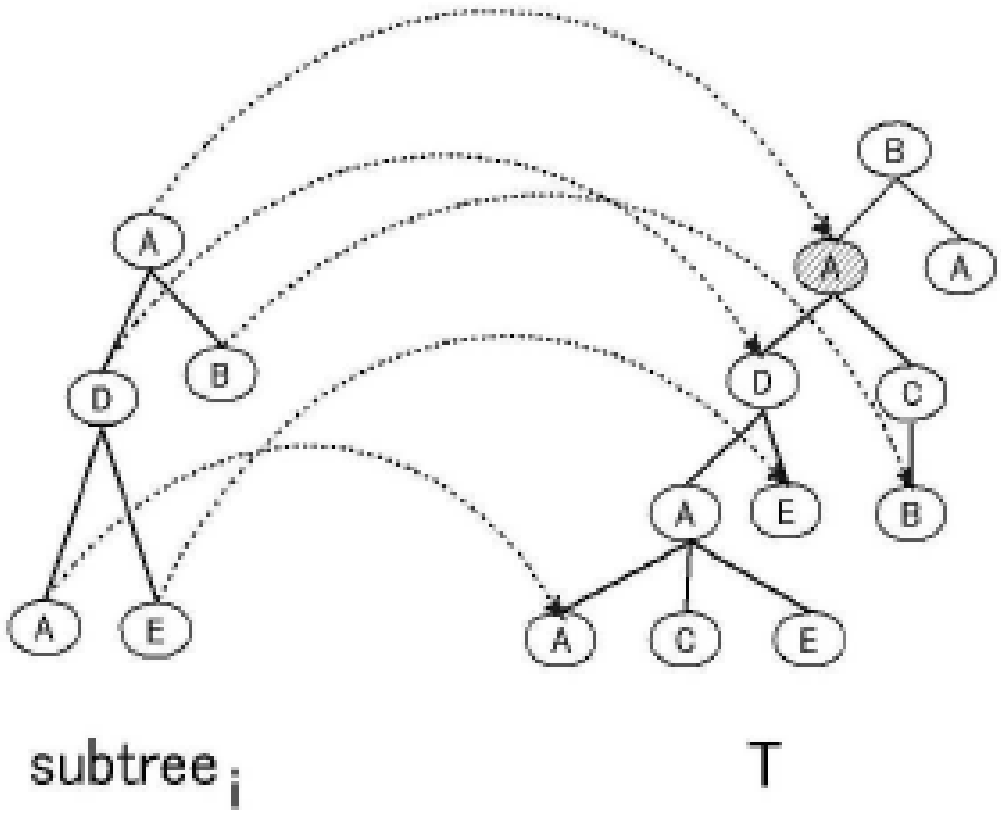


Extension: Label Mutations

- Only need to modify $C(n_S, n_T)$
- $C(n_S, n_T) = M(L(n_S), L(n_T) \times D(n_S, n_T, \#child(n_S), \#child(n_T))$
- $M(L(n_S), L(n_T)) = \Sigma f(L(n_S)|a) f(L(n_T)|a)$
- Sum over all possible ways in which the labels can be changed

Extension: Embedding

- Allow a subtree$_i$ to be embedded in T



Extension: Embedding

- For subtrees
  - all subtrees rooted at v can be constructed by combining subtrees rooted at each of its children
- For embedded subtrees
  - all subtrees rooted at v can be constructed by combining subtrees rooted at each of its descendants
- Allow the possibility of skipping nodes in the recurrence
- Complexity remains at O(|S||T|)

Summary of Kernel Methods

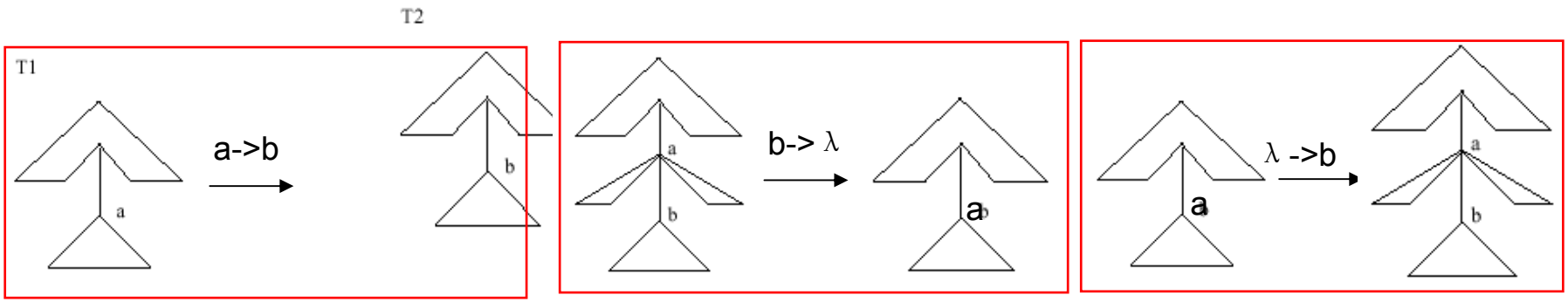
- Kernel methods avoid dealing with high dimensional vectors
- Dot product ~ similarity measure
- Dynamic programming to the rescue
- Extending the recurrence without increasing the complexity

Outline

- Importance of Trees
- Distance between Trees
- Kernel methods for Trees
- Fast Distance Computations
- Mining Frequent Subtrees

Similarity Measurement

- Edit Distance *EDist(T_1, T_2)*
  Edit Operation *e;* cost γ(e),



$s_i(e_{i1},e_{i2},\ldots,e_{ik})$: $T_1 \rightarrow T_2$; $cost(s_i) = \sum_j \gamma(e_{ij})$

$EDist(T_1,T_2) = \min_i(cost(s_i))$ unit cost: $EDist(T_1,T_2) = \min(k)$

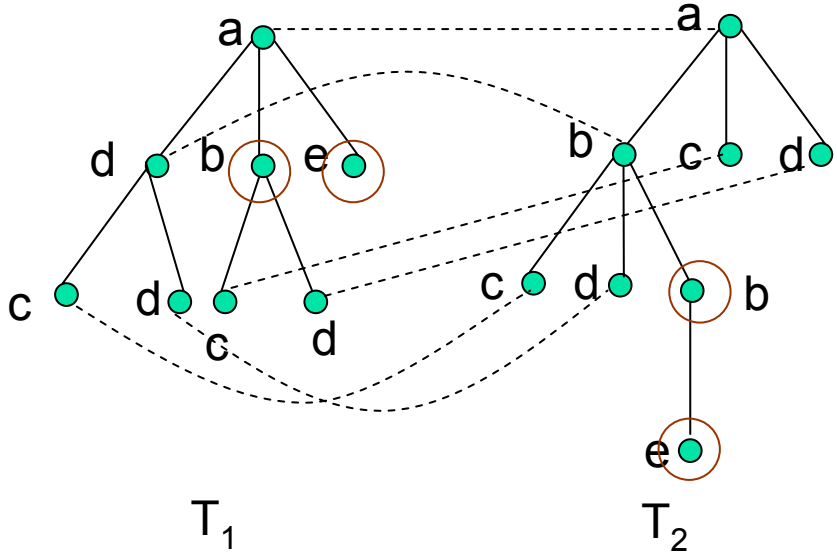
Computational Complexity:

$$O(|T_1| \times |T_2| \times \min(depth(T_1), leaves(T_1)) \times \min(depth(T_2), leaves(T_2)))$$

Edit Operation Mapping

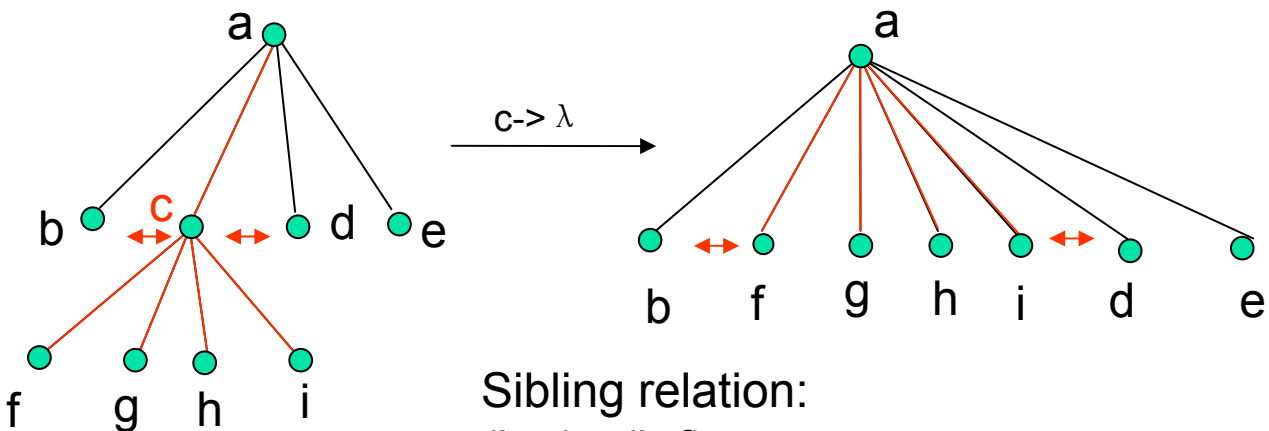
- Edit operations mapping
  - One-to-one
  - Preserve sibling order
  - Preserve ancestor order

$M(T_1,T_2)$



Observation

- Edit operations do not change many sibling relationship

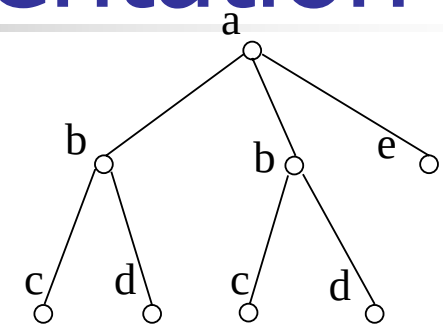


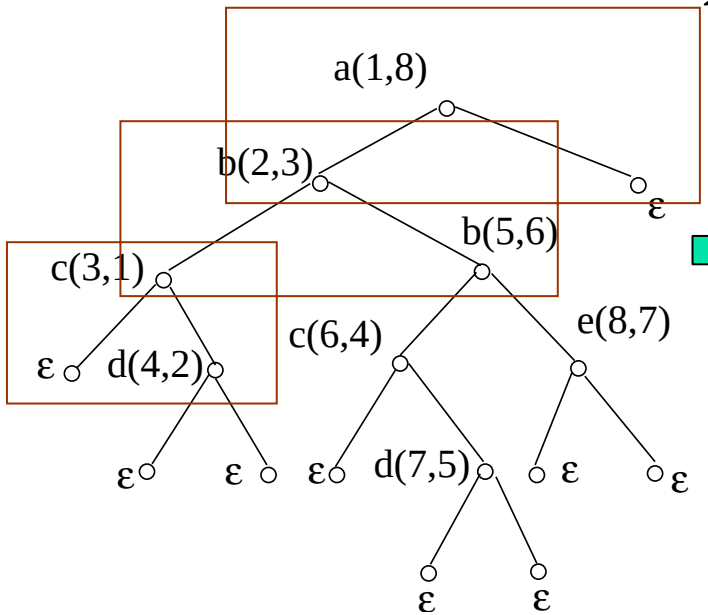
Sibling relation:
(b,c)->(b,f)
(c,d)->(i,d)

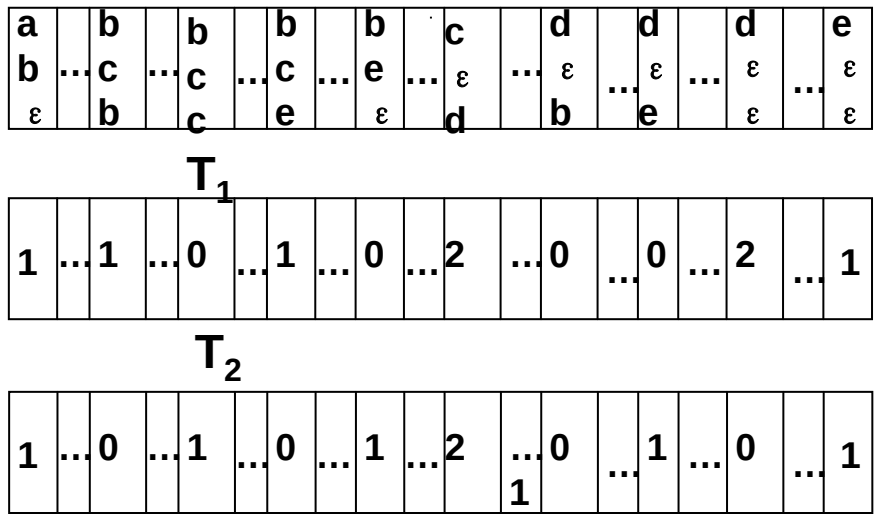
Node: Varying number of children v.s. at most 2 siblings

Binary Tree Representation

- Binary Tree Representation
  - Left-child, right sibling
- Normalized Binary Tree



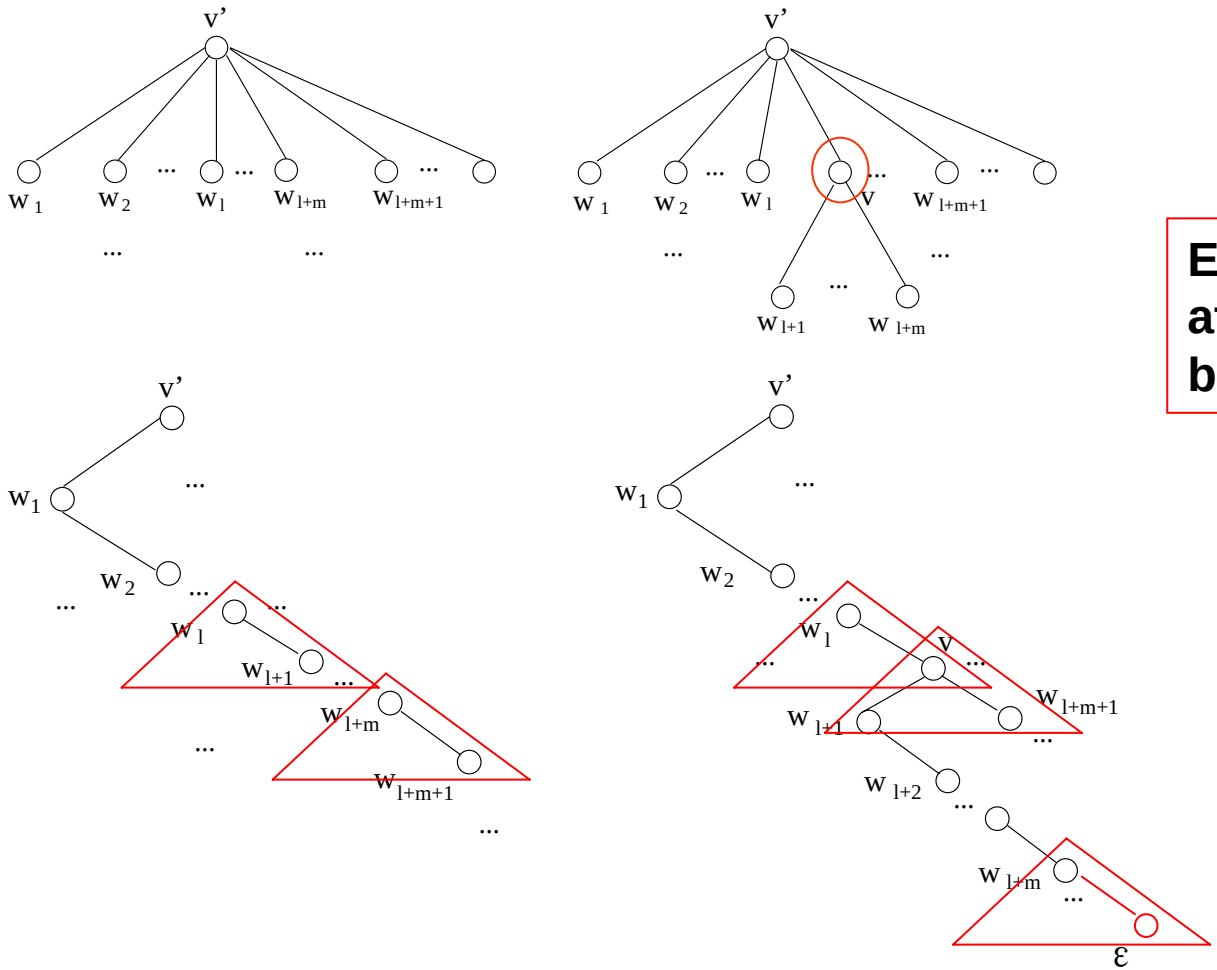




$$BBDist(T_1, T_2) = \sum_{i=1}^{|\Gamma|} | b_{1i} - b_{2i} | = 8$$

Triangular Inequality

One Edit Operation Effect



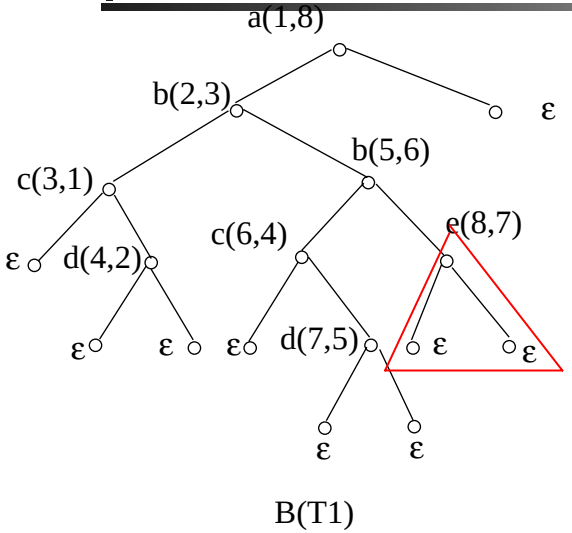
Each node appears in at most two binary branches

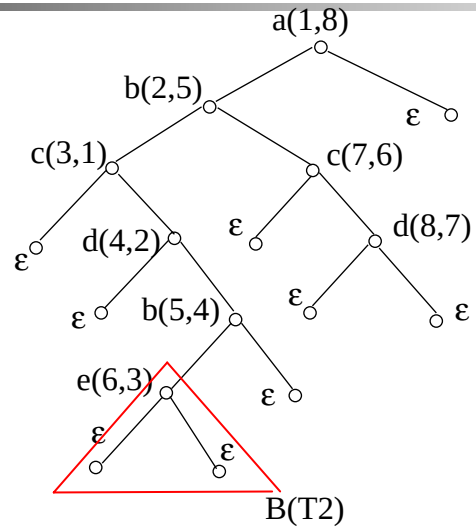
Theorem

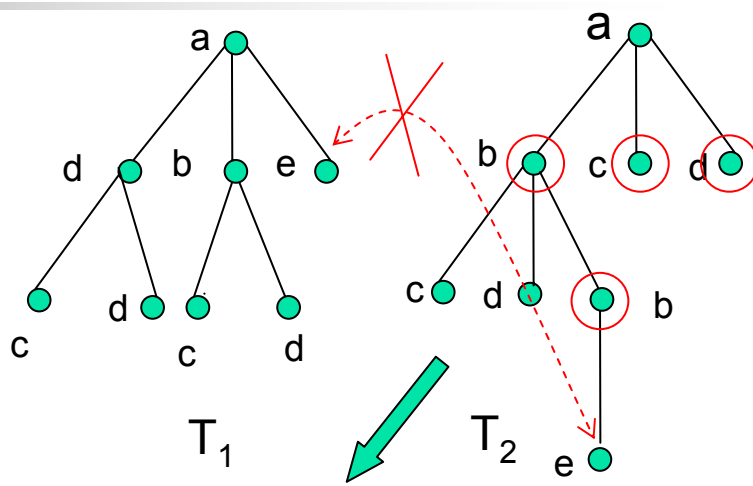
- 1 insertion/deletion incurs at most 5 difference on BBDist
- 1 rellabeling incurs at most 4 difference on BBDist
- T, T', EDIST(T, T') = k = $k_i + k_d + k_r$,
- *BDist*(*T, T'*) <= $4k_r+5k_i+5k_d$ <= 5k;

1/5 BDist is a lower bound of edit distance;

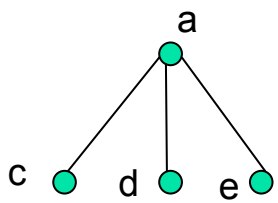
Positional Binary Branch







Incurs 0 difference for $BBDist(T_1, T_2)$



T'_2

Positional binary branch: PosBiB(T(u))

$PosBiB(T_1(e)) = (BiB(e, \varepsilon, \varepsilon), 8, 7)$ $\neq$ $PosBib(T_2(e)) = (BiB(e, \varepsilon, \varepsilon), 6, 3)$

Positional Binary Branch Distance

Computational Complexity

- D: dataset; |D|: dataset size;
- Vector construction part:
  - Traverse the data trees for once
  - Optimistic bound computation:

    time: each binary search $O(|T_i|+|T_q|)$,

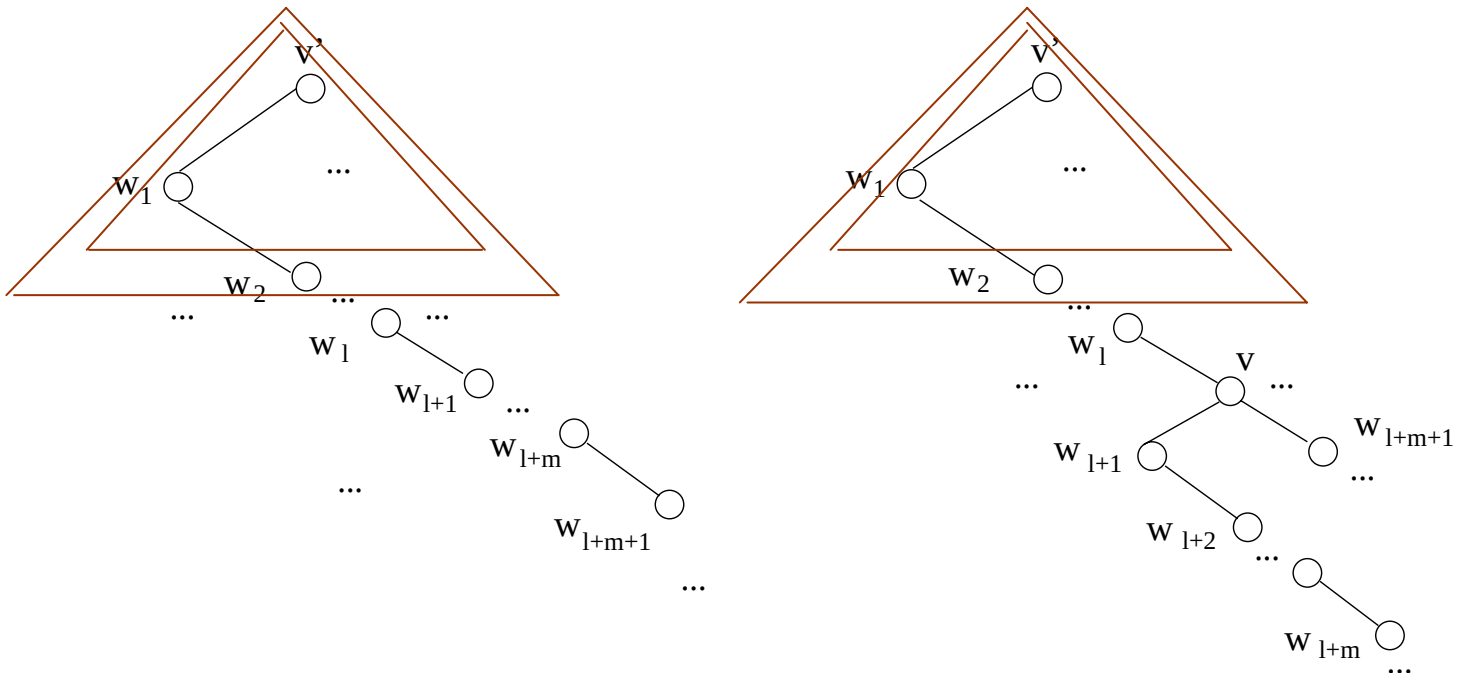
$$time,\ space:\ O(\sum_{i=1}^{|D|}|T_i|)$$

totally:
$$O(\sum_{i=1}^{|D|}(|T_i|+|T_q|)\log(\min(|T_i|,|T_q|)))$$

space:
$$O(\sum_{i=1}^{|D|}(|T_i|+|T_q|))$$

Generalized Study

- Extend the sliding window to q level
- The images vector gives multiple level binary branch profiles.
- $BDist_q(T,T') <= [4*(q-1)+1]*EDist(T,T')$



Query Processing Strategy

- Filter-and-refine frameworks
  - Lower bound distances filter out most objects
    - The lower bound computation is much succinct
    - Lower bound distance is a close approximation of the real dist
  - Remaining objects be validated by real distance

Experimental Settings

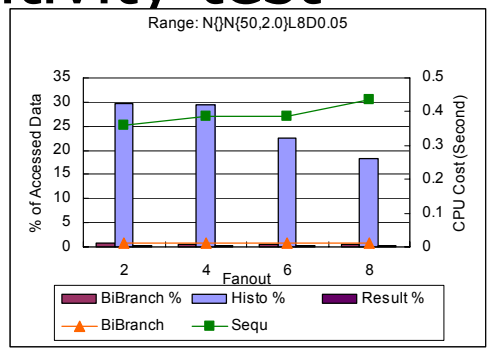
- Compare with histogram methods[KKSS04]
  - Lower bound: feature vector distance (Leaf Distance Height histogram vector, Degree histogram vector, Label histogram vector)
- Synthetic dataset:
  - Tree size, Fanout, Label, Decay factor
- Real dataset: dblp XML document
- Performance measure:
  - Percentage of data accessed:

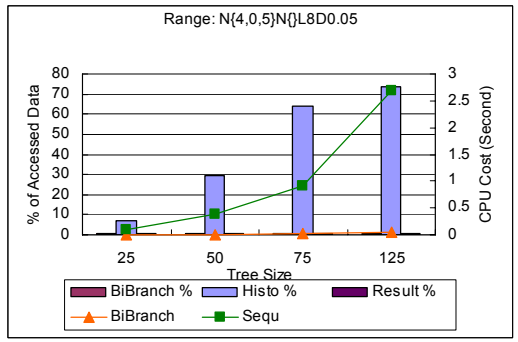
$$\frac{|\,false\ positive\,|+|\,true\ positive\,|}{|\,dataset\,|}\times 100\%$$

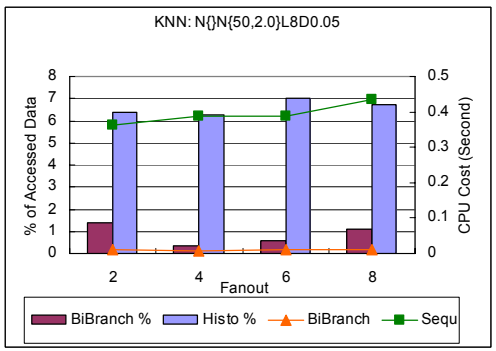
  - CPU time consumed
  - Space requirement

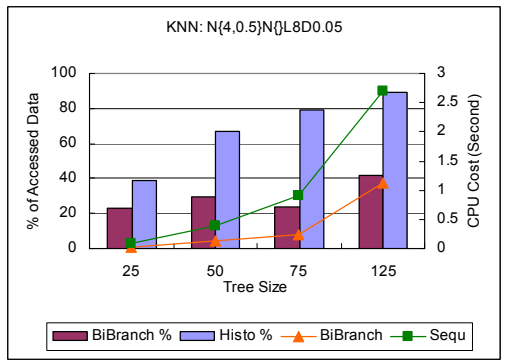
Sensitivity to the Data Properties

- Sensitivity test





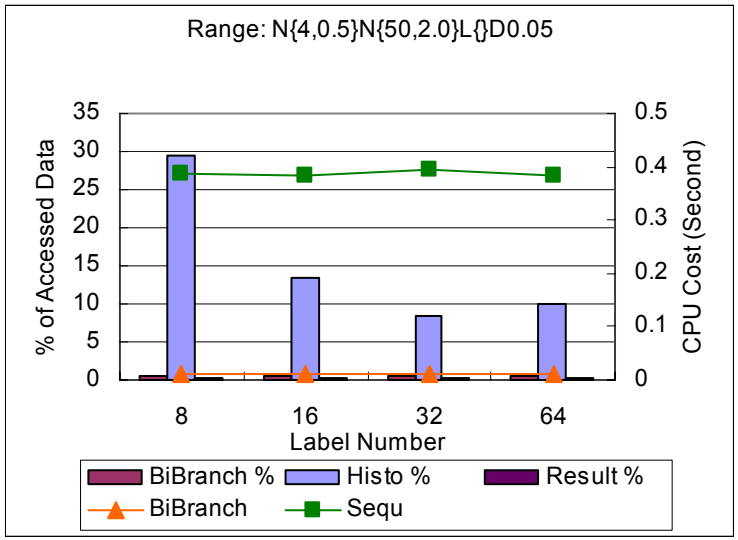


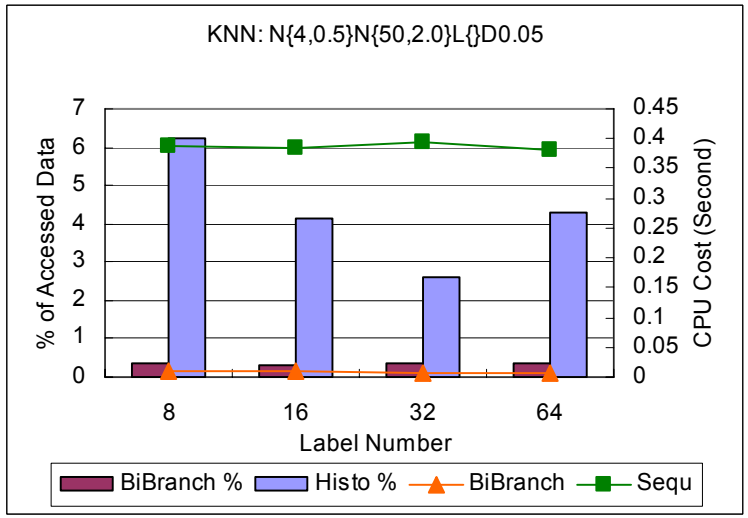


mean(fanout): 2 → 8;
mean(|T|): 50;
size(label): 8

mean(|T|): 25 → 125;
mean(fanout): 4;
size(label): 8

Sensitivity test (cont.)

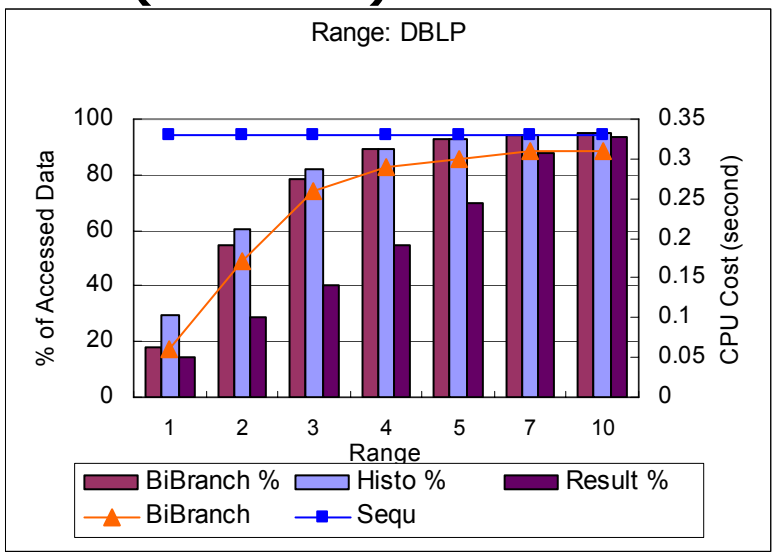


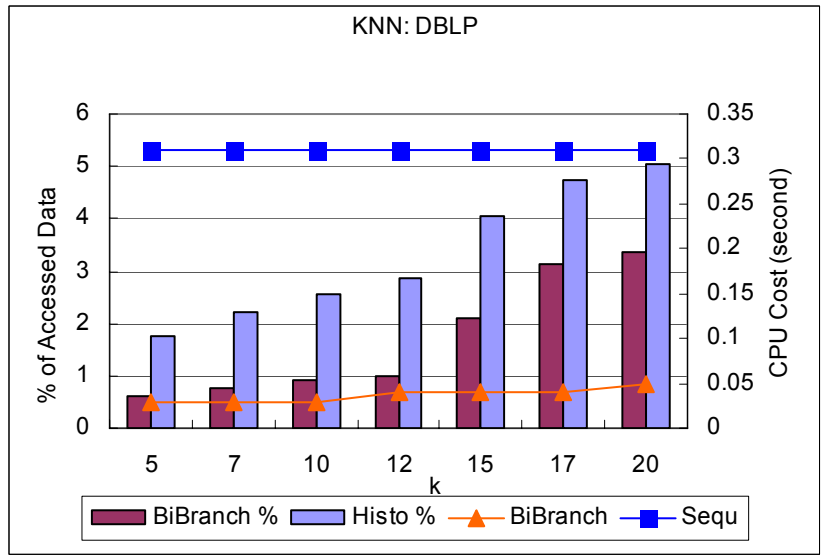


size(label): 8 → 64; mean(|T|): 50; mean(fanout): 4

Queries with Different Parameters

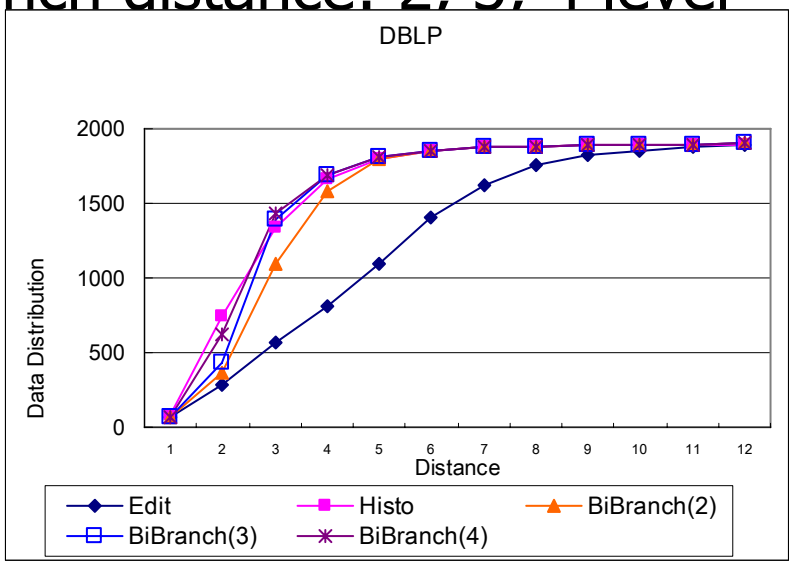
- Dblp data (avg. distance: 5.031)
- Range queries
- KNN (k:5-20)





Pruning Power of Different Level

- Data distribution according to distances
  - Edit distance
  - Histogram distance
  - Binary branch distance: 2, 3, 4 level



Citations on the Paper

- Surprisingly, attract citations and questions from software engineering! Expect more impact along software mining direction soon.
- DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones - all 2 versions »
  L Jiang, G Misherghi, Z Su, S Glondu - Proceedings of the 29th International Conference on Software …, 2007 - portal.acm.org
  Detecting code clones has many software engineering applications. Existing approaches either do not scale to large code bases or are not robust against minor code modifications. In this paper, we present an efficient **...**
- Fast Approximate Matching of Programs for Protecting Libre/Open Source Software by Using Spatial … - all 2 versions »
  AJM Molina, T Shinohara - Source Code Analysis and Manipulation, 2007. SCAM 2007. …, 2007 - doi.ieeecomputersociety.org
  To encourage open source/libre software development, it is desirable to have tools that can help to identify open source license violations. This paper describes the imple-mentation of a tool that matches open source programs ...

Outline

- Importance of Trees
- Distance between Trees
- Kernel methods for Trees
- Fast Distance Computations
- Mining Frequent Subtrees

Mining Complex Structures

- Frequent Structure Mining tasks:
  - Item sets (transactional, unordered data)
  - Sequences (temporal/positional: text, bioseqs)
  - **Tree patterns (semi-structured / XML data, web mining, bioinformatics, etc.)**
  - Graph patterns (bioinformatics, web data)
- "Frequent" used broadly:
  - Maximal or closed patterns in dense data
  - Correlation, other statistical metrics
  - Interesting, rare, non-redundant patterns

Tree Mining: Motivation

- Capture intricate (subspace) patterns
- Can be used (as features) to build global models (classification, clustering, etc.)
- Ideally suited for categorical, high-dimensional, complex and massive data
- Interesting Applications
  - XML, semi-structured data: Mine structure + content for Classification
  - Web usage mining: Log mining (user sessions as trees)
  - Bioinformatics: RNA sub-structures, phylogenetic trees

Contributions

- Mining embedded subtrees in rooted, ordered, and labeled trees (forest) or a single large tree
- Notion of node scope
- Representing trees as strings
- Scope-lists for subtree occurrences
- Systematic subtree enumeration
- Extensions for mining unlabeled or unordered subtrees or sub-forests

Tree Mining: Definitions

- Rooted tree: special node called root
- Ordered tree: child order matters
- Labeled tree: nodes have labels
- Ancestor (embedded child): $x \leq_l y$ (l length path x to y)
- Sibling nodes: two nodes having same parent
- Embedded siblings: two nodes having common ancestor
- Depth-first Numbering: node's position in a pre-order traversal of the tree
- A node has a number n_i and a label $l(n_i)$
- Scope of node n_l is $[l, r]$, n_r is rightmost leaf under n_l

Tree Mining: Definitions

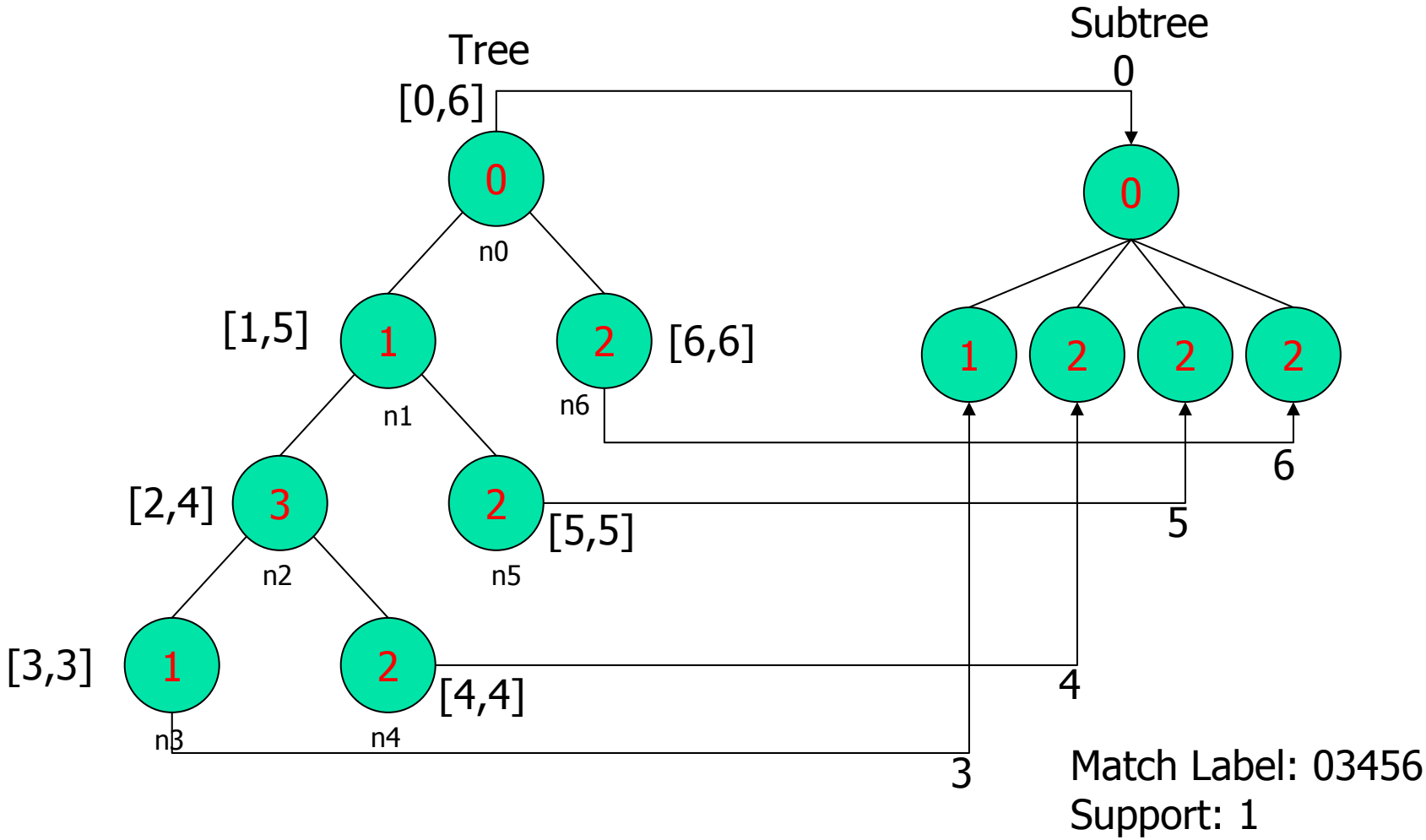
- Embedded Subtrees: $S = (N_s, B_s)$ is a subtree of

  $T = (N,B)$ if and only if (iff)
  - $N_s \subseteq N$
  - $b = (n_x, n_y) \in B_s$ iff $n_x \leq_l n_y$ in T (n_x ancestor of n_y)
  - Note: in an induced subtree $b = (n_x, n_y) \in B_s$ iff $(n_x, n_y) \in B$ (n_x is parent of n_y)
  - We say S *occurs* in T if S is a subtree of T
  - If S has *k* nodes, we call it a *k*-subtree
- Able to extract patterns hidden (embedded) deep within large trees; missed by traditional definition of induced subtrees

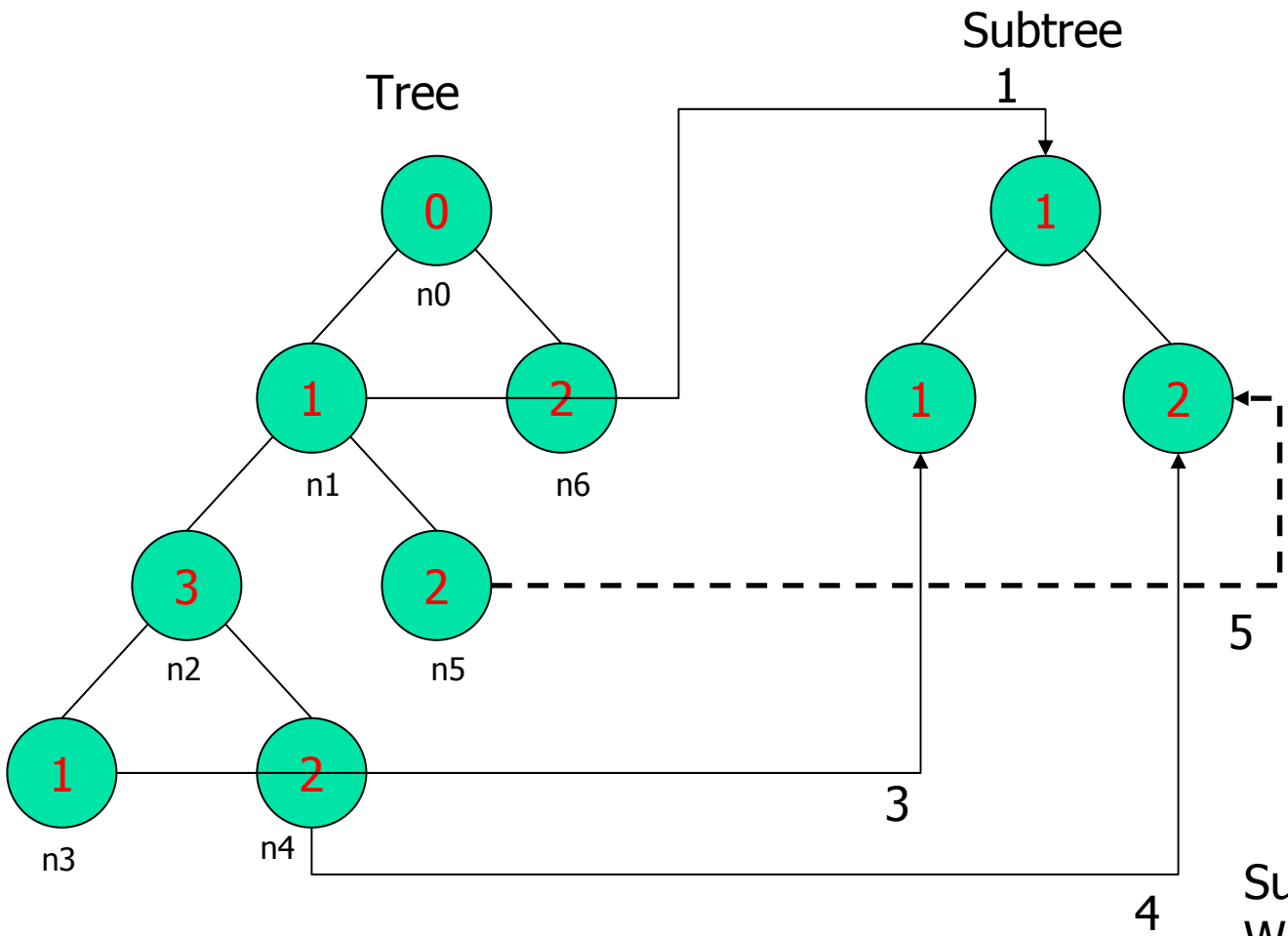
Tree Mining Problem

- Match labels of S in T
  - Positions in T where each node of S matches
  - Match label is unique for each occurrence of S in T
- *Support*: Subtree may occur more than once in a tree in D, but count it only once
- *Weighted Support*: Count each occurrence of a subtree (e.g., useful when |D| =1)
- Given a database (forest) D of trees, find all frequent embedded subtrees
  - Should occur in a minimum number of times
  - used-defined *minimum support (minsup)*

Subtree Example



Subtree Example



Match Labels:

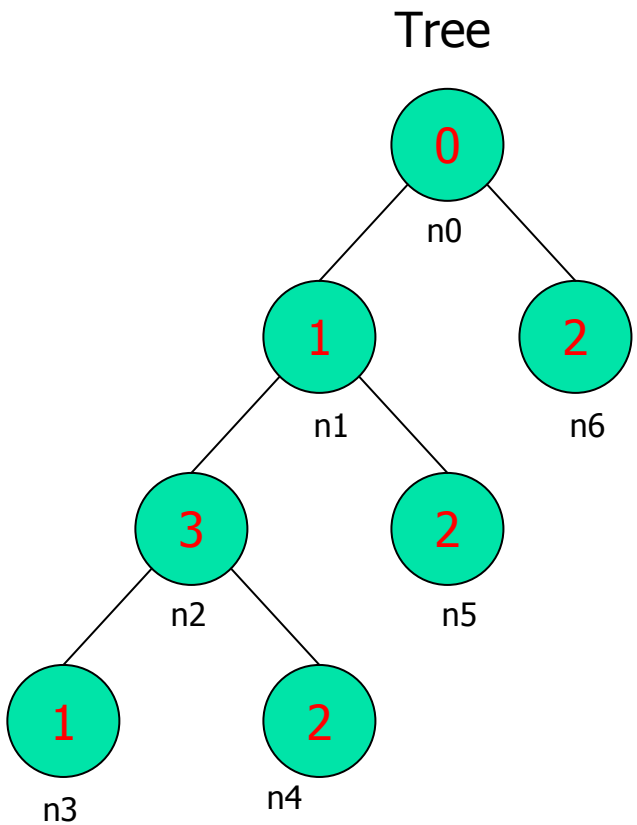
134

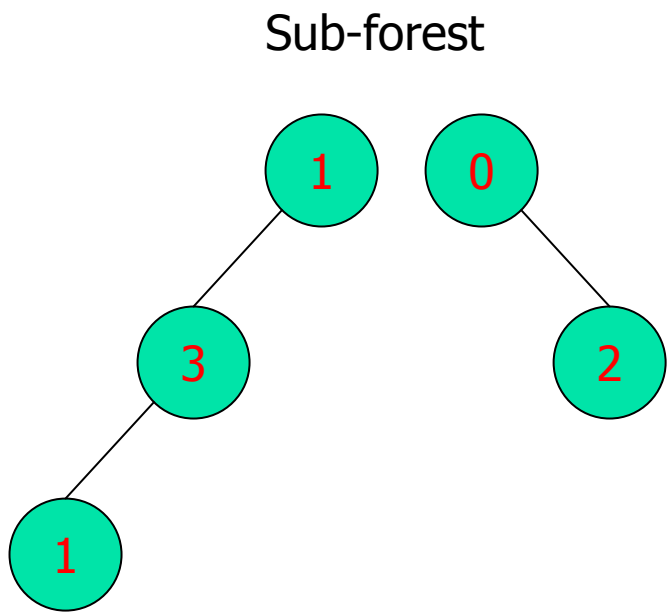
135

Support = 1
Weighted Support = 2

Example sub-forest (not a subtree)

By definition a subtree is connected
A disconnected pattern is a sub-forest

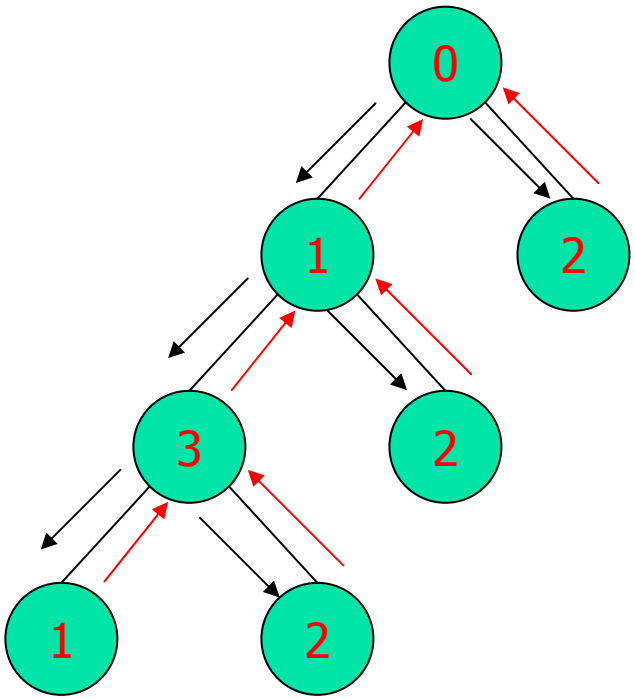




Tree Mining: Main Ingredients

- Pattern representation
  - Trees as strings
- Candidate generation
  - No duplicates
- Pattern counting
  - Scope-list based (TreeMiner)
  - Pattern matching based (PatternMatcher)

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

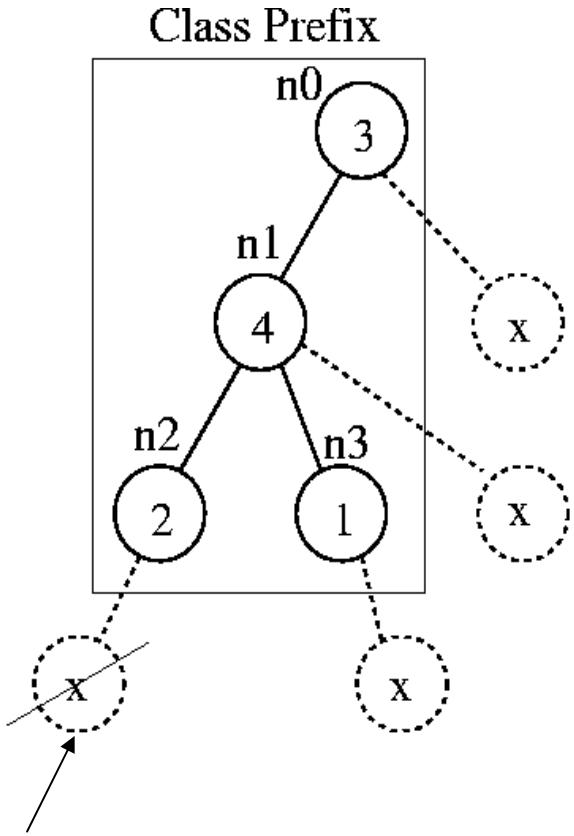
Adjacency List requires: 4N-2 space

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

String representation requires: 2N-1 space

Systematic Candidate Generation: Equivalence Classes

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



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

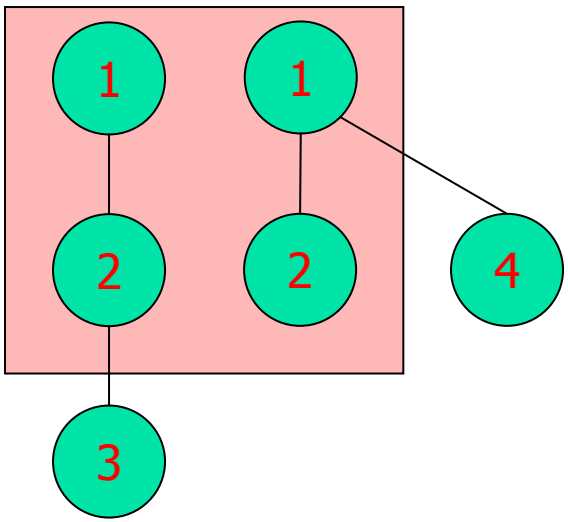
A valid element *x* attached to only the nodes lying on the path from root to rightmost leaf in prefix *P*

Candidate Generation

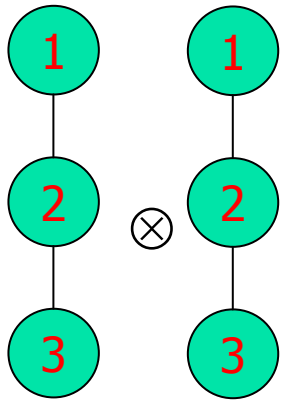
- Generate new candidates (k+1)-subtrees from equivalence classes of k-subtrees
- Consider each pair of elements in a class, including self-extensions
- Up to two new candidates from each pair of joined elements
- All possible candidates subtrees are enumerated
- Each subtree is generated only once!

Candidate Generation (Join operator ⊗)

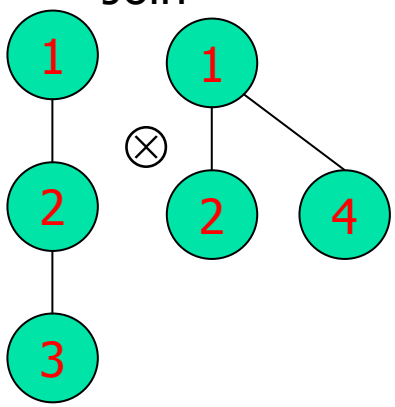
Equivalence Class
Prefix: 1 2, Elements: (3,1) (4,0)



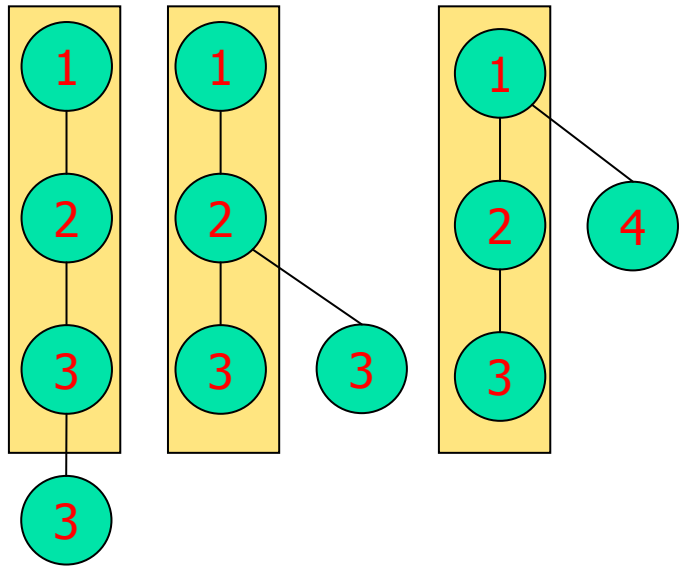
Self Join



Join

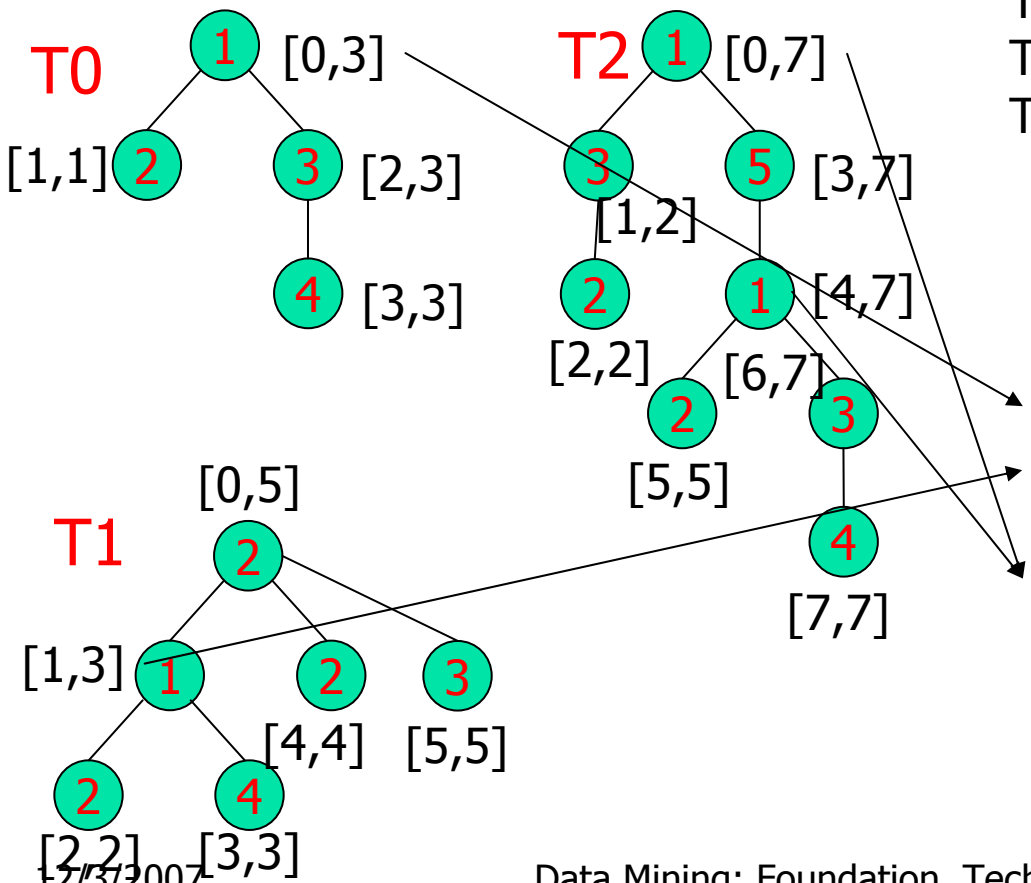


New Candidates



New Equivalence Class
Prefix: 1 2 3
Elements: (3,1) (3,2) (4,0)

TreeMiner:Scope List for Trees



String Representation

T0: 1 2 -1 3 4 -1 -1
T1: 2 1 2 -1 4 -1 -1 2 -1 3 -1
T2: 1 3 2 -1 -1 5 1 2 -1 3 4 -1 -1 -1 -1

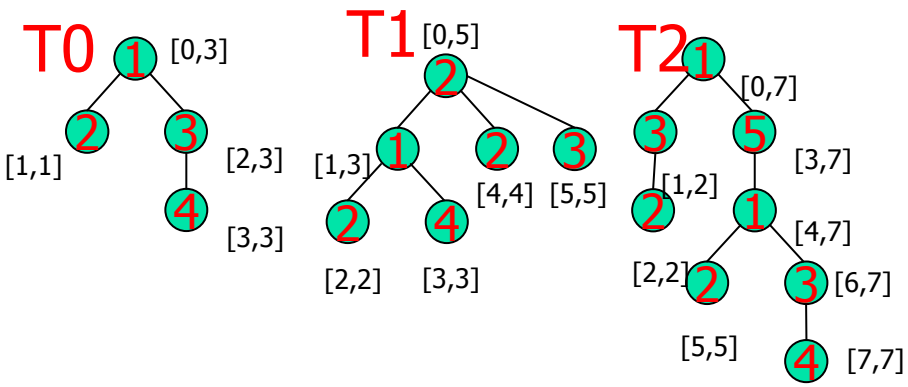
Scope-Lists

1	2	3	4
0, [0,3]	0, [1,1]	0, [2,3]	0,[3,3]
1, [1,3]	1, [0,5]	1, [5,5]	1,[3,3]
2, [0,7]	1, [2,2]	2, [1,2]	2,[7,7]
2, [4,7]	1, [4,4]	2, [6,7]	
	2, [2,2]		
	2, [5,5]		

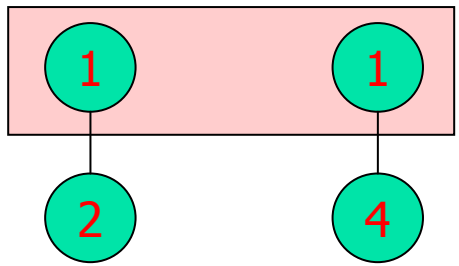
5

2, [3,7]

Frequency Computation: Scope List Joins ($\cap_{\otimes}$) – In Scope



Minsup = 3 (100%)



Interval Algebra
[0, 3] ⊃ [1, 1]

0, 0, [1,1]
1, 1, [2,2]
2, 0, [2,2]
2, 0, [5,5]
2, 4, [5,5]

0, 0, [3,3]
1, 1, [3,3]
2, 0, [7,7]
2, 4, [7,7]

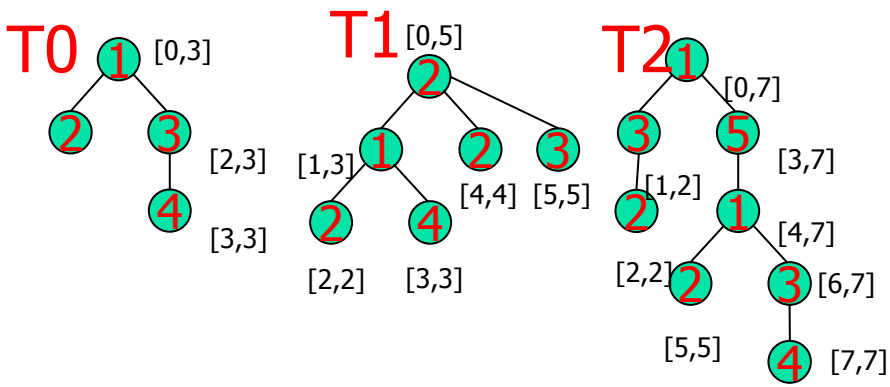
Count = 3

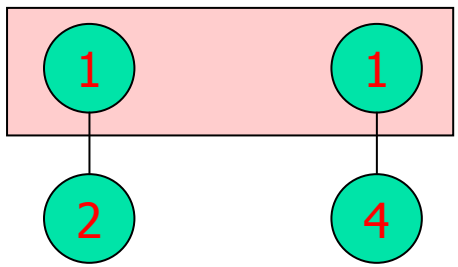
Equivalence Class: Prefix =∅
Elements: (1,-1) (2,-1) (3,-1) (4,-1)

1	2	3	4
0, [0,3]	0, [1,1]	0, [2,3]	0,[3,3]
1, [1,3]	1, [0,5]	1, [5,5]	1,[3,3]
2, [0,7]	1, [2,2]	2, [1,2]	2,[7,7]
2, [4,7]	1, [4,4]	2, [6,7]	
	2, [2,2]		
	2, [5,5]		

Tree Id, Prefix Match Label, Last Node Scope
0 0 [1,1]

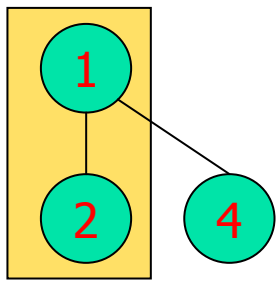
Scope List Joins: Out Scope





0, 0, [1,1]
1, 1, [2,2]
2, 0, [2,2]
2, 0, [5,5]
2, 4, [5,5]

0, 0, [3,3]
1, 1, [3,3]
2, 0, [7,7]
2, 4, [7,7]



0, 01, [3,3]
1, 12, [3,3]
2, 02, [7,7]
2, 05, [7,7]
2, 45, [7,7]

Interval Algebra
[1, 1] < [3, 3]

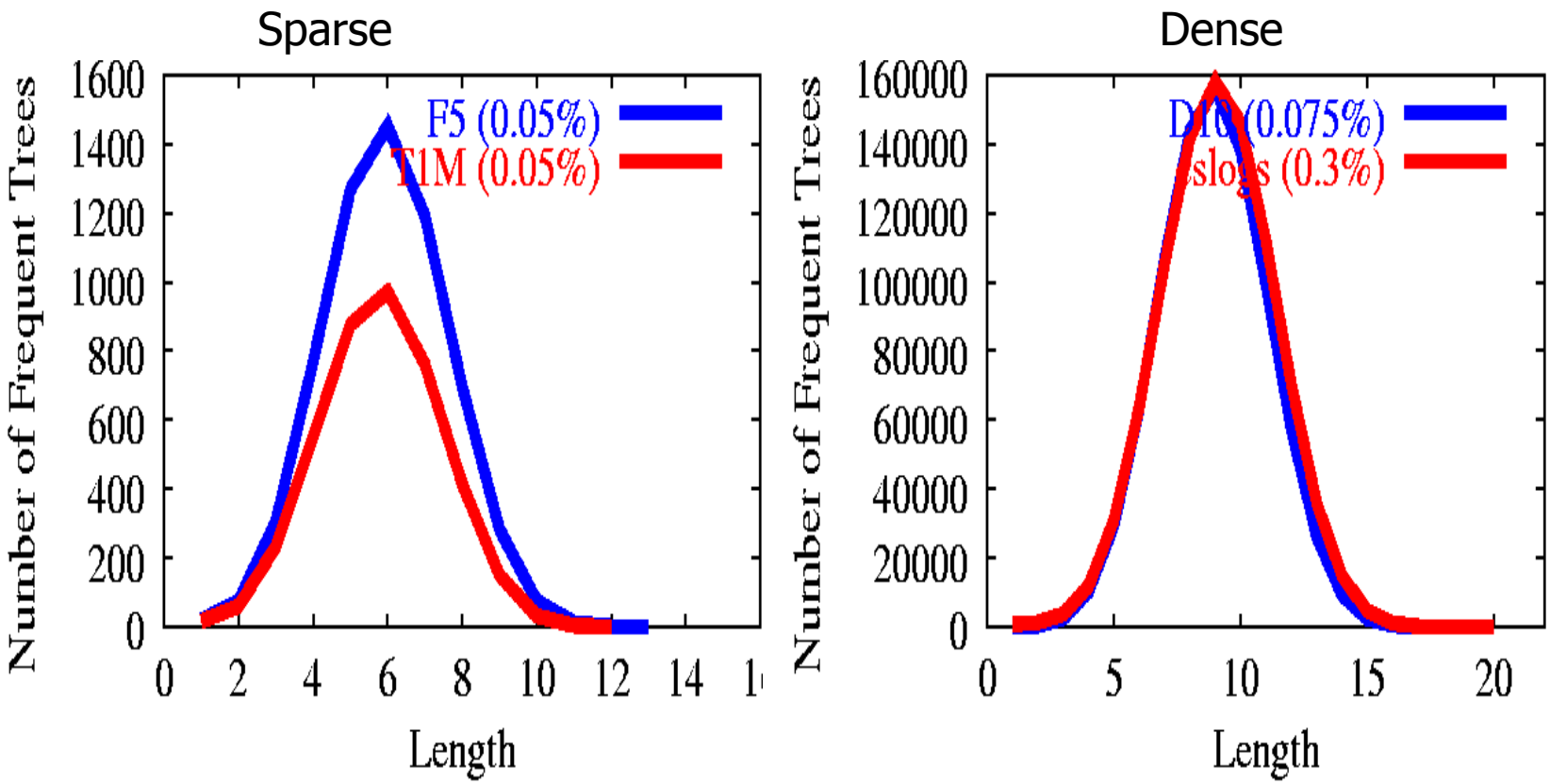
Pattern Matcher

- Level-wise Apriori Style Algorithm
- Uses optimized pattern matching
  - Equivalence classes
  - 3 step matching
    - Finding matching leafs
    - Prefix matching
    - Element matching

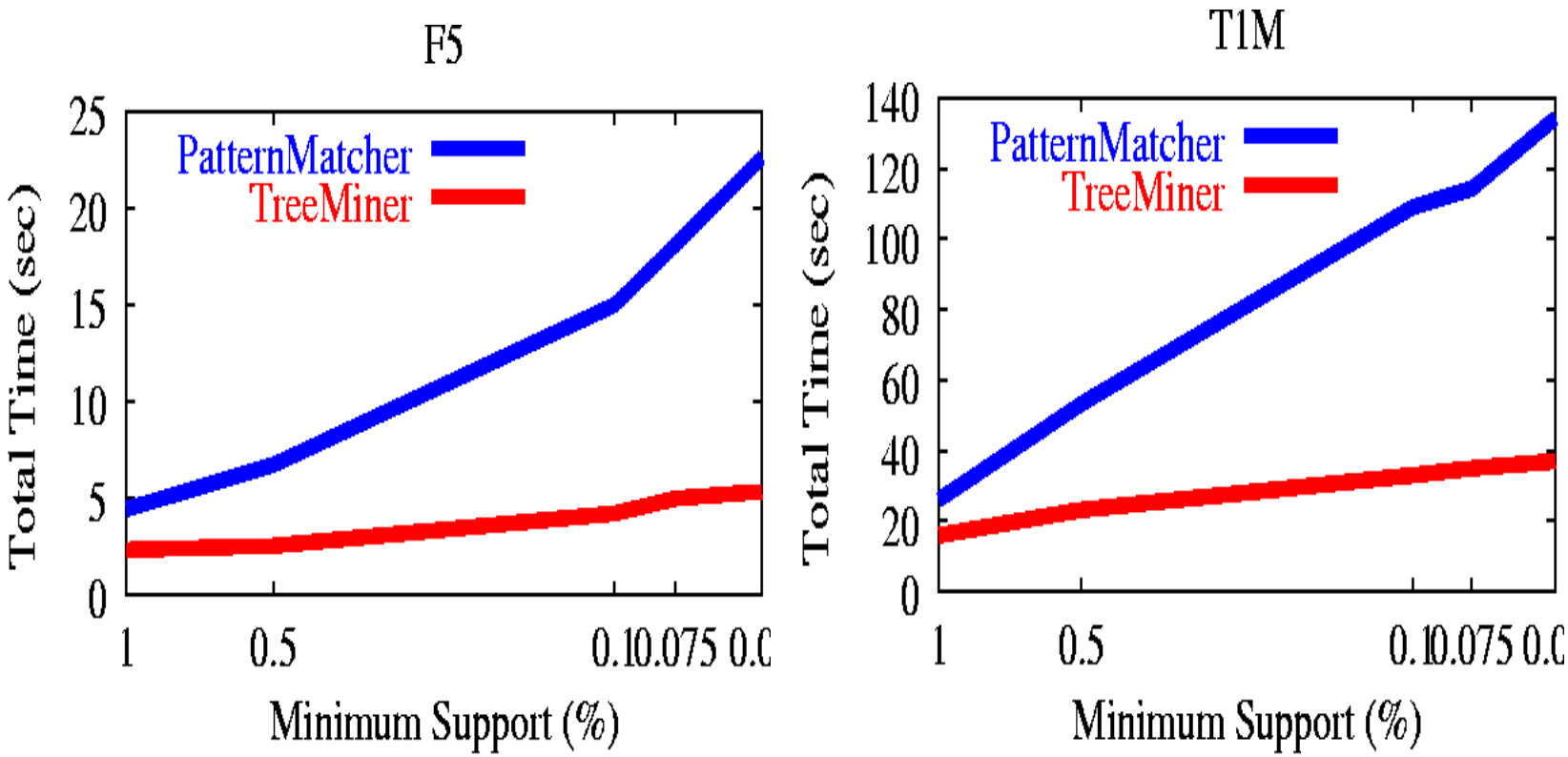
Experimental Results

- Machine: 500Mhz PentiumII, 512MB memory, 9GB disk, Linux 6.0
- Synthetic Data: Web browsing
  - Parameters: N = #Labels, M = #Nodes,
    F = Max Fanout, D = Max Depth, T = #Trees
  - Create master website tree W
    - For each node in W, generate #children (0 to F)
    - Assign probabilities of following each child or to backtrack; adding up to 1
    - Recursively continue until D is reached
  - Generate a database of T subtrees of W
    - Start at root. Recursively at each node generate a random number (0-1) to decide which child to follow or to backtrack.
  - Default parameters: N=100, M=10,000, D=10, F=10, T=100,000
  - Three Datasets: D10 (all default values), F5 (F=5), T1M ($T=10^6$)
- Real Data: CSLOGS – 1 month web log files at RPI CS
  - Over 13361 pages accessed (#labels)
  - Obtained 59,691 user browsing trees (#number of trees)
  - Average string length of 23.3 per tree

Distribution of Frequent Trees

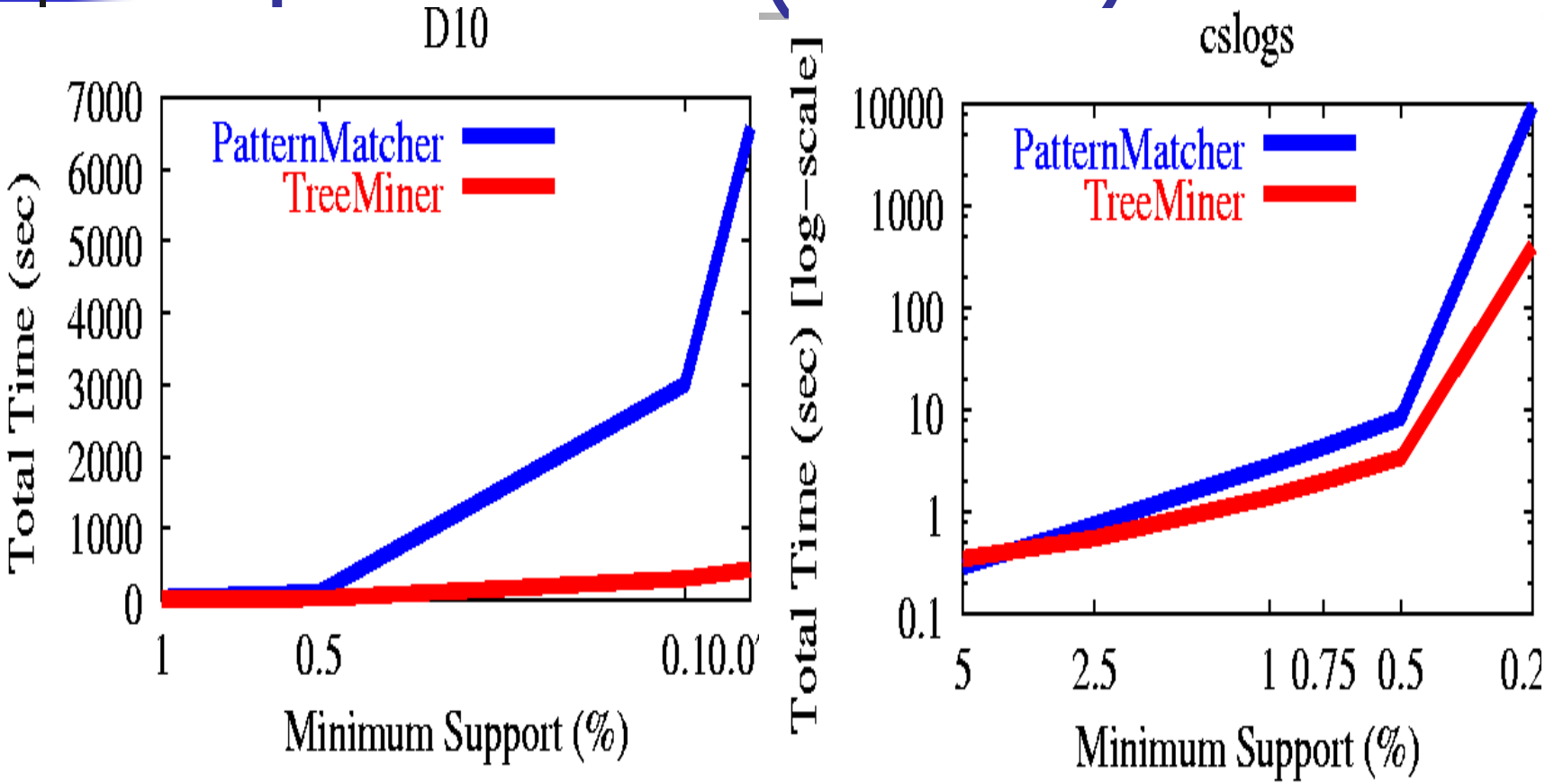


Experiments (Sparse)



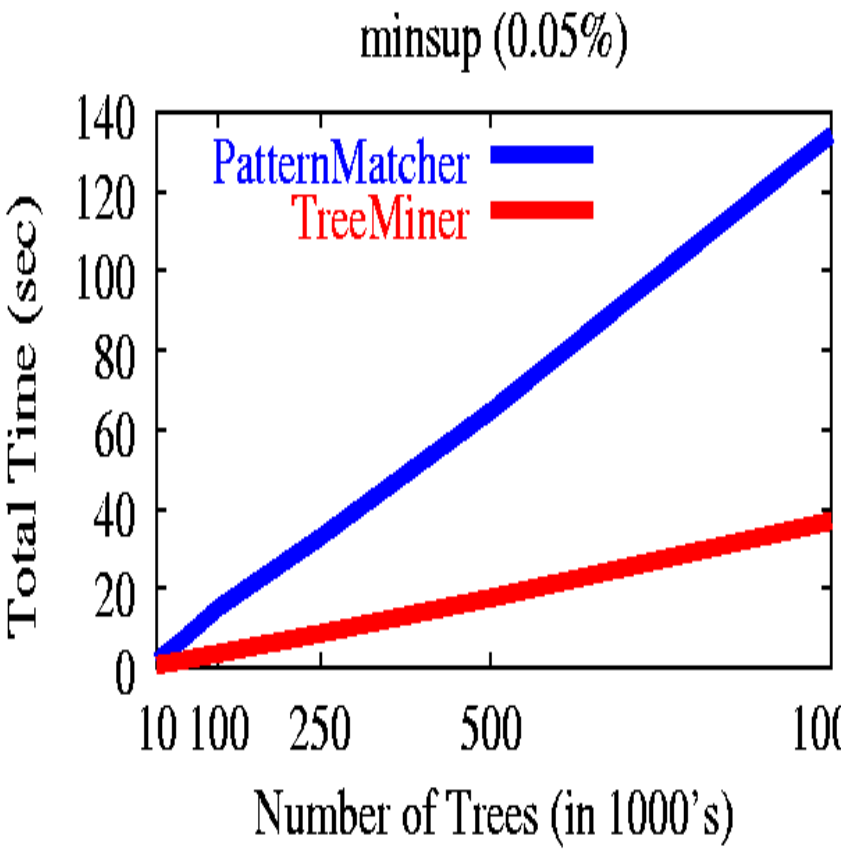
- Relatively short patterns in sparse data
- Level-wise approach able to cope with it
- TreeMiner about 4 times faster

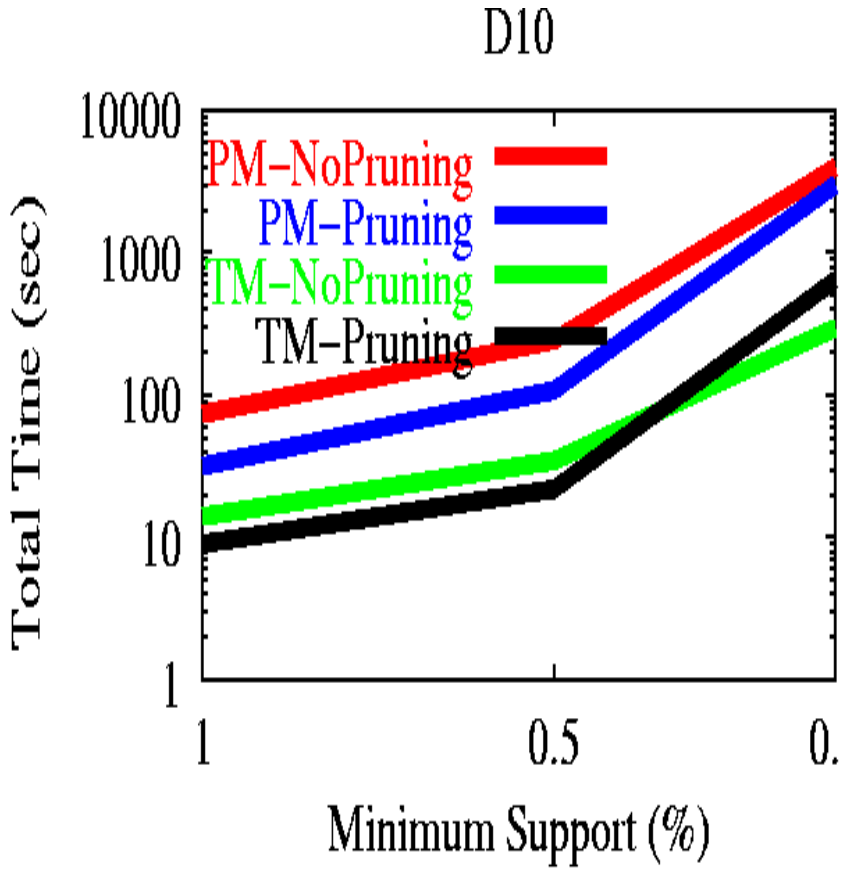
Experiments (Dense)



- Long patterns at low support (length=20)
- Level-wise approach suffers
- TreeMiner 20 times faster!

Scaleup


minsup (0.05%)
Total Time (sec)
140
120
100
80
60
40
20
0
PatternMatcher
TreeMiner
10 100
250
500
Number of Trees (in 1000's)


D10
Total Time (sec)
10000
1000
100
10
1
PM-NoPruning
PM-Pruning
TM-NoPruning
TM-Pruning
1
0.5
Minimum Support (%)

Application: Web/XML Mining

- LOGML (Log Markup Language) Documents
  - Log graph of the site in XGMML
  - Information for log reports
    - top visiting hosts, top user agents, top keywords, etc.
  - User session (subgraph of log graph)
    - list of edges referring to nodes of log graph with time stamps
- Extract user sessions from LOGML db to create User Graphs
  - Session Id (IP or host name)
  - Path count (# of edges)
  - Edge Information (source-destination)
  - Time when an edge is traversed
- Form a task-relevant (sets, sequences, trees, etc.) database from user graphs

User Graph

<userSession name="ppp0-69.ank2.isbank.net.tr" ...>
<path count="6">
<uedge source="5938" target="16470" utime="24/Oct/2000:07:53:46"/>
<uedge source="16470" target="24754" utime="24/Oct/2000:07:56:13"/>
<uedge source="16470" target="24755" utime="24/Oct/2000:07:56:36"/>
<uedge source="24755" target="47387" utime="24/Oct/2000:07:57:14"/>
<uedge source="24755" target="47397" utime="24/Oct/2000:07:57:28"/>
<uedge source="16470" target="24756" utime="24/Oct/2000:07:58:30"/>

Set Mining

Transaction Format: user name, number of nodes

accessed, node list

ppp0-69.ank2.isbank.net.tr 7 5938 16470 24754 24755 47387 47397 24756

Example from 1 day's logs at RPI CS dept.

Let Path=http://www.cs.rpi.edu/~sibel/poetry

FREQUENCY=16, NODE IDS = 16395 38699 38700 38698 593

Path/poems/akgun_akova/index.html

Path/poems/akgun_akova/picture.html

Path/poems/akgun_akova/biyografi.html

Path/poems/akgun_akova/contents.html

Path/sair_listesi.html

Sequence Mining

Format: user name, sequence id, nodes accessed (maximal forward paths)

ppp0-69.ank2.isbank.net.tr 1 5938 16470 24754
ppp0-69.ank2.isbank.net.tr 2 5938 16470 24755 47387
ppp0-69.ank2.isbank.net.tr 3 5938 16470 24755 47397
ppp0-69.ank2.isbank.net.tr 4 5938 16470 24756

Let Path=http://www.cs.rpi.edu/~sibel/poetry
FREQUENCY = 20, NODE IDS = 5938 -> 16395 -> 38698
Path/sair_listesi.html ->
Path/poems/akgun_akova/index.html ->
Path/poems/akgun_akova/contents.html

Tree Mining

Format: user name, tree string

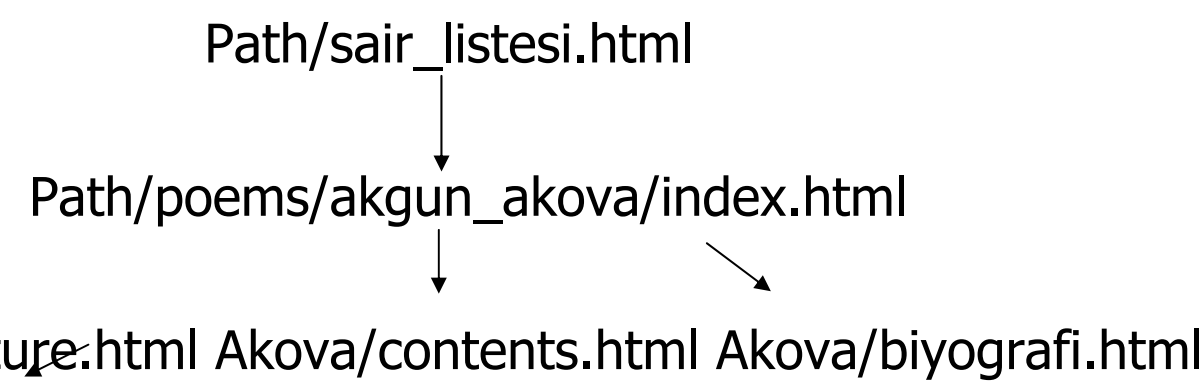
ppp0-69.ank2.isbank.net.tr, 5938 16470 24754 -1 24755 47387 -1 47397

-1 -1 24756 -1 -1

Let Path=http://www.cs.rpi.edu/~sibel/poetry

Let Akova = Path/poems/akgun_akova

FREQUENCY=59, NODES = 5938 16395 38699 -1 38698 -1 38700



Summary

- TreeMiner: Novel tree mining approach
  - Non-duplicate candidate generation
  - Scope-list joins for frequency computation
- Framework for Tree Mining Tasks
  - Frequent subtrees in a forest of rooted, labeled, ordered trees
  - Frequent subtrees in a single tree
  - Unlabeled or unordered trees
  - Frequent Sub-forests
- Outperforms pattern matching approach
- Future Work: constraints, maximal subtrees, inexact label matching

Conclusion

- How to compute distance between two trees?
  - Edit distance
  - Kernel method
- How to quickly approximate the distance?
  - L_1 metric using vector representation
- Which structures occurs frequently in a database of trees?
  - Mining frequent embedded subtrees

References

- Philip Bille . A survey on tree edit distance and related problems. Theoretical Computer Science. Volume 337 , Issue 1-3 (June 2005)
- Rui Yang, Panos Kalnis, Anthony K. H. Tung: Similarity Evaluation on Tree-structured Data. SIGMOD 2005.
- Hisashi Kashima Teruo Koyanagi. Kernels for Semi-Structured Data. Proceedings of the Nineteenth International Conference on Machine Learning. 291-298. 2002.
- Mohammed J. Zaki. "Efficiently mining frequent trees in a forest". Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. Pg. 71-80. 2002

Optional References:

- JP Vert. "A tree kernel to analyze phylogenetic profiles" - Bioinformatics, 2002 - Oxford Univ Press