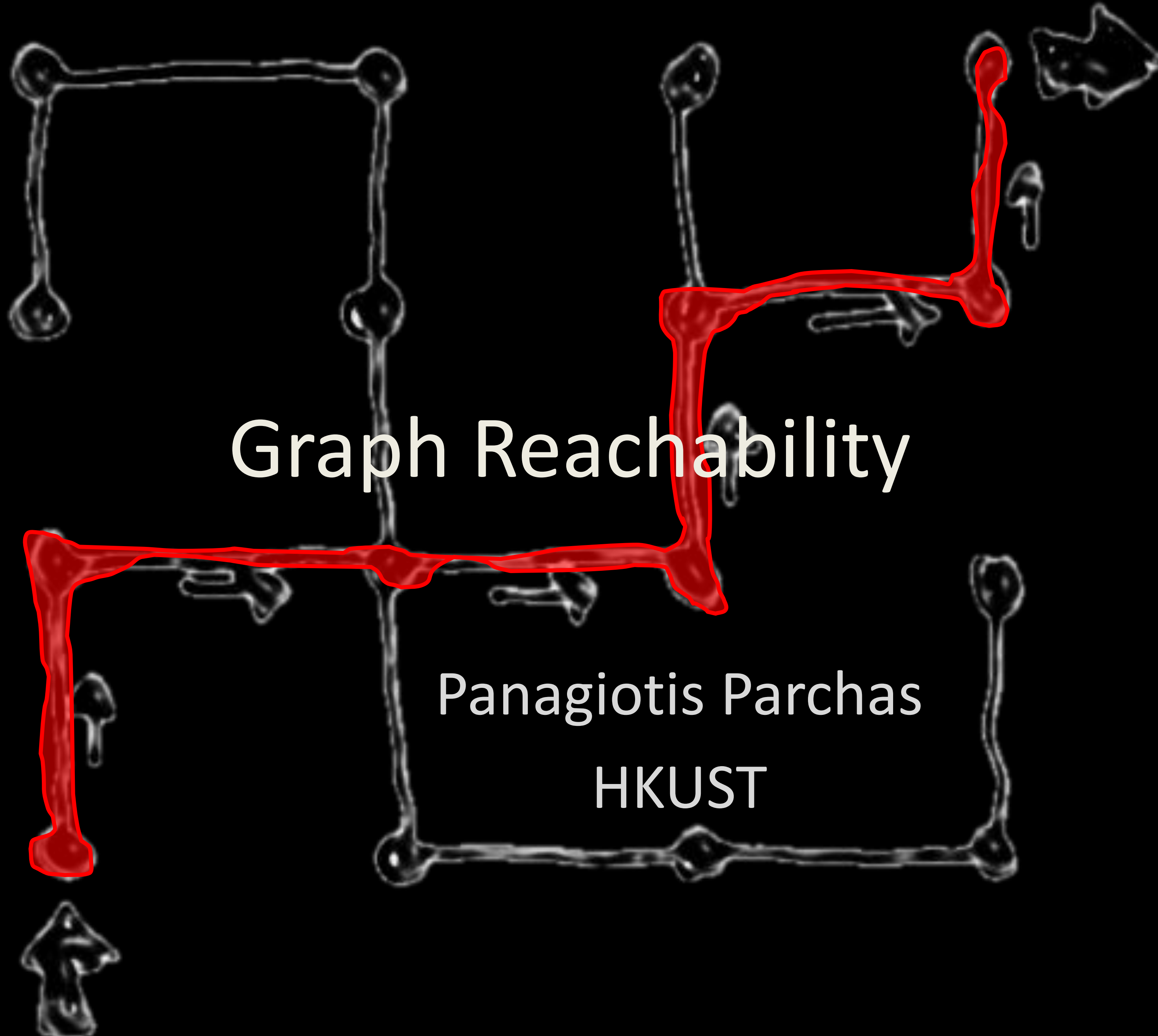


Graph Reachability

Panagiotis Parchas
HKUST

Where are the graphs?

- Some famous graph data
  - Social Networks
  - Web sites/internet
  - XML Documents
  - Biological/chemical networks

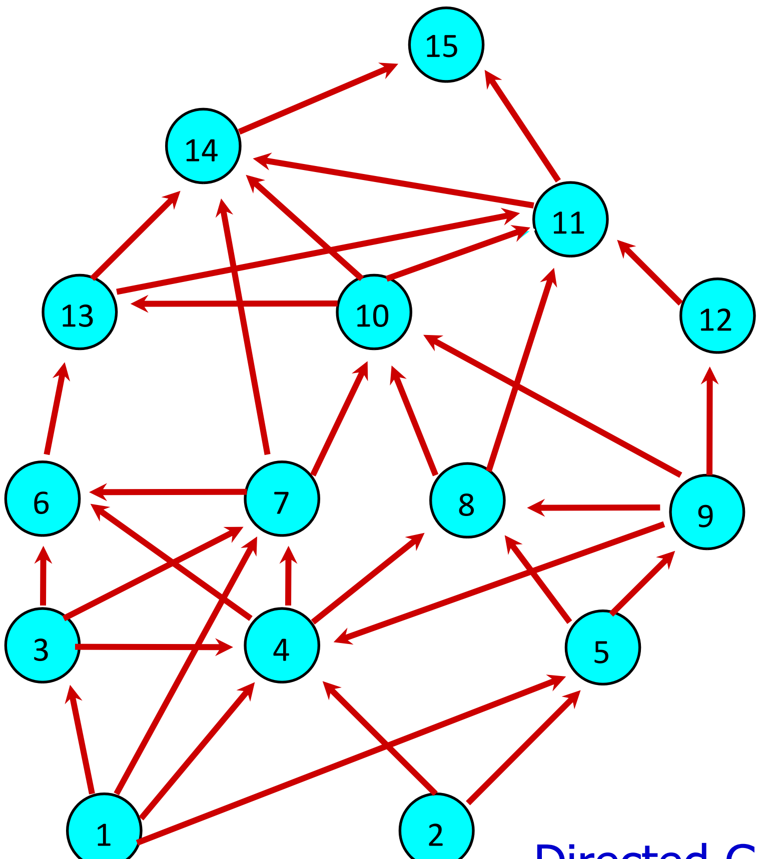
Why graph analysis?

- Knowledge discovery in social networks
  - Targeted advertising.
  - Pattern extraction for behavioral analysis.
  - Useful statistics and conclusions for a wide range of scientists
- Mining in Web Data
  - Interesting navigation patterns
  - XML documents
- Biological Data Analysis
  - Drug discovery
  - DNA Analysis
- Geographic Information Systems

- ………And much more……..

Problem formulation

Given a *directed graph G* and two nodes u and v, is there a path connecting u to v (denoted u↝v)?



3↝11? YES

3↝12? NO

Directed Graph → DAG (directed acyclic graph) by coalescing the strongly connected components

Motivation

- Classical problem in graph theory.
- Studying the influence flow in social networks
  - Even undirected graphs (facebook) are converted to directed w.r.t a certain attribute distribution
- Security: finding possible connections between suspects
- Biological data: is that protein involved- directly or indirectly- in the expression of a gene?
- Primitive for many graph related problems (pattern matching)

Methods Overview

	Method	Query time	Construction	Index size
Naïve	DFS/BFS	O(n+m)	O(n+m)	O(n+m)
	Transitive Closure	O(1)	$O(nm)=O(n^3)$	$O(n^2)$
Tree Cover	Optimal Tree Cover (Agrawal et al., SIGMOD'89)	O(n)	$O(nm)=O(n^3)$	$O(n^2)$
	GRIPP (Trißl et al., SIGMOD'07)	O(m-n)	O(n+m)	O(n+m)
	Dual-Labeling (Wang et al., ICDE'06)	O(1)	$O(n+m+t^3)$	$O(n+t^2)$
Chain Cover	Optimal Chain Cover (Jagadish, TODS'90)	O(k)	O(nm)	O(nk)
	Path-Tree (Jin, et al., SIGMOD'08)	$\log^2 k'$	O(mk')/O(mn)	O(nk')
HOP Cover	2-HOP (SODA 2002)	$O(nm^{1/2})$	$O(n^3\|T_C\|)=O(n^5)$	$O(m^{1/2})$
	3-HOP (Yang Xiang et al., SIGMOD '09)	O(log n +k)	$O(kn^2)$	O(nk)

Depth First Traversal

- ? u⤳v
- Depth First Traversal (DFT) starting from u
- if node v is discovered:
  - then stop search, report YES
- If all nodes have been visited:
  - then report NO

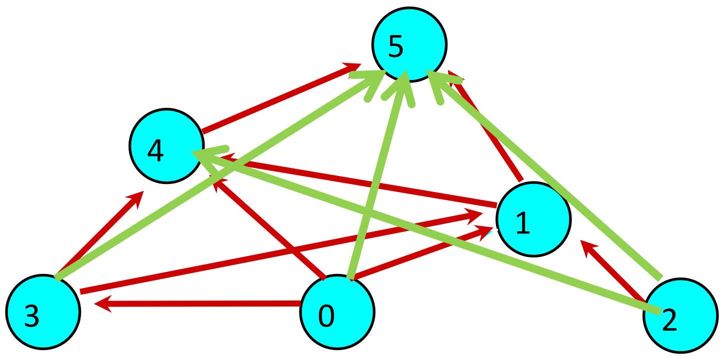
No index and thus no construction overhead and no extra space consumption

TOO GOOD

Query time: O(m+n) the entire graph should be traversed in the worst case

TOO BAD

Transitive Closure (TC)



	0	1	2	3	4	5
0	1	1	0	1	1	1
1	0	1	0	0	1	1
2	0	1	1	0	1	1
3	0	1	0	1	1	1
4	0	0	0	0	1	1
5	0	0	0	0	0	1

- It can be done by dynamic programming algorithm *Floyd–Warshall* in $\Theta(n^3)$
- It takes $O(n^2)$ space

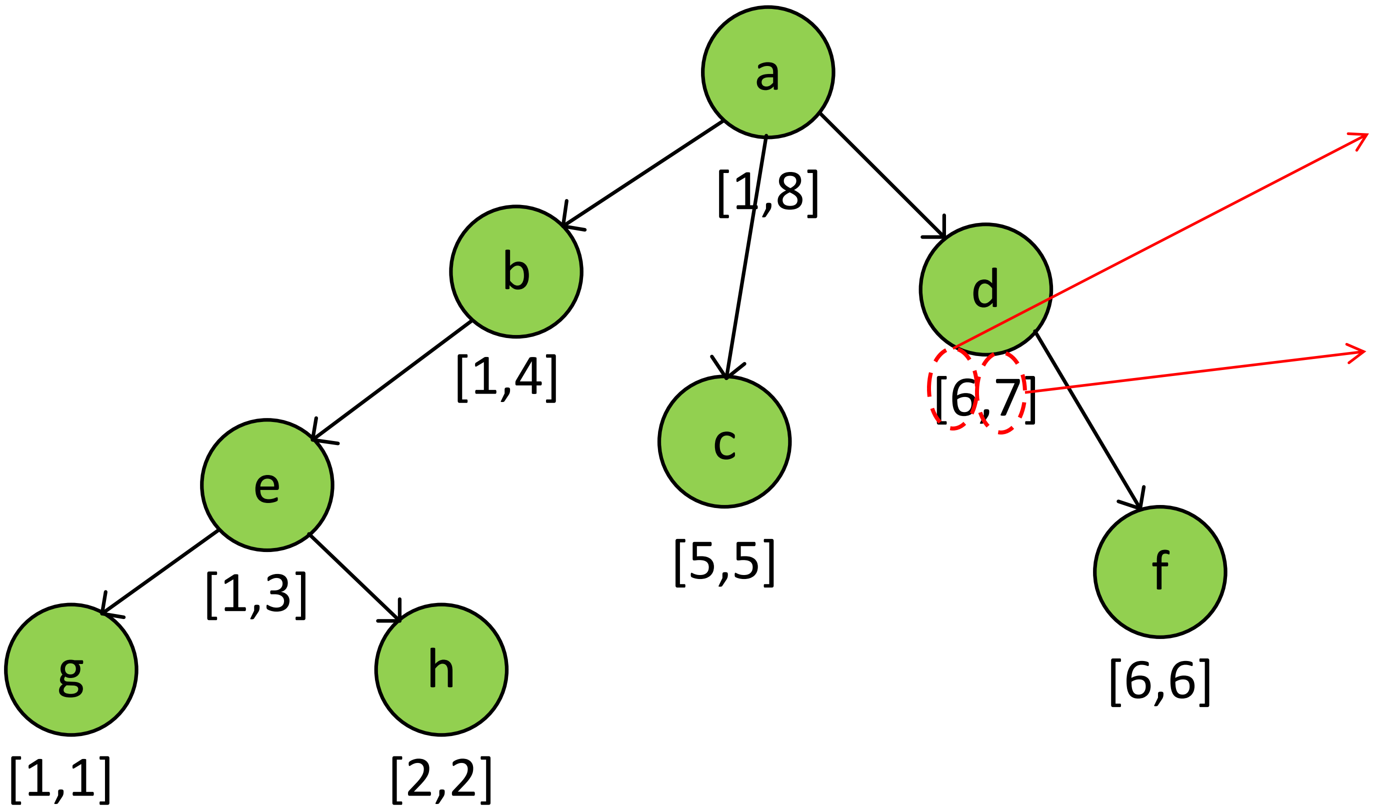
TOO BAD

- BUT, queries can be answered in constant time $O(1)$

TOO GOOD

Optimal Tree Cover [idea]

SIGMOD ‘89



- *Index*: smallest postorder number of descendants
- DFT postorder number

Postorder: left-right-root

(example) $?(b \rightsquigarrow h) \Longrightarrow ?(1 \leq 2 < 4) \Rightarrow YES$

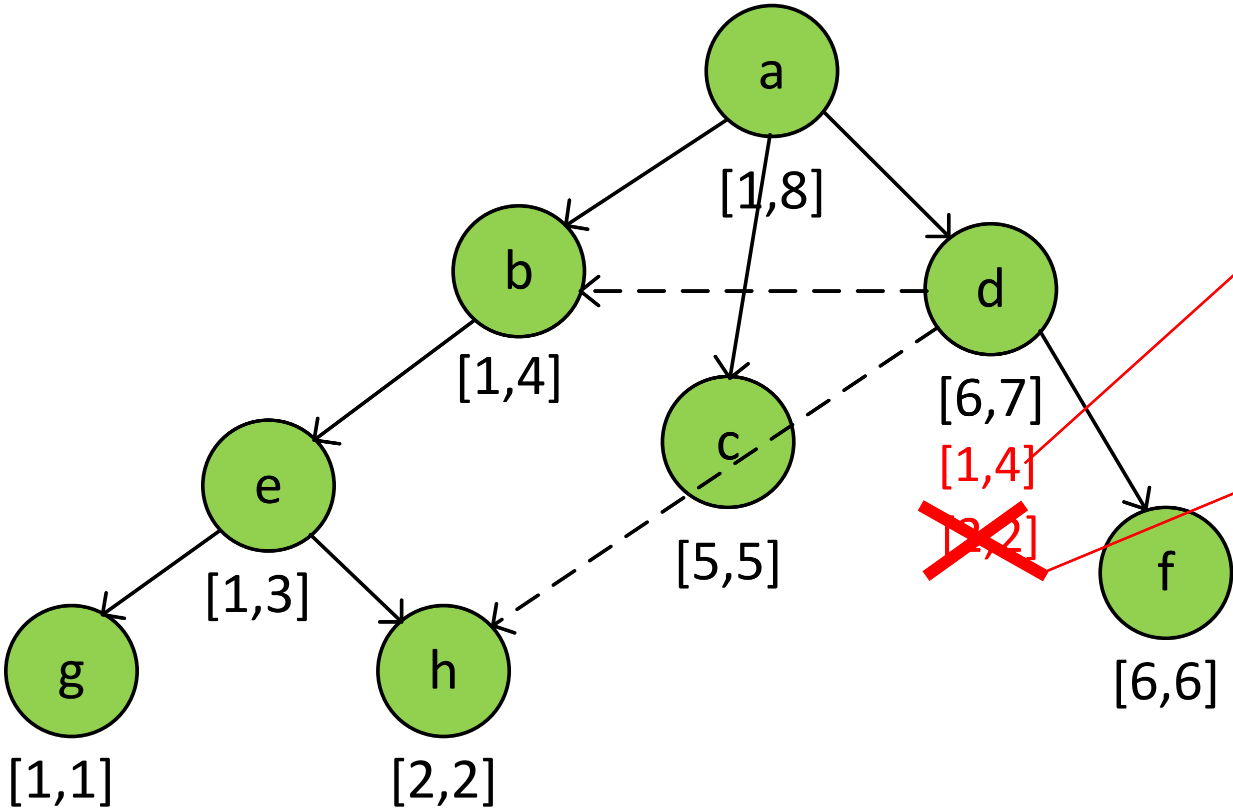
$?(b \rightsquigarrow c) \Longrightarrow ?(1 \leq 5 < 4) \Rightarrow NO$

$$label(u) = [u_{start}, u_{end}]$$

Query Processing: $?(u \rightsquigarrow v) \Longrightarrow$

$$?(u_{start} \leq v_{end} < u_{end})$$

Optimal Tree Cover



- *Inherited* from b
- [2,2]$\subseteq$ [1,4]

Optimal Tree Cover

Topological sorting: parent comes before the child

- Index Construction: O(nm) time = O (n^3)
  1. Connect all nodes with no predecessors to a dummy node-root.
  2. Find a spanning tree for the DAG (+root) – requires a topological sorting first
  3. Label nodes according to tree edges (postorder).
  4. For each non-tree edge (u,v) – in reverse topological order of the nodes- :

     $$label(v) = label(v) \cup label(u)$$

  5. If $label(a) \subset label(b)$ for labels inherited from nodes a, b, then keep only $label(b)$.

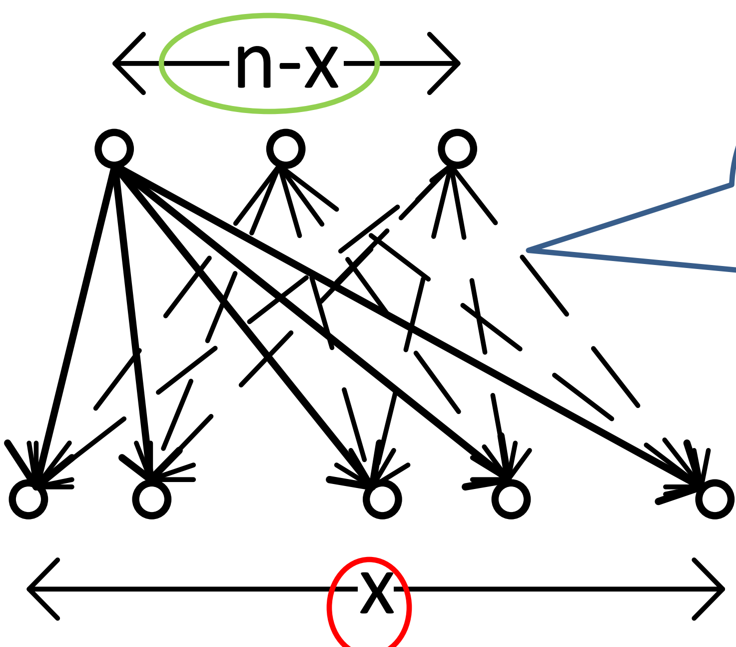
Optimal Tree Cover

- **Query processing:** O(n) time

$$label(u) = \{[u_{start}, u_{end}], [u_{start_1}, u_{end_1}] \dots\}$$

$$(u \rightsquigarrow v) \text{ iff } \exists i: (u_{start_i} \leq v_{end} < u_{end_i})$$

- **Space:** O(n^2) worst case



Bipartite Graph:
every one of these (n-x) vertices will inherit all x labels, resulting in total (n-x)(x+1) labels, yielding O(n^2).

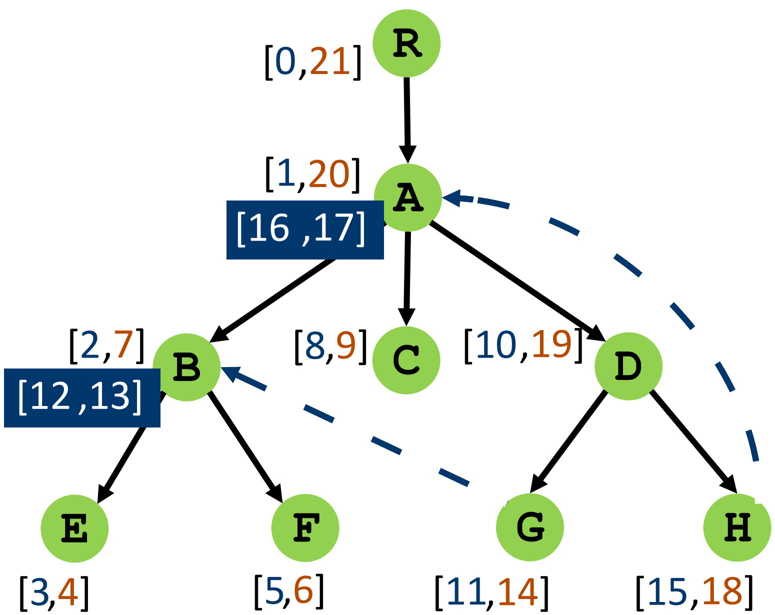
Optimal Tree Cover

- The authors propose an algorithm for finding the optimal spanning tree (in terms of total label size).
- They also suggest a maintenance mechanism with non-consecutive numbering of nodes.
- Although asymptotically equivalent to the straightforward method of transitive closure, in the experiments the method performs better in orders of magnitude.
- The experiments are quite primitive (graphs with 1000 nodes)

GRIPP Index Creation

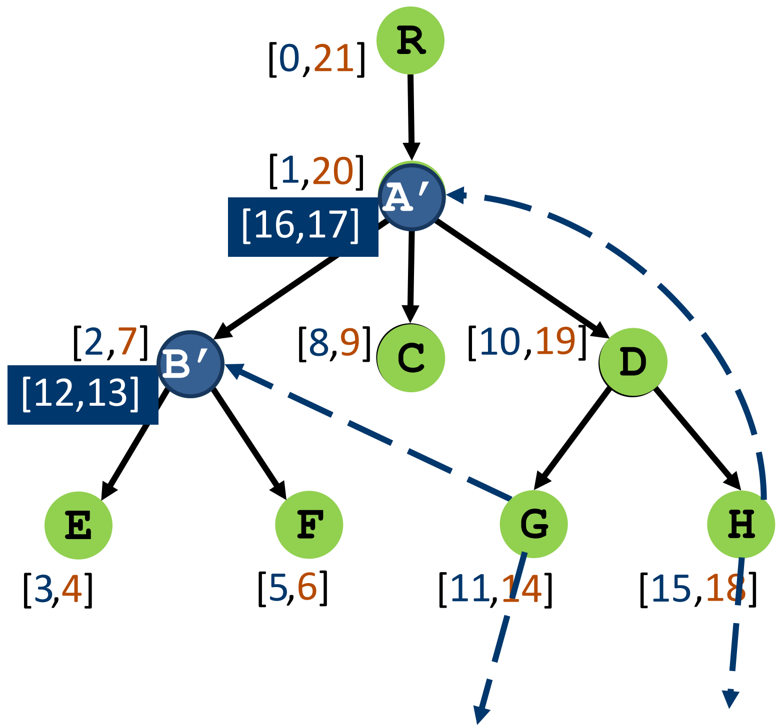
SIGMOD '07

- Depth-first traversal of *G*



- We reach a node *v*
  - for the first time
    - add *tree instance* of *v* to *IND(G)*
    - proceed traversal
  - again
    - add *non-tree instance* of *v to IND(G)*
    - do not traverse child nodes of *v*

GRIPP Index Table, *IND(G)*



Graph, G

node	pre	post	inst
R	0	21	tree
A	1	20	tree
B	2	7	tree
E	3	4	tree
F	5	6	tree
C	8	9	tree
D	10	19	tree
G	11	14	tree
B`	12	13	Non-tree
H	15	18	tree
A`	16	17	Non-tree

GRIPP index, *IND(G)*

- Is node C reachable from node D?

GRIPP Query answering

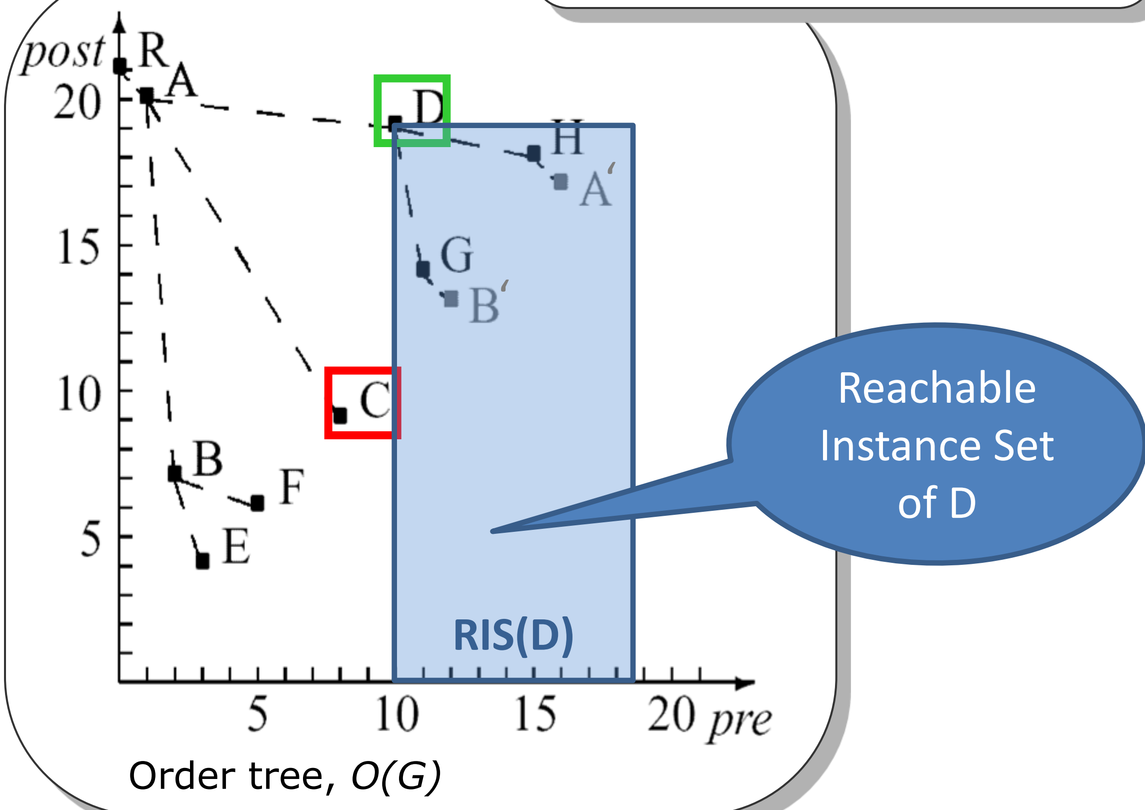
?(D ⤳ C)

node	pre	post	inst
R	0	21	tree
A	1	20	tree
B	2	7	tree
E	3	4	tree
F	5	6	tree
C	8	9	tree
D	10	19	tree
G	11	14	tree
B`	12	13	non
H	15	18	tree
A`	16	17	non

RIS(D)

If $D_{pre} < C_{pre} < D_{post}$

C reachable from *D*



GRIPP Query answering

?(D ⇝ C)

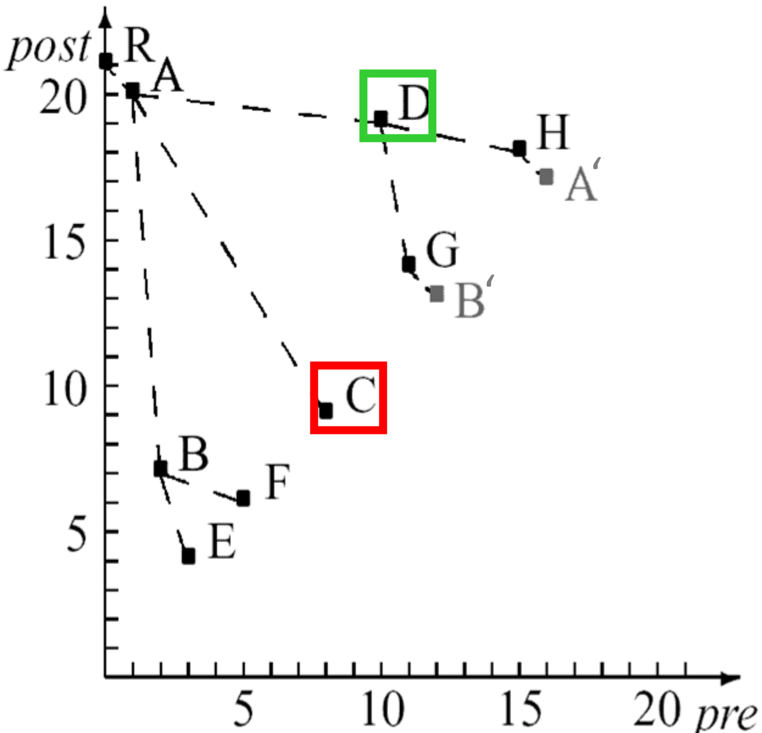
Step 1: Retrieve RIS (D).

If $C \in RIS(D)$ **then** answer YES and finish.

Step 2: **Else**

foreach non_tree entry h' $\in RIS(D)$ do:

recursively issue the query *?(h ⇝ C)*



Gripp [Facts]

- Index Construction Time: Depth First Traversal, linear **O(m+n)**
- Storage: O(n) nodes of the graph + O(m-n) non-tree nodes yields **O(m+n)** storage
- Query Time: in the worst case we ask for **O(m-n)** recursive calls.

The authors use some pruning techniques and heuristics and claim that their algorithm has **almost constant** query time for various types of graphs.

The order of the traversal is crucial. The same for the order of the recursive hops.

Dual Labeling [assumptions]

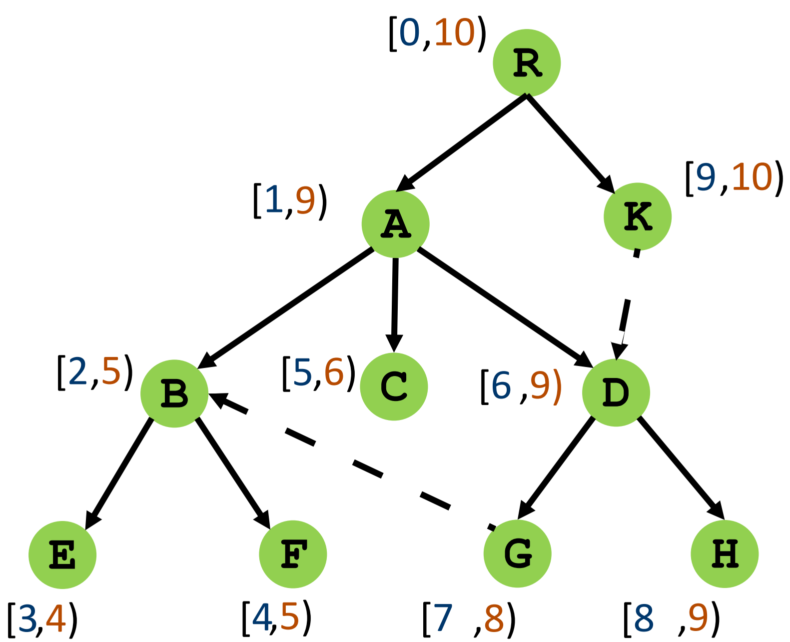
- Basic assumption: most practical graphs are sparse.
  - Examples of biological data and XML documents
  - Average degree *~1.2 (edges/node)*
- The authors will use this fact to build nearly optimal algorithms for tree-like sparse graphs
- Thus, they assume that $t<<n$.

number of non-tree edges

Dual Labeling [idea]

ICDE '06

1. DFT to Compute a spanning tree and the Transitive Link Table (for the non-tree edges)
2. Compute the transitive link closure.



TRANSITIVE LINK TABLE

7->[2,5)

9->[6,9)

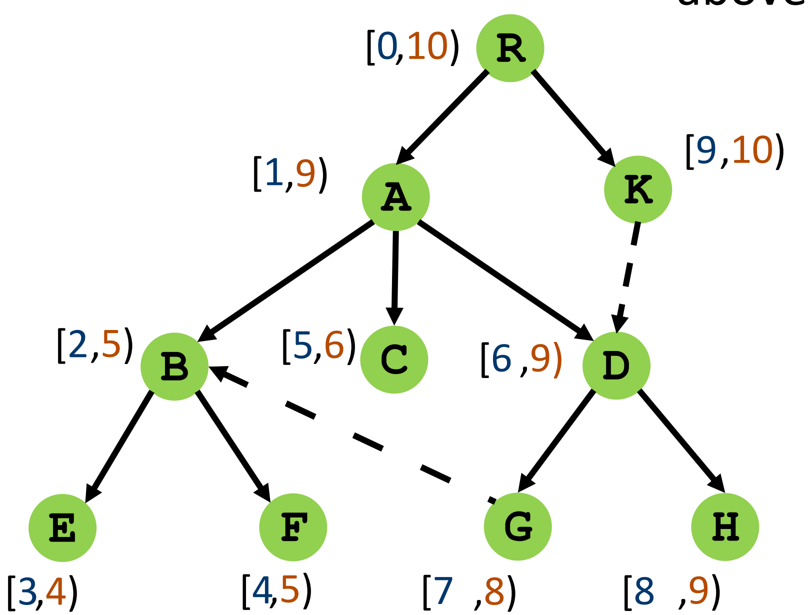
9->[2,5)

Dual Labeling

- Now we can answer queries by checking the tree labels + the transitive link table(TLT)
  - examples: ?A $\rightsquigarrow$ *G* *YES:* *7* $\in$ *[1,9)*

    ?D $\rightsquigarrow$ *F* *YES:* Although *4* $\notin$ *[6,9)* if we search TLT we find edge *7->[2,5)* for which *7* $\in$ *[6,9)* and *4* $\in$ *[2,5)*

    ?D $\rightsquigarrow$ *C* *NO:* *5* $\notin$ *[6,9)* and there is no entry in TLT with the above property



TRANSITIVE LINK TABLE

7->[2,5)

9->[6,9)

9->[2,5)

Dual Labeling

- The size of TLT is $O(t^2)$ since it contains the transitive closure of t non-tree edges.
- Given the above indexing scheme, query time might take $O(t^2)$ for the linear search of TLT

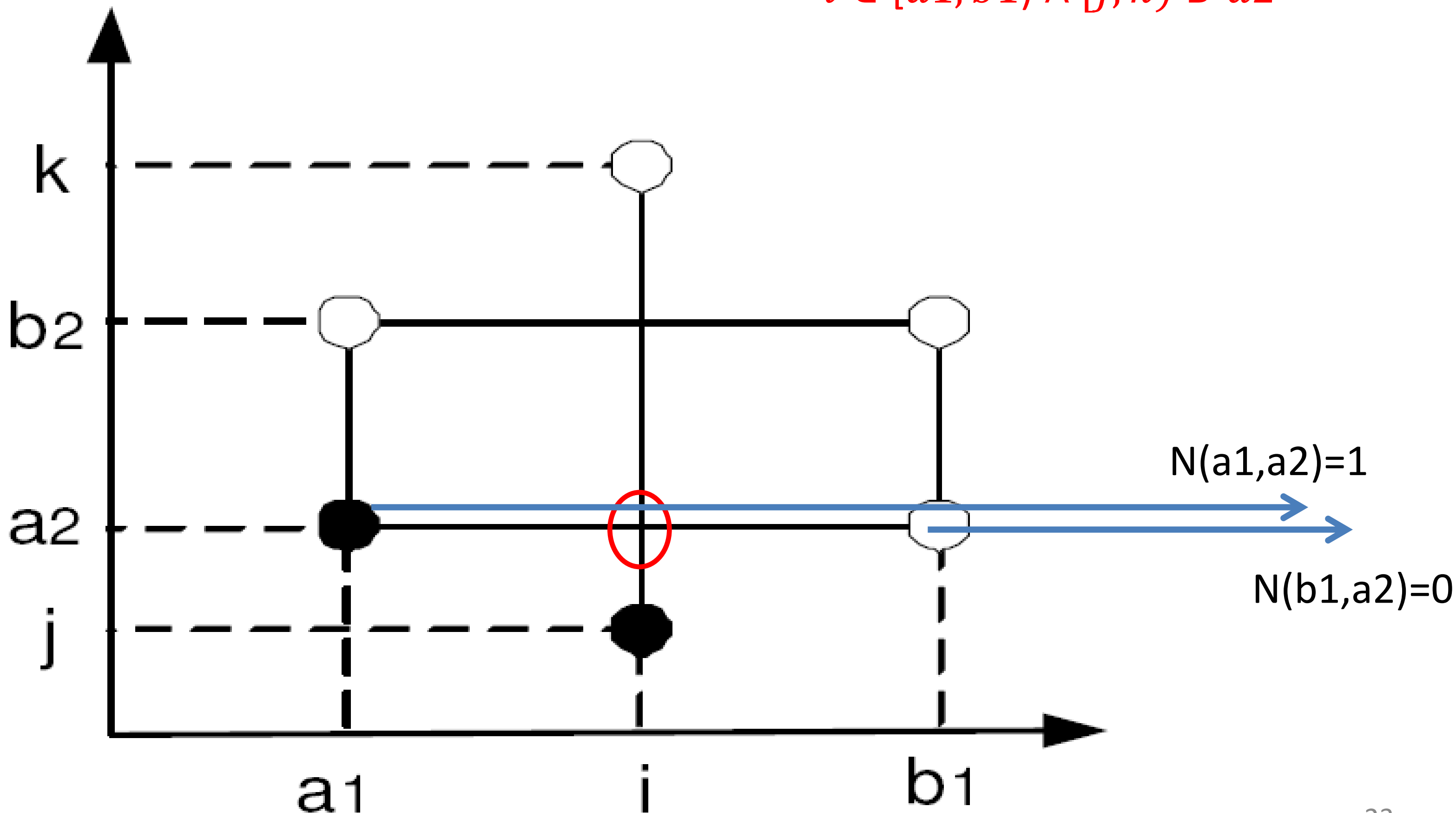
- The goal is to reduce query time to O(1).
- We woudn’t mind to put them in a table (since t is small) in order to reduce the query time in O(1) but we cannot!
  - TLT consists of entries of the form: $i \rightarrow [x, y)$
  - We would need 3D table
- The authors propose one solution

Dual Labeling

$?u \rightsquigarrow v$
$u = [a1, b1)$
$v = [a2, b2)$

$a2 \notin [a1, b1)$ so unreachable from tree only edges
What is the property of an entry $i \rightarrow [j, k)$ in TLT?

$i \in [a1, b1) \wedge [j, k) \ni a2$



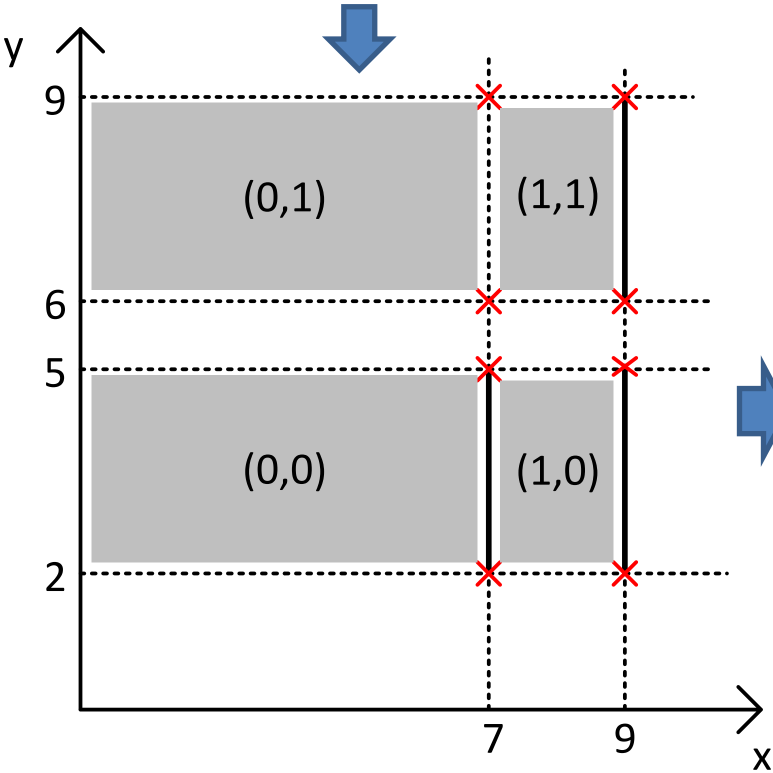
Dual Labeling

TRANSITIVE LINK TABLE

7->[2,5)

9->[6,9)

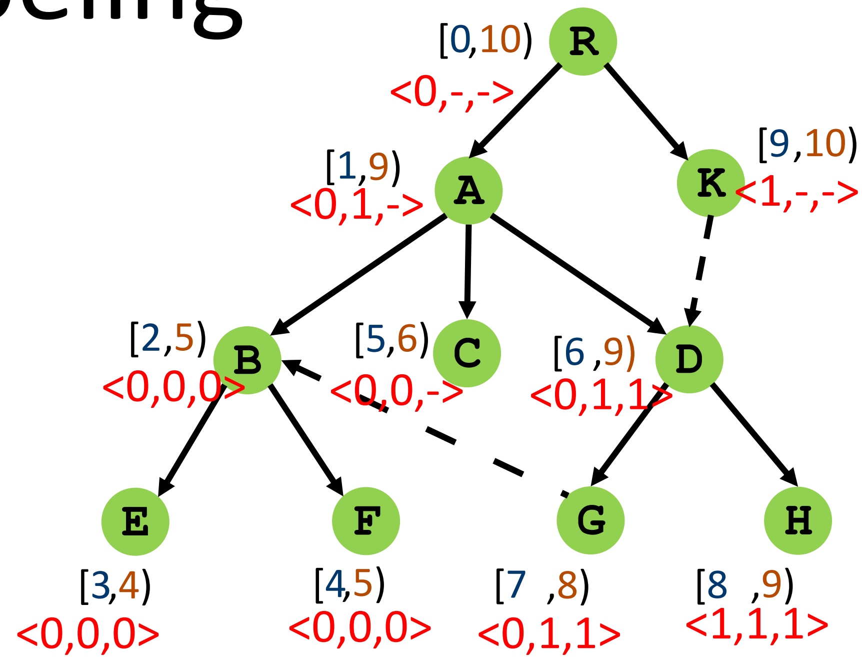
9->[2,5)



TLT Grid

1	1	1
0	2	1
y \ x	**0**	**1**

TLT Matrix



- $x = index_x(a')$, where $a' = \min\{i \mid i \to [j,k) \in T \wedge i \geq a)\}$. If such an a' does not exist, let x be the special symbol "$-$."
- $y = index_x(b')$, where $b' = \min\{i \mid i \to [j,k) \in T \wedge i \geq b)\}$. If such a b' does not exist, let y be "$-$."
- $z = index_y(a^*)$, where a^* is the start interval label of the lowest (tree) ancestor of u with a non-tree incoming edge. If such an a^* does not exist, let z be "$-$."

Dual Labeling

Now reachability can be defined in constant time by the values of the two labels.

u: $([a1, b1), < x1, y1, z1 >)$

v: $([a2, b2), < x2, y2, z2 >)$

$u \rightsquigarrow v \Leftrightarrow$

- $a2 \in [a1, b1)\ or$
- $\boldsymbol{N}[x1, z2] - \boldsymbol{N}[y1 - z2] > 0$

1	1	1
0	2	1
y╲x	**0**	**1**

TLT Matrix

e.g.:

$?\, K \rightsquigarrow E$

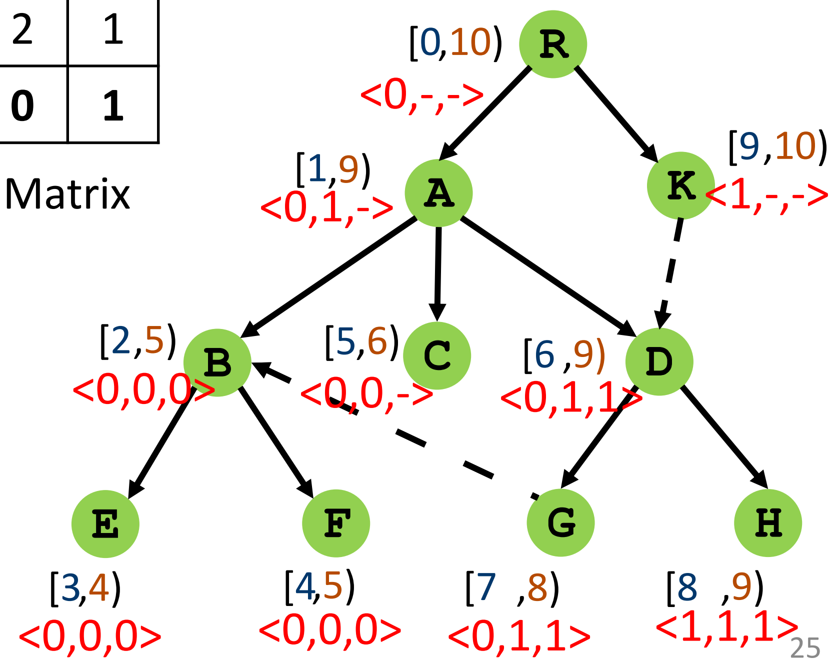
- $3 \notin [9,10)$
- $N[1,0] - N[-, 0] = 1 - 0 > 0$

So the answer is YES.

$?\, H \rightsquigarrow B$

- $2 \notin [8,9)$
- $N[1,0] - N[1,0] = 1 - 1 \ngtr 0$

So the answer is NO.



Dual Labeling [sum up]

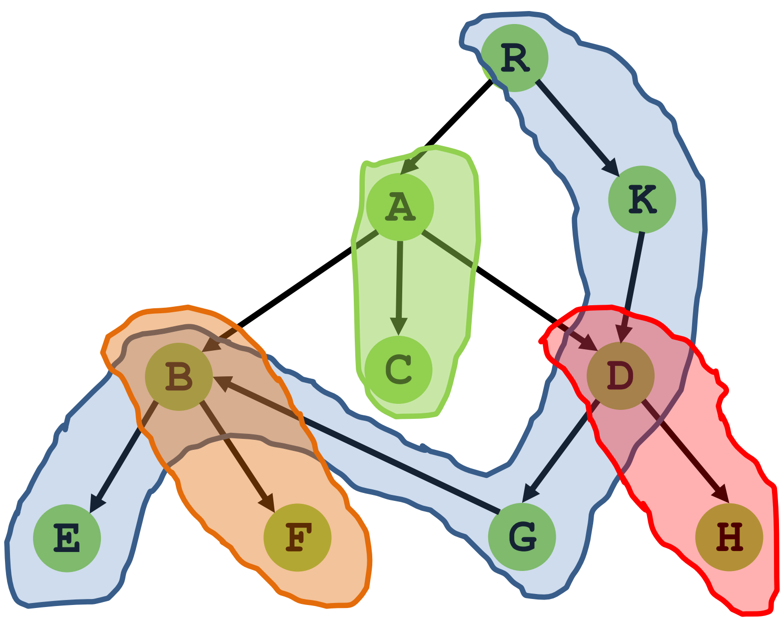
- Index Construction Time: Depth First Traversal, linear O(m+n) **+** transitive link closure construction O(t^3) yields **O(m+n+t^3)** ≈ O(m+n) for *t<<n.*
- Storage: O(n) nodes of the graph O(t^2) for the TLT matrix yields **O(n+ t^2)** storage
- Query Time: **O(1)**

Of course if *t* is comparable to *n* (there are a lot of non-tree edges) then Dual Labeling performs as bad as the naïve approach of the Transitive closure of the graph.

Chain Cover

ACM Trans. Database Syst. ‘90

- Enough with the spanning trees!
- Let’s partition the graph into chains



R ↝K ↝G ↝E

Chain 0: R ↝K ↝~~D~~ ↝G ↝~~B~~ ↝E

Chain 1: D ↝H

Chain 2: A ↝C

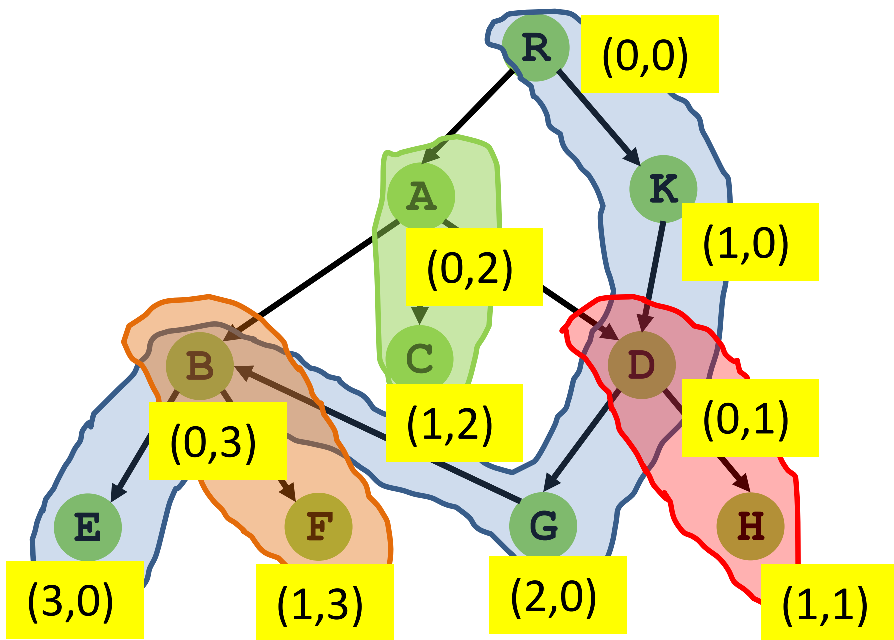
Chain 3: B ↝F

Chain Cover

- A chain cover of G is a partition of G into disjoint sets called chains.
- Let G=(V,E) a directed graph and $c_i \subseteq V$ s.t. if $\mathrm{u}, \mathrm{v} \in c_i$ then $\mathrm{u} \leadsto \mathrm{v}$. Now let C={$c_1$,$c_2$,…,$c_k$} the set containing such sets.

  If $\forall u \in V\ \exists i{:}\, u \in c_i$ and $\forall i \neq j, c_i \cap c_j = \emptyset$ then C is a chain cover of G.

Chain Cover [index]



The idea behind the chain cover is again to produce a compressed transitive closure of the graph based on the chains.

e.g
?K $\leadsto$ F

- Find label of F: (1,3)
- Find K's entry for chain no 3 : (0,3)
- 0≤1 so answer is YES

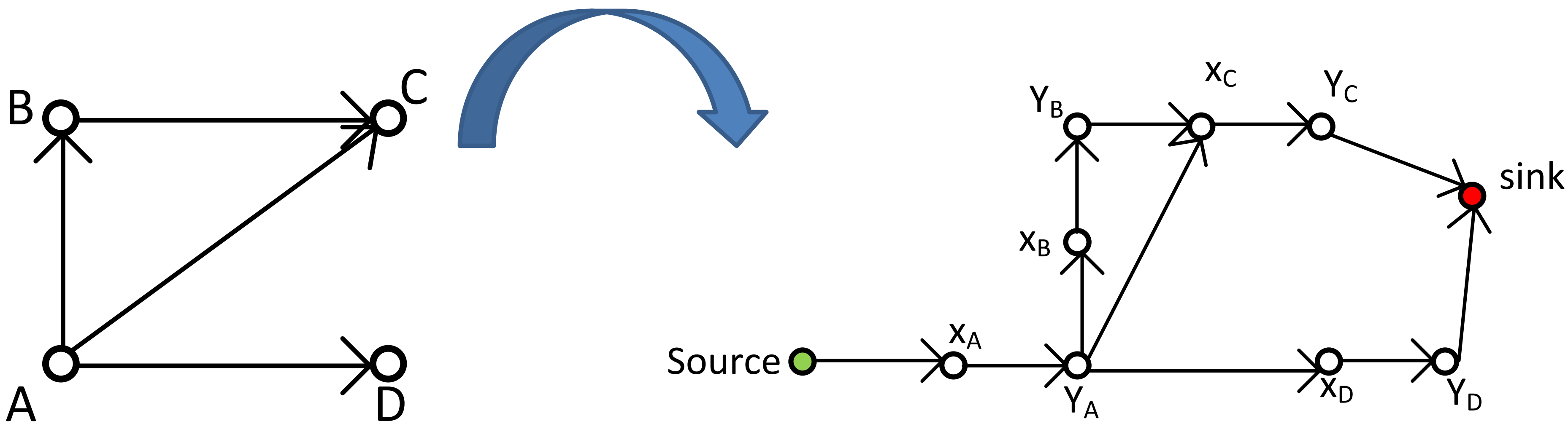
	R	A	B	C	D	E	F	G	H	K
c_0	(1,0)	(2,0)	(3,0)	-	(2,0)	-	-	(3,0)	-	(2,0)
c_1	(0,1)	(0,1)	-	-	(1,1)	-	-	-	-	(0,1)
c_2	(0,2)	(2,1)	-	-	-	-	-	-	-	-
c_3	(0,3)	(0,3)	(1,3)	-	(0,3)	-	-	(0,3)	-	(0,3)

Chain Cover

- The efficiency depends heavily in the initial chain covering (not unique of course).
- The smaller the number of chains the better.
- Optimal chain cover can be found in polynomial time.
- How? Transform the problem to a *min-flow* problem.

Find optimal Chain Cover

- We'll use a simpler graph for illustration:



Graph G

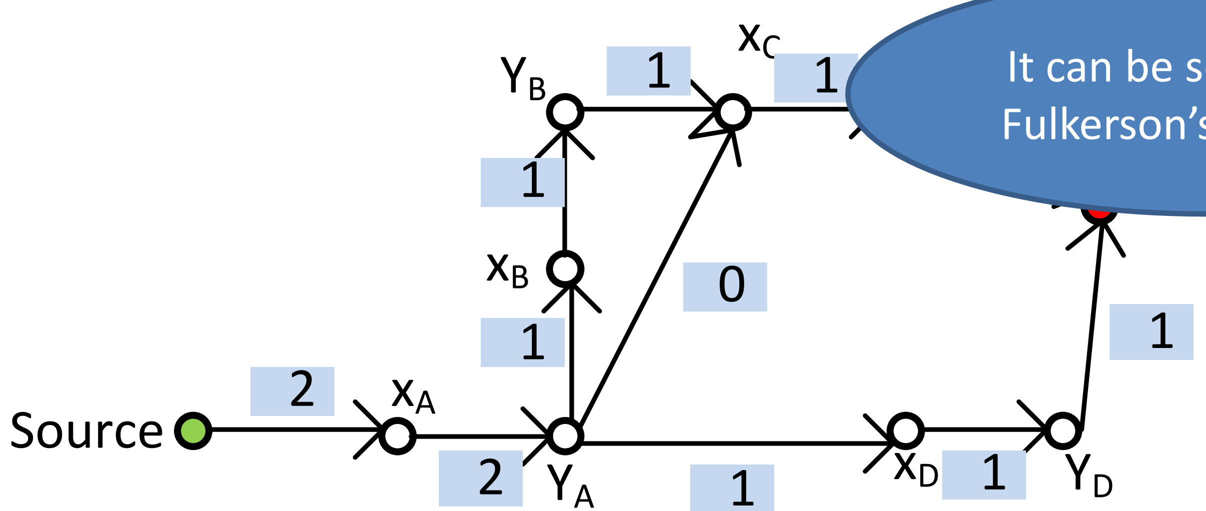
Transformed Graph G

Min-Flow problem

- Solve the min flow problem for the transformed graph under the constraints:
  - flow $(x_i y_i) > 0 \; \forall i$
  - flow $(x_i y_j) \geq 0 \; \forall i \neq j$
  - no flow accumulation in the nodes

The problem can be formulated by LP(Linear Program)

It can be solved Ford Fulkerson's algorithm



Chain Cover [facts]

- Index Construction Time: It takes $O(n^3)$ to compute the transitive closure and find the min chain cover
  - there are faster (and a lot more complicated) methods that use bipartite maching and drop the complexity to $O(n^2 + kn\sqrt{k})$

k is the number of the chains

- Storage: $O(nk)$ [worst case: k=O(n) so there is no real compression]
- Query Time: O(1) if *k* is small enough to store the index in a 2D table. If not storing it into lists and indexing the lists yields O(logn+ k) query time

Path-Tree Cover

SIGMOD '08

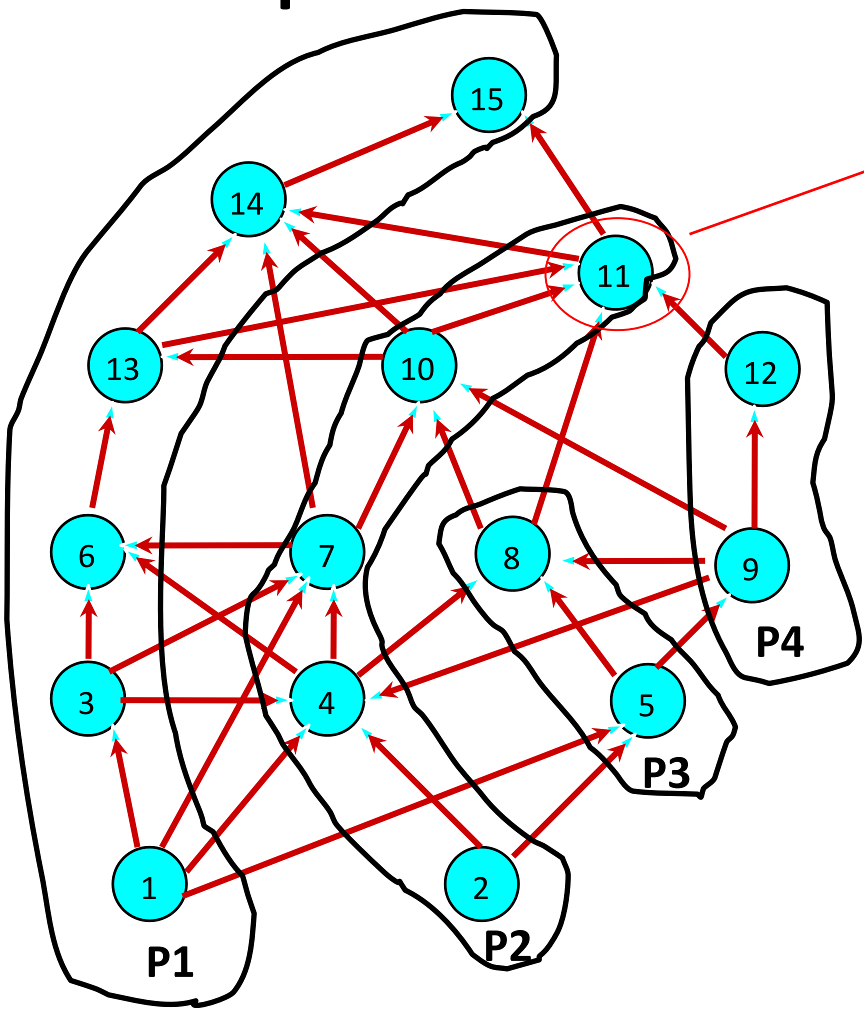
- In the tree covering approaches, we tried to:
    1. build a spanning tree T,
    2. give some labels w.r.t. T and then
    3. find a solution for the extra reachability induced by the non-tree edges

- In this paper the authors pay special attention to the first of the above steps.
- They generalize the notion of spanning tree to spanning graph
- They try to compute the best spanning graph in order to reduce the complexity of the third step (the non-spanning edges)

Constructing Path-Tree

- Step 1: Path-Decomposition of DAG
- Step 2: Minimal Equivalent Edge Set between any two paths
- Step 3: Path-Graph Construction
- Step 4: Path-Tree Cover Extraction

Step 1: Path-Decomposition



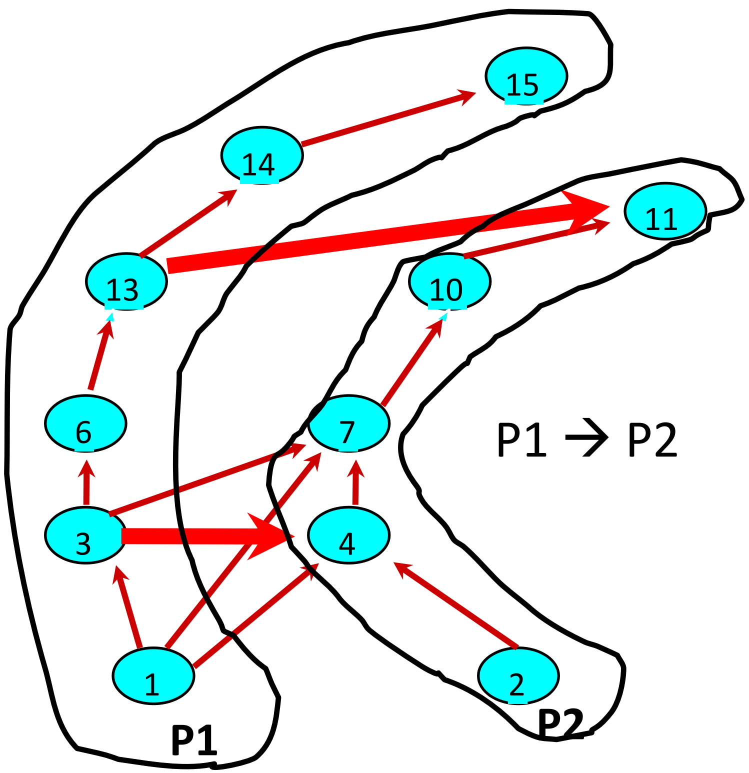
(PID,SID)
=(2, 5)

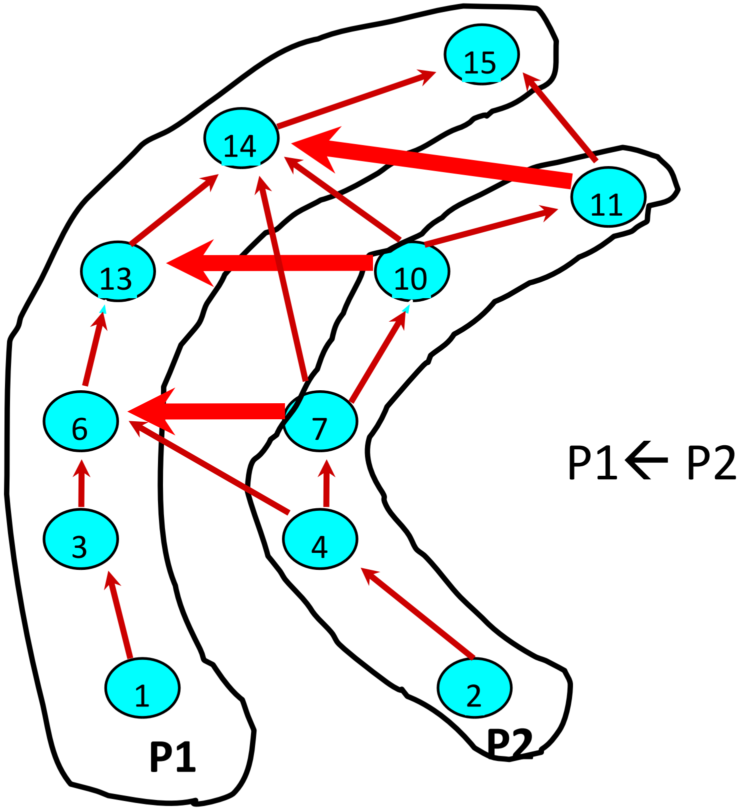
For any two nodes (u, v)
in the same path,
u → v if and only if (u.sid ≤ v.sid)

Simple linear algorithm based on topological sort can achieve a path-decomposition

Step 2: Minimal equivalent edge set

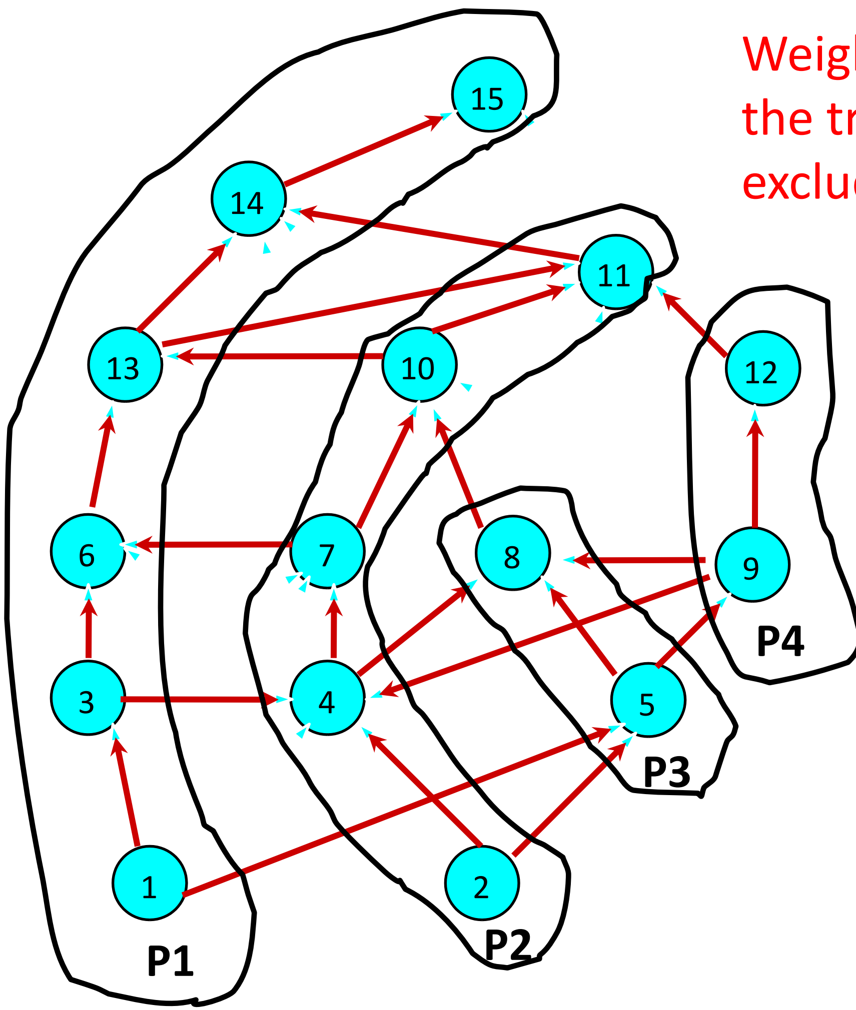
The reachability between any two paths can be captured by a unique minimal set of edges



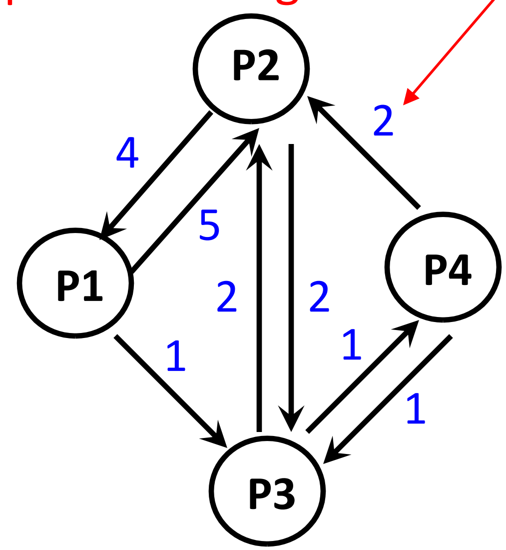


The edges in the minimal equivalent edge set do not cross (always parallel)!

Step 3: Path-Graph Construction

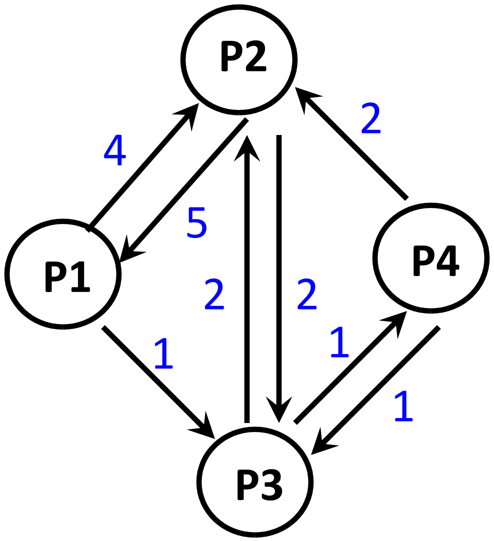


Weight reflects the cost we have to pay for the transitive closure computation if we exclude this path-tree edge

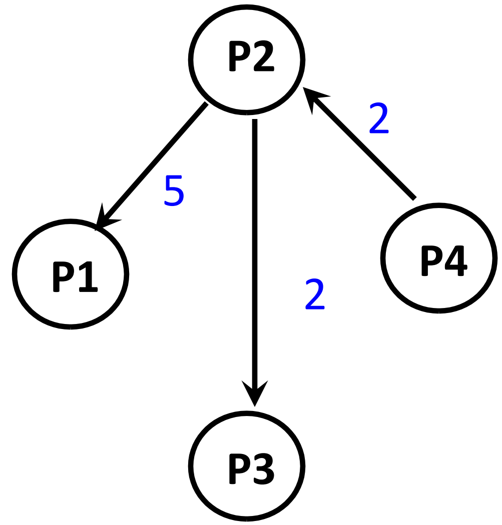


Weighted Directed Path-Graph

Step 4: Extracting Path-Tree Cover



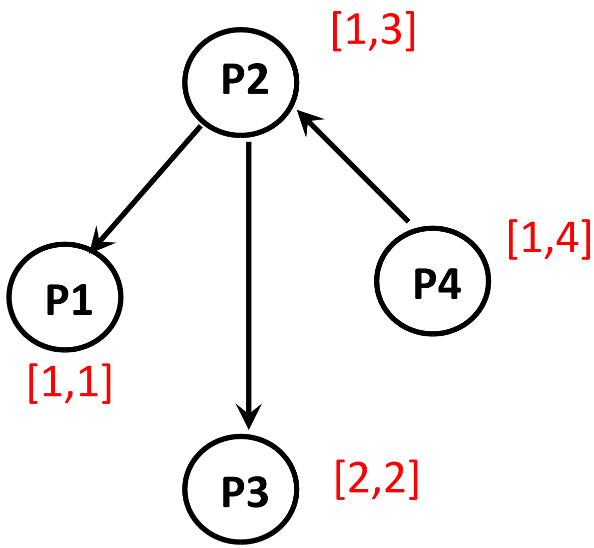
Weighted Directed Path-Graph



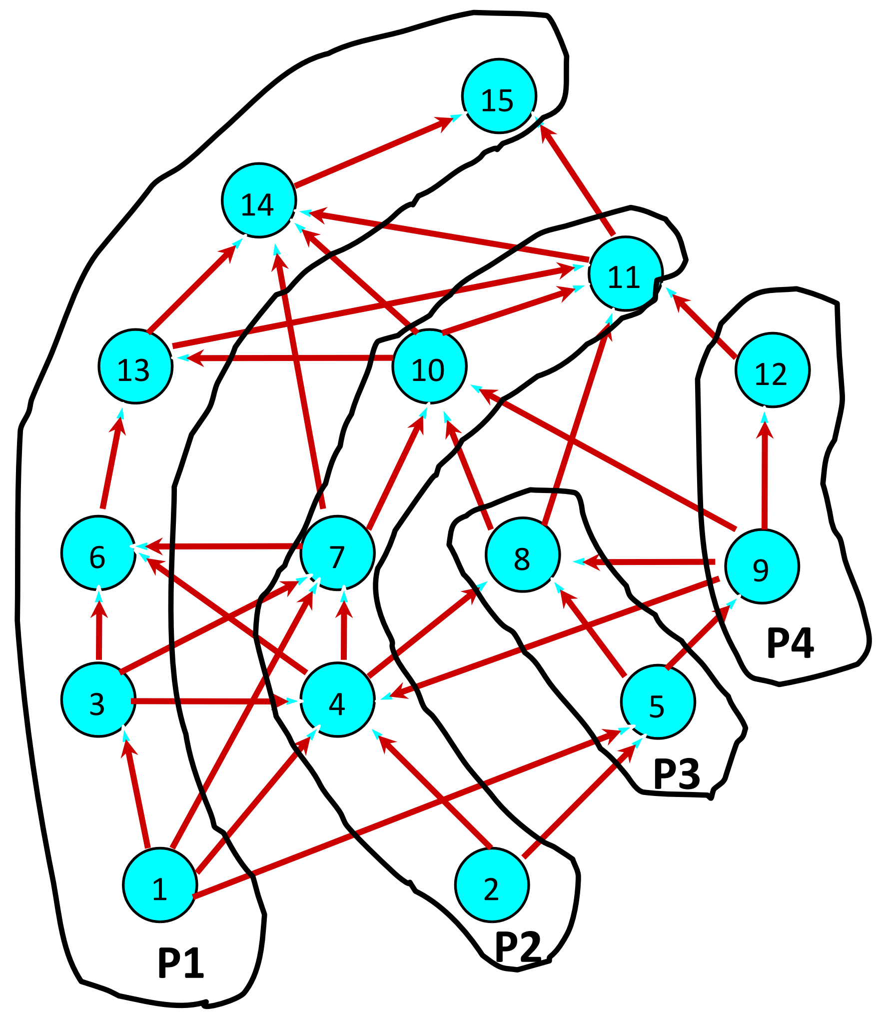
Maximal Directed Spanning Tree

Chu-Liu/Edmonds algorithm, *O(m'+ k logk)*

3-Tuple Labeling for Reachability

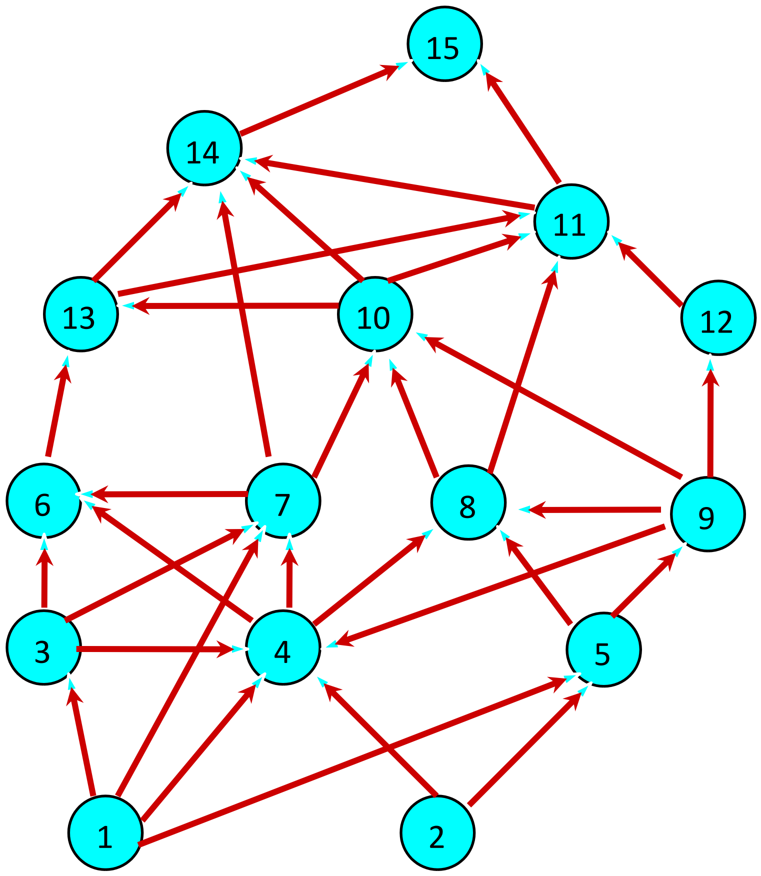


Interval labeling (2-tuple)
High-level description about paths
Pi → Pj ?



DFS labeling (1-tuple)

Transitive Closure Compression



Path-tree cover (including labeling) can be constructed in *O(m + n logn)*

An efficient procedure can compute and compress the transitive closure in *O(mk)*, *k* is number of paths in path-tree

Path-tree cover [and then?]

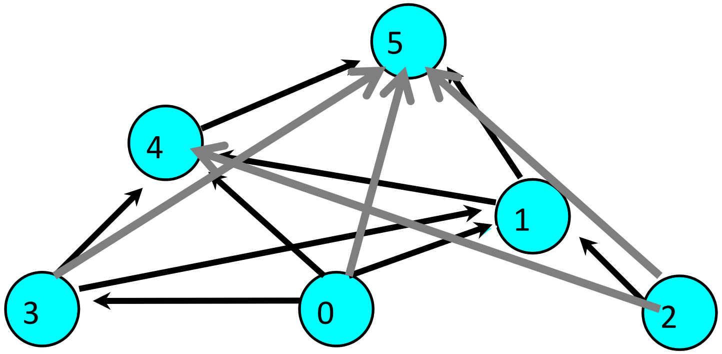
- After building this complex index structure, they use the techniques discussed above (GRIPP, Dual Labeling etc.).

2 HOP-Cover [idea]

SODA '02

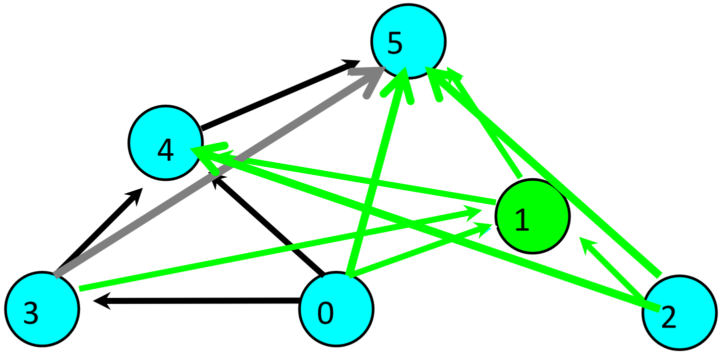
- Transitive closure compression (again…)

- If I choose node 1 as a center node, then I know that all 1's ancestors can reach 1's descendants.
- So by choosing node 1, all the green edges are covered.
- The goal is to choose nodes so as to cover all edges in TC(G)



2 HOP-Cover [idea]
SODA '02

- Based on that, we can label nodes as follows:
  - each node u will have a label L(u)
  - $L(u) = \{L_{in}(u), L_{out}(u)\}$
  - $L_{in}(u), L_{out}(u) \subseteq V$
- After choosing node 1, we add it at
  - $L_{out}(2) = \{1\},\ L_{out}(3) = \{1\},$ $L_{out}(0) = \{1\}$
  - $L_{in}(4) = \{1\}, L_{in}(5) = \{1\}$
  - $L_{in}(1) = \{1\},\ L_{out}(1) = \{1\}$
- After covering all edges of TC(G), nodes are labeled



	0	1	2	3	4	5
L_{in}		{1}		{0}	{1,4}	{1,4}
L_{out}	{1,4,0}	{1}	{1}	{1,4}	{4}	

2 HOP-Cover [idea]

SODA '02

Compression for each center node w

$$TC_{size} = Ans_w \cdot Desc_w$$

2-HOP = $Ans_w + Desc_w$

- Now reachability queries can be answered using the labels:
  - $?\, u \rightsquigarrow v$

    $$L_{out}(u) \cap L_{in}(v) \neq \emptyset$$

e.g.

$?\, 0 \rightsquigarrow 5$

$$L_{out}(0) \cap L_{in}(5) = \{1,4,0\} \cap \{1,4\} \neq \emptyset$$

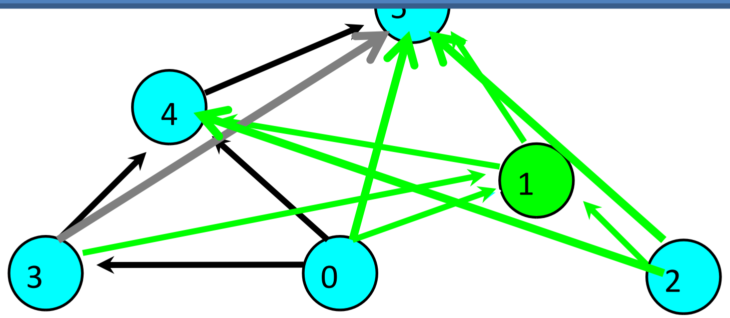
YES

$?\, 4 \rightsquigarrow 1$

$$L_{out}(4) \cap L_{in}(1) = \{4\} \cap \{1\} = \emptyset$$

NO

Table with redundancy



	0	1	2	3	4	5
L_{in}		{1}		{0}	{1,4}	{1,4}
L_{out}	{1,~~4~~,0}	{1}	{1}	{1,4}	{4}	

2 HOP-Cover

Compression for each center node w

$$TC_{size} = Ans_w \cdot Desc_w$$

2-HOP = $Ans_w + Desc_w$

- Problem: How do you find a minimum 2-Hop Cover in the graph?
  - exact solution is NP hard
- Approximate Solution: Greedily pick the node with the highest compression as a center node
  - log n approximation ratio
  - but for each node we should compute all the subsets of ancestors and descendants to see which yields the highest compression
    - exponential combinations: another 2-approximation algorithm to find the approximate highest compression node (algorithm based on bipartite matching- works in linear time)

2-HOP cover

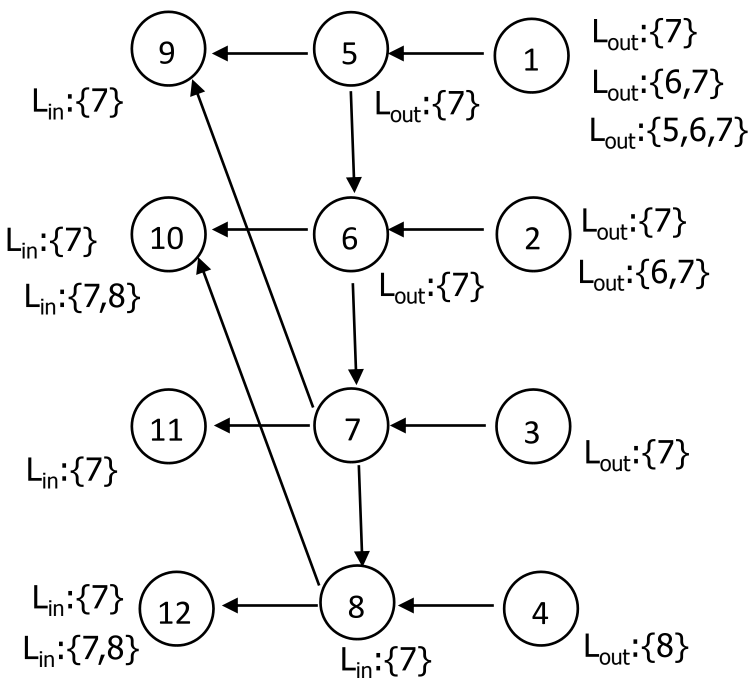
- Problem: How do you find a minimum 2-Hop Cover in the graph?
  - exact solution is NP hard
- There have been also other methods:
  - Heuristics
  - Geometrical Approach
  - Graph partitioning methods

2-HOP cover

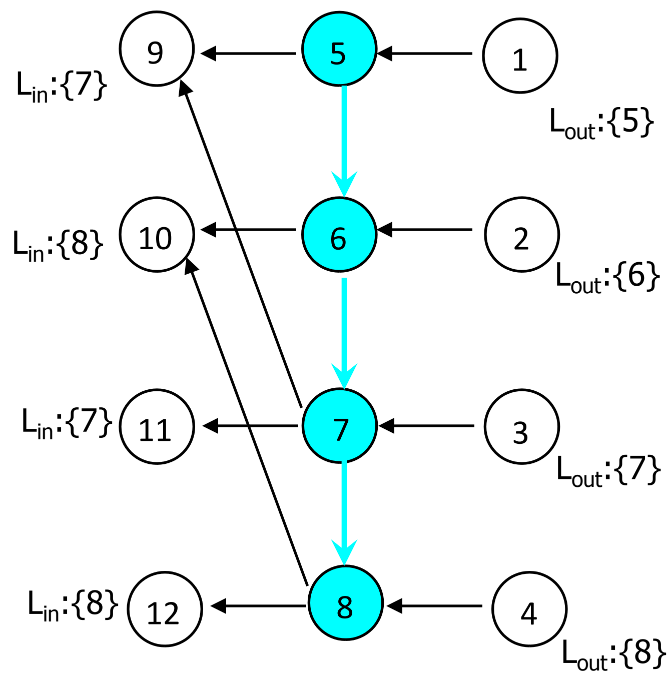
- Index Construction Time: complicated and costly O($\boldsymbol{n^3} \cdot |\boldsymbol{TC}|$)=O($n^5$)
- Storage: O($\boldsymbol{n\sqrt{m}}$)
- Query Time: O($\boldsymbol{\sqrt{m}}$).

3 HOP-Cover [intuition]

SIGMOD ‘09



2-Hop



3-Hop

3-HOP Cover [overview]

- Vertex→Vertex→Vertex (2 hop)
- Vertex→Chain(Vertex→Vertex)→Vertex (Initial motivation of 3-HOP)
- Chain(Vertex→Vertex) → Chain(Vertex→Vertex) → Chain(Vertex→Vertex) (3 hop contour)
- Chain decomposition is a spanning structure of G
- Some special vertices in the graph are labeled by L_{out} (a subset of vertices it can reach) and/or L_{in} (a subset of vertices it can be reached from).
- Chain decomposition plus the set of L_{out} and L_{in} are all that we need to design efficient reachability answering schemes.

Conclusion

	Method	Query time	Construction	Index size
Naïve	DFS/BFS	O(n+m)	O(n+m)	O(n+m)
	Transitive Closure	O(1)	$O(nm)=O(n^3)$	$O(n^2)$
Tree Cover	Optimal Tree Cover (Agrawal et al., SIGMOD'89)	O(n)	$O(nm)=O(n^3)$	$O(n^2)$
	GRIPP (Trißl et al., SIGMOD'07)	O(m-n)	O(n+m)	O(n+m)
	Dual-Labeling (Wang et al., ICDE'06)	O(1)	$O(n+m+t^3)$	$O(n+t^2)$
Chain Cover	Optimal Chain Cover (Jagadish, TODS'90)	O(k)	O(nm)	O(nk)
	Path-Tree (Jin, et al., SIGMOD'08)	$\log^2 k'$	O(mk')/O(mn)	O(nk')
HOP Cover	2-HOP (SODA 2002)	$O(nm^{1/2})$	$O(n^3\|T_C\|)=O(n^5)$	$O(m^{1/2})$
	3-HOP (Yang Xiang et al., SIGMOD '09)	O(log n +k)	$O(kn^2)$	O(nk)

THANK YOU

APPENDIX

Optimal Tree Cover[results]

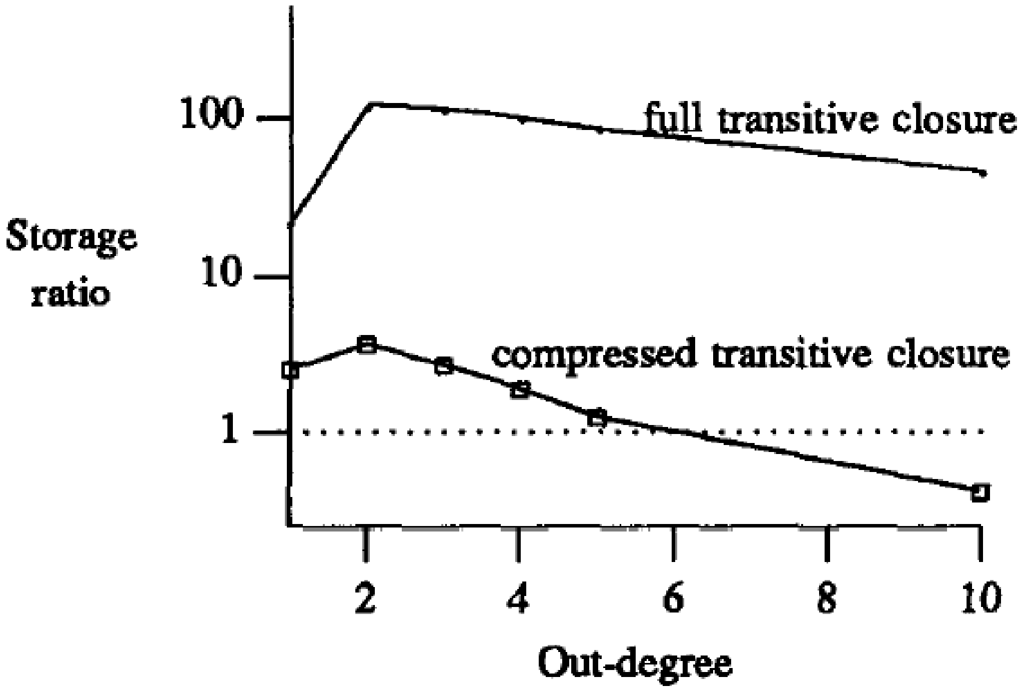
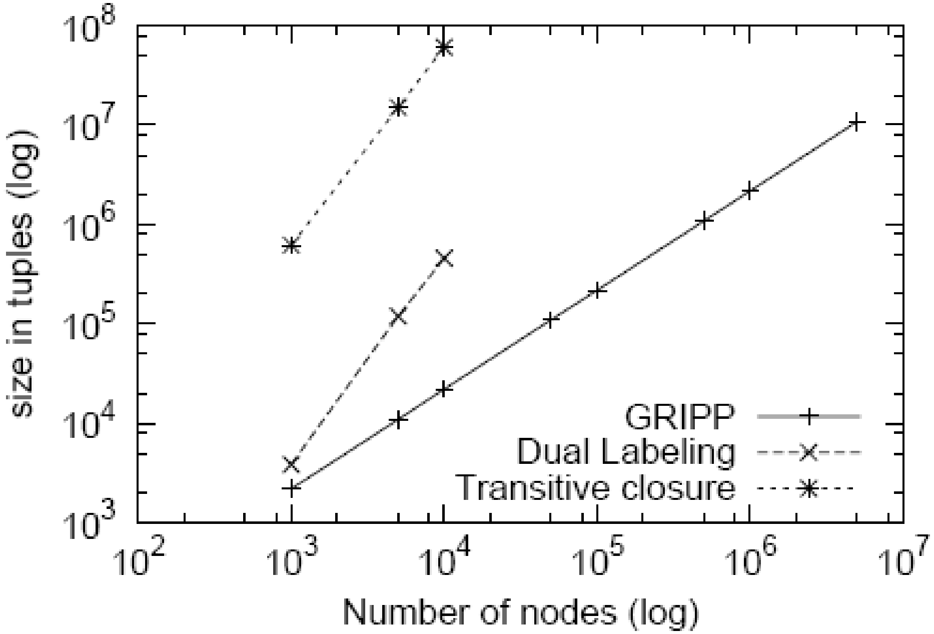
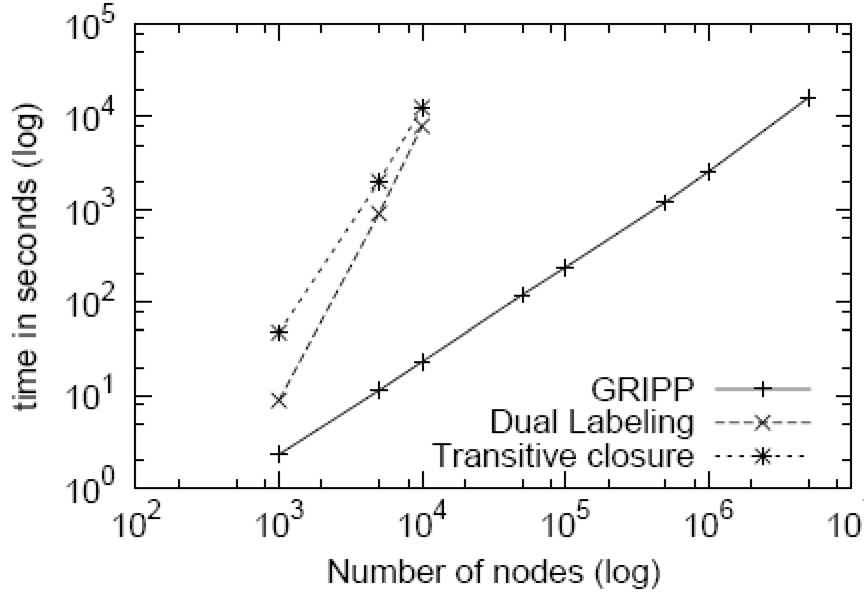


Figure 3.9. Storage required for a 1000 node graph as a function of average degree

GRIPP[results]

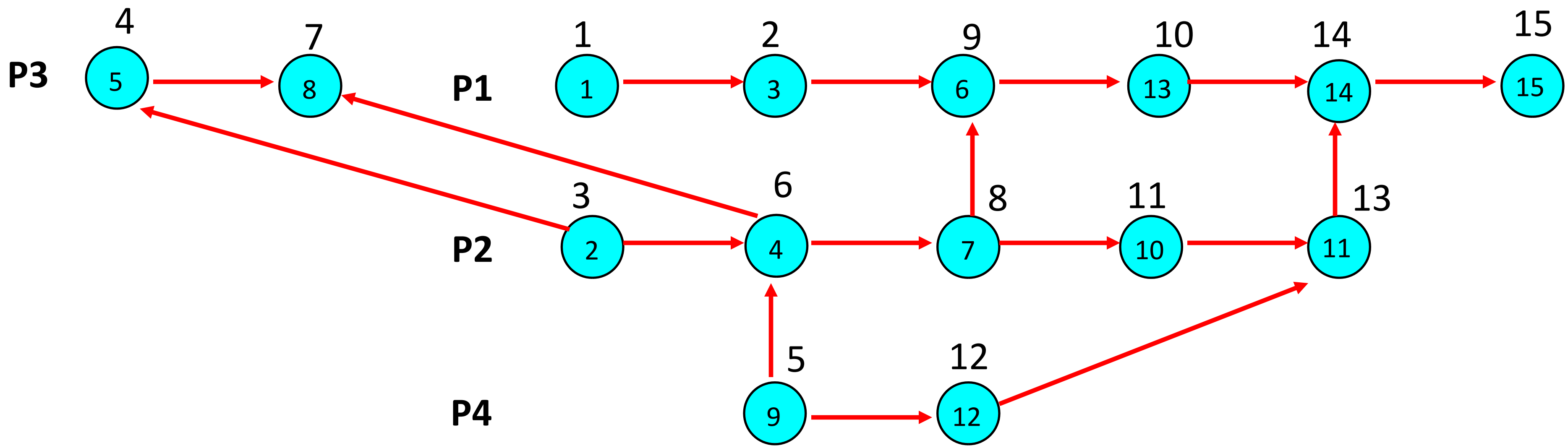


(a) Average size (tuples)



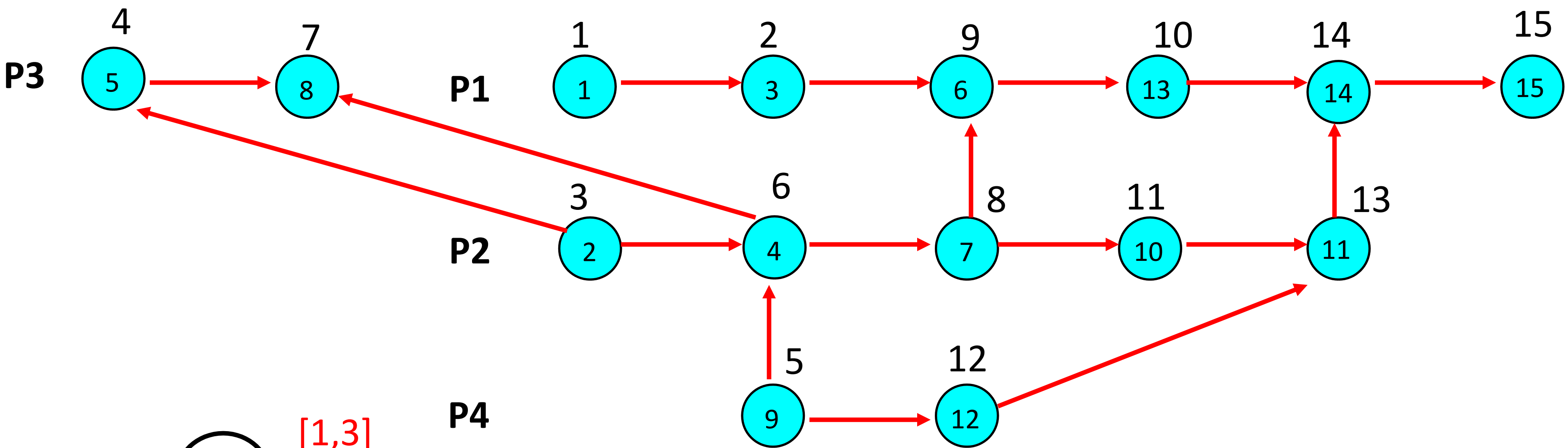
(b) Average time (sec)

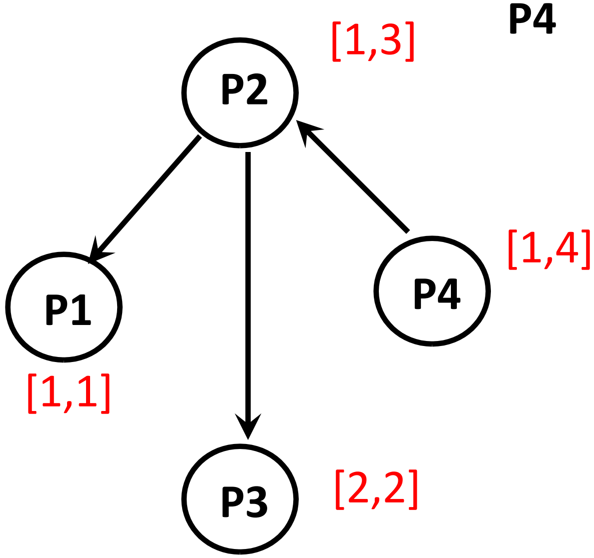
DFS labeling



1. Starting from the first vertex in the root-path
2. Always try to visit the next vertex in the same path
3. Label a node when all its neighbors has been visited
   L(v)=N-x, x is the # of nodes has been labeled

3-Tuple Labeling for Reachability





u→v if and only if 1) Interval label I(u) $\supseteq$ I(v)
2) DFS label L(u) $\leq$ L(v)

?Query(9,15)
P4[1,4] $\supseteq$ P1[1,1] and 5 < 15
Yes
?Query(9,2)
?Query(5,9)

3-HOP Cover [results]

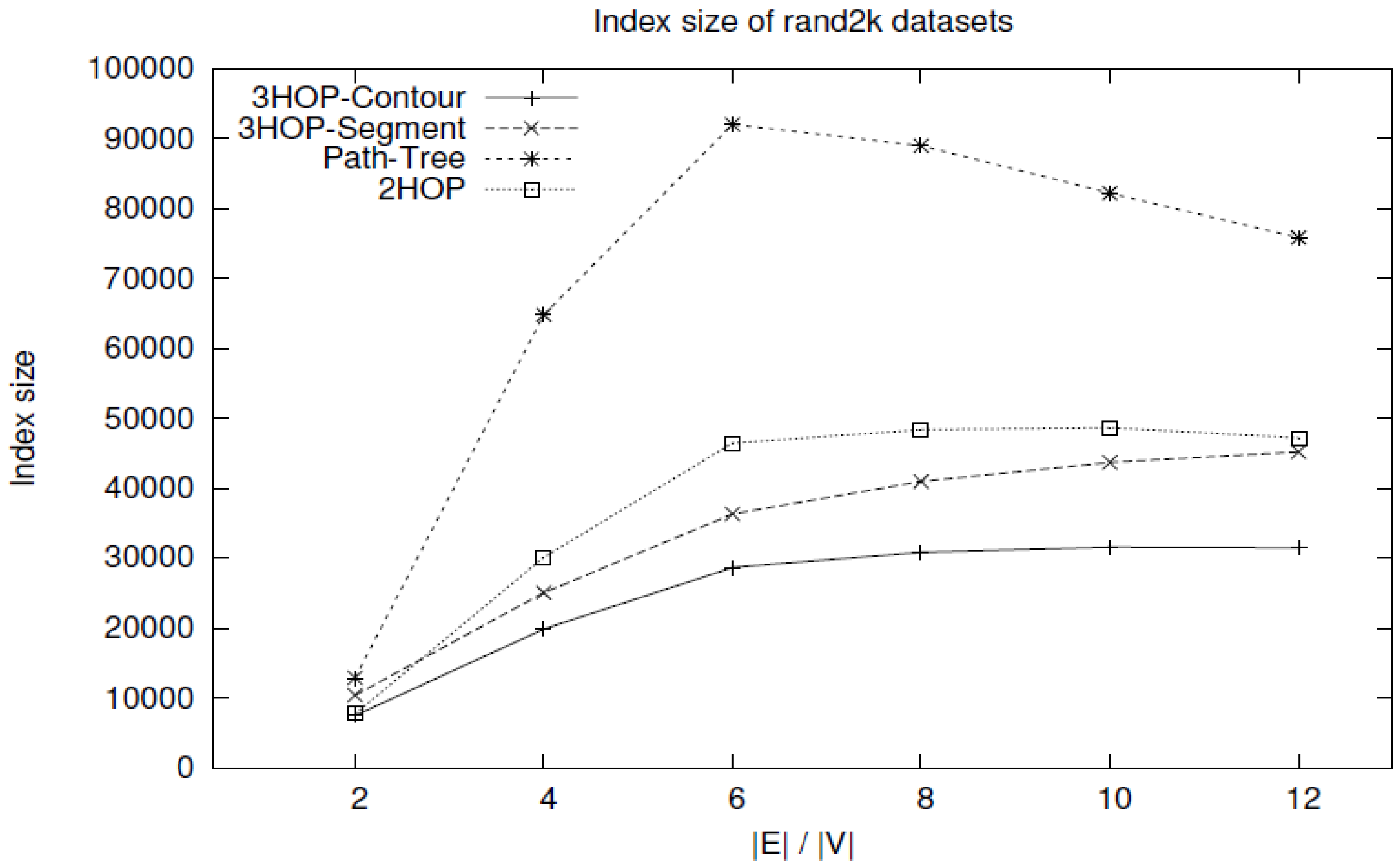


Figure 8: Index size of Synthetic Datasets (2K)