An Overview of Reachability Indexes on Graphs

Chao Zhang[1], Angela Bonifati[2], and M. Tamer Özsu[1]

[1]University of Waterloo

[2]Lyon 1 University

UNIVERSITY OF WATERLOO | DSg Data Systems Group

Lyon 1

Our Objectives

1. What is the problem?

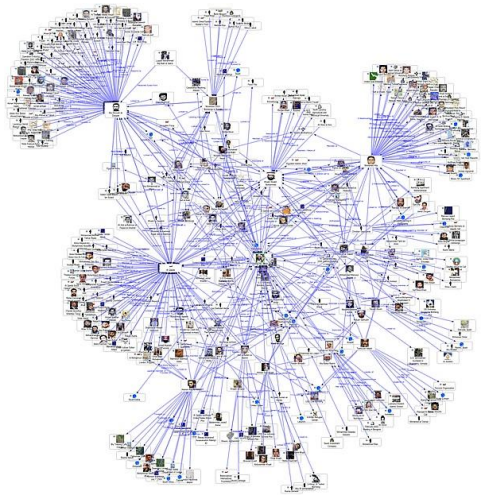
   Reachability queries *on graphs*

2. What are the available approaches?

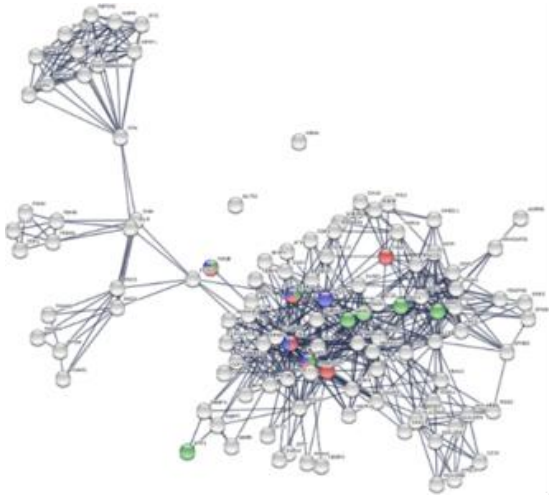
   An overview of ***reachability indexes***

3. What needs to be done?

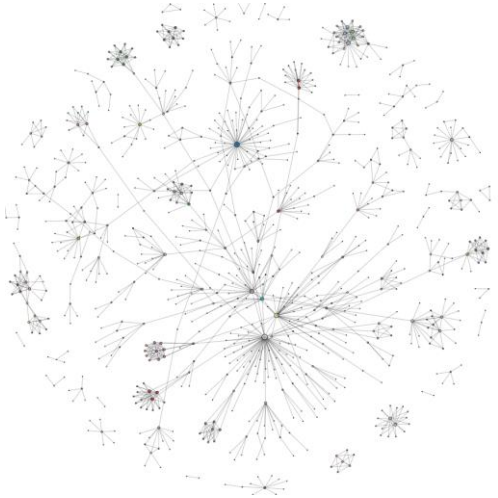
   Research challenges and perspectives



Social networks



Biological networks



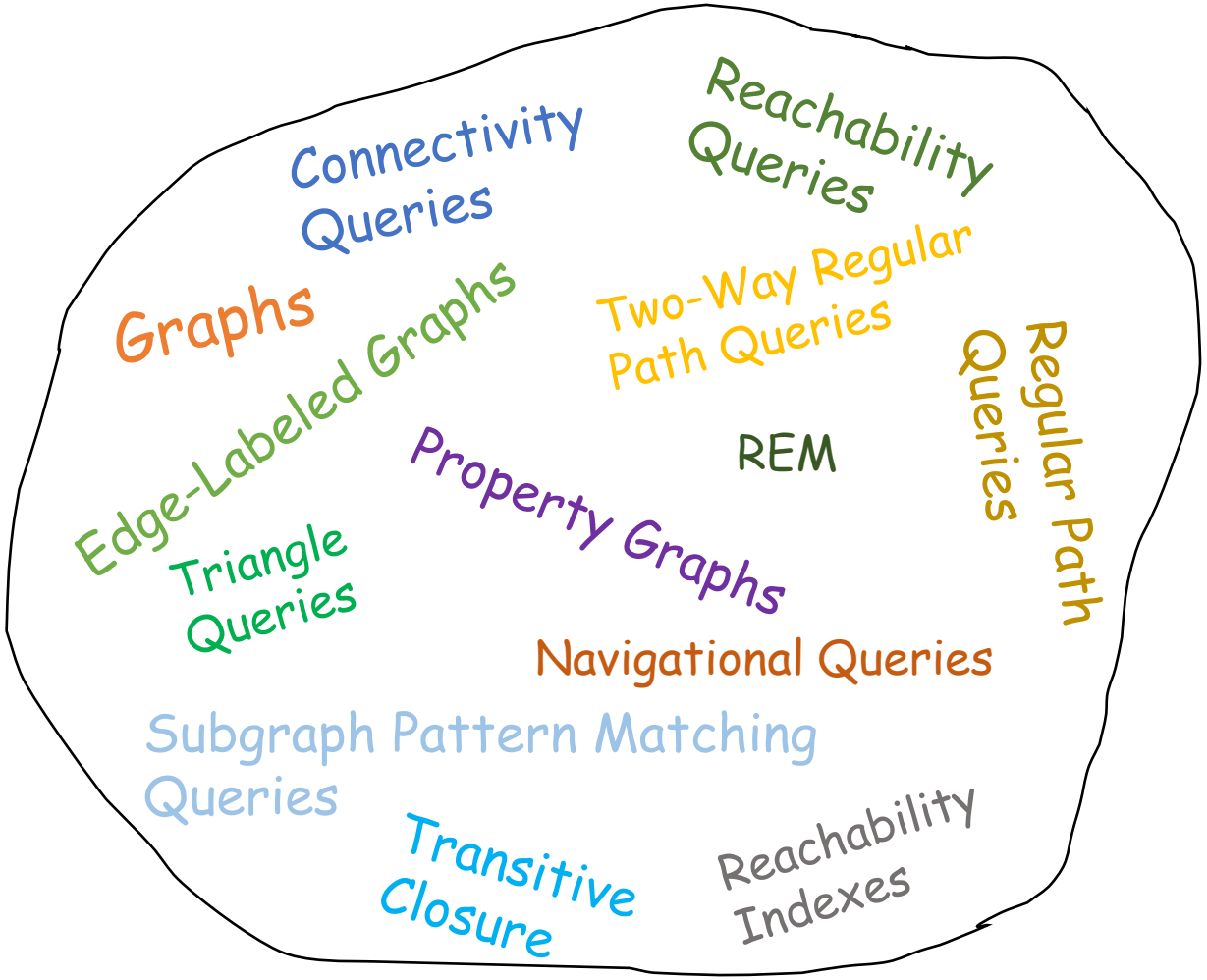
Web graphs

…

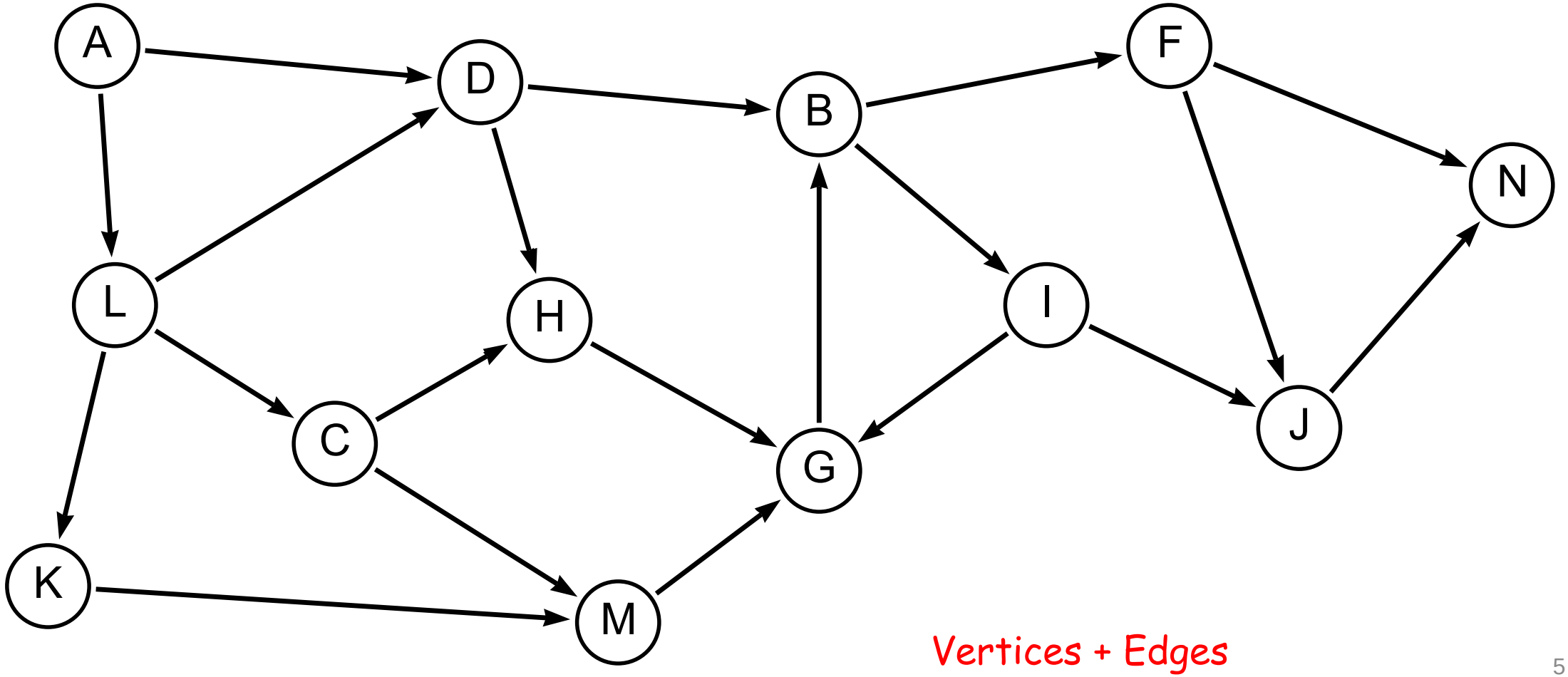
Graphs are everywhere

One of the fundamental graph processing operations [Sah20]: **Reachability Queries**

[Sah20] S. Sahu et al. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. VLDB J. 29(2-3): 595-618 (2020)

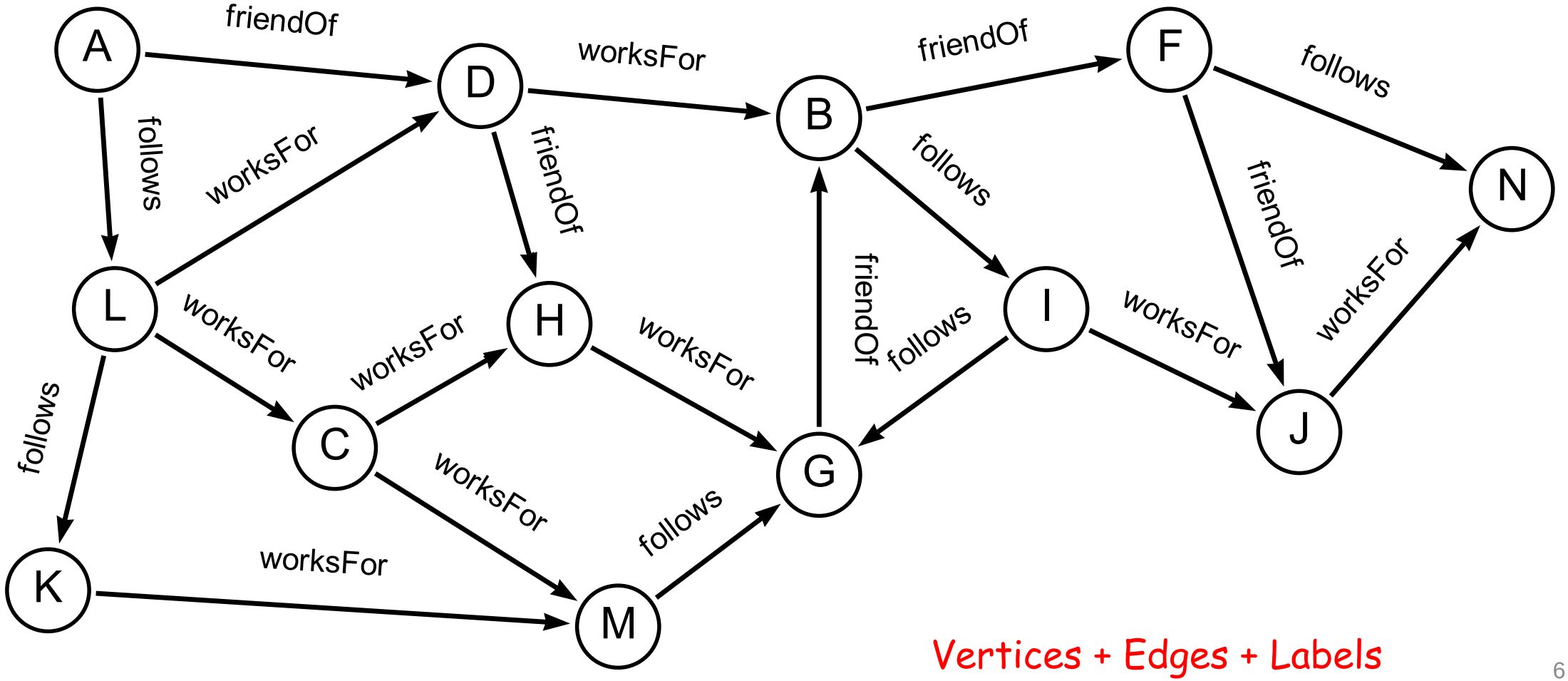

Connectivity Queries
Reachability Queries
Graphs
Edge-Labeled Graphs
Two-Way Regular Path Queries
Regular Path Queries
REM
Property Graphs
Triangle Queries
Navigational Queries
Subgraph Pattern Matching Queries
Transitive Closure
Reachability Indexes

Plain Graphs



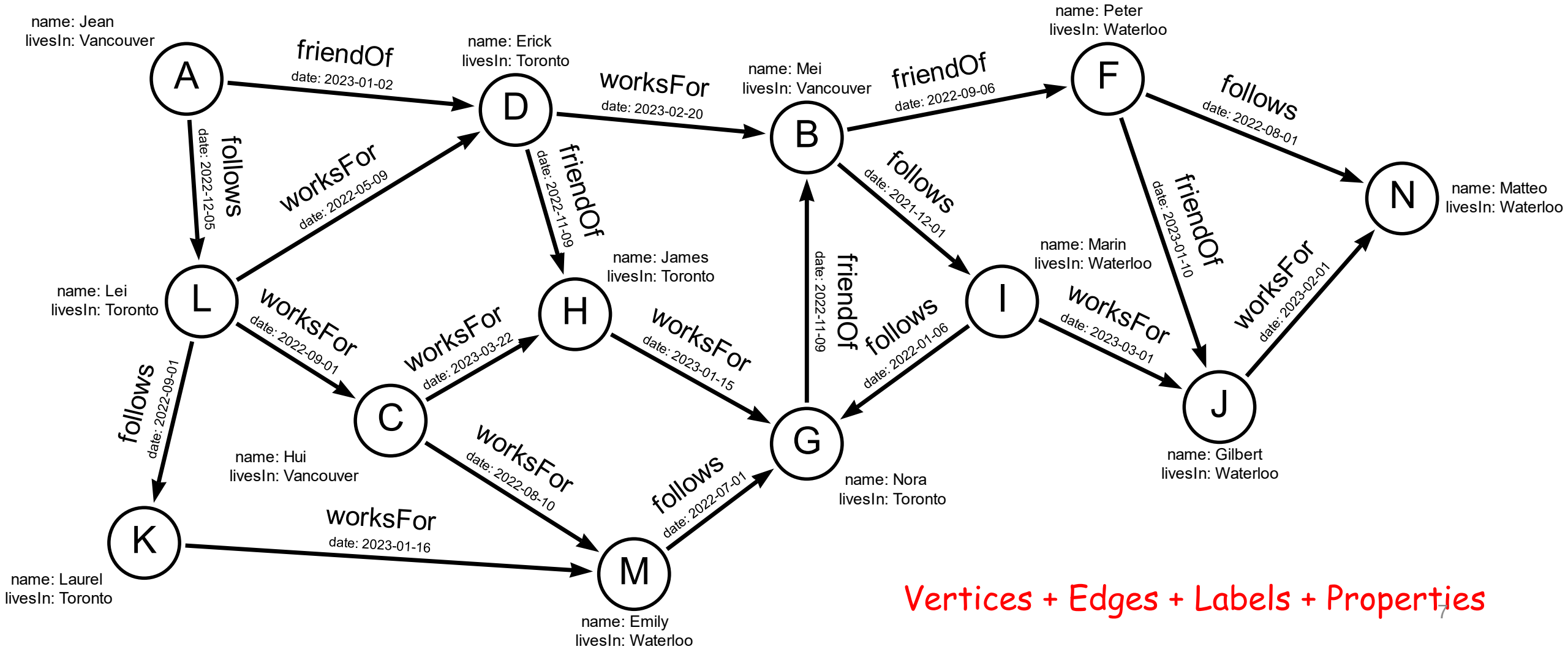
Vertices + Edges

Edge-Labeled Graphs



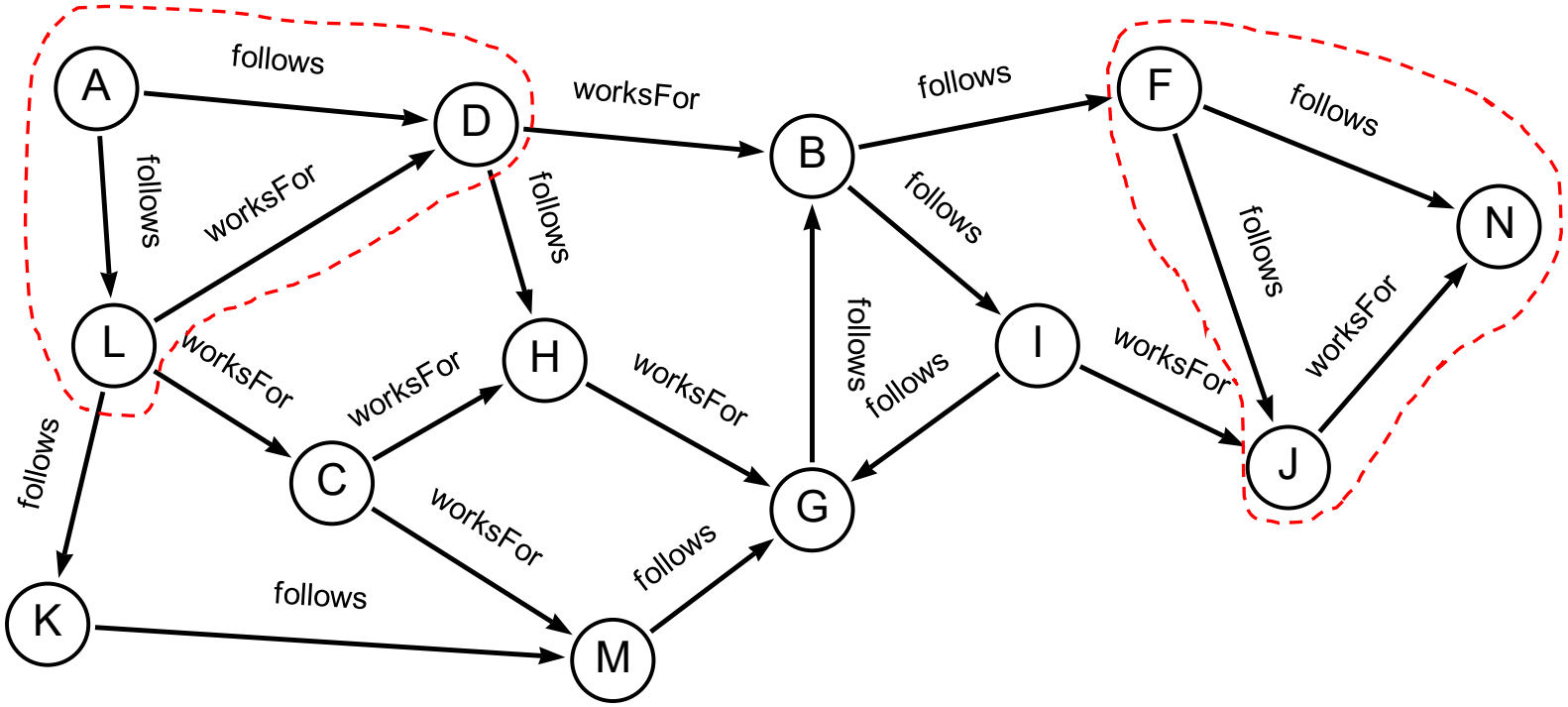
Vertices + Edges + Labels

Property Graphs

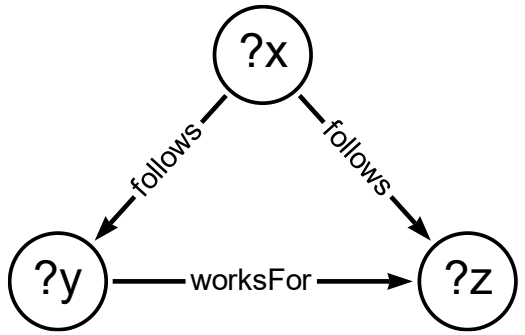


Vertices + Edges + Labels + Properties

Graph Queries: Pattern Matching Queries



Query

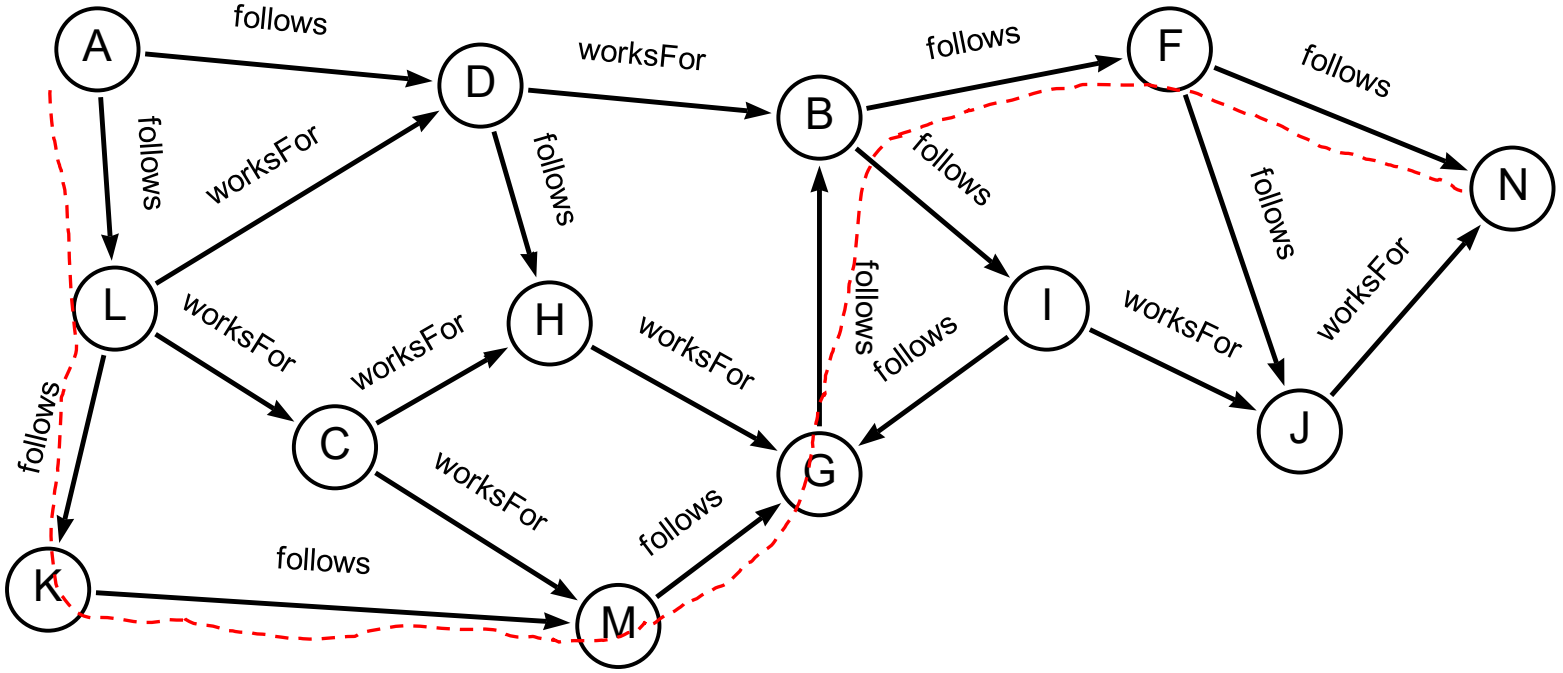


Result

?x	?y	?z
A	L	D
F	J	N

R. Angles et al. Foundations of Modern Query Languages for Graph Databases. ACM Comput. Surv. 50(5): 68:1-68:40 (2017)

Graph Queries: Navigational Queries



Query

$$A \xrightarrow{follows^*} N$$

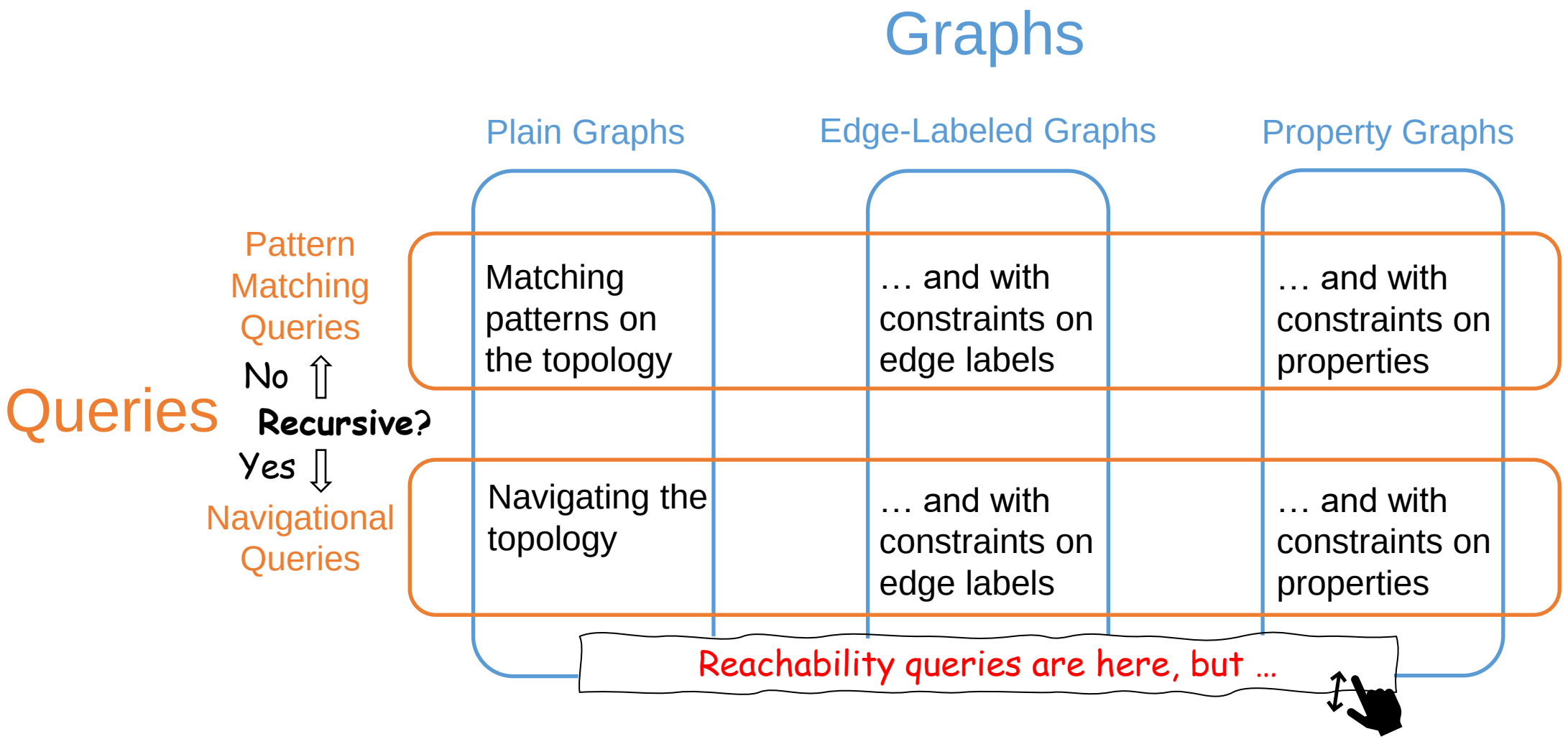
Whether A reaches N through paths with only *follows*-edges?

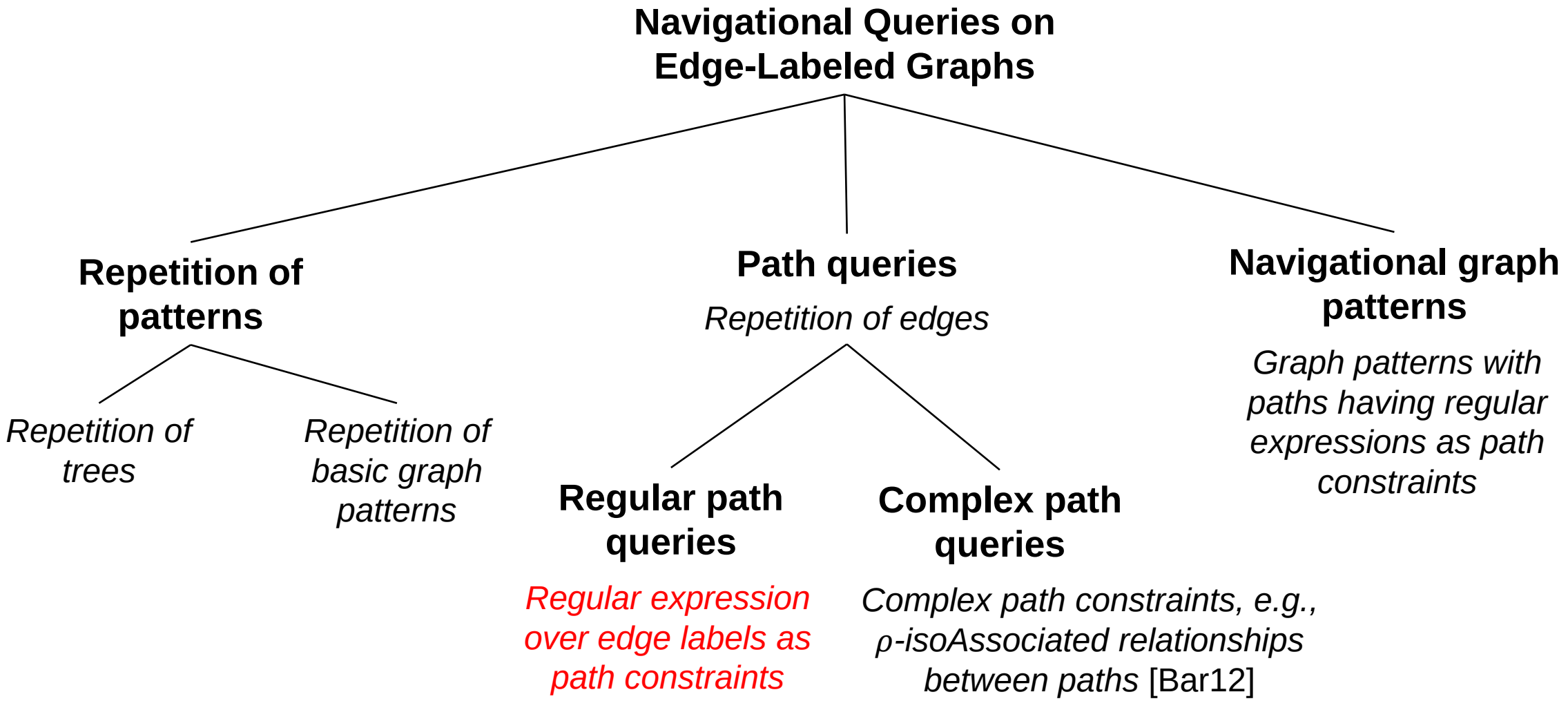
Result

True

R. Angles et al. Foundations of Modern Query Languages for Graph Databases. ACM Comput. Surv. 50(5): 68:1-68:40 (2017)

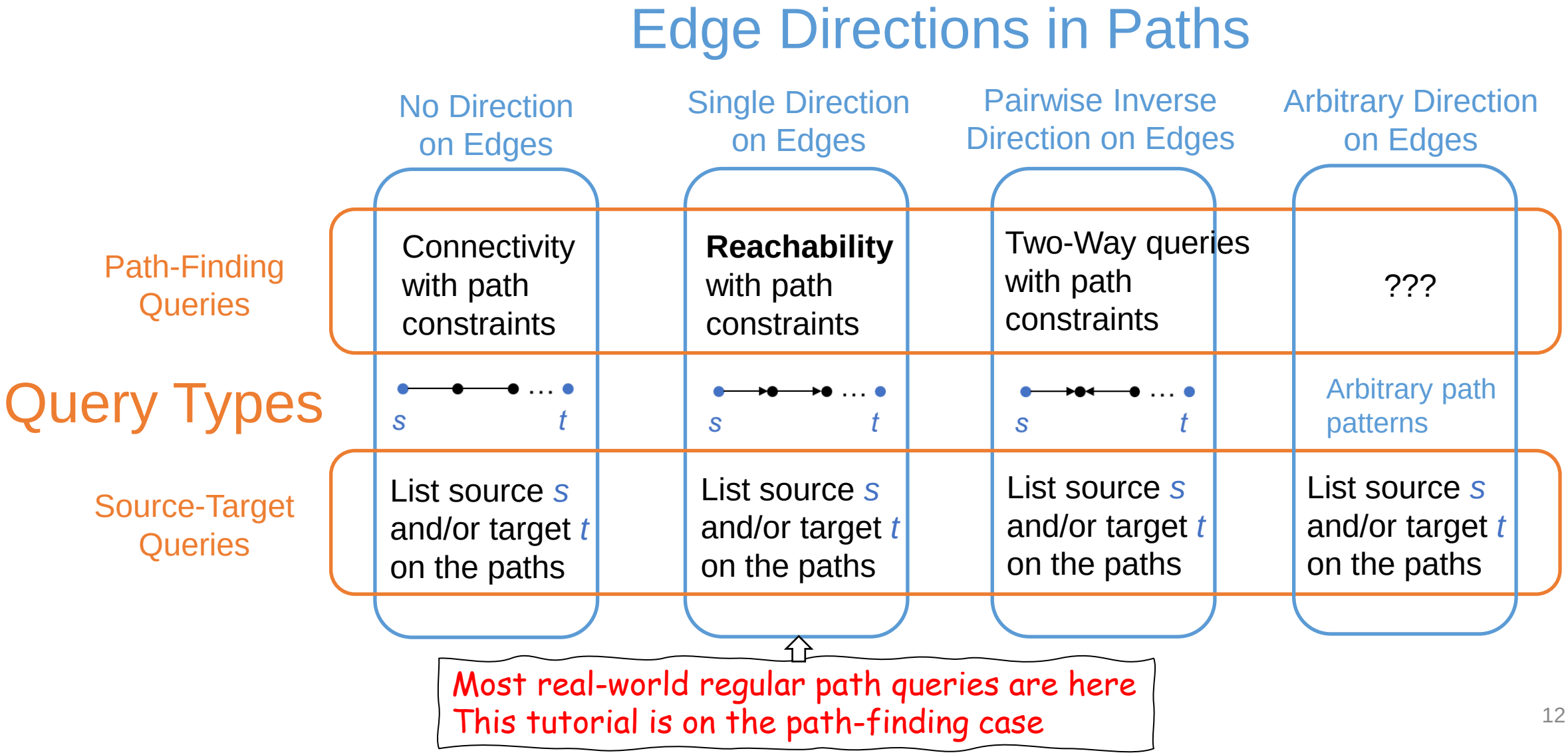
Queries on Graphs





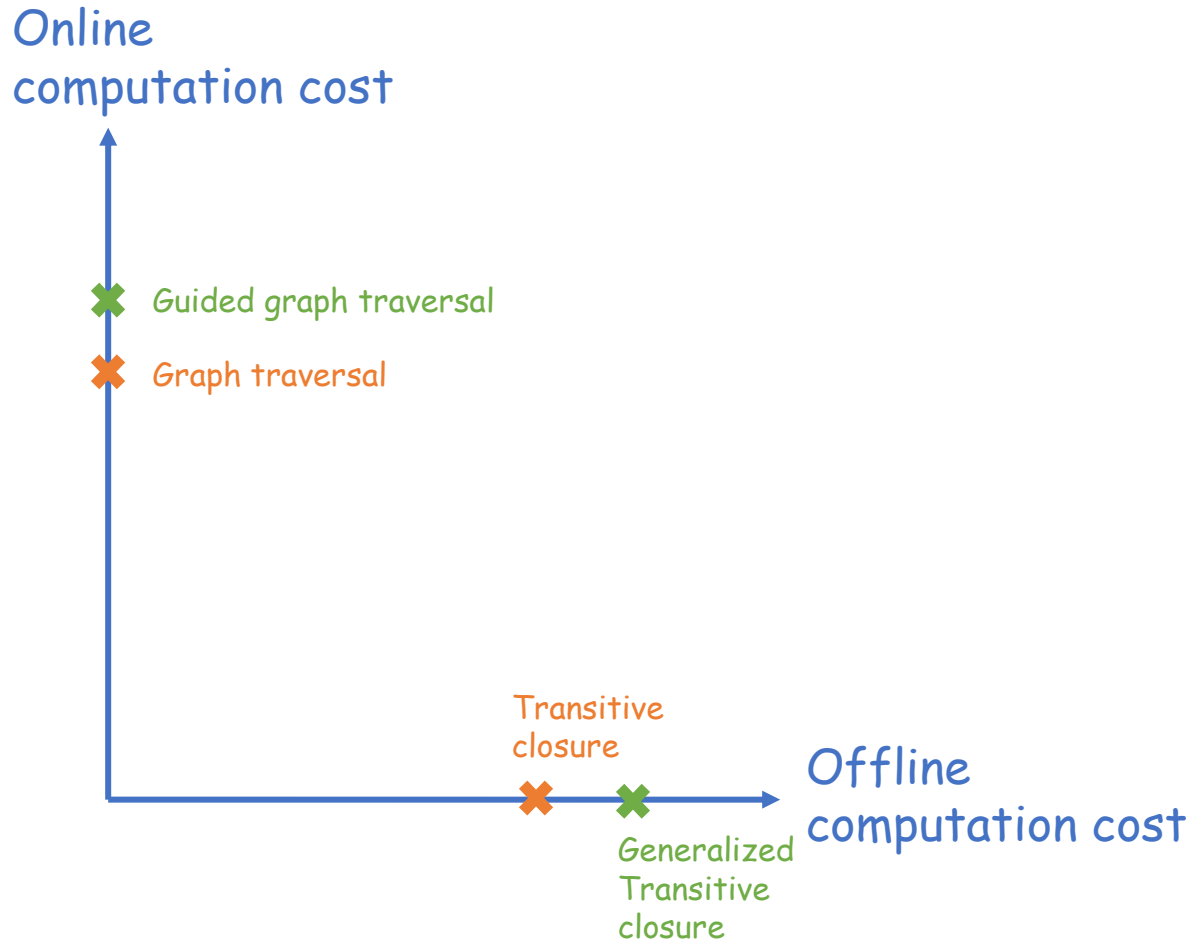
[Bar12] P. Barceló et al. Expressive Languages for Path Queries over Graph-Structured Data. ACM Trans. Database Syst. 37(4): 31:1-31:46 (2012)

Regular Path Queries: Taxonomy



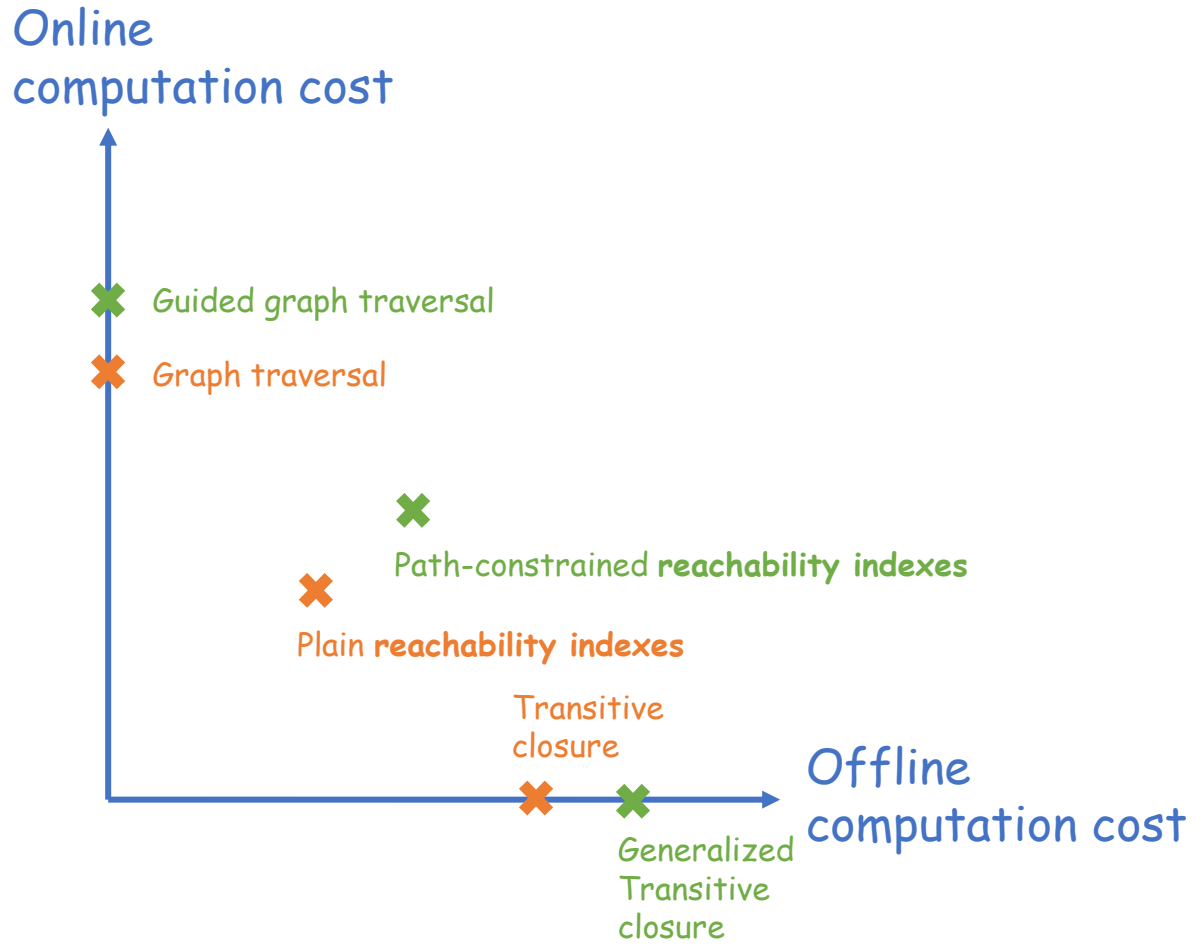
Reachability Query Processing

- Reachability on plain graphs
  - Full online computation:
    - Graph traversal
  - Full offline approach:
    - Transitive closure
- Reachability on edge-labeled graphs
  - Full online computation:
    - Guided graph traversal
  - Full offline approach:
    - Generalized transitive closure



Reachability Indexes

- Plain graphs:
  - Plain reachability indexes
- Edge-labeled graphs:
  - Path-constrained reachability indexes

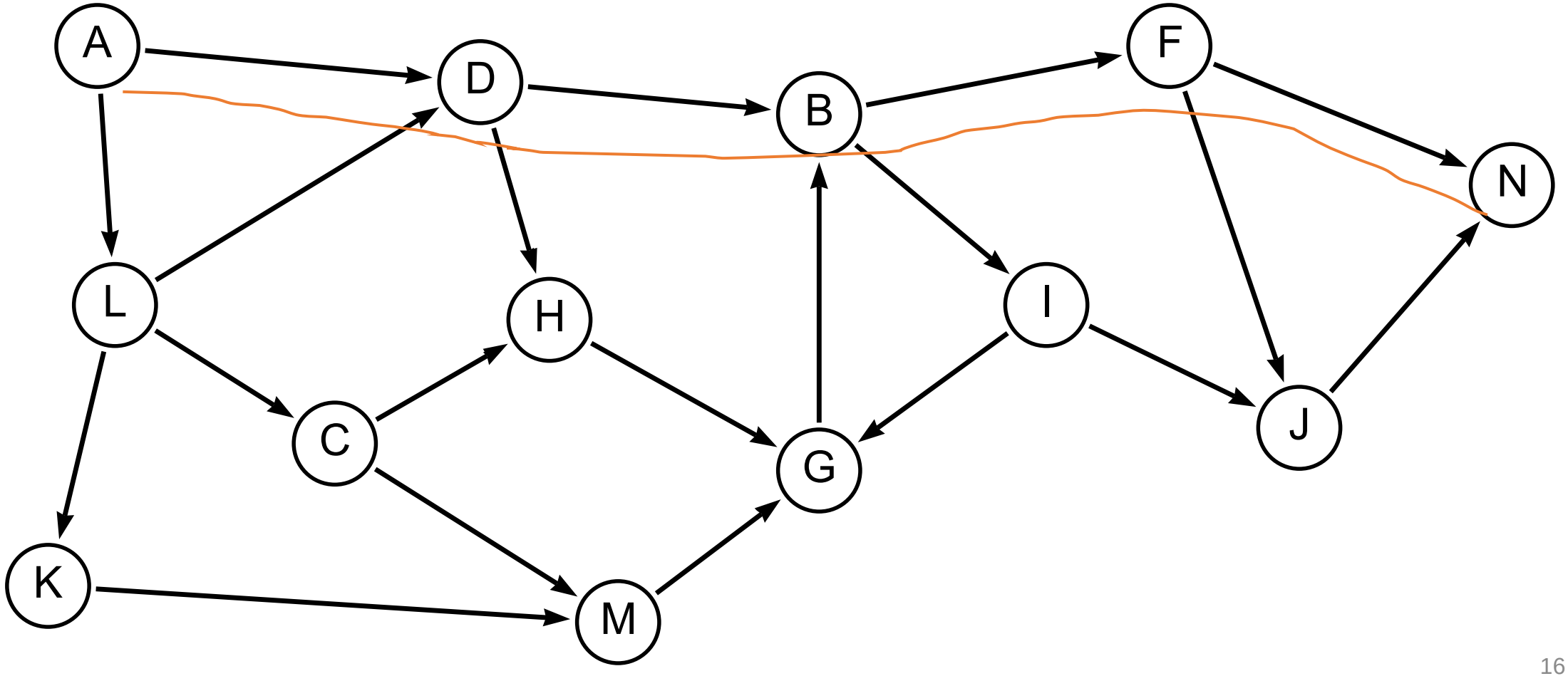


Outline

1. Plain Reachability Indexes
    a) Tree-Cover Indexes
    b) 2-Hop Indexes
    c) Approximate Transitive Closures
    d) Other Techniques
2. Path-Constrained Reachability Indexes
    a) Indexes for Alternation-Based Queries
    b) Indexes for Concatenation-Based Queries
3. Open Challenges

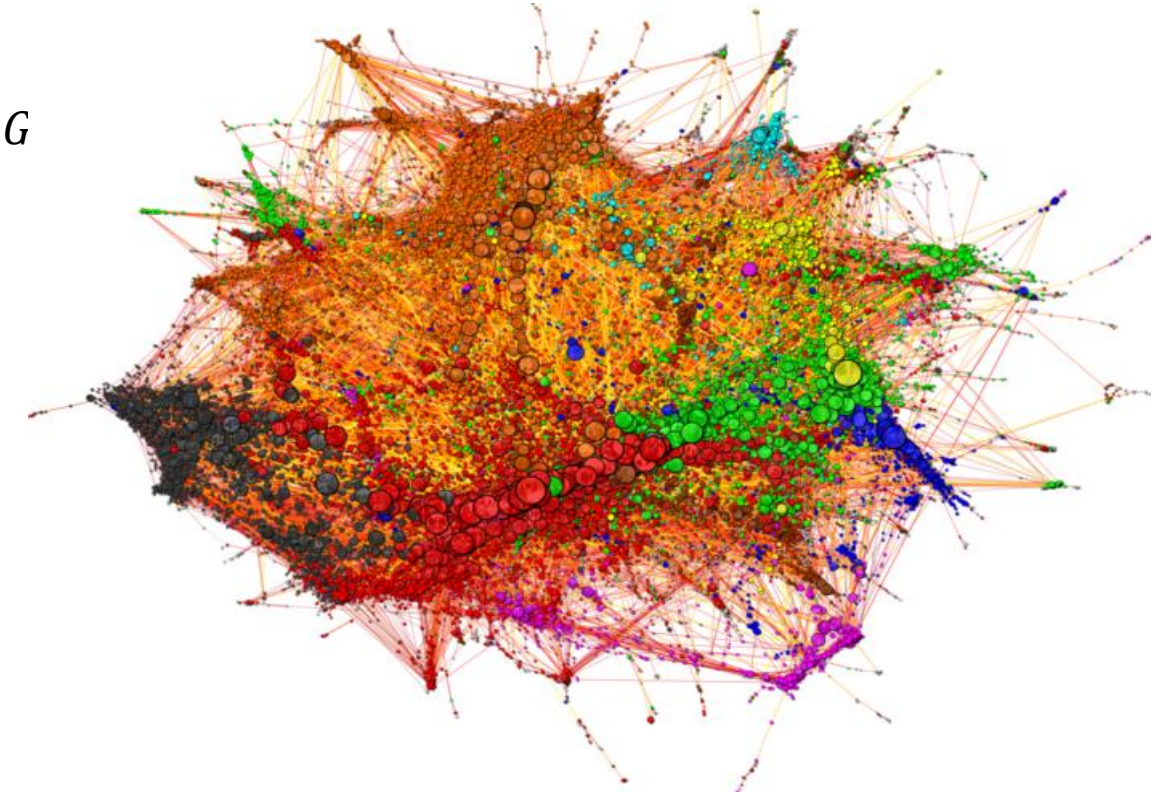
Plain Reachability Query

Is there a path from vertex A to vertex N?



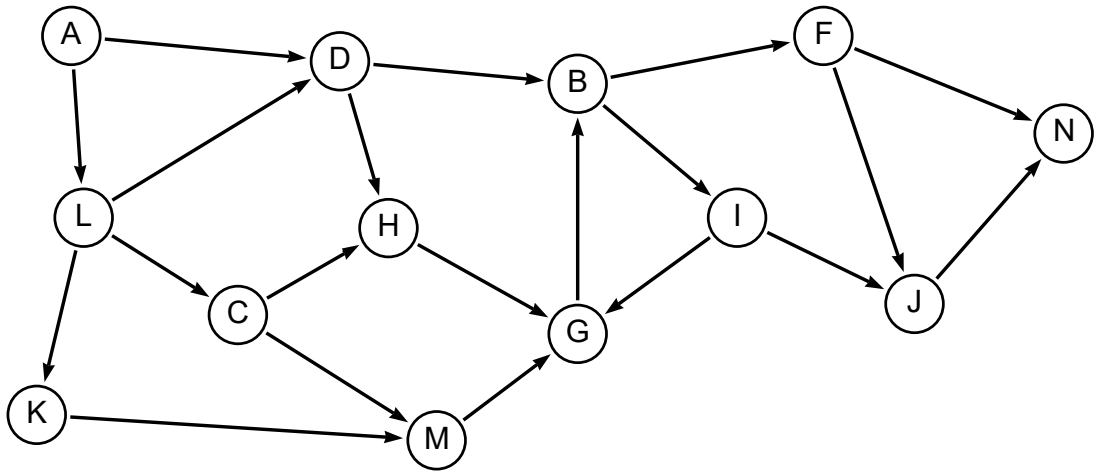
Plain Reachability Query

- $Q_r(s, t)$ on a directed graph G
  - Checking the existence of a path from s to t in G
  - Return either True or False
- Query evaluation
  - Graph traversal: BFS, DFS, and BiBFS
  - Problem: graphs are large
- A naïve index for reachability queries
  - Transitive closure (TC)



Music Recommendation Graph

Transitive Closure



Checking whether vertex A can reach vertex N can be done in constant time

Target Vertex

Source Vertex

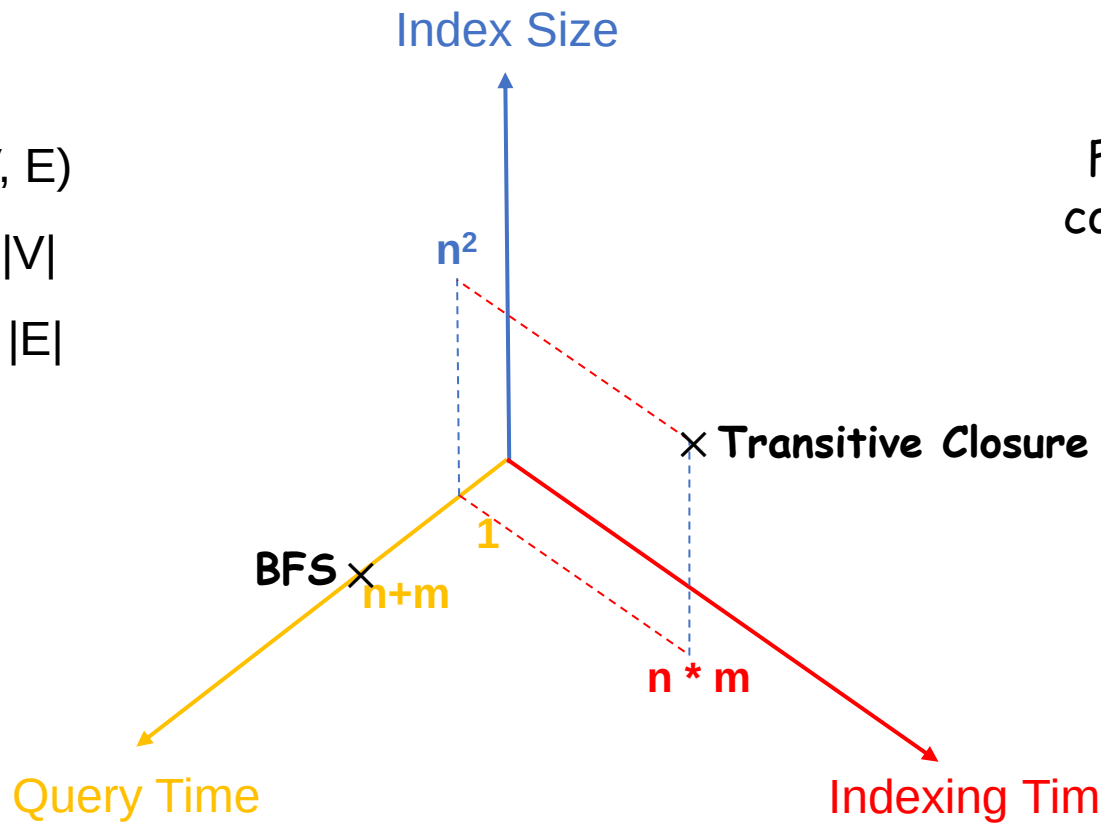
	A	B	C	D	F	G	H	I	J	K	L	M	N
A	0	1	1	1	1	1	1	1	1	1	1	1	1
B	0	1	0	0	1	1	0	1	1	0	0	0	1
C	0	1	0	0	1	1	1	1	1	0	0	1	1
D	0	1	0	0	1	1	1	1	1	0	0	0	1
F	0		0	0	0	0	0	0	1	0	0	0	1
G	0	1	0	0	1	1	0	1	1	0	0	0	1
H	0	1	0	0	1	1	0	1	1	0	0	0	1
I	0	0	0	0	0	0	0		1	0	0	0	1
J	0	0	0	0	0	0	0	0	0	0	0	0	1
K	0	1	0	0	1	1	0	1	1	0	0	1	1
L	0	1	0	0	1	1	1	1	1	0	0	1	1
M	0	1	0	0	1	1	0	1	1	0	0	0	1
N	0	0	0	0	0	0	0	0	0	0	0	0	0

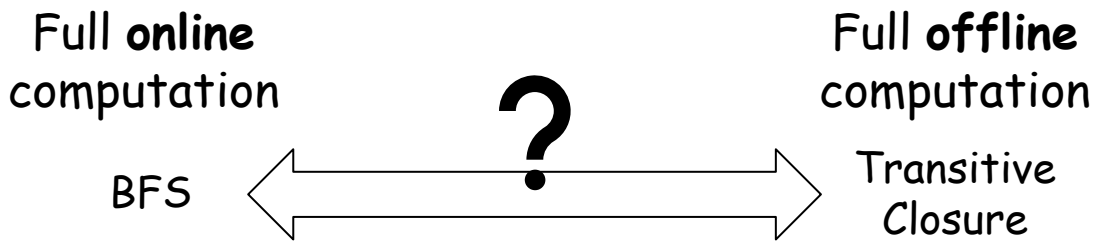
Complexity

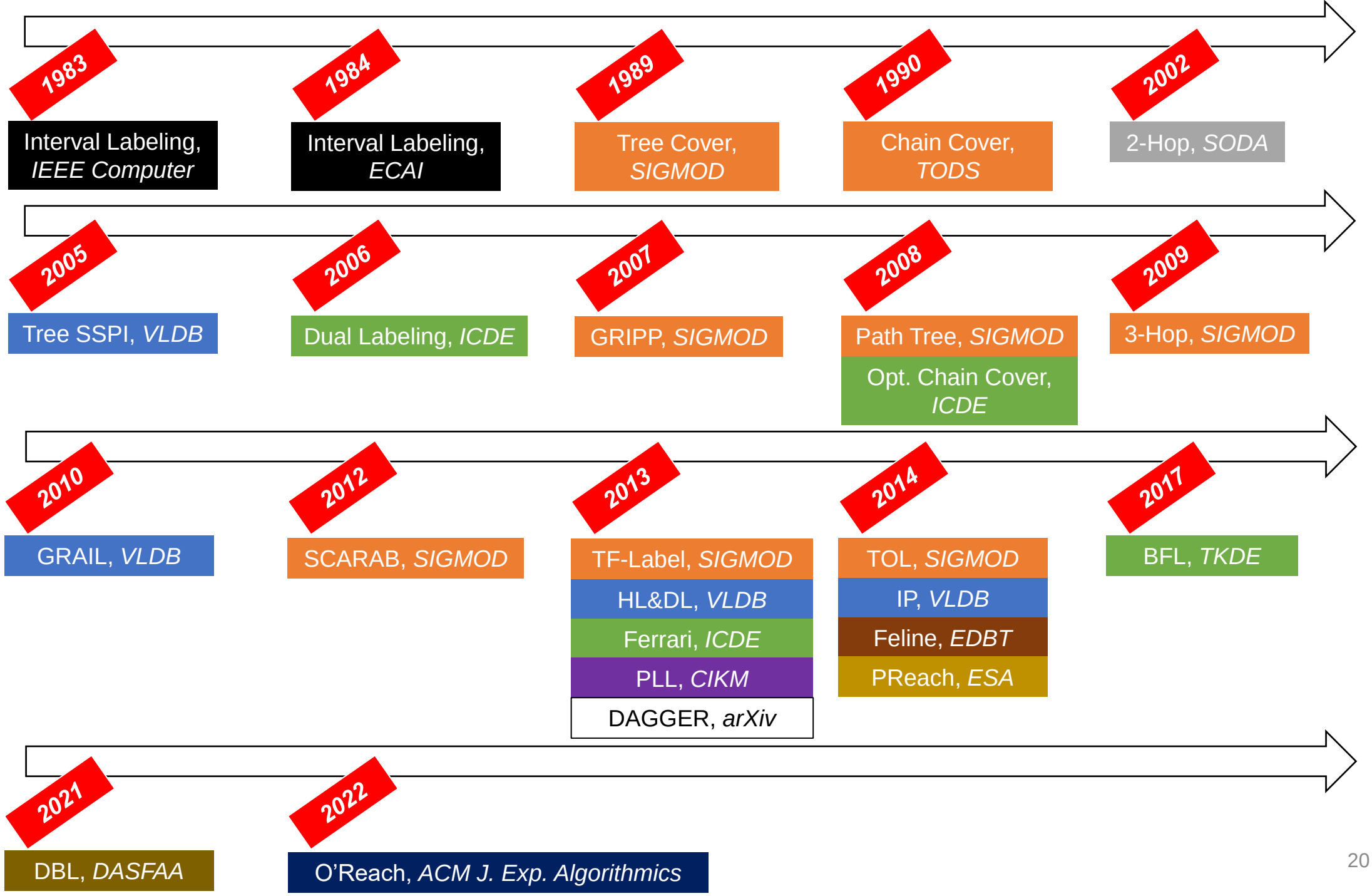
G (V, E)

n = |V|

m = |E|






1983
Interval Labeling, IEEE Computer
1984
Interval Labeling, ECAI
1989
Tree Cover, SIGMOD
1990
Chain Cover, TODS
2002
2-Hop, SODA
2005
Tree SSPI, VLDB
2006
Dual Labeling, ICDE
2007
GRIPP, SIGMOD
2008
Path Tree, SIGMOD
Opt. Chain Cover, ICDE
2009
3-Hop, SIGMOD
2010
GRAIL, VLDB
2012
SCARAB, SIGMOD
2013
TF-Label, SIGMOD
HL&DL, VLDB
Ferrari, ICDE
PLL, CIKM
DAGGER, arXiv
2014
TOL, SIGMOD
IP, VLDB
Feline, EDBT
PReach, ESA
2017
BFL, TKDE
2021
DBL, DASFAA
2022
O'Reach, ACM J. Exp. Algorithmics

Indexing technique	Framework	Index type	Input	Dynamic	References
Tree cover	Tree cover	Complete	DAG	No	[Agr89]
Tree+SSPI	Tree cover	Partial	DAG	No	[Che05]
Dual labeling	Tree cover	Complete	DAG	No	[Wan06]
GRIPP	Tree cover	Partial	General Graph	No	[Tri07]
Path-Tree	Tree cover	Complete	DAG	Yes	[Jin08,Jin11]
GRAIL	Tree cover	Partial	DAG	No	[Yil10]
Ferrari	Tree cover	Partial	DAG	No	[Seu13]
DAGGER	Tree cover	Partial	DAG	Yes	[Yil13]
2-Hop	2-Hop	Complete	General Graph	No	[Coh02]
Ralf et al.	2-Hop	Complete	General Graph	Yes	[Sch05]
3-Hop	2-Hop	Complete	DAG	No	[Jin09]
U2-Hop	2-Hop	Complete	DAG	Yes	[Bra10]
Path-Hop	2-Hop	Complete	DAG	No	[Cai10]
TFL	2-Hop	Complete	DAG	No	[Che13]
DL	2-Hop	Complete	General Graph	No	[Jin13]
PLL	2-Hop	Complete	General Graph	No	[Yan13]
TOL	2-Hop	Complete	DAG	Yes	[Zhu14]
DBL	2-Hop	Partial	General Graph	Yes	[Lyu21]
O'Reach	2-Hop	Partial	DAG	No	[Han21]
IP	Approximate TC	Partial	DAG	Yes	[Wei14,Wei18]
BFL	Approximate TC	Partial	DAG	No	[Su17]
Chain Cover	Chain Cover	Complete	DAG	Yes	[Jag90]
Opt. Chain Cover	Chain Cover	Complete	DAG	Yes	[Che08]
HL	-	Complete	DAG	No	[Jin13]
Feline	-	Partial	DAG	No	[Vel14]
Preach	-	Partial	DAG	No	[Mer14]

Complete index: index-only query processing

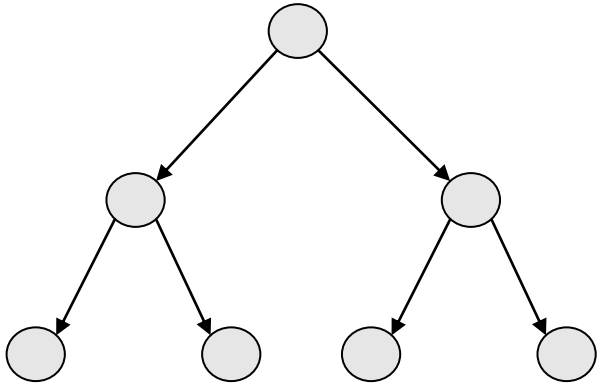
Partial index: index-graph query processing

Three widely used frameworks:

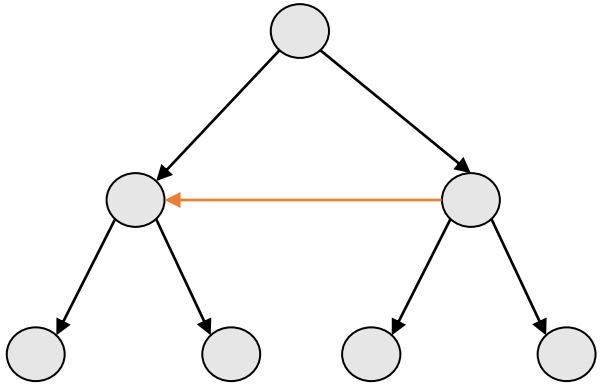
- Tree cover
- 2-Hop
- Approximate TC

Outline

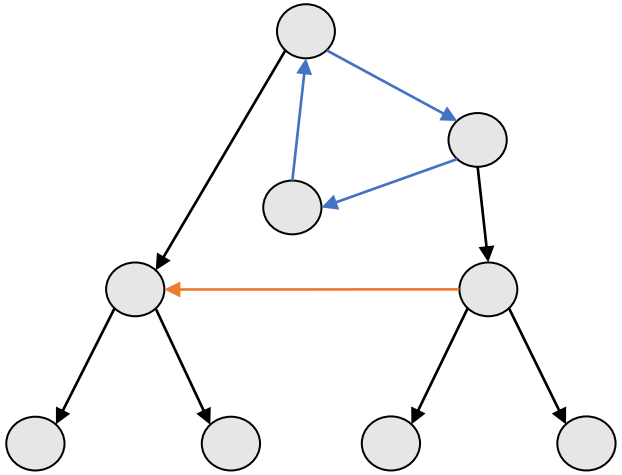
1. Plain Reachability Indexes
    a) Tree-Cover Indexes
    b) 2-Hop Indexes
    c) Approximate Transitive Closures
    d) Other Techniques
2. Path-Constrained Reachability Indexes
    a) Indexes for Alternation-Based Queries
    b) Indexes for Concatenation-Based Queries
3. Open Challenges



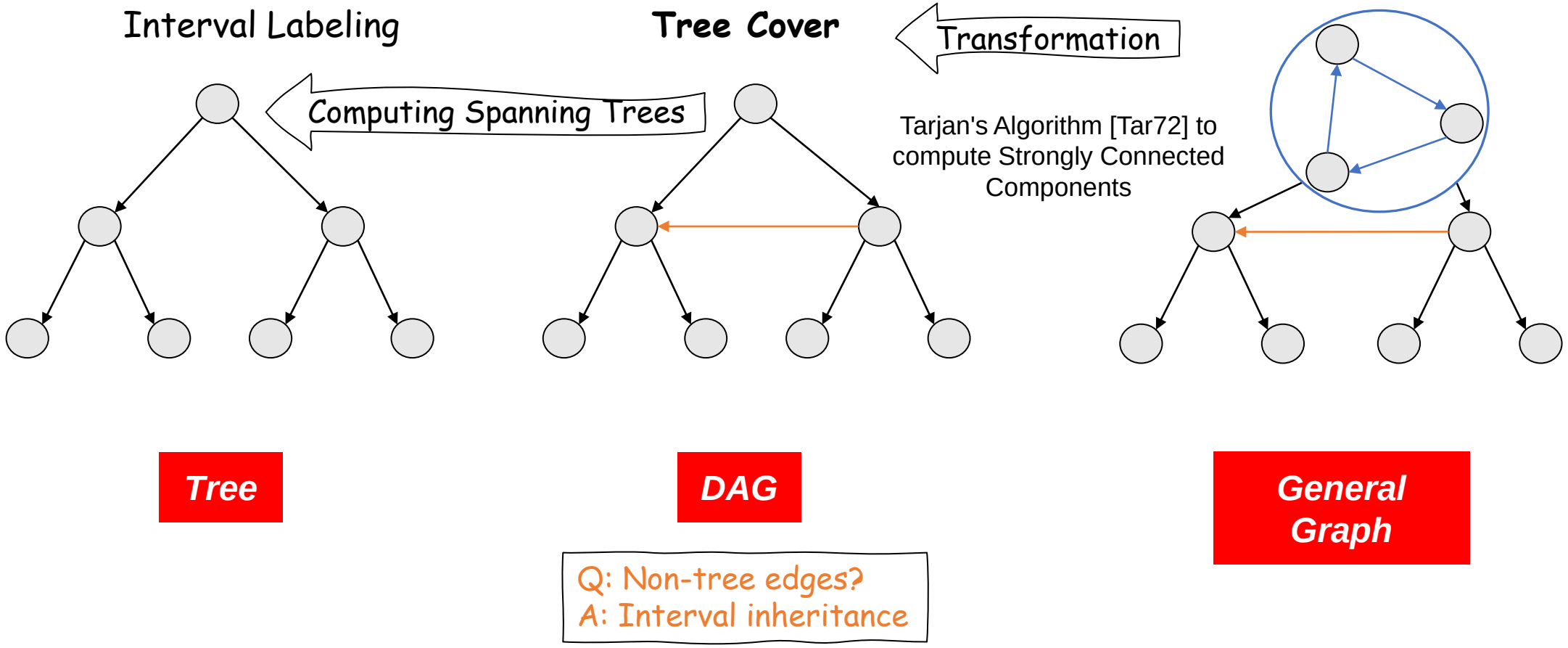
Tree



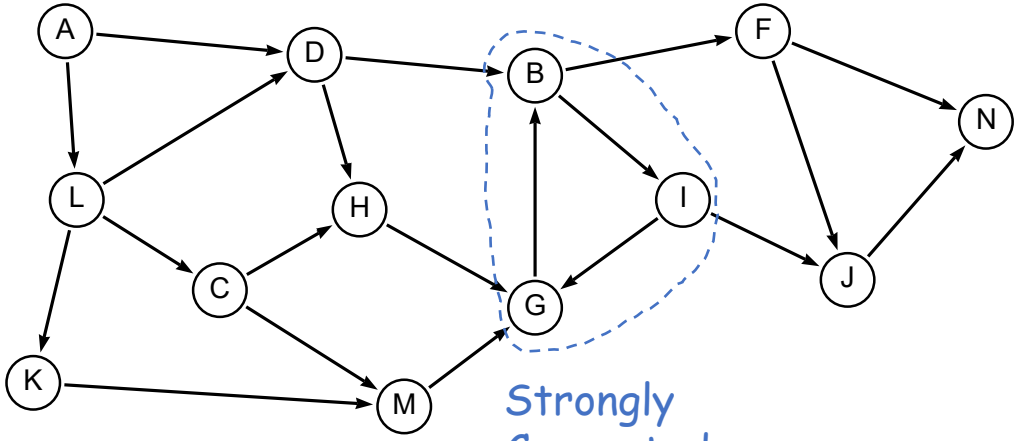
DAG



General Graph

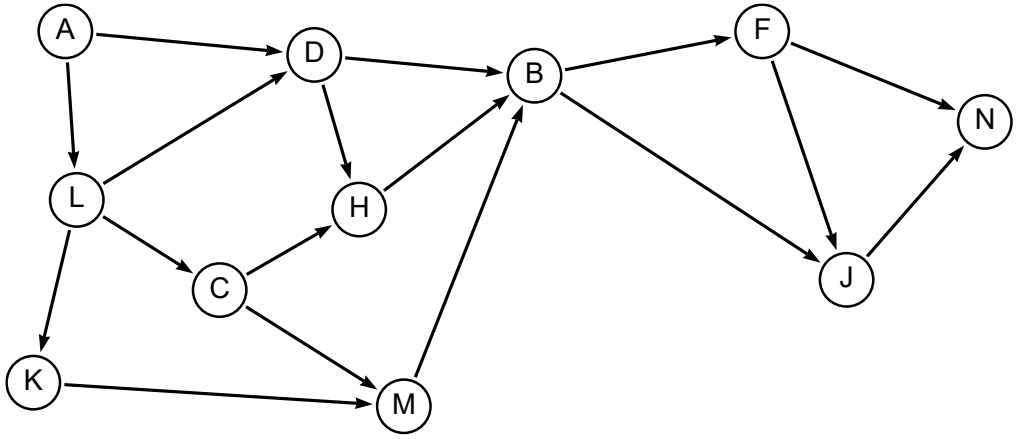


[Tar72] R. Tarjan. Depth-First Search and Linear Graph Algorithms. SIAM J. Comput. 1(2): 146-160 (1972)



Strongly Connected Component

Transformation

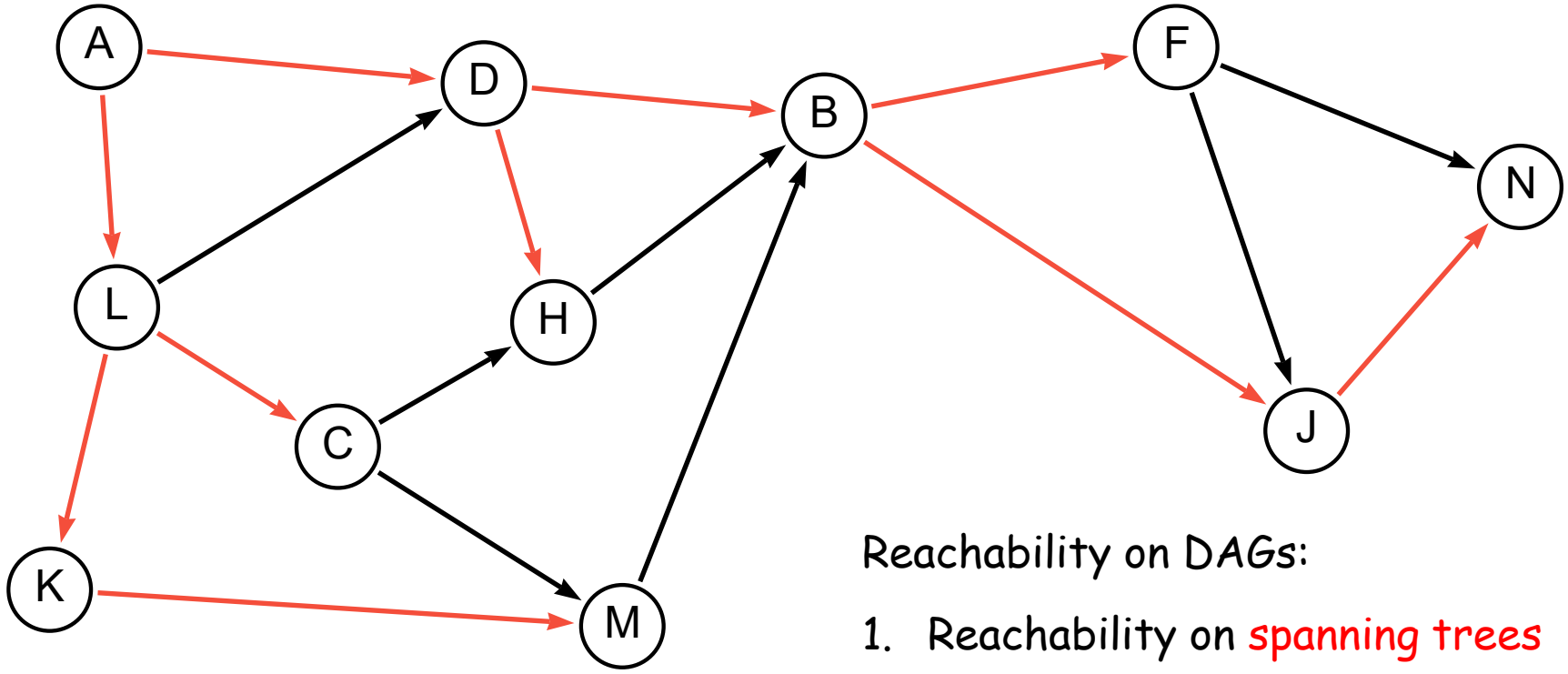


$Q_r(A, I)$: whether A reaches I

Reduction

$Q_r(A, B)$: whether A reaches B

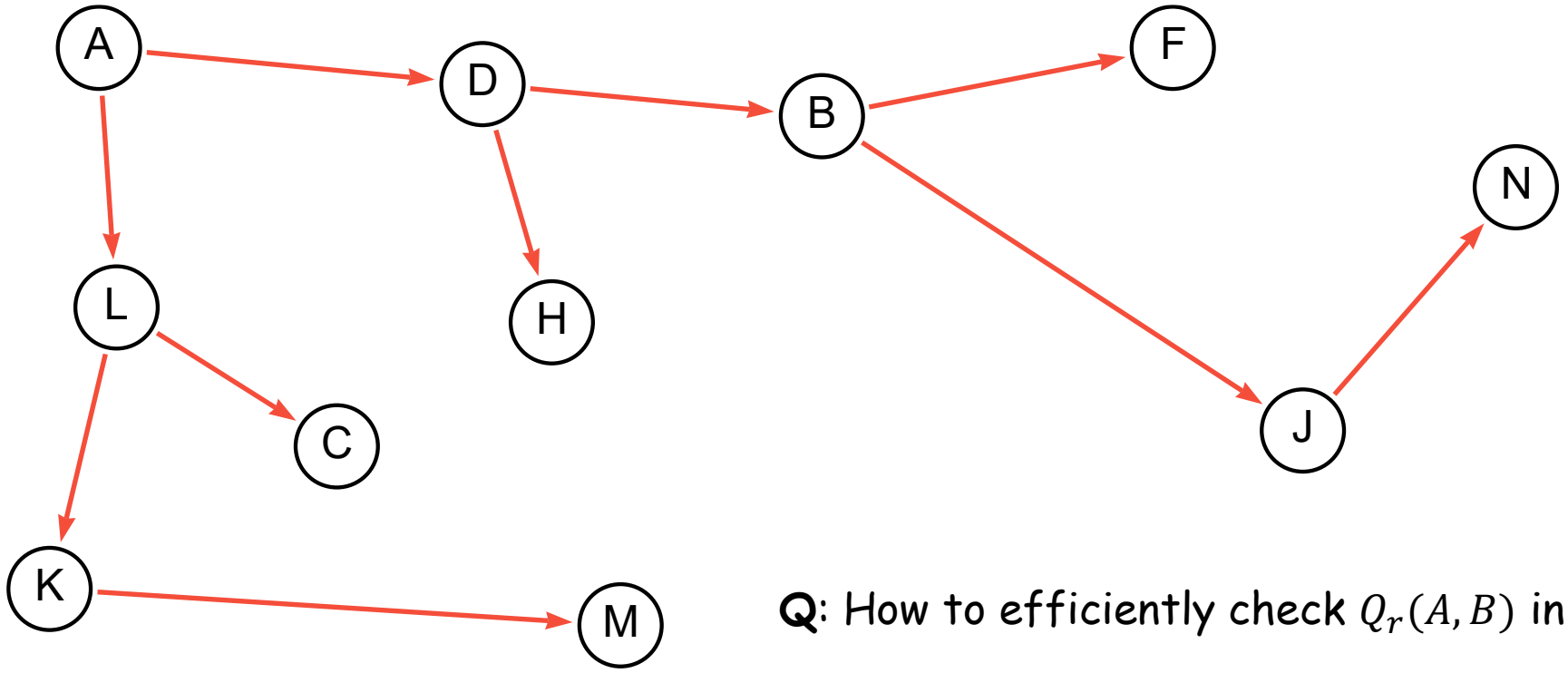
Reachability on DAGs



Reachability on DAGs:

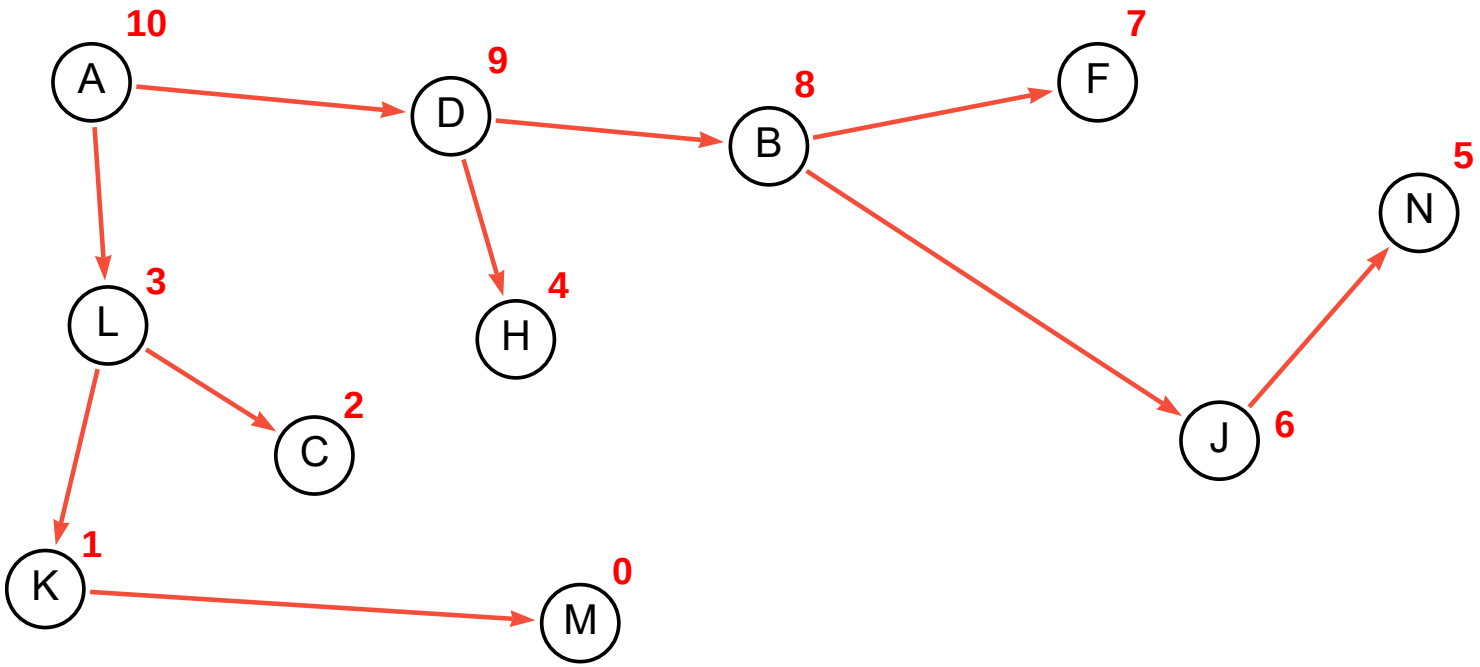
1. Reachability on spanning trees
2. Extension for non-tree edges

Reachability on Spanning Trees

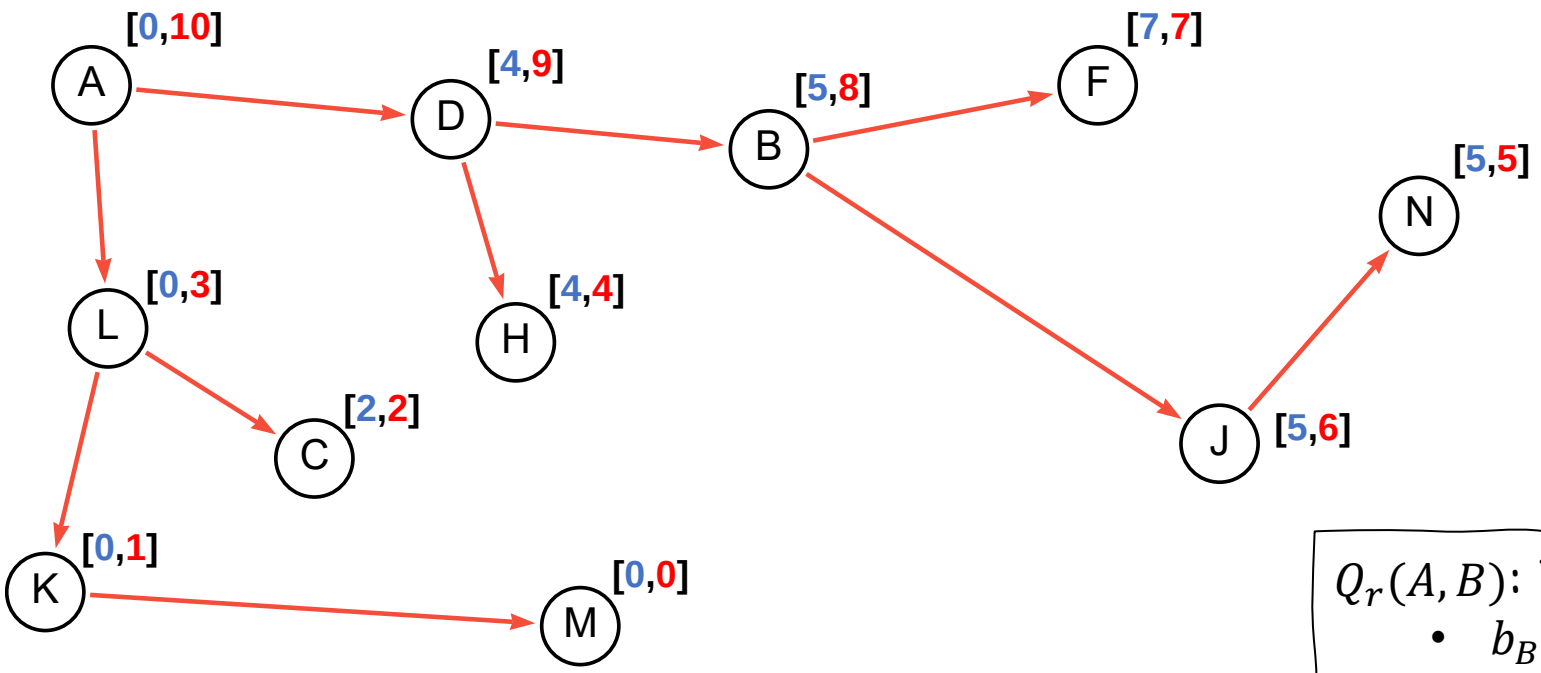


Q: How to efficiently check $Q_r(A, B)$ in the tree?

Postorder Numbers



Interval Labeling



Assign an interval $[a_v, b_v]$ to each vertex v, denoted as $\mathcal{L}_v$

a_v: The lowest postorder number of all the descendants of v

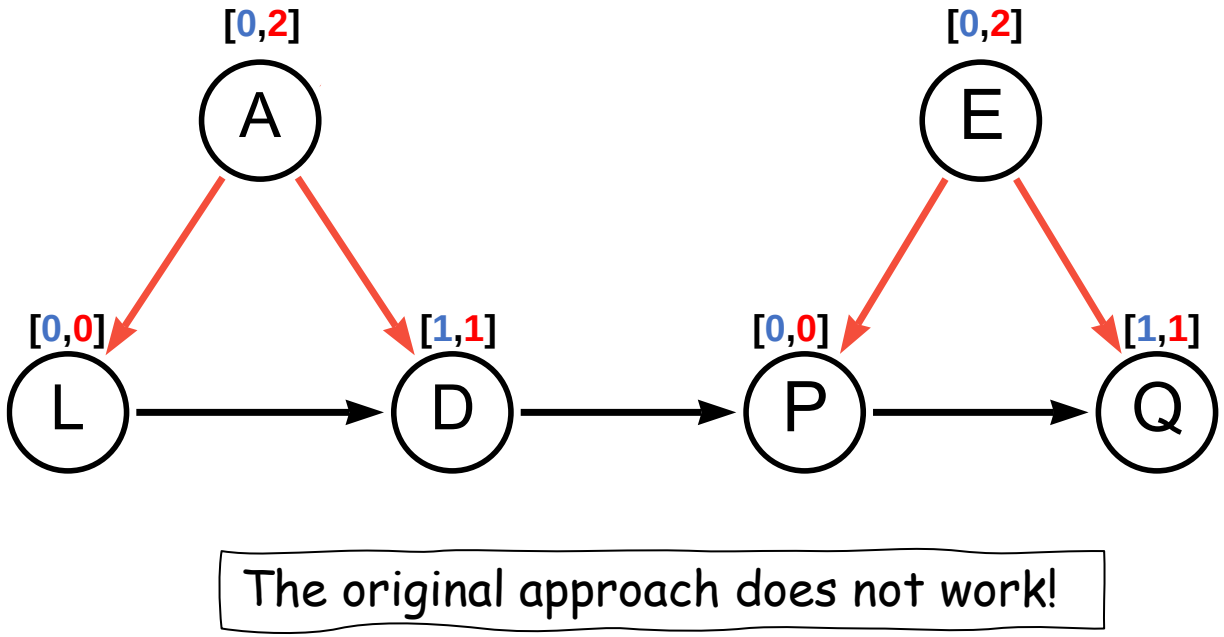
b_v: Postorder number of v

$Q_r(A, B)$: True

- $b_B \in \mathcal{L}_A$
  - $[0,10]$ contains 8
- Equivalently, $\mathcal{L}_B \subseteq \mathcal{L}_A$
  - $[5,8] \subseteq [0,10]$

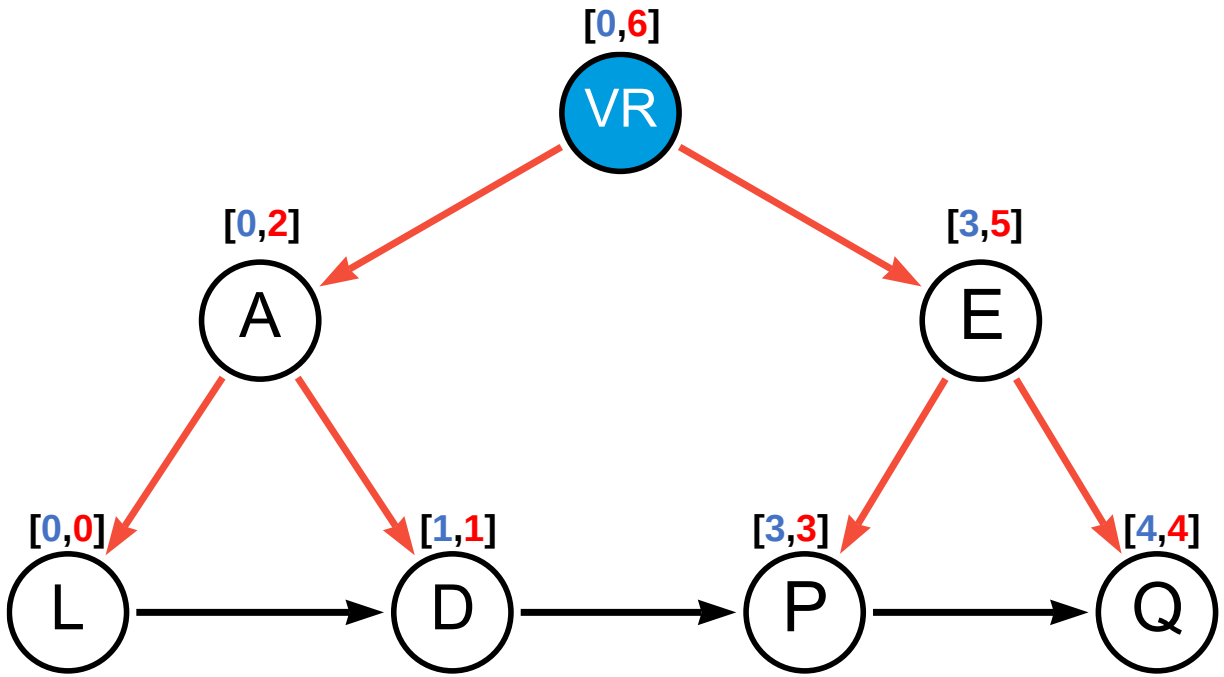
From Spanning Trees to DAGs

- Problem 1: *spanning forests*
  - $Q_r(E, L)$: E does not reach L, but the interval of E contains the postorder number of L
- Problem 2: *non-tree edges*
  - $Q_r(L, Q)$: L can reach Q, but it cannot be derived by the intervals



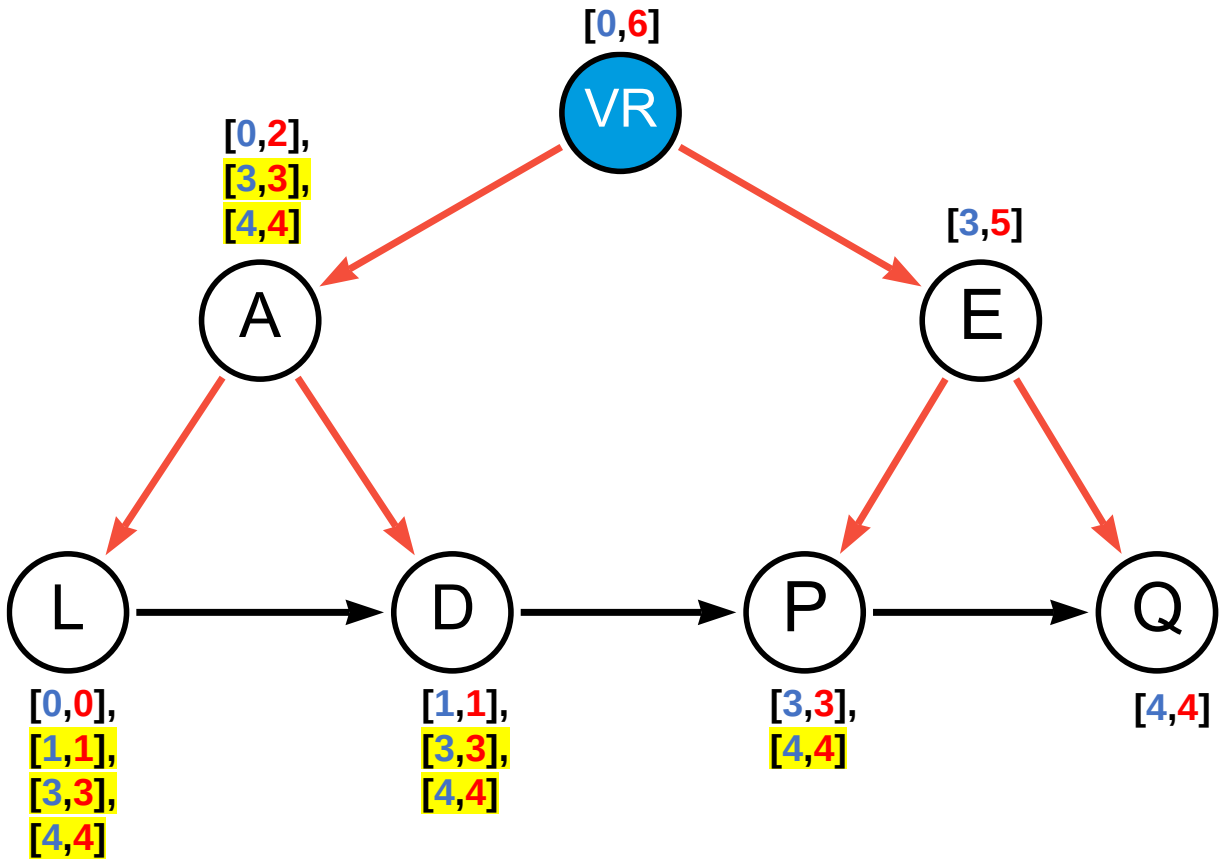
Problem 1: Spanning Forest

- Solution:
  - Creating a virtual root (VR) linking to the roots of the spanning trees
  - Computing intervals on the unified tree
- Query:
  - $Q_r(E, L)$: $b_L \notin \boldsymbol{\mathcal{L}}_E, i.e., 0 \notin [3,5]$, thus False

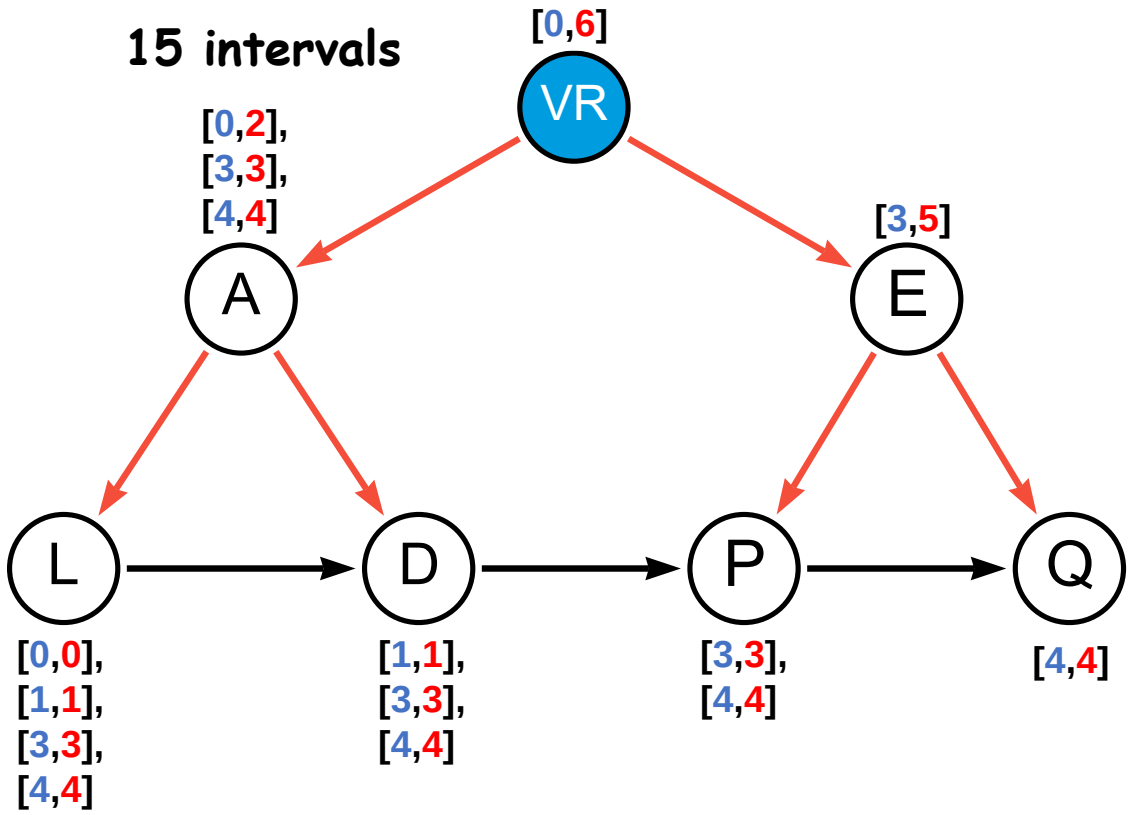


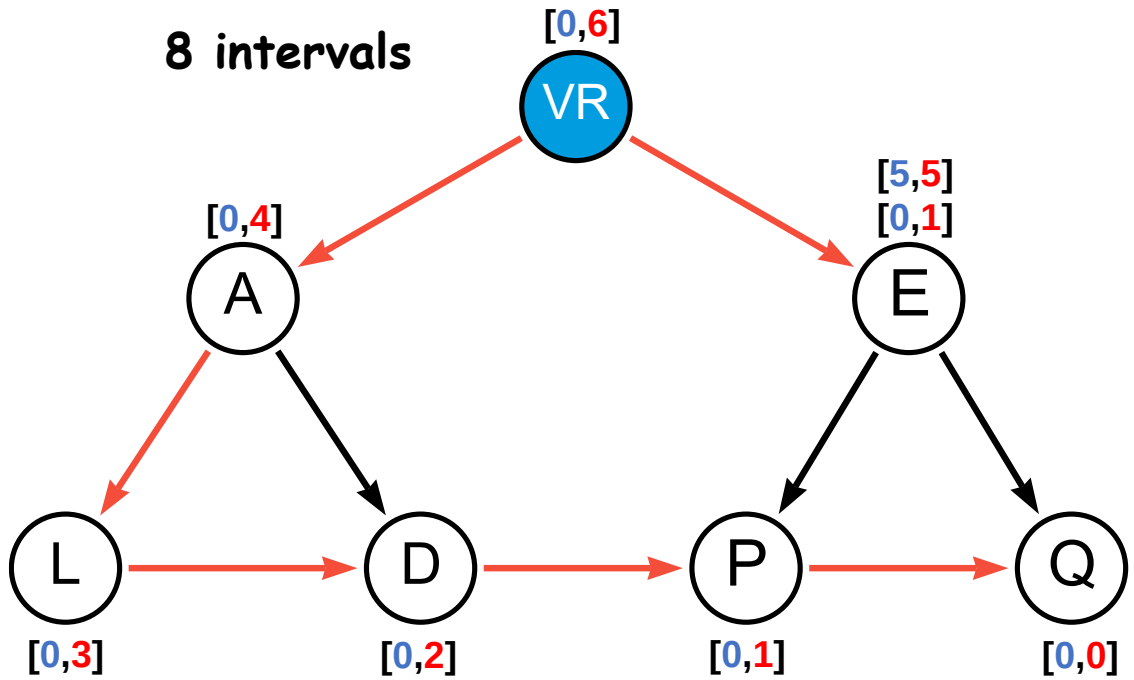
Problem 2: Non-Tree Edges

- Solution:
  - Examine vertices in the reverse topological order
  - For each edge (u, v), u inherits all the intervals of v
  - Discard **subsumed** intervals
    - E.g., A inherits [1,1] from L but [1,1] is subsumed by [0,2]
- Query:
  - $Q_r(L, Q)$: [4,4] contains 4, thus True



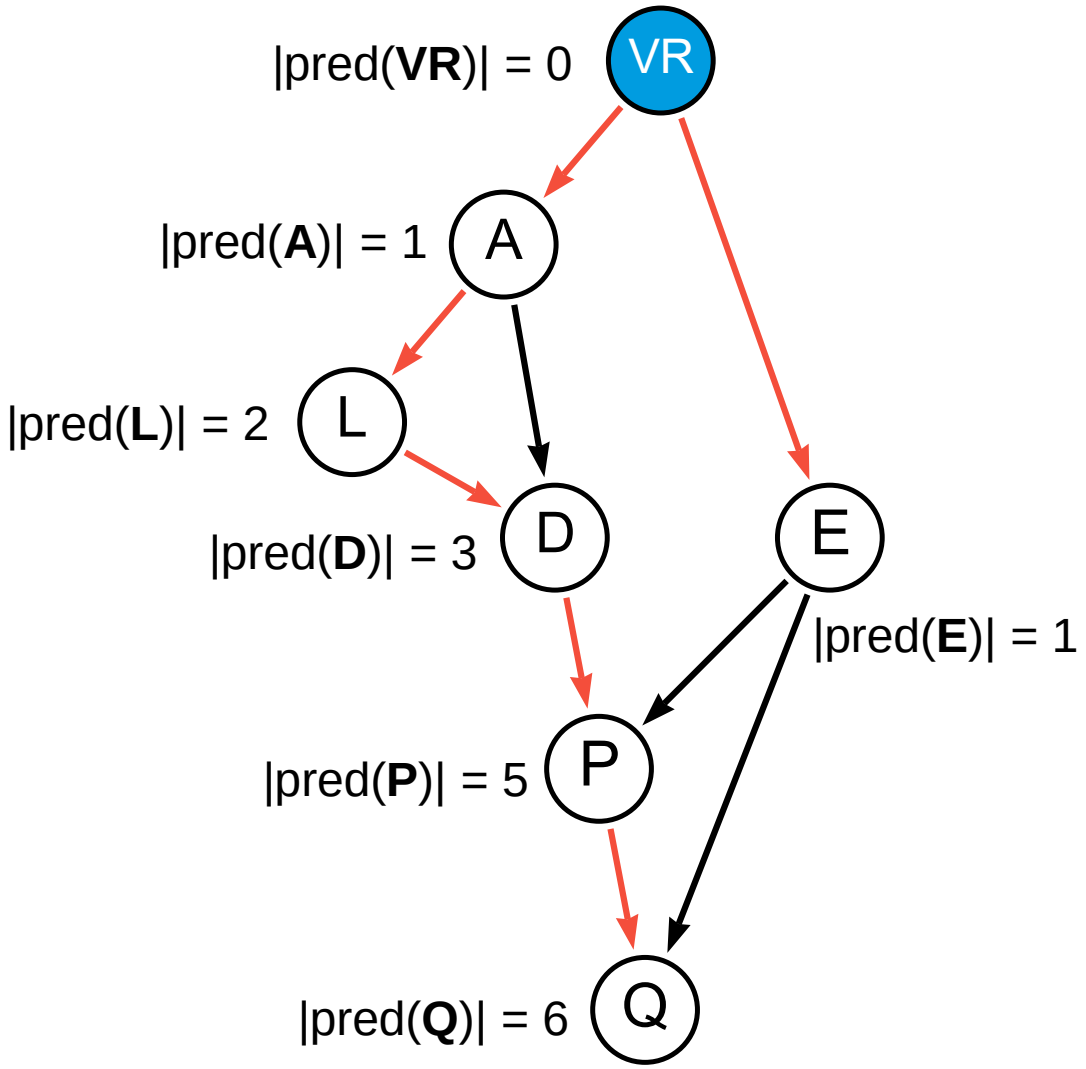
What is the impact of using different spanning trees?





Optimum Tree Cover

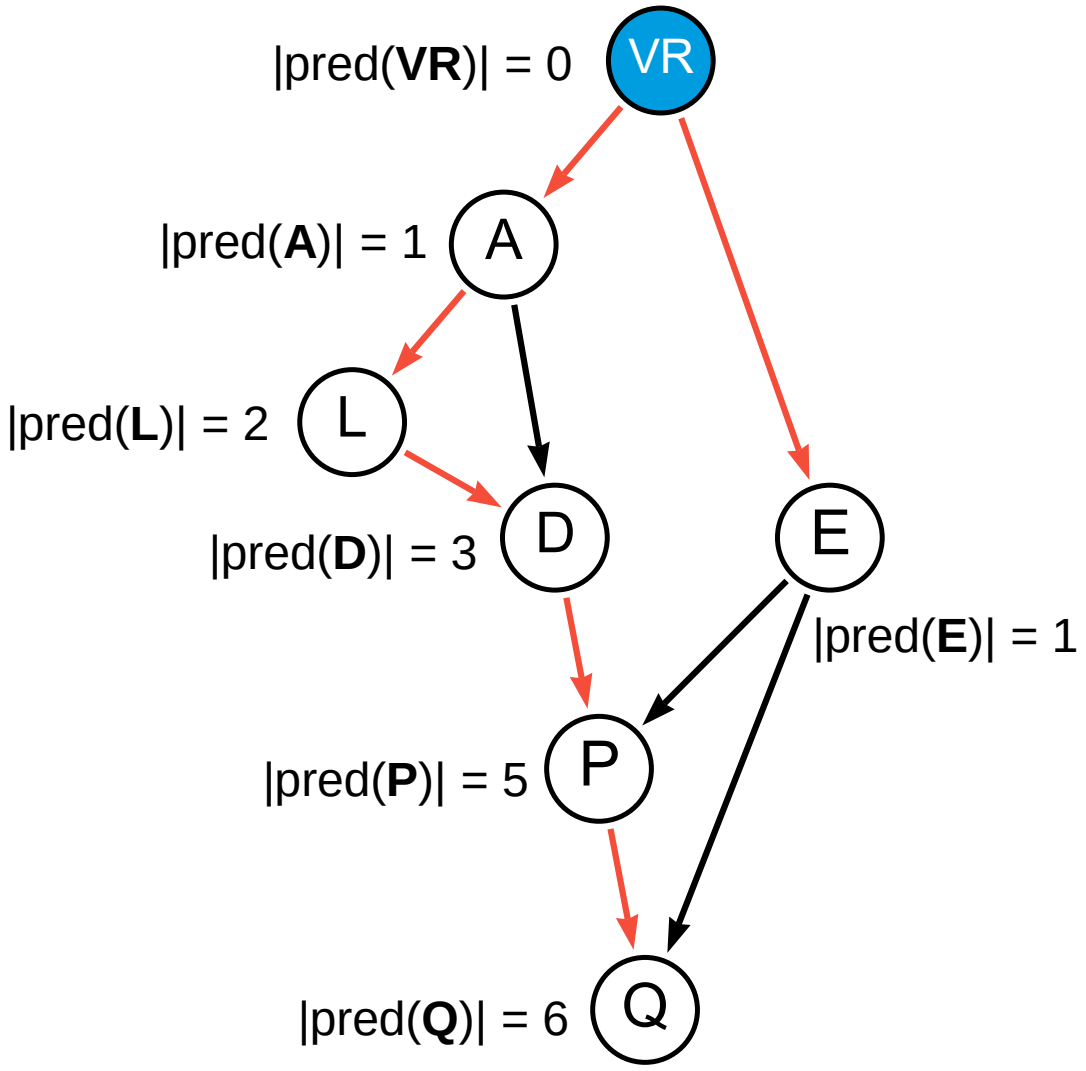
- The optimum tree cover:
  - The spanning tree that can lead to the minimum number of intervals
- Computing the optimum tree cover
  - Topologically sort DAG
  - From the top to the bottom:
    - For each vertex j, keep the incoming edge from i to j, if i has the largest number of predecessors
    - Example: For D, choosing the edge from L instead of the edge from A
  - Predecessors of each vertex are incrementally computed



R. Agrawal et al. Efficient Management of Transitive Relationships in Large Data and Knowledge Bases. SIGMOD Conference 1989: 253-262

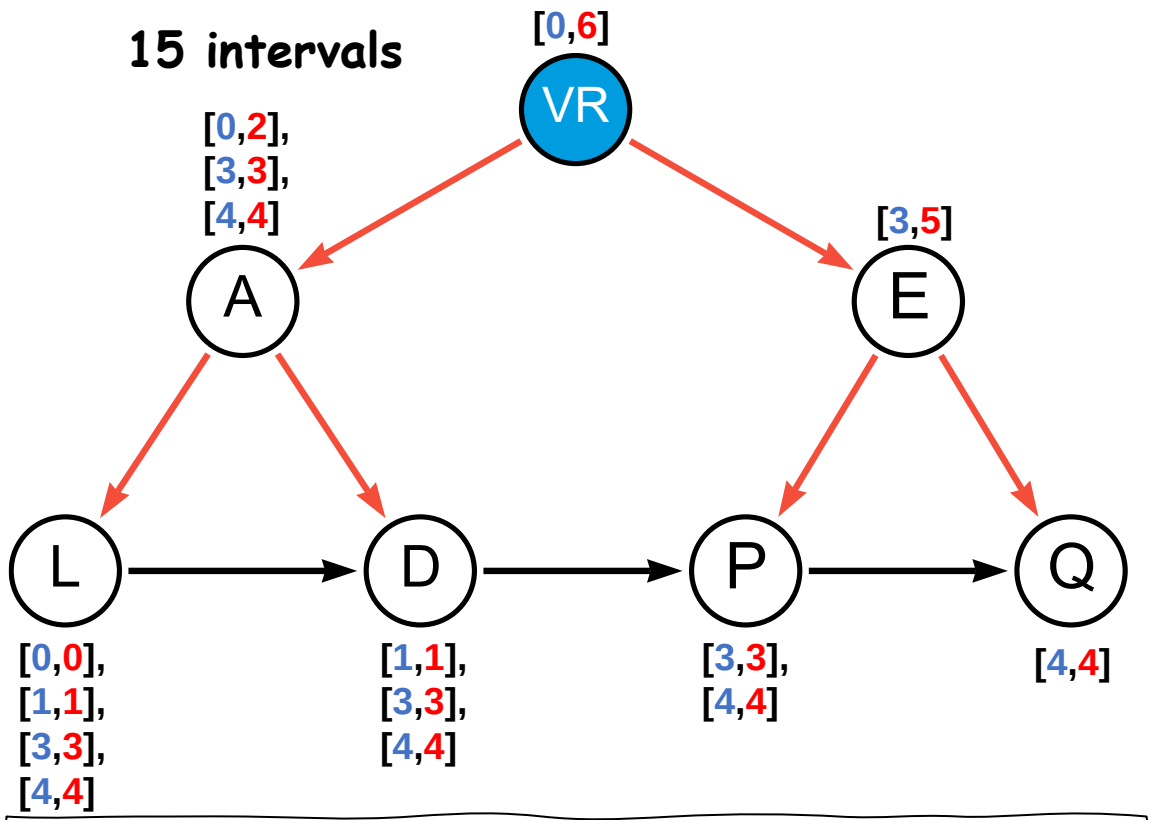
Complexity

- The optimum tree cover can be computed efficiently
- However, the complexity of computing **the minimum index** is still high
  - Because of inheriting and then removing subsumed intervals
- The same as the complexity of computing the transitive closure



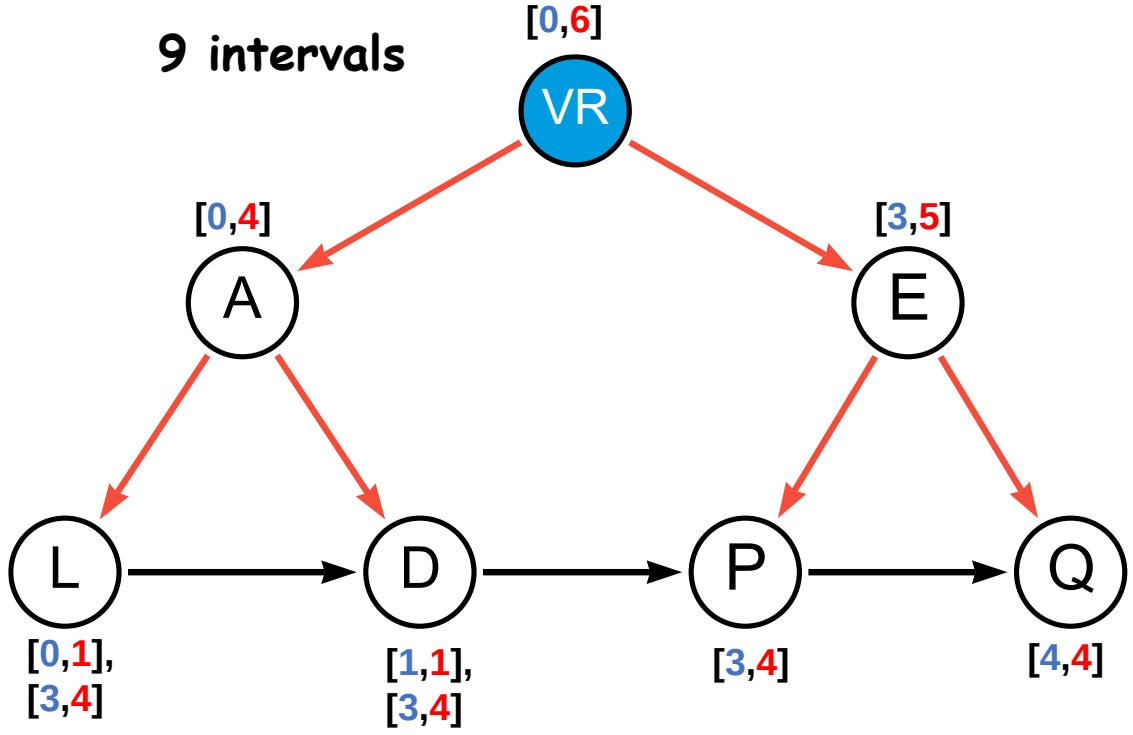
R. Agrawal et al. Efficient Management of Transitive Relationships in Large Data and Knowledge Bases. SIGMOD Conference 1989: 253-262

Optimization: Merging Intervals



Optimization: merging adjacent intervals

E.g., for vertex A, the intervals [0,2], [3,3], and [4,4] can be merged into [0,4]



The effect of merging intervals is not considered in the optimal tree cover

Summary: Tree Cover

- Building a tree-cover index
  - Transforming a general graph to a DAG
  - Computing a spanning tree (or forest) on a DAG
  - Computing intervals on the spanning tree
  - Inheriting intervals
- Bottleneck:
  - Many intervals caused by non-tree edges
- Question:
  - How to reduce the number of intervals

Reducing the Number of Intervals

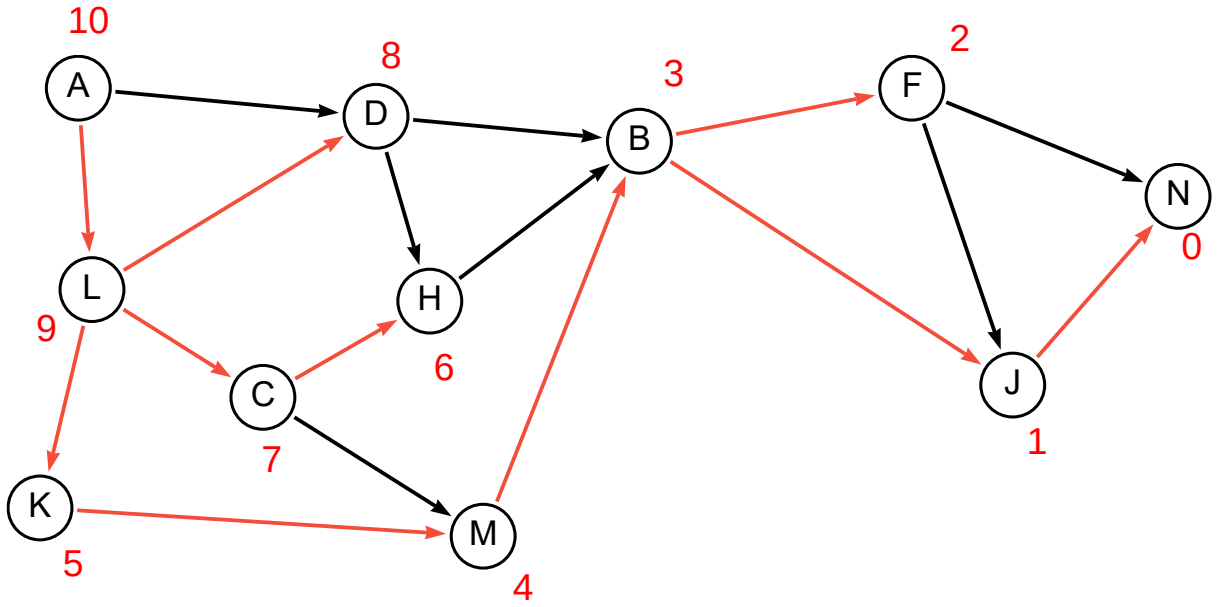
- Bounding the number of intervals
  - GRAIL [Yil10]:
    - Each vertex has **exactly k** intervals by computing k spanning trees
  - Ferrari [Seu13]:
    - Each vertex has **at most k** intervals by merging non-adjacent intervals
- Partial indexes without false negatives
- Resort to online search
  - Guided DFS by querying the incomplete indexes

[Yil10] H. Yildirim et al. GRAIL: Scalable Reachability Index for Large Graphs. Proc. VLDB Endow. 3(1): 276-284 (2010)
[Seu13] S. Seufert et al. FERRARI: Flexible and efficient reachability range assignment for graph indexing. ICDE 2013: 1009-1020

Interval Labeling in GRAIL

Assign an interval $[a_v, b_v]$ to each vertex v, denoted as $\mathcal{L}_v$

b_v: Postorder number of v



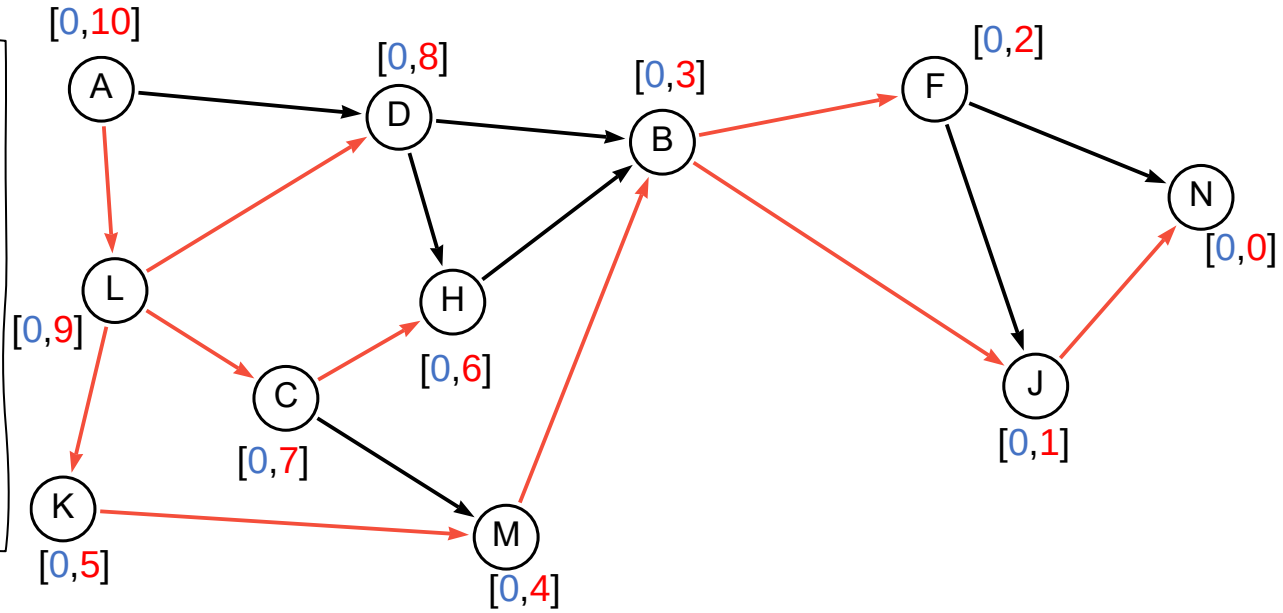
Interval Labeling in GRAIL

Assign an interval $[a_v, b_v]$ to each vertex v, denoted as $\mathcal{L}_v$

b_v: Postorder number of v

~~a_v (Tree Cover): The lowest postorder number of all the descendants of v~~

a_v (GRAIL): $min(c, b_v), c = min\{a_u: u \in outNei(v)\}$



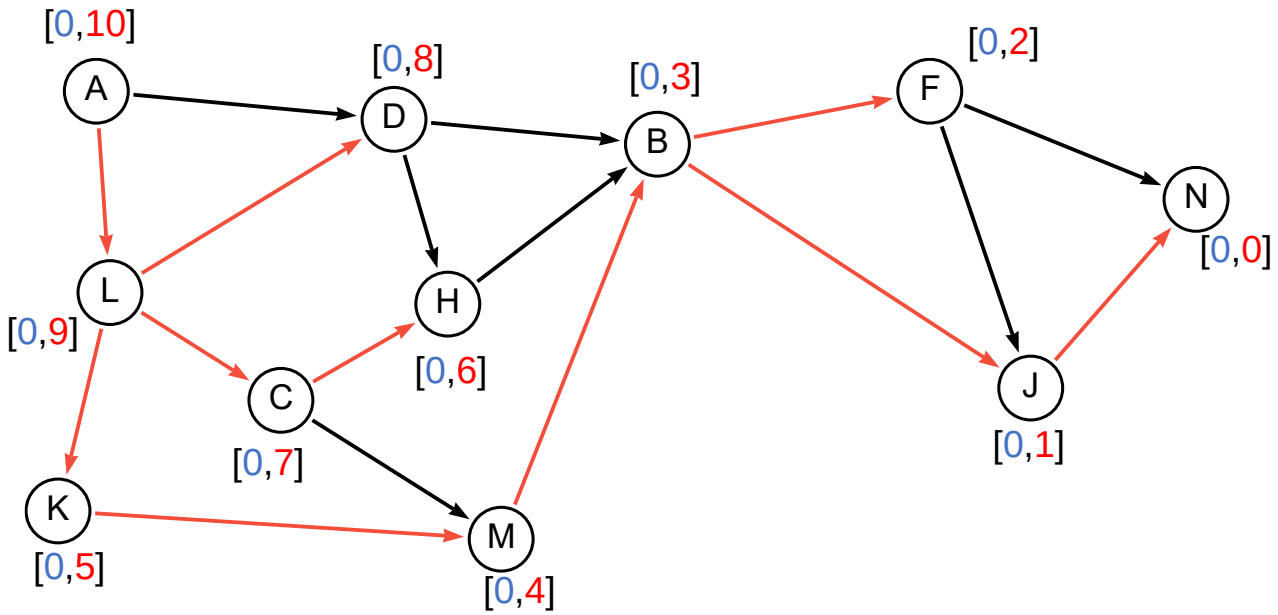
No interval inheritance in GRAIL

Reachability Processing in GRAIL

The interval labeling approach in GRAIL does not have false negatives but may have false positives

$Q_r(K, C)$
- $\mathcal{L}_C \nsubseteq \mathcal{L}_K, i.e., [0,7] \nsubseteq [0,5]$
- C is not reachable from K (**no false negatives**)

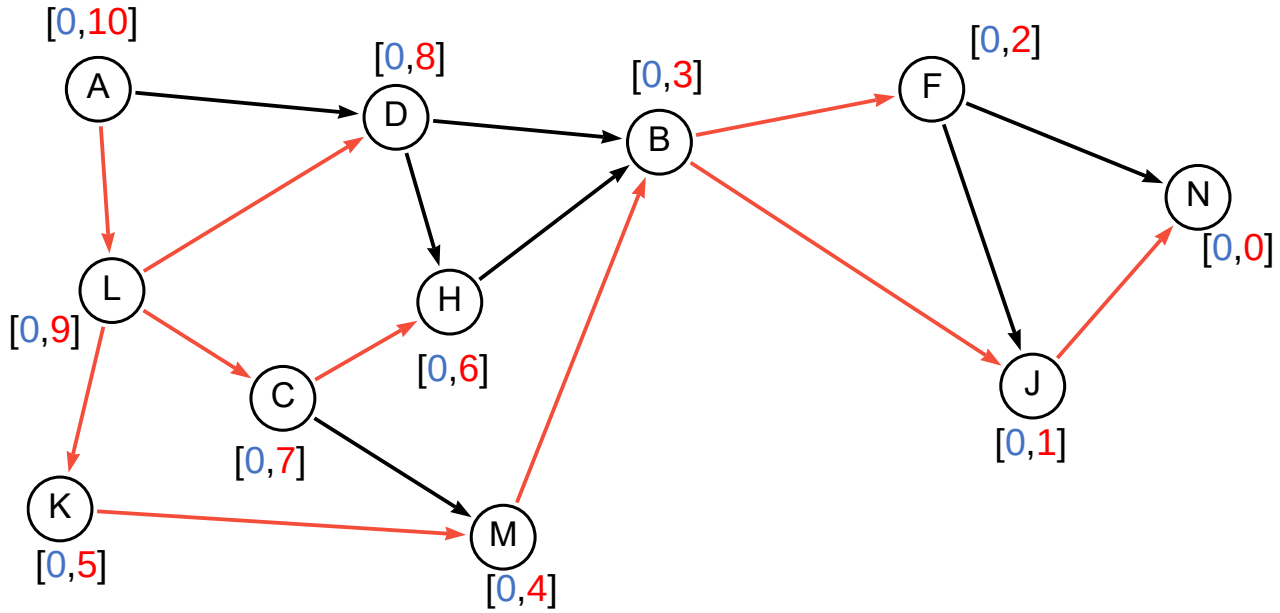
$Q_r(D, M)$
- $\mathcal{L}_M \subseteq \mathcal{L}_D, [0,4] \subseteq [0,8]$
- M is not reachable from D (**false positive**)



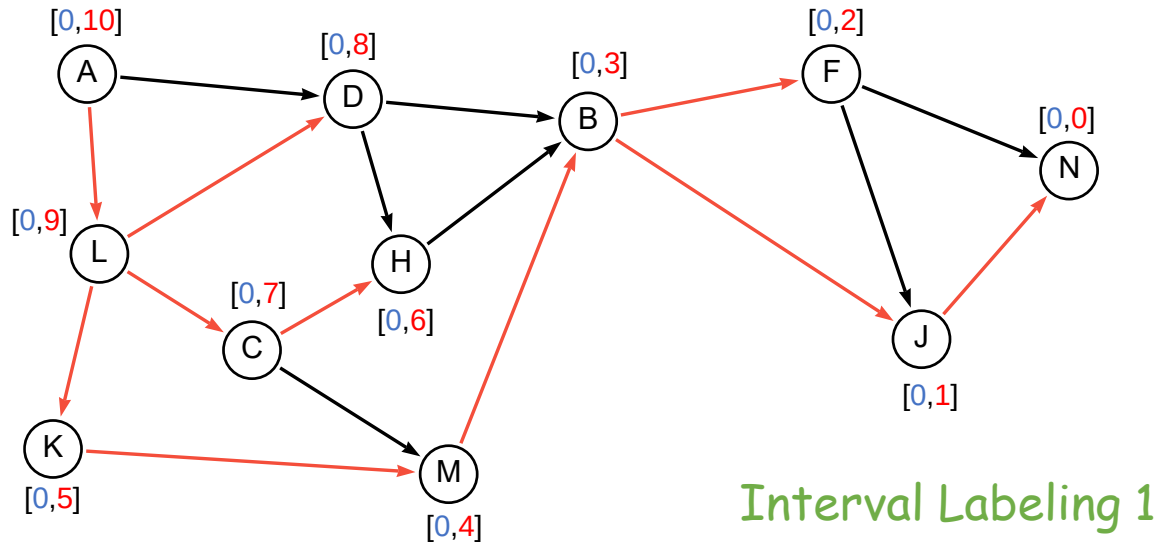
Guided DFS in GRAIL

False positive in $Q_r(D, M)$
- $\mathcal{L}_M \subseteq \mathcal{L}_D$, $[0,4] \subseteq [0,8]$
- But M is not reachable from D

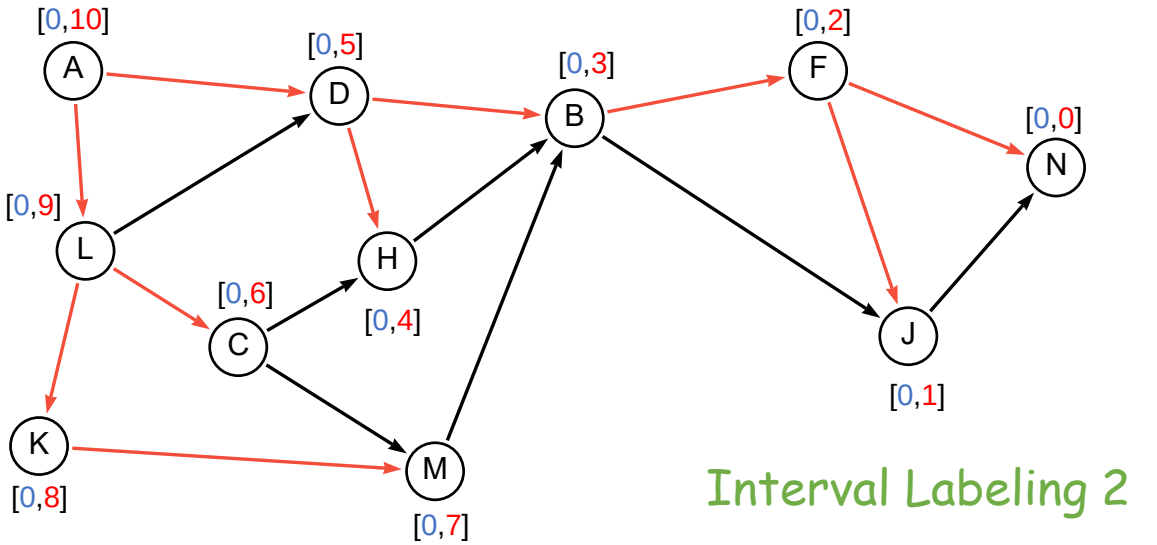
Guided DFS from D
- Outgoing neighbours of D
  - B: is M reachable from B?
    - $\mathcal{L}_M \not\subseteq \mathcal{L}_B$, thus B is not further explored
  - H: is M reachable from H?
    - $\mathcal{L}_M \subseteq \mathcal{L}_H$, i.e., false positive, then visiting outgoing neighbours of H
      - B: M is not reachable from B
  - Return False



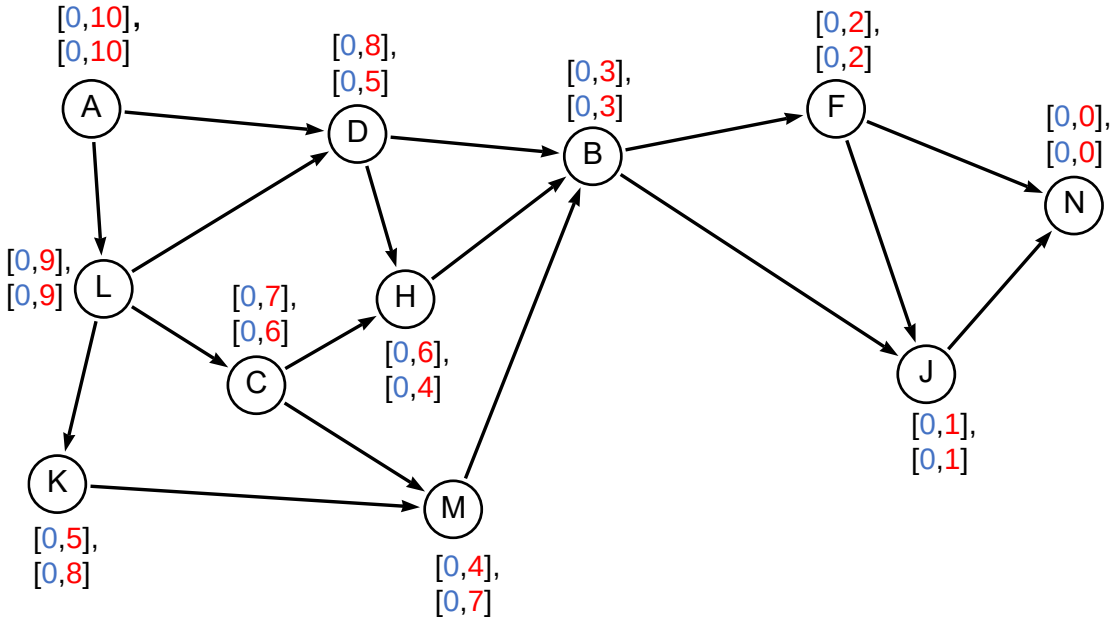
K Intervals in GRAIL



Interval Labeling 1



Interval Labeling 2



K Intervals in GRAIL

Each vertex v has k intervals

- $\mathcal{L}_v = (\mathcal{L}_v^1, \ldots, \mathcal{L}_v^k)$

Query processing: $Q_r(s,t)$

- $\mathcal{L}_t \subseteq \mathcal{L}_s$ **if and only if** $\mathcal{L}_t^i \subseteq \mathcal{L}_s^i$ for all $i \in [1,k]$

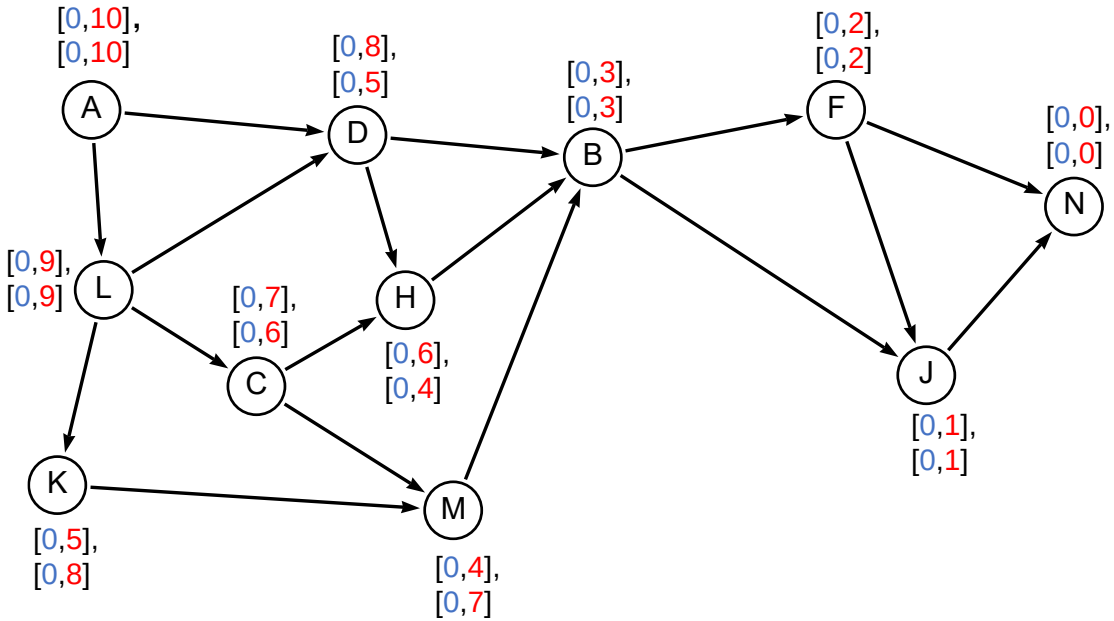
Example: $k = 2$

$Q_r(D,M)$

- $\mathcal{L}_M \not\subseteq \mathcal{L}_D$, because $[0,4] \subseteq [0,8]$ but $[0,7] \not\subseteq [0,5]$
- M is not reachable from D

Optimization: topological ordering

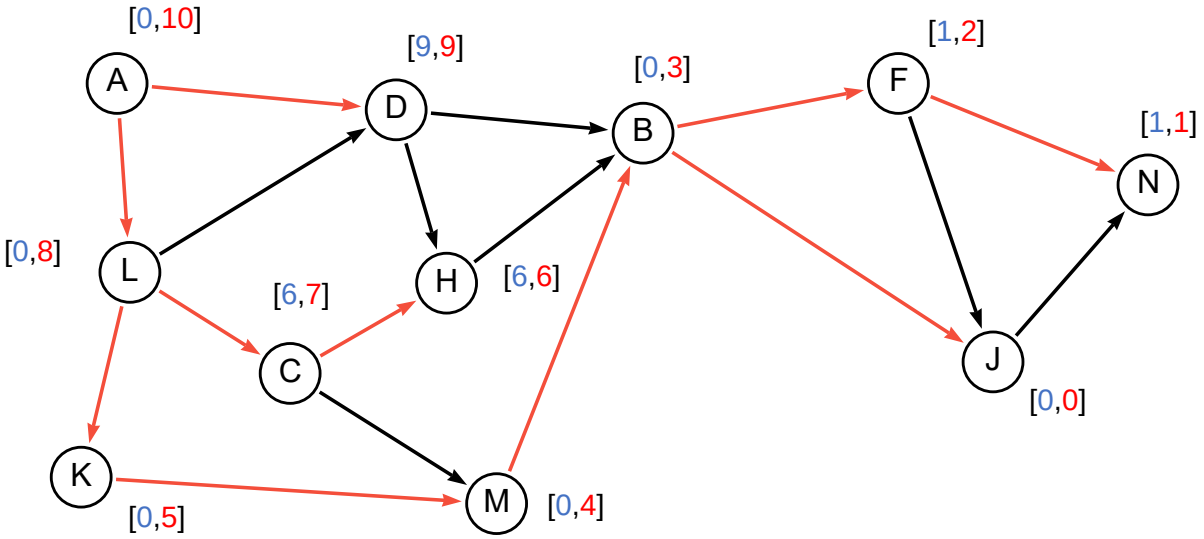
False positives might still exist, which calls for the guided DFS



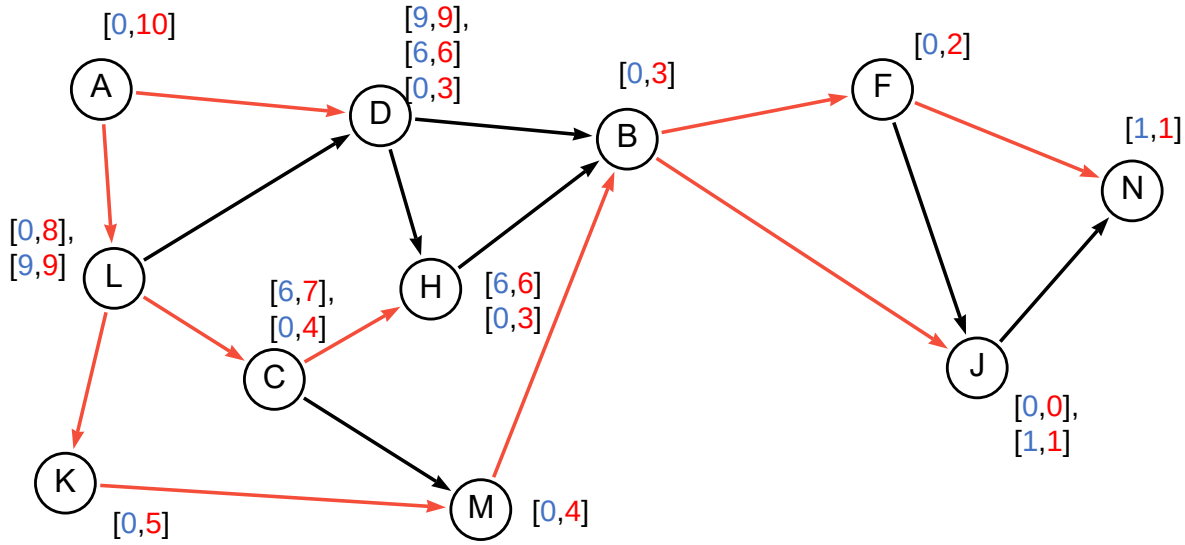
FERRARI

1. Using the tree-cover approach to compute the intervals of vertices in the DAG

1.1. Interval labeling on a spanning tree



1.2. Inheriting and merging intervals



Merging intervals without gaps

- F: inheriting $[0,0]$ from J, then merging it with $[1,2]$ to have $[0,2]$

FERRARI

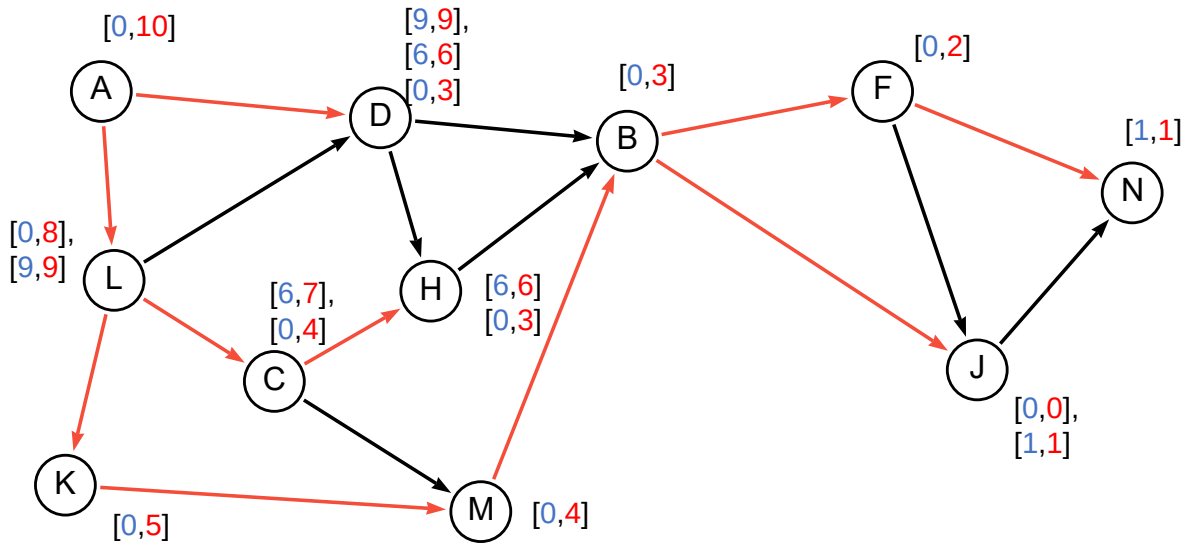
2. Merging intervals with gaps to have at most k intervals for each vertex

Merging intervals with gaps

- $[i, j]$ and $[i', j']$ are merged into $[i, j']^*$ if $i' > j + 1$
- $[i, j']^*$ is an approximate interval

Example: $k = 2$

- D: [0,3],[6,6], [9,9]



FERRARI

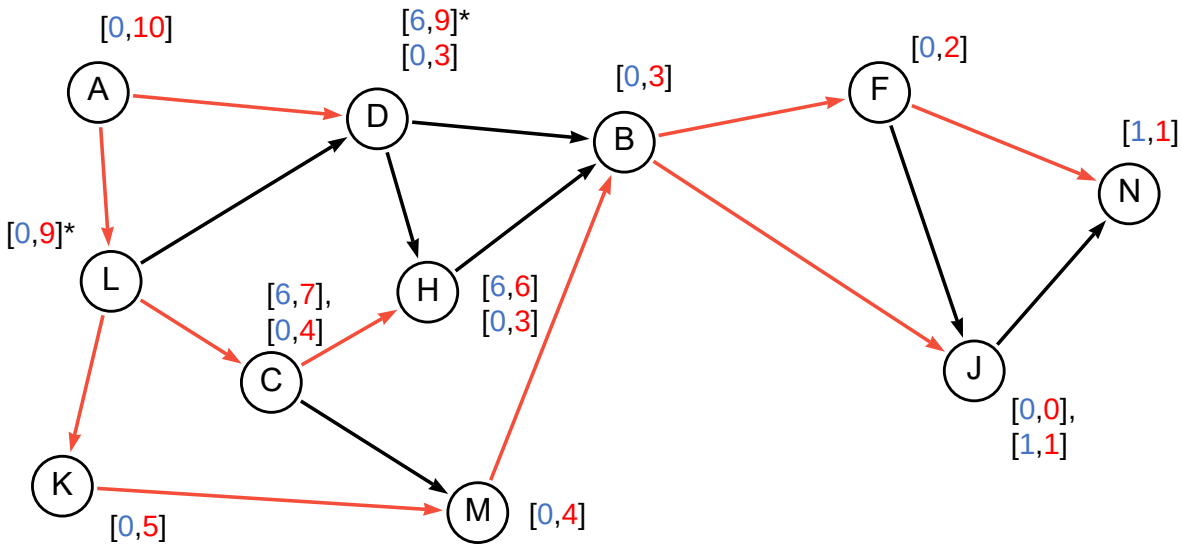
2. Merging intervals with gaps to have at most k intervals for each vertex

Merging intervals with gaps

- $[i, j]$ and $[i', j']$ are merged into $[i, j']^*$ if $i' > j + 1$
- $[i, j']^*$ is an approximate interval

Example: $k = 2$

- D: $[0,3]$,$[6,6]$, $[9,9]$
  - Option 1 (bad): $[0,6]^*$, $[9,9]$
  - Option 2 (good): $[0,3]$, $[6,9]^*$
  - FERRARI uses **Option 2** as the size of the approximate interval is smaller
- L: inheriting $[6,9]^*$ from D, and then merging it with $[0,8]$ to have $[0,9]^*$ (merging without gaps)



Query Processing in FERRARI

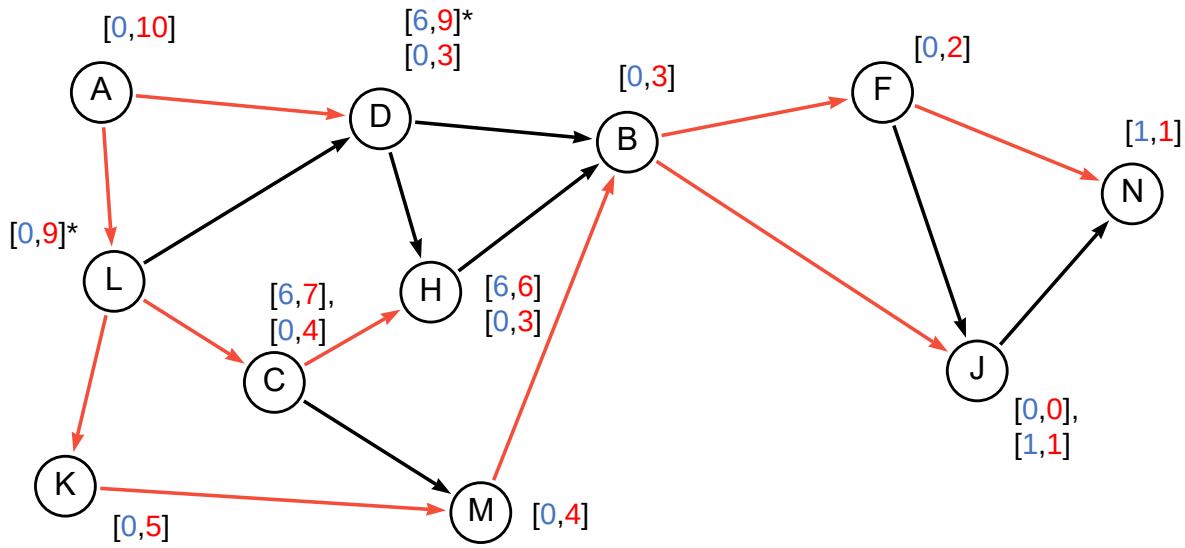
Exact intervals: no false positive, but may have false negatives
Approximate intervals: no false negatives, but may have false positives

$Q_r(D, N)$:

- Postorder number of N: 1
- Exact interval of D: $[0,3]$
  - $1 \in [0,3]$ (no false positive)
- Return True

$Q_r(D, K)$:

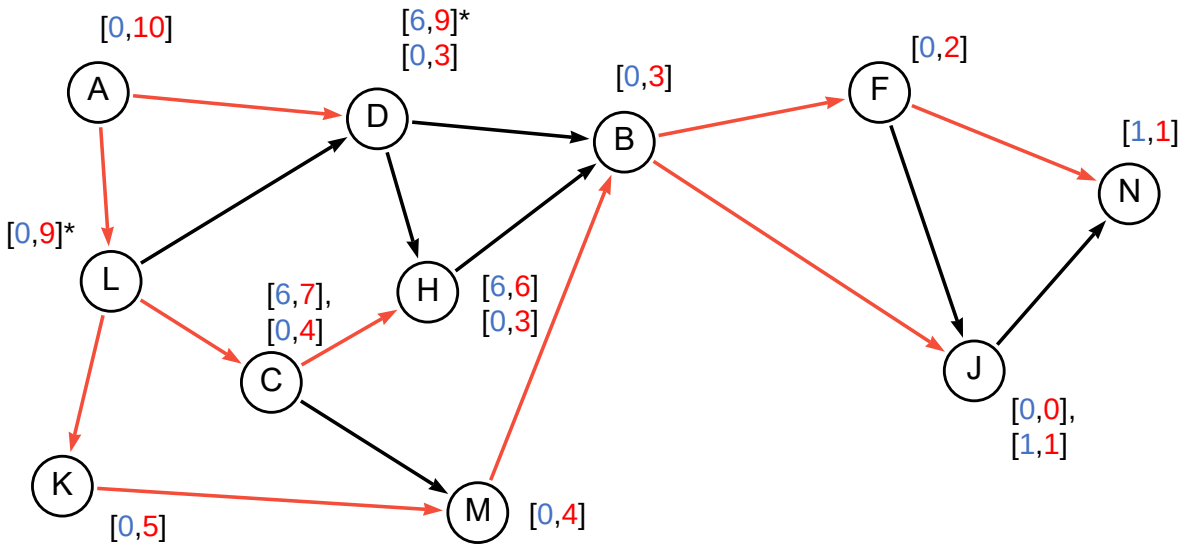
- Postorder number of K: 5
- Exact interval of D: $[0,3]$
  - $5 \notin [0,3]$
- Approximate interval of D: $[6,9]^*$
  - $5 \notin [6,9]^*$ (no false negative)
- Return False



Query Processing in FERRARI

$Q_r(D, C)$:

- Postorder number of C: 7
- Approximate interval of D: $[6,9]^*$
  - $7 \in [6,9]^*$ (false positives)
- Guided DFS from D
- Return False



Other Interval-Based Techniques

- Reachability for non-tree edges:
  - Tree + SSPI [Chen05]
    - Computing predecessors that are incident to non-tree edges
  - Dual-labeling [Wan06]
    - Computing the transitive closure for the reachability of non-tree edges
  - GRIPP [Tri07]
    - Recursive using intervals for reachability of non-tree edges

[Chen05] L. Chen et al. Stack-based Algorithms for Pattern Matching on DAGs. VLDB 2005: 493-504
[Wan06] H. Wang et al. Dual Labeling: Answering Graph Reachability Queries in Constant Time. ICDE 2006: 75
[Tri07] S. Tril et al. Fast and practical indexing and querying of very large graphs. SIGMOD Conference 2007: 845-856

Other Interval-Based Techniques

- Extended covers: Path-tree labeling [Jin08, Jin11]
  - Path-tree cover:
    - Spanning trees on a path-tree graph that is constructed based on a path-partition
  - Labeling in the path-tree graph:
    - Interval labeling with additional information on vertices in paths
  - Reachability for non-tree edges in the path-tree graph:
    - Transitive closure, i.e., the dual-labeling approach

[Jin08] R. Jin et al. Efficiently answering reachability queries on very large directed graphs. SIGMOD Conference 2008: 595-608
[Jin11] R. Jin et al. Path-tree: An efficient reachability indexing scheme for large directed graphs. ACM Trans. Database Syst. 36(1): 7:1-7:44 (2011)

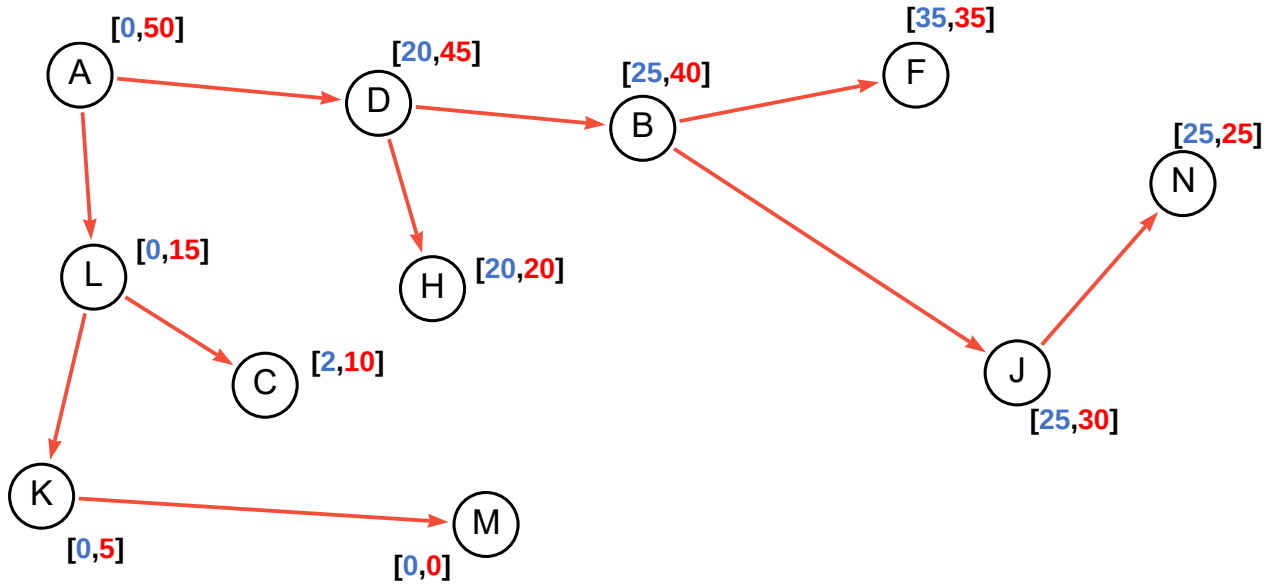
Tree-Cover Indexes on Dynamic Graphs

- Two level reachability in tree-cover-based indexes
  1. Within strongly connected components (SCC)
  2. Within DAGs
- Edge insertion or deletion can lead to
  - SCC maintenance: merging or splitting strongly connected components
    - Requiring recomputation
  - Interval maintenance: updating interval labeling
    - Tailored design for optimization

Our focus in this tutorial

Tree-Cover Indexes on Dynamic Graphs

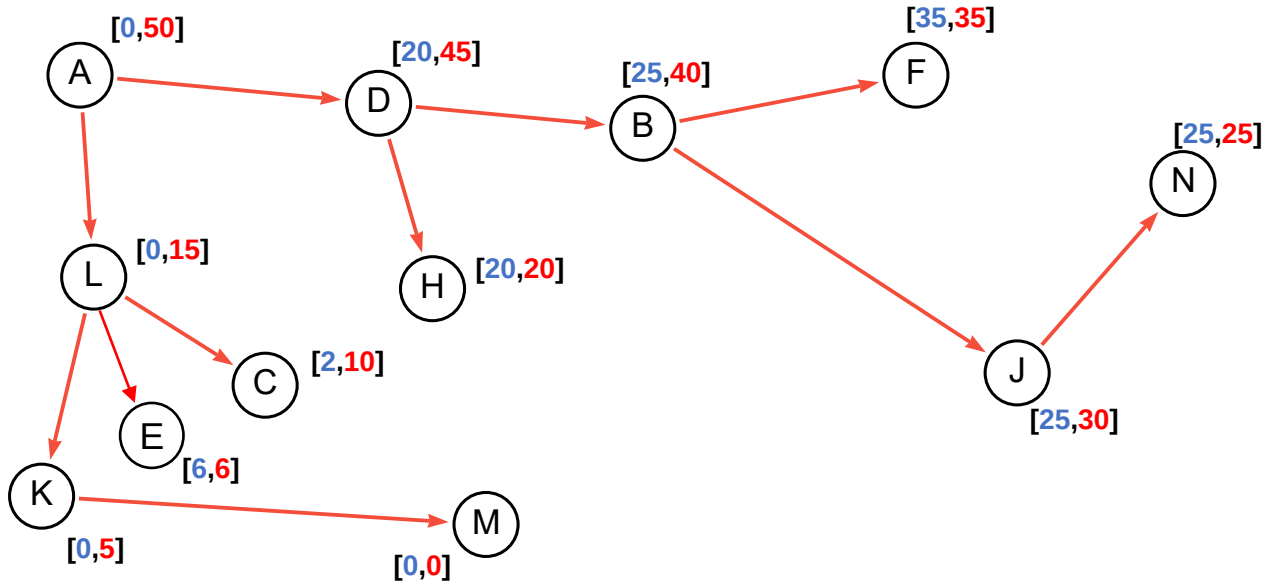
- Tree-Cover [Agr89]:
  - The postorder numbers assigned to vertices **do not** need to be **contiguous**
    - Contiguous:
      0, 1, 2, 3, 4, 5, 6, 7, 8, 10
    - Noncontiguous:
      0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50
- Tailored design:
  - Leveraging the ‘gaps’ between **noncontiguous** postorder numbers to deal with tree edge insertions



[Agr89] R. Agrawal et al. Efficient Management of Transitive Relationships in Large Data and Knowledge Bases. SIGMOD Conference 1989: 253-262

Tree-Cover Indexes on Dynamic Graphs

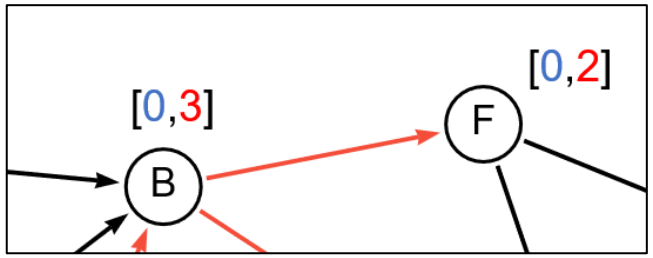
- Example:
  - Inserting edge (L, E)
  - The interval of E: **[6,6]**



[Agr89] R. Agrawal et al. Efficient Management of Transitive Relationships in Large Data and Knowledge Bases. SIGMOD Conference 1989: 253-262

Tree-Cover Indexes on Dynamic Graphs

- Dagger [Yil13]
  - Extension of GRAIL [Yil10] to dynamic graphs
  - *Partial index without false negatives*
  - *False positives are allowed in the index*
- Tailored design:
  - If an edge in DAG is deleted, no need for updates
    - Only introducing false positives without invalidating the index
- Example:
  - Deleting the edge (B, F) without changing intervals only introduces false positives



[Yil10] H. Yildirim et al. GRAIL: Scalable Reachability Index for Large Graphs. Proc. VLDB Endow. 3(1): 276-284 (2010)
[Yil13] H. Yildirim et al. DAGGER: A Scalable Index for Reachability Queries in Large Dynamic Graphs. CoRR abs/1301.0977 (2013)

Outline

1. Plain Reachability Indexes
    a) Tree-Cover Indexes
    b) 2-Hop Indexes
    c) Approximate Transitive Closures
    d) Other Techniques
2. Path-Constrained Reachability Indexes
    a) Indexes for Alternation-Based Queries
    b) Indexes for Concatenation-Based Queries
3. Open Challenges

Path Concatenation

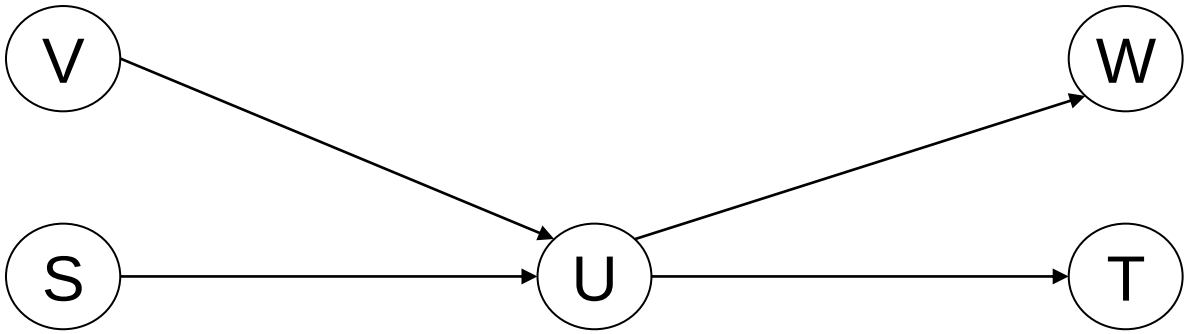
source \ target	S	U	T
S		1	1
U			1
T			

The existence of the path from S to T can be derived by the existence of

- the path from S to U, and
- the path from U to T

Path Concatenation

source \ target	S	U	T	V	W
S		1	1		1
U			1		1
T					
V		1	1		1
W					



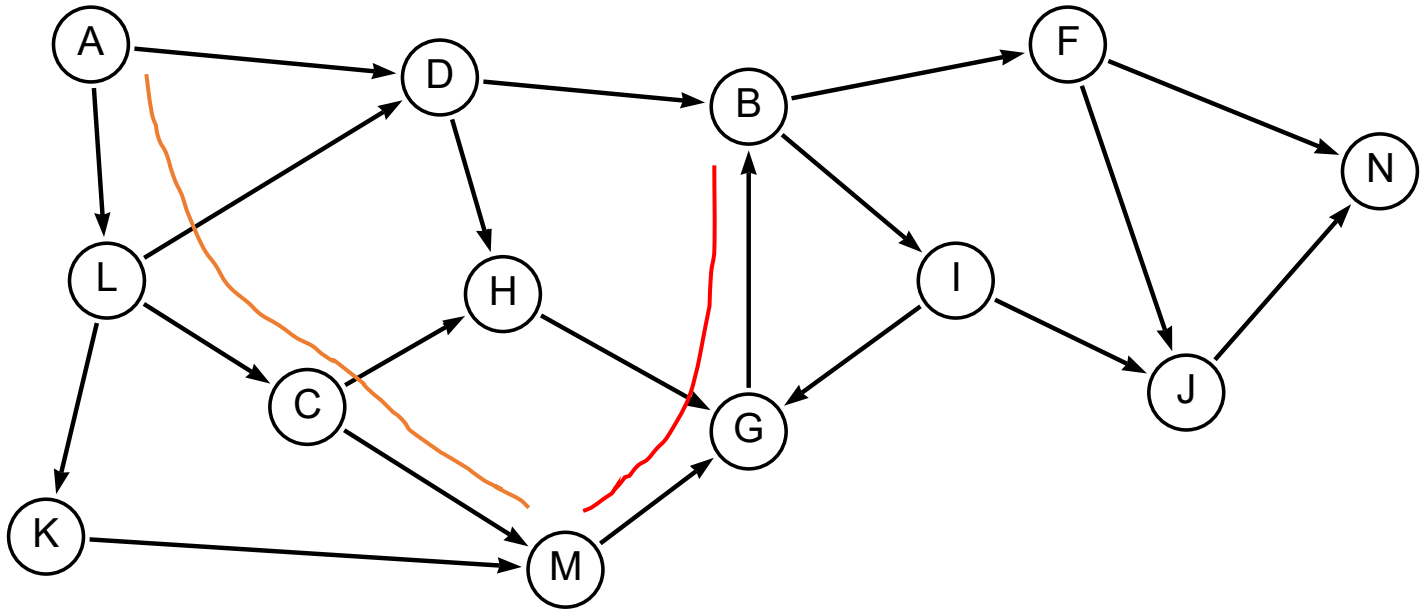
The number of the recordings in TC can be reduced

2-Hop Labeling

Assigning $L(v) = (L_{in}(v), L_{out}(v))$ for each v, such that

$\forall\, u \in L_{in}(v)$, $\exists$ a path from u to v

$\forall\, w \in L_{out}(v)$, $\exists$ a path from v to w

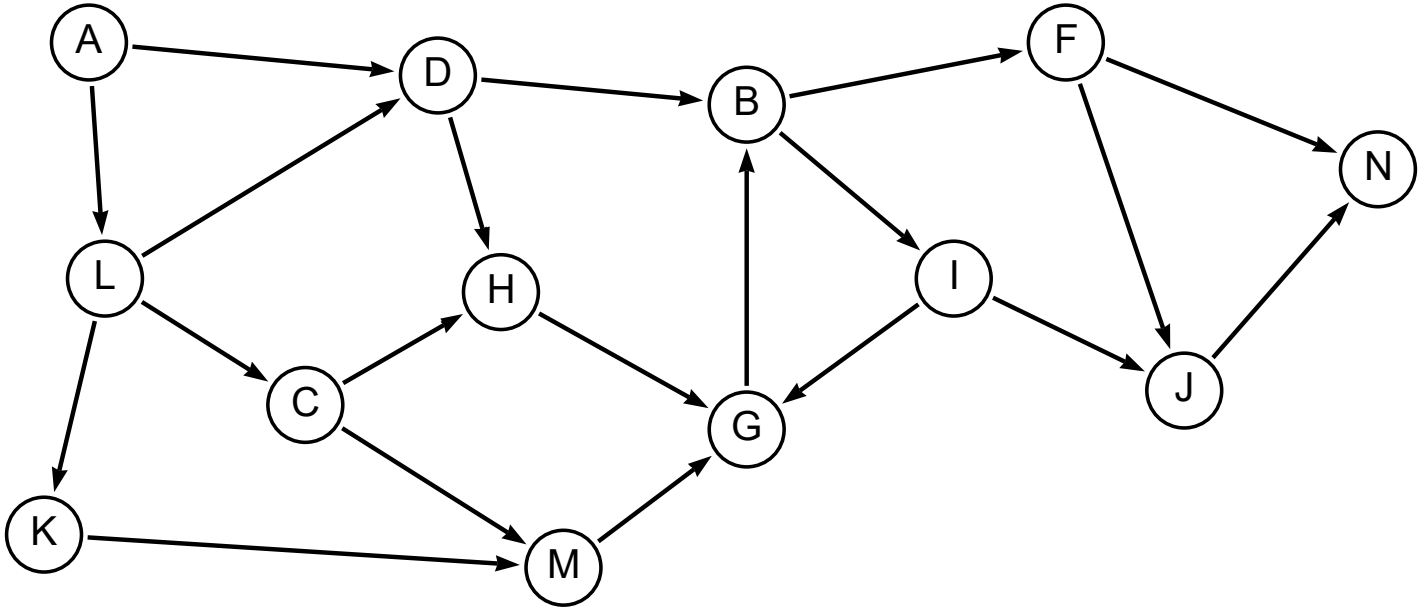


v	$L_{in}(v)$	$L_{out}(v)$
A		M, D, C, K
B	M, D, C, B	
C		M
D		
F	M, D, C, B	N
G	M, D, C, B	B
H	D, C	B, G
I	M, D, C, B	N, G
J	M, D, C, B, F, I	N
K	A	M
L	A	M, D, C, K
M		
N	M, D, C, B	

The 2-hop labeling can be computed directly on a general graph (no need for transformation)

E. Cohen et al. Reachability and distance queries via 2-hop labels. SODA 2002: 937-946

2-Hop Labeling



Case 1: $Q(L, M) = True,\ M \in L_{out}(L)$

Case 2: $Q(M, B) = True,\ M \in L_{in}(B)$

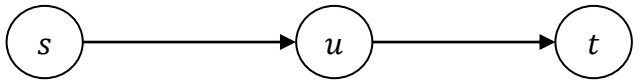
Case 3: $Q(A, N) = True,\ L_{out}(A) \cap L_{in}(N) \neq \emptyset$

v	$L_{in}(v)$	$L_{out}(v)$
A		M, D, C, K
B	M, D, C, B	
C		M
D		
F	M, D, C, B	N
G	M, D, C, B	B
H	D, C	B, G
I	M, D, C, B	N, G
J	M, D, C, B, F, I	N
K	A	M
L	A	M, D, C, K
M		
N	M, D, C, B	

E. Cohen et al. Reachability and distance queries via 2-hop labels. SODA 2002: 937-946

Minimum 2-Hop Labeling

- Index size: $\sum_{v \in V} |L_{in}(v)| + |L_{out}(v)|$
- **Minimum** 2-hop labeling: the index with the minimum index size
  - Intuition: maximumly compress the transitive closure
- NP-hard problem [Coh02]
- Efficient heuristics for building 2-hop indexes
  - TFL [Che13], PLL [Aki13], DL [Jin13], and TOL [Zhu14]



Labeling 1

v	$L_{in}(v)$	$L_{out}(v)$
s		u
u		
t	u	

Smaller

Labeling 2

v	$L_{in}(v)$	$L_{out}(v)$
s		
u	s	t
t	s	

Larger

[Coh02] E. Cohen et al. Reachability and distance queries via 2-hop labels. SODA 2002: 937-946
[Che13] J. Cheng et al. TF-Label: a topological-folding labeling scheme for reachability querying in a large graph. SIGMOD Conference 2013: 193-204
[Jin13] R. Jin et al. Simple, Fast, and Scalable Reachability Oracle. Proc. VLDB Endow. 6(14): 1978-1989 (2013)
[Aki13] E. Akiba et al. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. SIGMOD Conference 2013: 349-360
[Zhu14] A. Zhu et al. Reachability queries on large dynamic graphs: a total order approach. SIGMOD Conference 2014: 1323-1334

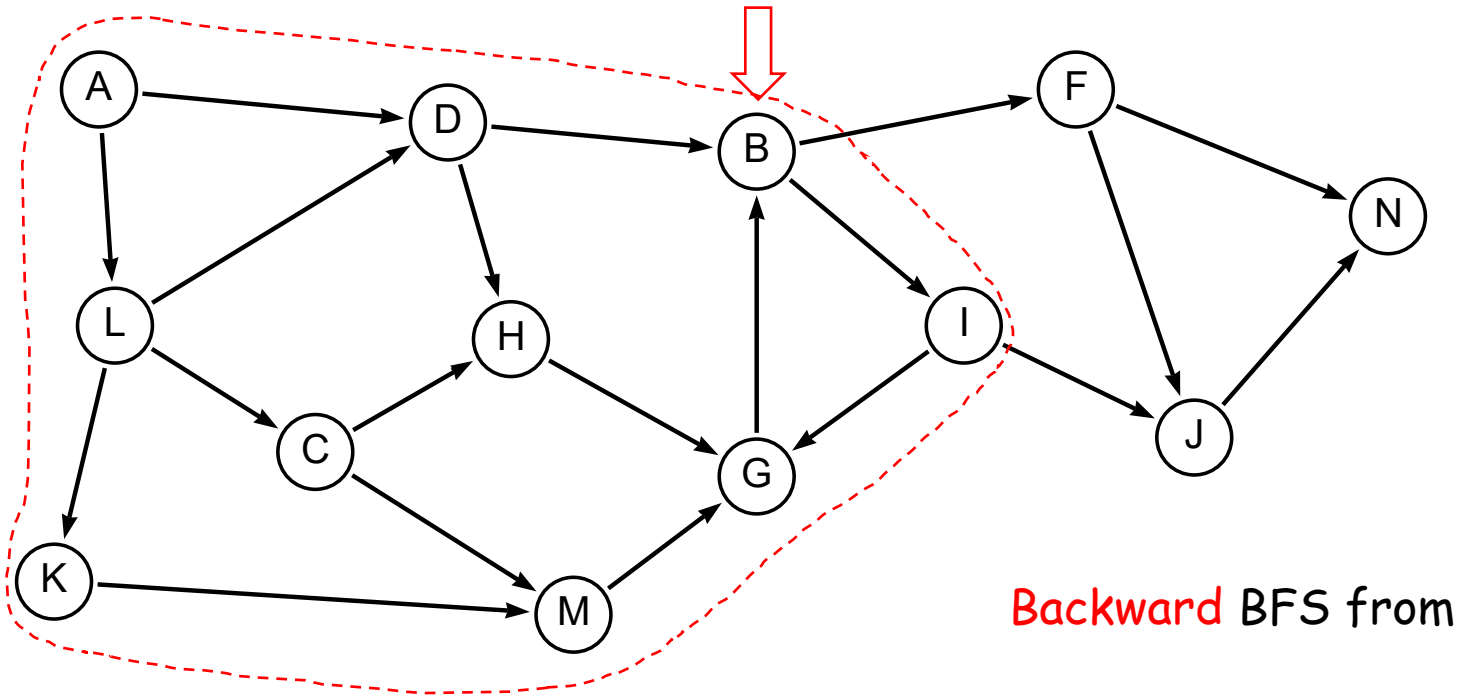
Pruned Landmark Labeling (PLL)

- PLL [Aki13, Yan13]:
  - Vertex accessing order :
    - Example: descending order according to $(d_{in}(v) + 1) \times (d_{out}(v) + 1)$
  - Iteratively processing vertices according to the accessing order:
    - Performing backward and forward BFS from each vertex to compute L_{out} and L_{in}
  - Pruning the search space for each BFS:

[Aki13] E. Akiba et al. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. SIGMOD Conference 2013: 349-360

[Yan13] Y. Yano, et al. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In CIKM'13.

PLL

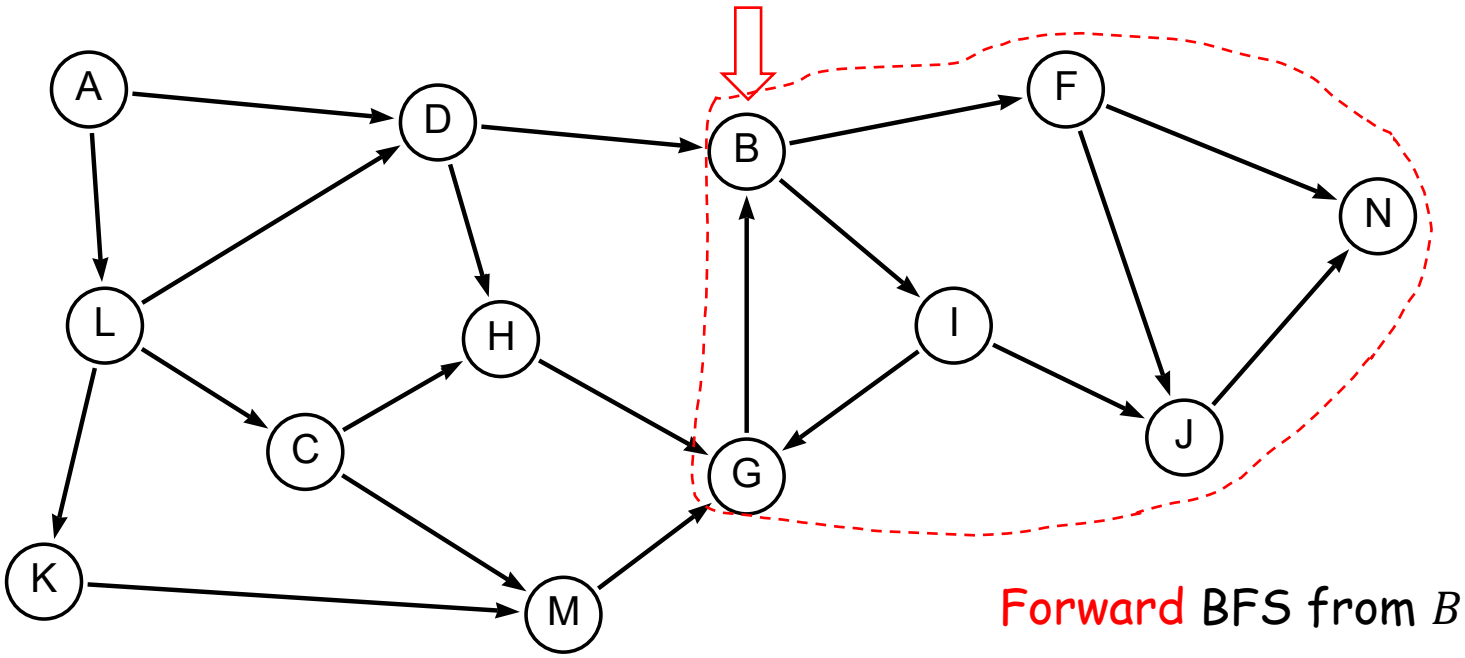


Backward BFS from B

Vertex accessing order: $(B, D, G, L, C, F, H, I, J, M, K, A, N)$

v	$L_{in}(v)$	$L_{out}(v)$
A		B
B		B
C		B
D		B
F		
G		B
H		B
I		B
J		
K		B
L		B
M		B
N		

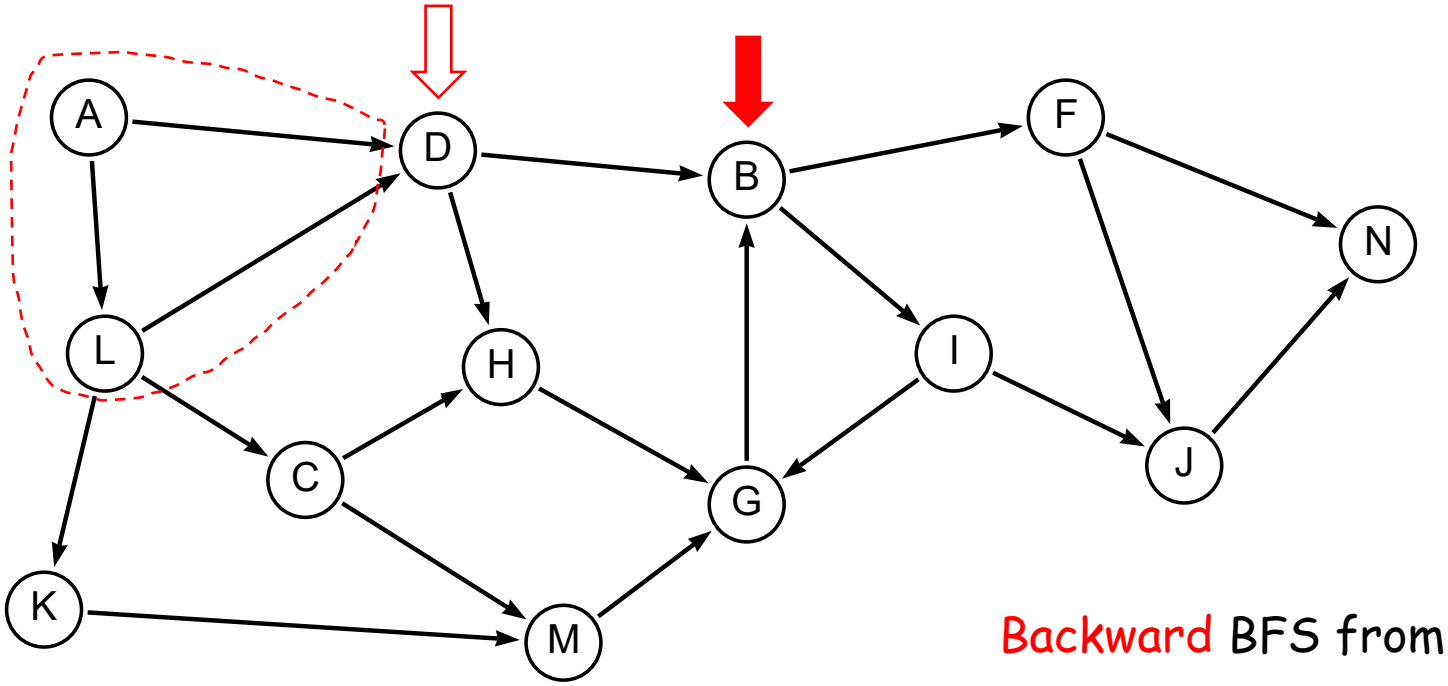
PLL



Vertex accessing order: $(B, D, G, L, C, F, H, I, J, M, K, A, N)$

v	$L_{in}(v)$	$L_{out}(v)$
A		B
B		B
C		B
D		B
F	B	
G	B	B
H		B
I	B	B
J	B	
K		B
L		B
M		B
N	B	

PLL



Backward BFS from D

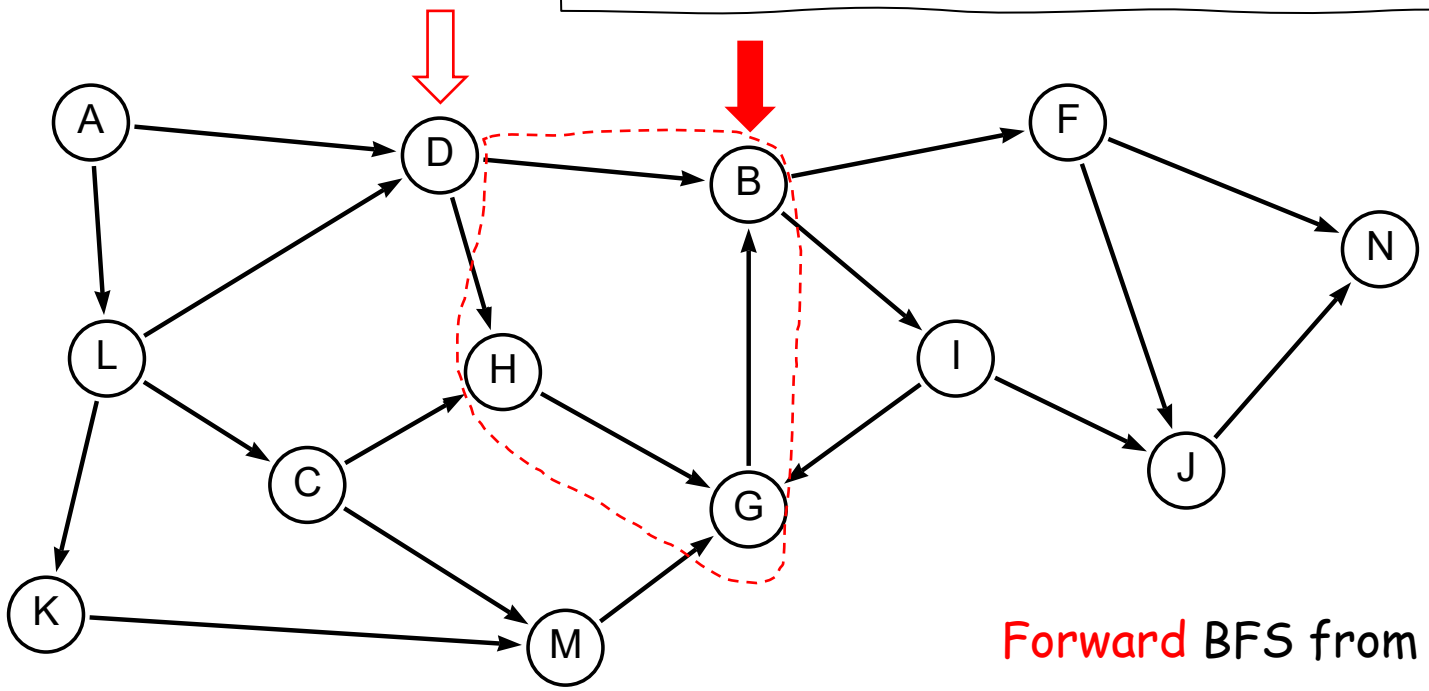
Vertex accessing order: $(B, D, G, L, C, F, H, I, J, M, K, A, N)$

v	$L_{in}(v)$	$L_{out}(v)$
A		B, D
B		B
C		B
D		B
F	B	
G	B	B
H		B
I	B	B
J	B	
K		B
L		B, D
M		B
N	B	

PLL

B is before D in the order so that the BFS from D skips B and the descendants of B

Why? All such reachability has been recorded in the BFSs from B, e.g., the reachability from D to N is recorded as $B \in L_{out}(D)$ and $B \in L_{in}(N)$

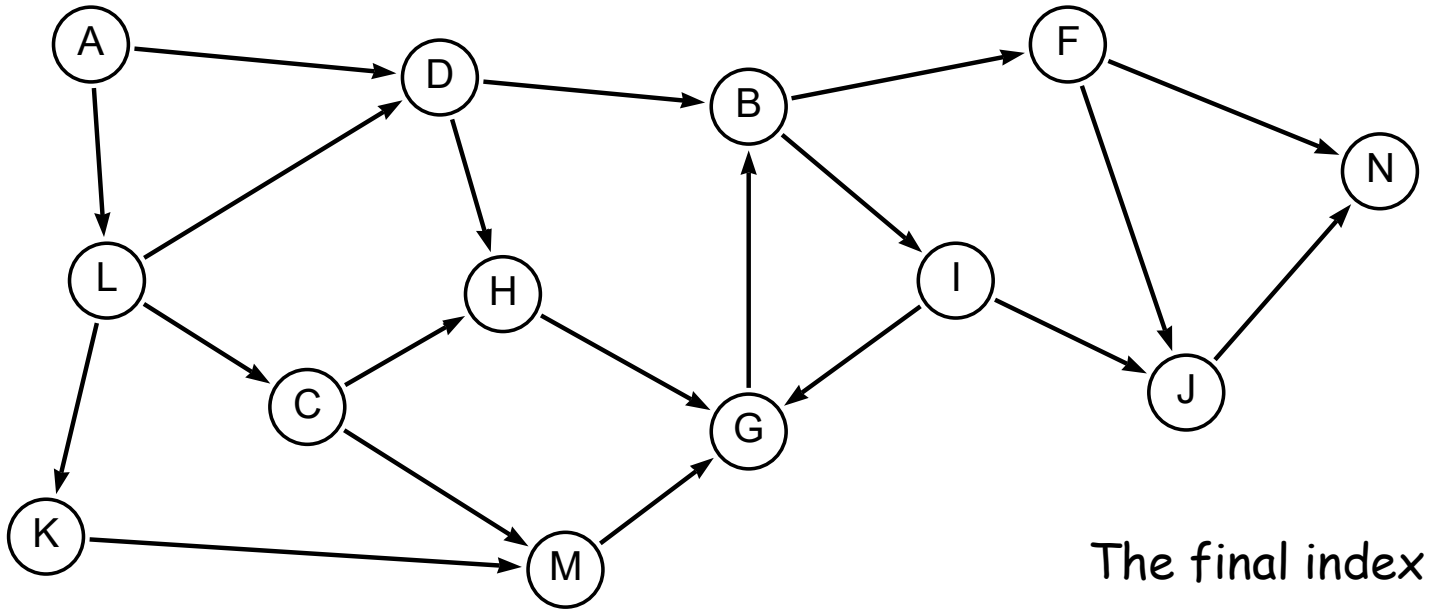


Forward BFS from D

Vertex accessing order: $(B, D, G, L, C, F, H, I, J, M, K, A, N)$

v	$L_{in}(v)$	$L_{out}(v)$
A		B, D
B		B
C		B
D		B
F	B	
G	B, D	B
H	D	B
I	B	B
J	B	
K		B
L		B, D
M		B
N	B	

PLL



The final index

v	$L_{in}(v)$	$L_{out}(v)$
A		B, D, G, L
B		B
C	L	B, G
D		B
F	B	
G	B, D	B
H	D, L, C	B, G
I	B	B, G
J	B, F, I	
K	L	B, G, M
L		B, D, G
M	L, C	B, G
N	B, F, I, J	

Vertex accessing order: $(B, D, G, L, C, F, H, I, J, M, K, A, N)$

PLL

Index built with the order:
$(M, D, C, B, A, K, N, F, G, H, I, J, L)$

v	$L_{in}(v)$	$L_{out}(v)$
A		M, D, C, K
B	M, D, C, B	
C		M
D		
F	M, D, C, B	N
G	M, D, C, B	B
H	D, C	B, G
I	M, D, C, B	N, G
J	M, D, C, B, F, I	N
K	A	M
L	A	M, D, C, K
M		
N	M, D, C, B	

Index Size

47 vs. 39

Finding the optimal order that leads to the minimum index size is NP-hard [Wel14]

Index built with the order:
$(B, D, G, L, C, F, H, I, J, M, K, A, N)$

v	$L_{in}(v)$	$L_{out}(v)$
A		B, D, G, L
B		B
C	L	B, G
D		B
F	B	
G	B, D	B
H	D, L, C	B, G
I	B	B, G
J	B, F, I	
K	L	B, G, M
L		B, D, G
M	L, C	B, G
N	B, F, I, J	

[Wel14] M. Weller: Optimal Hub Labeling is NP-complete. CoRR abs/1407.8373 (2014)

Total Order Labeling (TOL)

- TOL [Zhu14] is a framework for building 2-hop indexes (**generalization of PLL**)
  - Computing a total order o of vertices
  - Iteratively processing vertices according to the order o:
    - Performing backward and forward BFSs from each v to compute $L_{out}(v)$ and $L_{in}(v)$
  - Pruning the search space of each BFS according to o
    - If the forward BFS from v visits u and $o(v) > o(u)$, then u and its descendants are skipped
    - Similarly for the backward BFS

[Zhu14] A. Zhu et al. Reachability queries on large dynamic graphs: a total order approach. SIGMOD Conference 2014: 1323-1334

Instantiations of the Total Order

- Vertex degree (approximating centrality):
  - DL [Jin13] and PLL [Aki13]

  It has been proven that they are equivalent
- Topological folding number (TF):
  - TFL [Che13]

  TF is computed based on a topological ordering of a DAG
- Heuristics for building the minimum 2-hop index

[Che13] J. Cheng et al. TF-Label: a topological-folding labeling scheme for reachability querying in a large graph. SIGMOD Conference 2013: 193-204
[Jin13] R. Jin et al. Simple, Fast, and Scalable Reachability Oracle. Proc. VLDB Endow. 6(14): 1978-1989 (2013)
[Aki13] E. Akiba et al. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. SIGMOD Conference 2013: 349-360

Partial 2-Hop Indexes

- DBL [Lyu21]:
  - Landmark vertices:
    - Top-k vertices with the highest degree
  - Building a partial 2-hop index:
    - Performing the backward and forward BFSs from landmark vertices only
  - Querying processing:
    - Index lookups and online BFS
  - DBL adds a partial BFL [Su17] (an index based on Approximate TC)
    - Avoid online BFS as much as possible

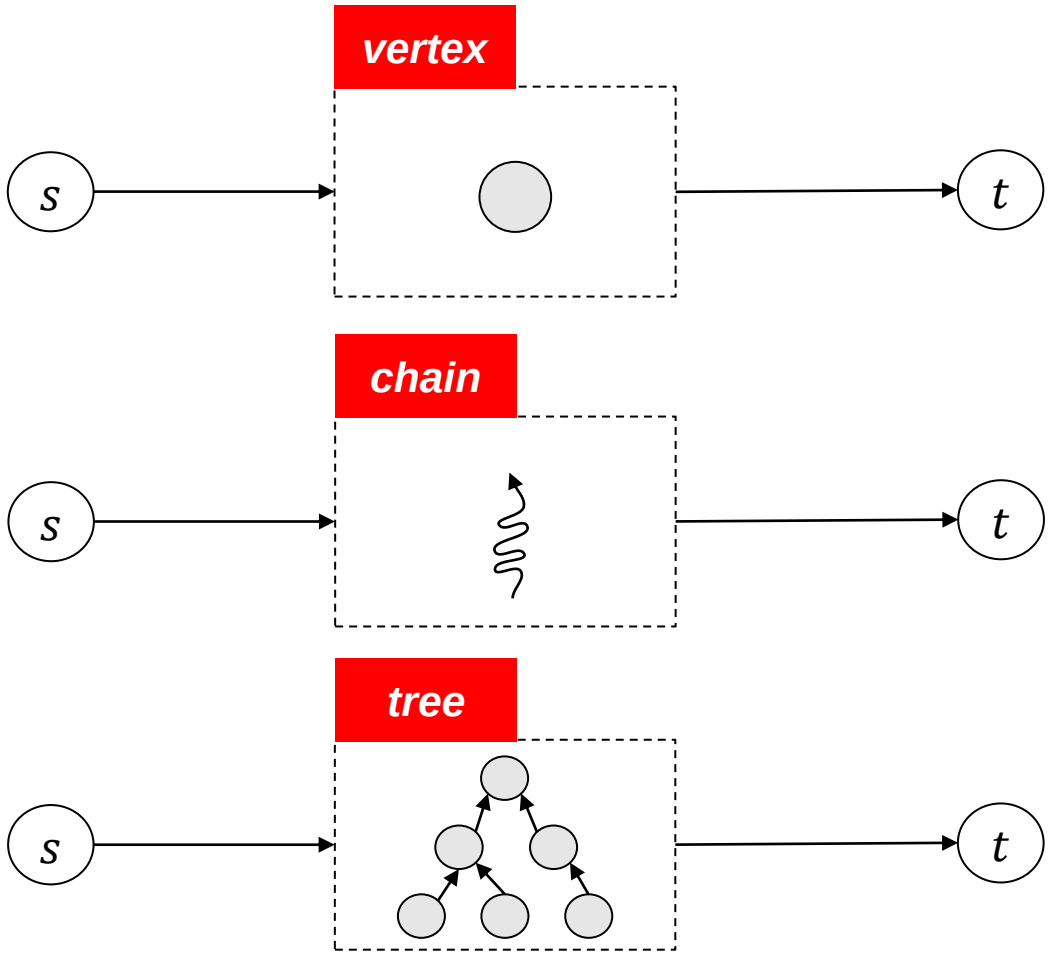
[Lyu21] Q. Lyu et al. DBL: Efficient Reachability Queries on Dynamic Graphs. DASFAA (2) 2021: 761-777
[Su17] J. Su et al. Reachability Querying: Can It Be Even Faster? IEEE Trans. Know. Data Eng. 29(3): 683-697 (2017)

Partial 2-Hop Indexes

- O’Reach [Han21]:
  - Supportive vertices:
    - K vertices that are in the middle of a topological ordering
  - Computing reachability via the k supportive vertices
  - Query processing:
    - Index lookup and online BFS

[Han21] K. Han et. O'Reach: Even Faster Reachability in Large Graphs. SEA 2021: 13:1-13:24

Other 2-Hop-Based Techniques



2-Hop

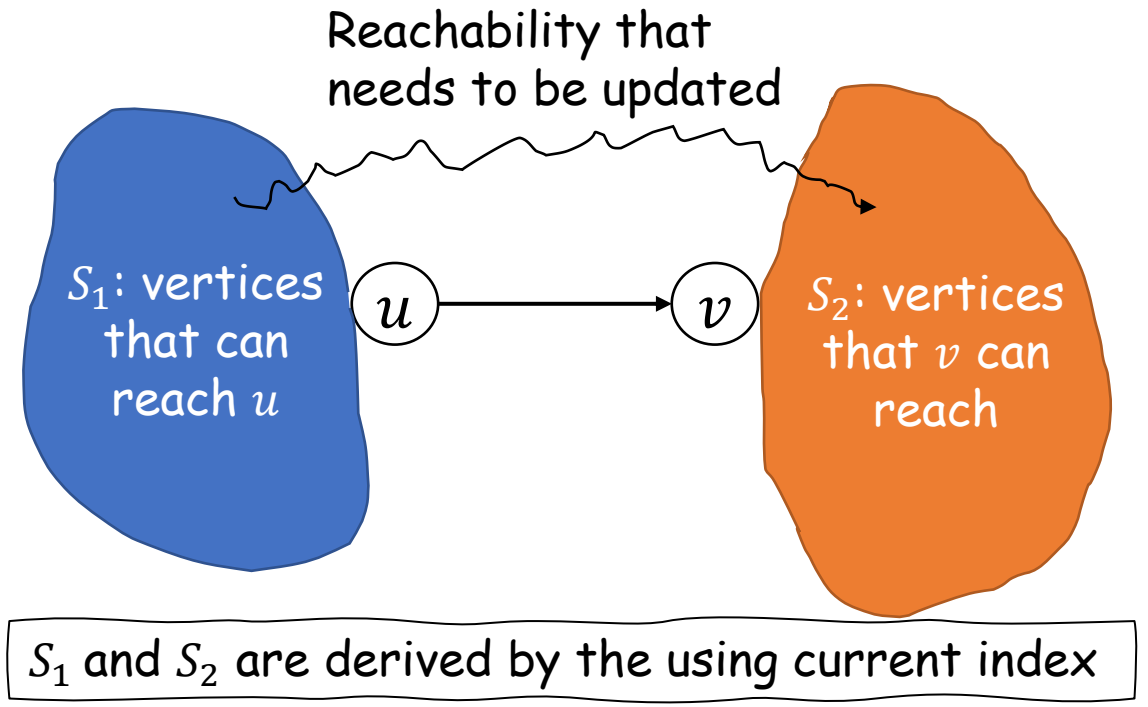
3-Hop [Jin09]

Path-Hop [Cai10]

[Jin09] R. Jin et al. 3-HOP: a high-compression indexing scheme for reachability query. SIGMOD Conference 2009: 813-826
[Cai10] J. Cai et al. Path-hop: efficiently indexing large graphs for reachability queries. CIKM 2010: 119-128

2-Hop Indexes on Dynamic Graphs

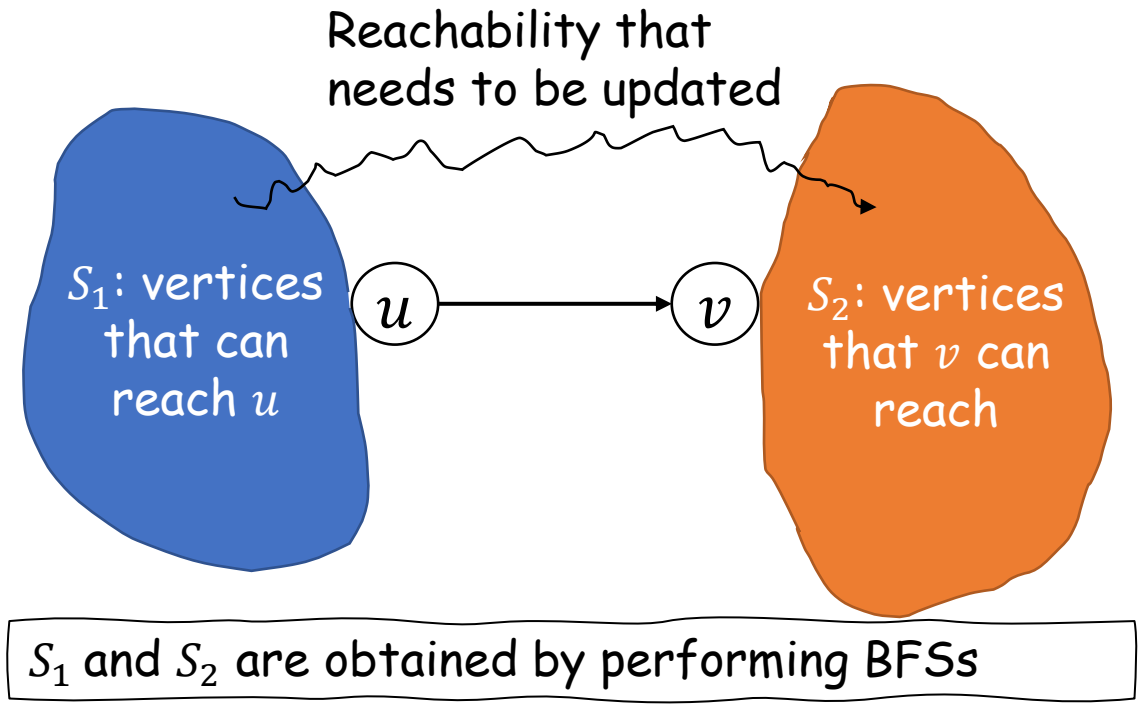
- TOL [Zhu14]: a **complete** 2-hop index
  - **Fully dynamic** graphs:
    - Edge insertions and deletions
  - Assuming DAGs as input:
    - Requires maintaining strongly connected components
  - Algorithm:
    - Leveraging the current index to avoid recomputing from scratch



[Zhu14] A. Zhu et al. Reachability queries on large dynamic graphs: a total order approach. SIGMOD Conference 2014: 1323-1334

2-Hop Indexes on Dynamic Graphs

- DBL [Lyu21]: a **partial** 2-hop index
  - **Insertion-only** graphs:
    - No edge deletions
  - General graphs as input:
    - No need to maintain strongly connected components
  - Algorithm:
    - Performing BFSs from the endpoints of the updated edge to reflect the changes



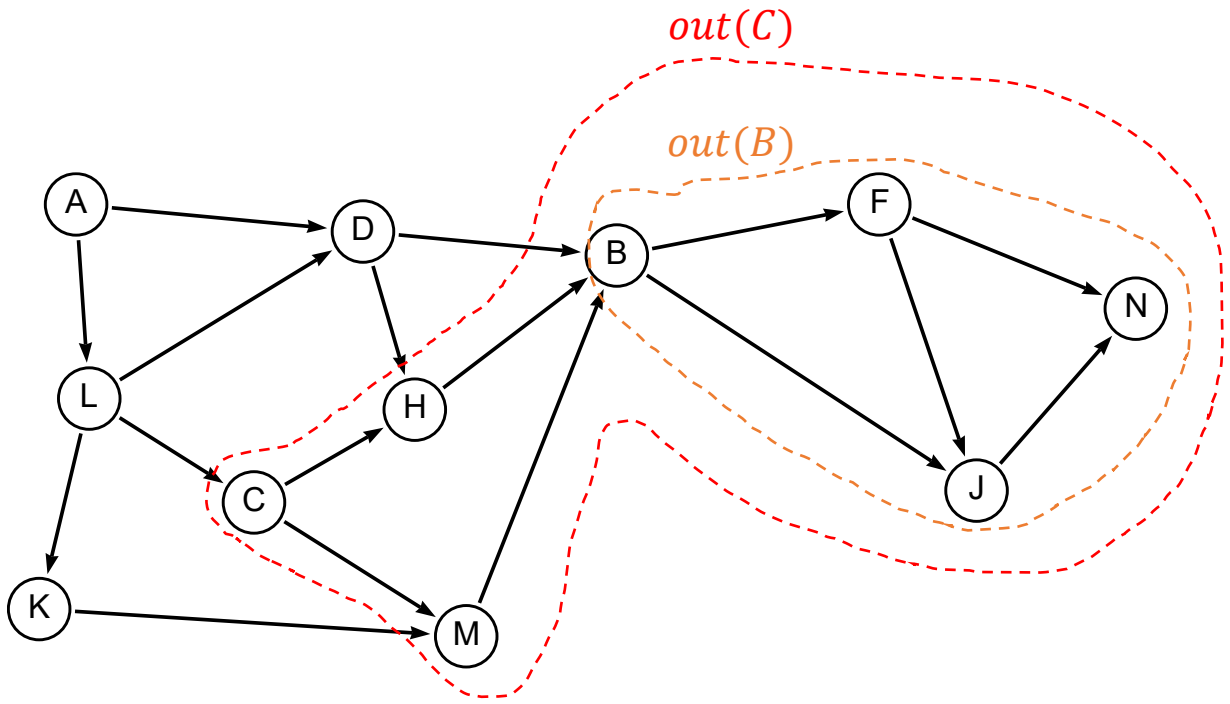
[Lyu21] Q. Lyu et al. DBL: Efficient Reachability Queries on Dynamic Graphs. DASFAA (2) 2021: 761-777

Outline

1. Plain Reachability Indexes
    a) Tree-Cover Indexes
    b) 2-Hop Indexes
    c) Approximate Transitive Closures
    d) Other Techniques
2. Path-Constrained Reachability Indexes
    a) Indexes for Alternation-Based Queries
    b) Indexes for Concatenation-Based Queries
3. Open Challenges

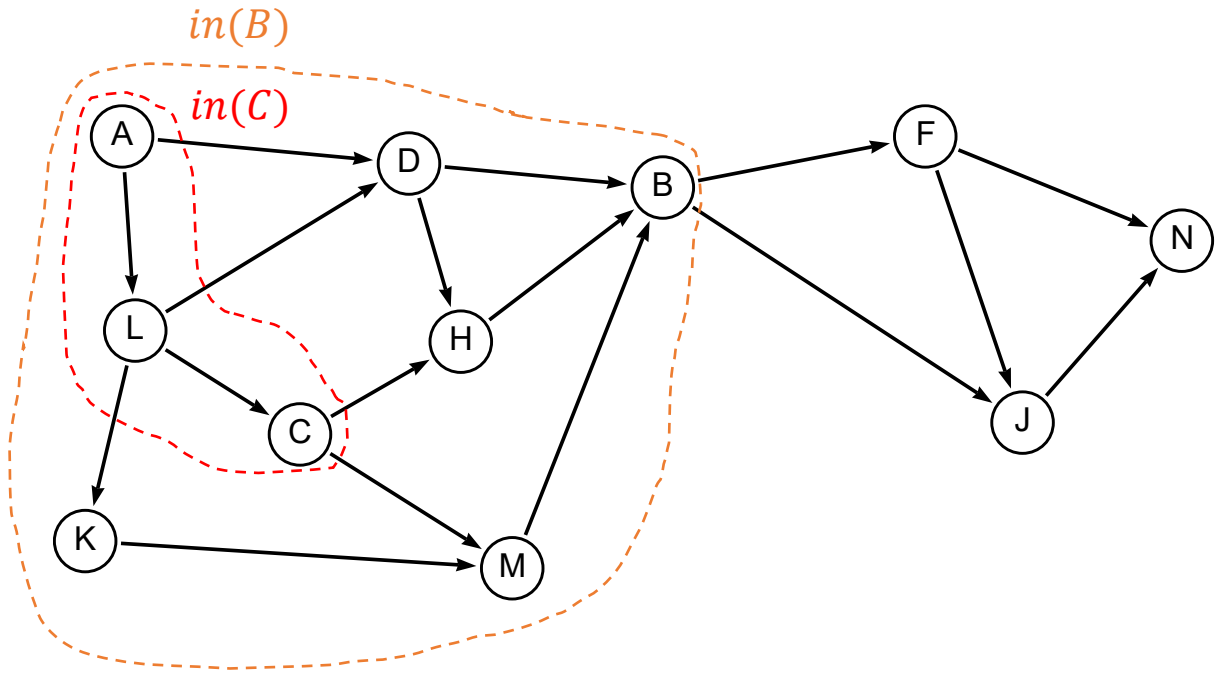
Rethinking of Transitive Closure

- $out(v)$:
  - v and all the vertices that v can reach
- **Observation**:
  - If u can reach v, then $out(v) \subseteq out(u)$
- Example: C can reach B
  - $out(B) = \{B, G, I, F, J, N\}$
  - $out(C) = \{B, G, I, F, J, N, C, H, M\}$
- **Contrapositive condition 1**:
  - If $out(v) \not\subseteq out(u)$, then u cannot reach v



Rethinking of Transitive Closure

- $in(v)$:
  - v and all the vertices that can reach v
- **Observation**:
  - If u can reach v, then $in(u) \subseteq in(v)$
- Example: C can reach B
  - $in(B) = \{C, L, A, B, D, G, H, M, K\}$
  - $in(C) = \{C, L, A\}$
- **Contrapositive condition 2**:
  - If $in(u) \not\subseteq in(v)$, then u cannot reach v



Approximate Transitive Closure

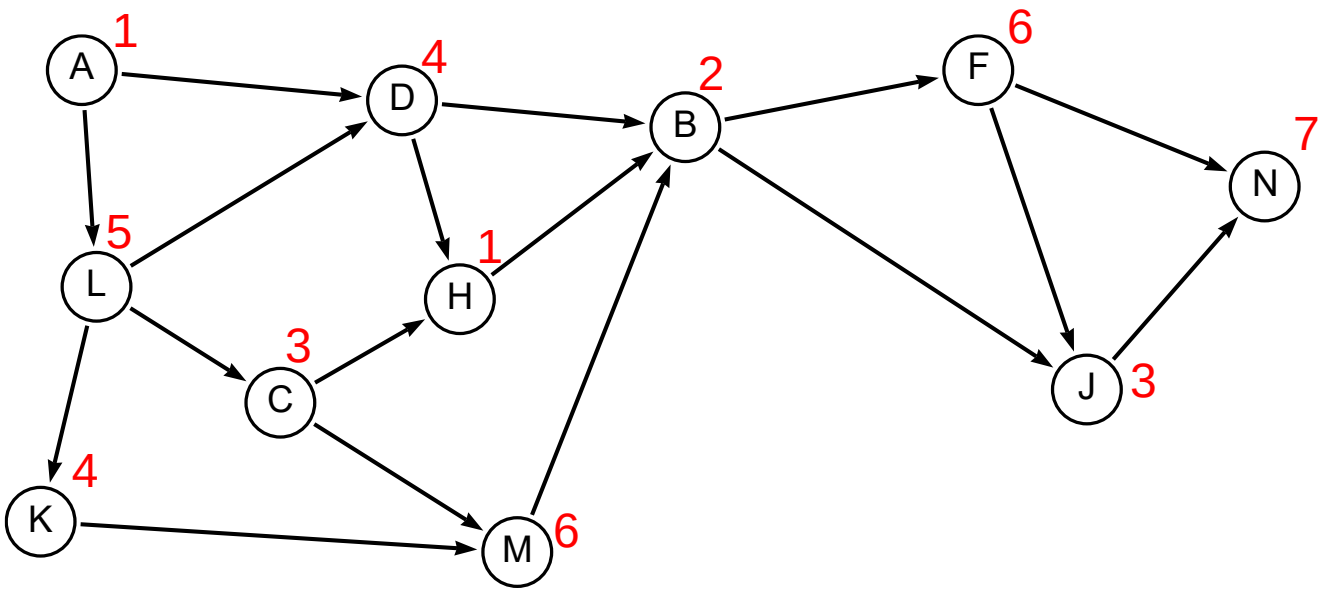
- Infeasible to compute $out(v)$ and $in(v)$ for each vertex v
  - Equivalent to computing the transitive closure
- $AP()$: an approximate function for computing $out(v)$ and $in(v)$
  - $|AP(out(v))| \ll |out(v)|$
  - $|AP(in(v))| \ll |in(v)|$
- The $AP()$ function should satisfy the following condition:
  - If $AP(out(v)) \not\subseteq AP(out(u))$, then $out(v) \not\subseteq out(u)$
  - Thus, we can derive u cannot reach v

The Approximate Functions

- K-min-wise independent permutation: IP [Wei14, Wei18]
  - In $out(v)$ and $in(v)$, recoding k smallest numbers obtained by a permutation of vertices
- Bloom filter: BFL [Su17]
  - In $out(v)$ and $in(v)$, recording the hash codes of vertices
- Partial indexes without false negatives
- Query processing $Q_r(s,t)$
  - If $in(s) \not\subseteq in(t)$ or $out(t) \not\subseteq out(s)$, then return False
  - Otherwise, performing a guided DFS from s with index lookups

[Wei14] H. Wei et al. Reachability Querying: An Independent Permutation Labeling Approach. Proc. VLDB Endow. 7(12): 1191-1202 (2014)
[Su17] J. Su et al. Reachability Querying: Can It Be Even Faster? IEEE Trans. Know. Data Eng. 29(3): 683-697 (2017)
[Wei18] H. Wei et al. Reachability querying: an independent permutation labeling approach. VLDB J. 27(1): 1-26 (2018)

BFL



v	$in(v)$	$out(v)$
A	$\{1\}$	$\{1,2,3,4,5,6,7\}$
B	$\{1,2,3,4,5,6\}$	$\{2,3,6,7\}$
C	$\{1,3,5\}$	$\{1,2,3,6,7\}$
D	$\{1,4,5\}$	$\{1,2,3,4,6,7\}$
F	$\{1,2,3,4,5,6\}$	$\{6,3,7\}$
H	$\{1,3,4,5\}$	$\{1,2,3,6,7\}$
J	$\{1,2,3,4,5,6\}$	$\{3,7\}$
K	$\{1,4,5\}$	$\{2,3,4,6,7\}$
L	$\{1,5\}$	$\{1,2,3,4,5,6,7\}$
M	$\{1,3,4,5,6\}$	$\{2,3,6,7\}$
N	$\{1,2,3,4,5,6,7\}$	$\{7\}$

- $Q_r(B, C)$:
  - Index lookup: $out(C) \not\subseteq out(B)$, thus immediately return False
- $Q_r(D, M)$:
  - Index lookup: $out(M) \subseteq out(D)$ and $in(D) \subseteq in(M)$, thus perform guided DFS from D
  - None of the out-neighbors of D can reach M, thus return False

Outline

1. Plain Reachability Indexes
    a) Tree-Cover Indexes
    b) 2-Hop Indexes
    c) Approximate Transitive Closures
    d) Other Techniques
2. Path-Constrained Reachability Indexes
    a) Indexes for Alternation-Based Queries
    b) Indexes for Concatenation-Based Queries
3. Open Challenges

Other Reachability Indexes

- Feline [Vel14]:
  - Partial index based on **dominance drawing** (no false negatives)
  - For each vertex v, assigning (v_x, v_y), such that
    - If v is reachable from u, then $u_x \leq v_x$ and $u_y \leq v_y$
  - Query processing $Q_r(s,t)$: guided DFS with index lookups
- PReach [Mer14]:
  - **Reachability contraction hierarchies**: recursively removing *sink* vertices (out-degree of zero) and *source* vertices (in-degree of zero)
  - Vertex orderings: the order when a vertex is removed
  - Query processing $Q_r(s,t)$: bidirectional search from s and t, looking for vertices with higher orderings

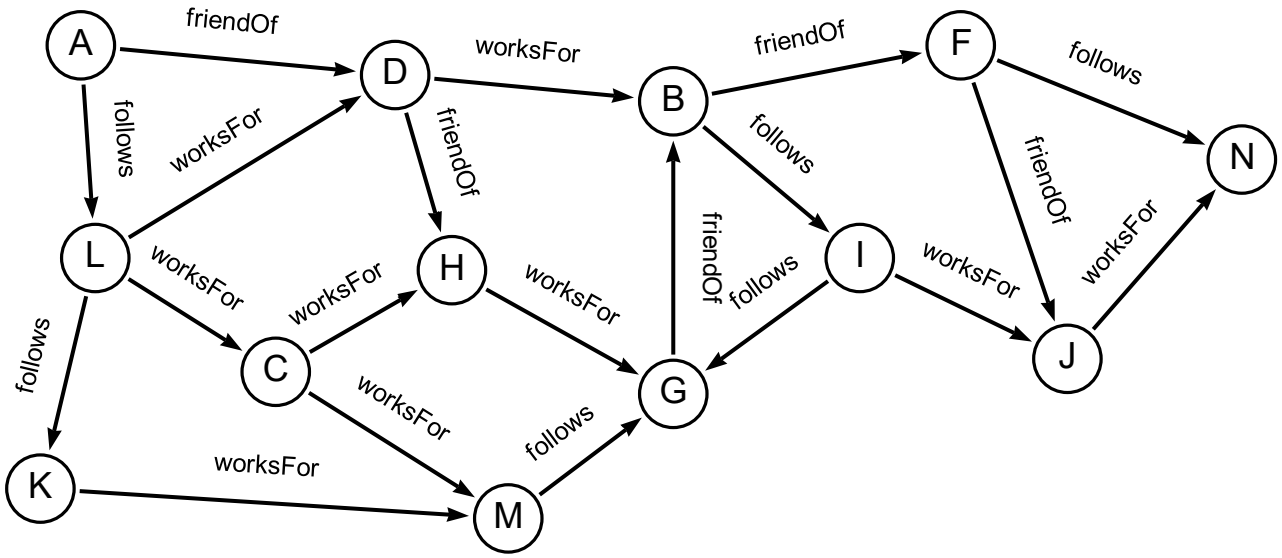
[Vel14] R. Veloso et al. Reachability Queries in Very Large Graphs: A Fast Refined Online Search Approach. EDBT 2014: 511-522
[Mer14] F. Merz et al. PReaCH: A Fast Lightweight Reachability Index Using Pruning and Contraction Hierarchies. ESA 2014: 701-712

Outline

1. Plain Reachability Indexes
    a) Tree-Cover Indexes
    b) 2-Hop Indexes
    c) Approximate Transitive Closures
    d) Other Techniques
2. Path-Constrained Reachability Indexes
    a) Indexes for Alternation-Based Queries
    b) Indexes for Concatenation-Based Queries
3. Open Challenges

Path-Constraint Expression

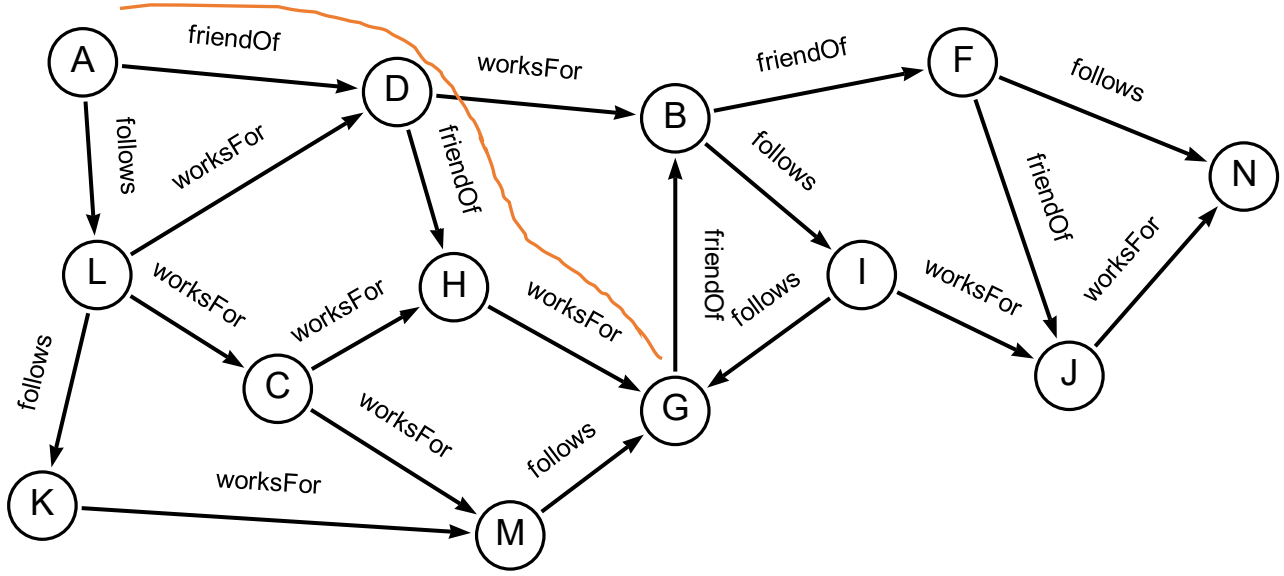
- $G = (V, E, \mathcal{L})$, each edge e is assigned a label $l \in \mathcal{L}$
- Path-constraint expression: regular expression α
  - Literal characters: $l \in \mathcal{L}$
  - Meta characters:
    - Concatenation '$\cdot$',
    - Alternation '$\cup$',
    - The Kleene operators (star '*' or plus '$^+$')
  - Grammar: $\alpha ::= l | \alpha \cdot \alpha | \alpha \cup \alpha | \alpha^+ | \alpha^*$
- Example: $\alpha = (friendOf \cup follows)^*$



R. Angles et al. Foundations of Modern Query Languages for Graph Databases. ACM Comput. Surv. 50(5): 68:1-68:40 (2017)

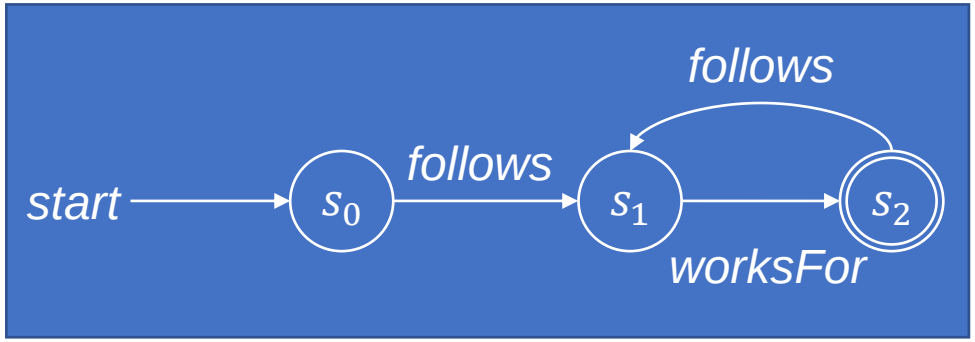
Path-Constrained Reachability Queries

- Path-constrained reachability $Q_r(s, t, \alpha)$
  - checking the existence of a path from s to t, and
  - checking whether the edge labels in the path can satisfy the path constraint α
- Example
  - $Q_r(A, G, \alpha)$, with $\alpha = (friendOf \cup worksFor)^*$: True
  - $Q_r(A, G, \alpha)$, with $\alpha = (friendOf \cup follows)^*$: False
- A subclass of regular path queries



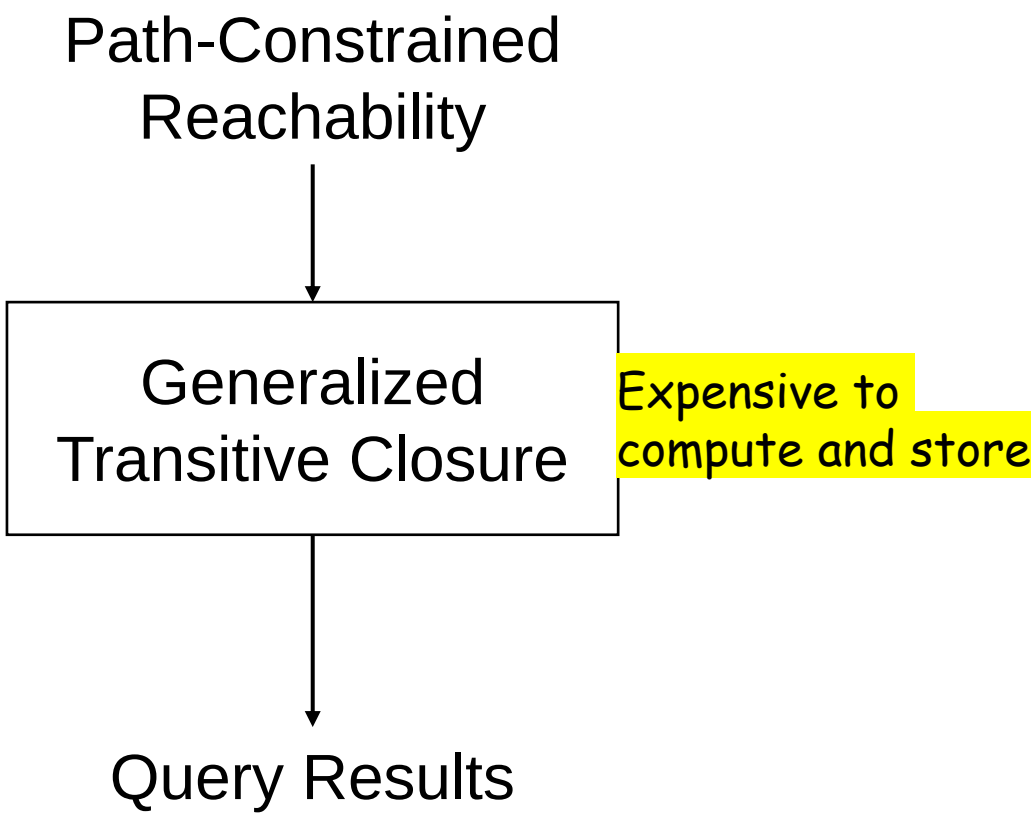
Path-Constrained Reachability Processing

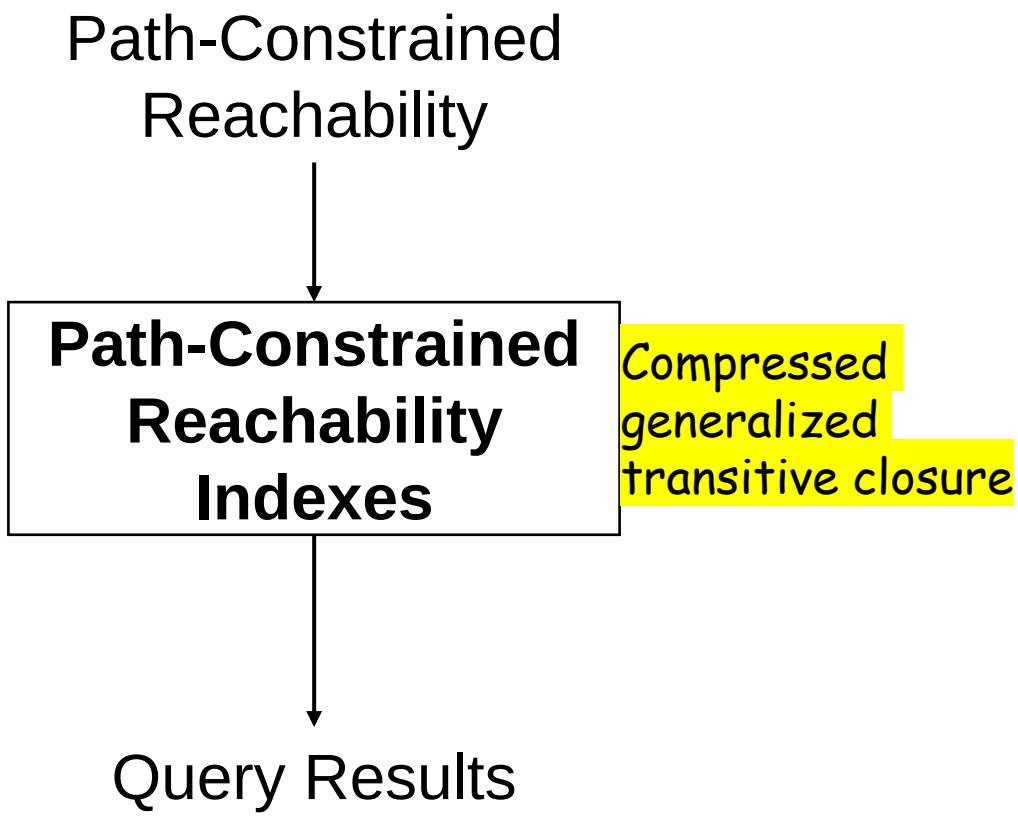
- Query processing
  - Building an FA (Finite Automata) based on α in $Q_r(s, t, \alpha)$
  - Online traversal guided by an FA
- Path semantics
  - Simple paths: non-repeated vertices or edges
  - **Arbitrary paths**: vertices or edges can repeat
- Recursive queries: the Kleene star (or plus)
- Indexes for efficient query processing



FA of (*follows*·*worksFor*)+

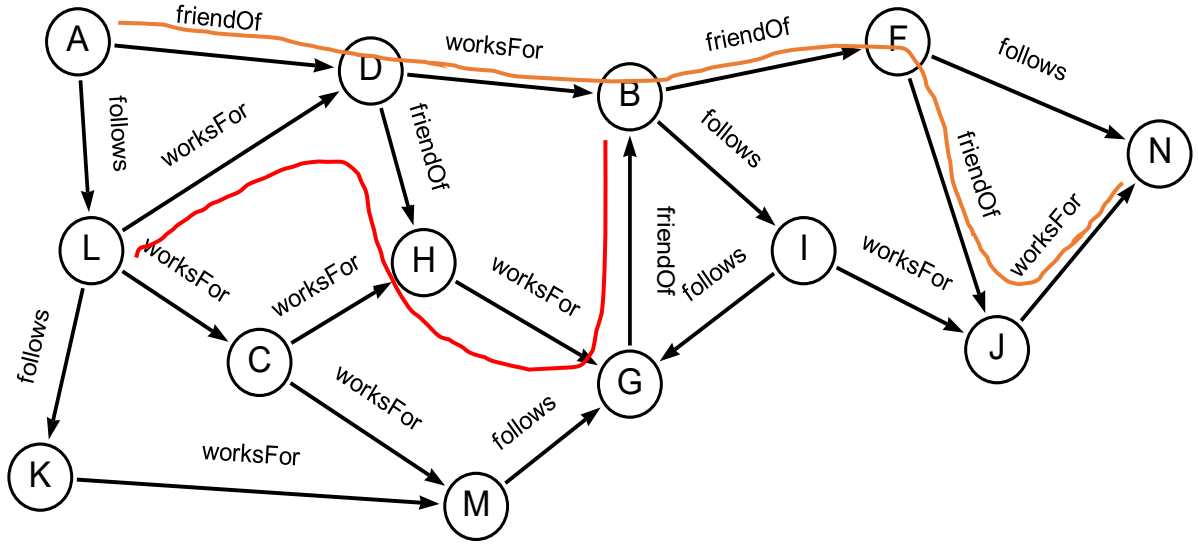
Indexes for Path-Constrained Reachability





Types of Path-Constrained Reachability Queries

- Indexes are specifically designed for each type
- $Q_r(s,t,\alpha), \alpha = (\boldsymbol{l_1} \cup \cdots \cup \boldsymbol{l_k})^*$
  - Alternation-based reachability
  - E.g., $Q_r(A,N,(worksFor \cup friendOf)^*)$ = True
- $Q_r(s,t,\alpha), \alpha = (\boldsymbol{l_1} \cdot \ldots \cdot \boldsymbol{l_k})^*$
  - Concatenation-based reachability
  - E.g., $Q_r(L,B,(worksFor \cdot friendOf)^*)$ = True

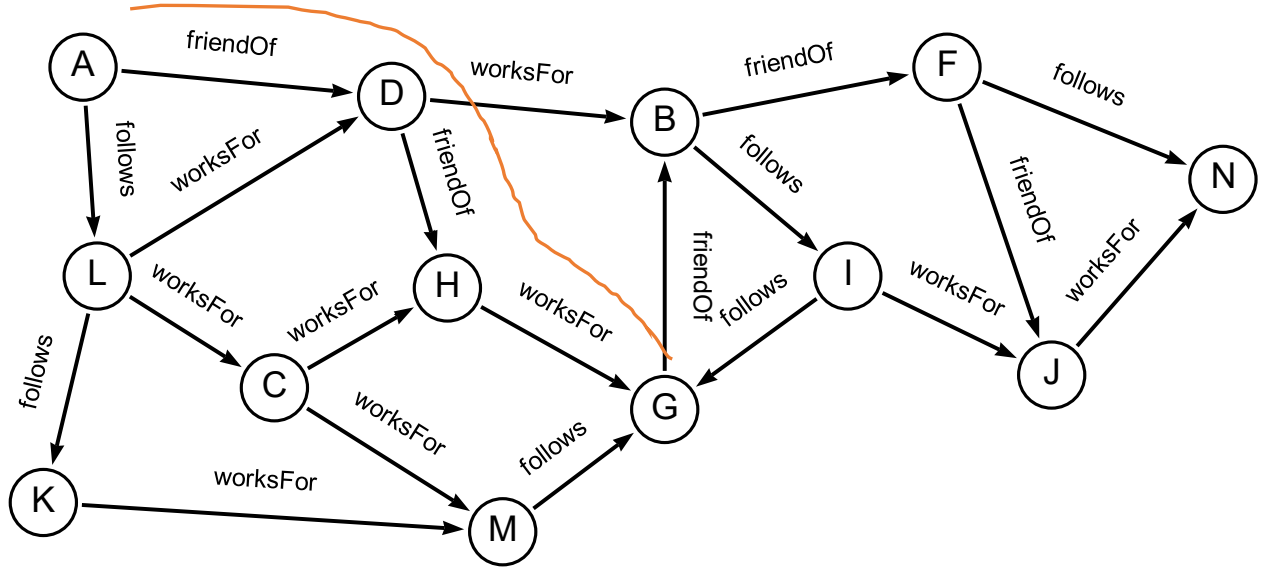


Outline

1. Plain Reachability Indexes
   a) Tree-Cover Indexes
   b) 2-Hop Indexes
   c) Approximate Transitive Closures
   d) Other Techniques
2. Path-Constrained Reachability Indexes
   a) Indexes for Alternation-Based Queries
   b) Indexes for Concatenation-Based Queries
3. Open Challenges

Alternation-Based Reachability

- $G = (V, E, \mathcal{L})$
- $Q_r(s, t, \alpha)$, $\alpha = (l_1 \cup \cdots \cup l_k)^*$
  - α defines a set $\mathcal{L}' \subseteq \mathcal{L}$, and the path should only contain edges of labels in $\mathcal{L}'$
- Supported languages:
  - SPARQL
  - openCypher
  - SQL/PGQ
  - GQL



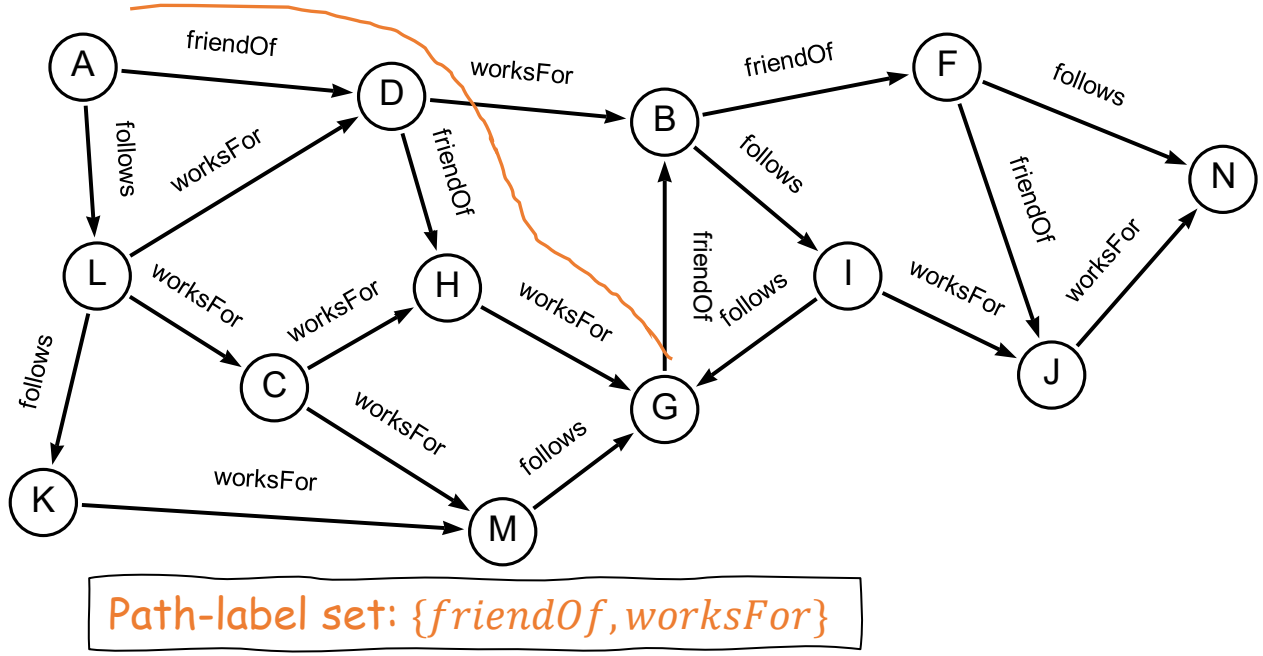
Example:

- $\alpha = (friendOf \cup worksFor)^*$, $Q_r(A, G, \alpha)$ = True
- SPARQL query:

```
ASK WHERE {:A (:friendOf|:worksFor)* :G}
```

Indexing for Alternation-Based Reachability

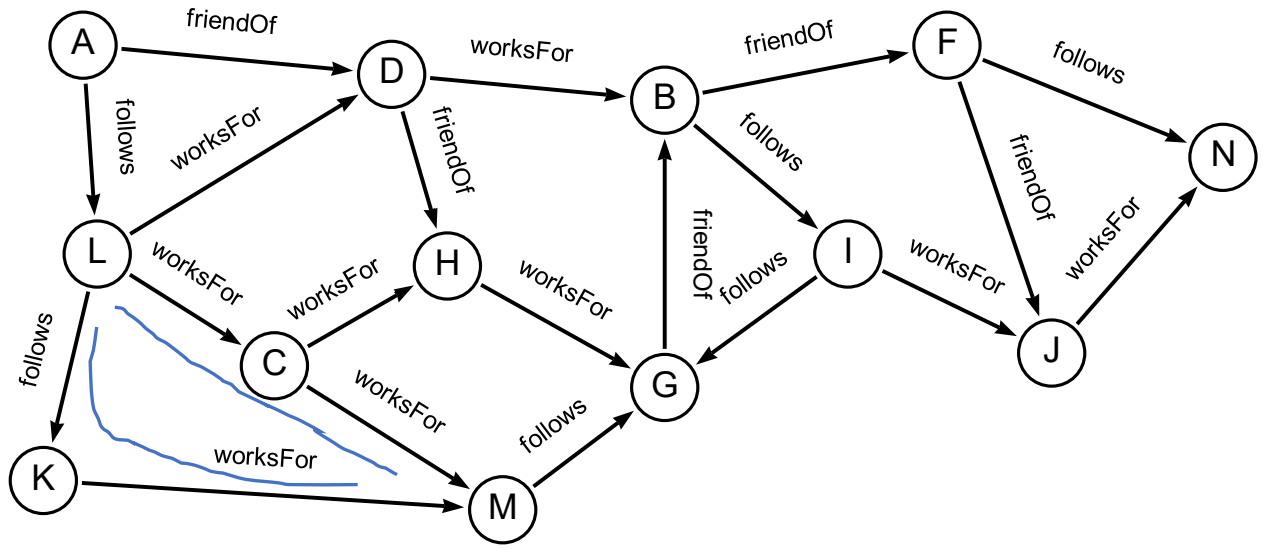
- Index-based evaluation for $Q_r(s, t, \alpha)$, $\alpha = (l_1 \cup \cdots \cup l_k)^*$
  - Path-label set: the set of edge labels in a path
  - The path-label sets from s to t are mandatory to record
- Foundations on storing and computing path-label sets [Jin10]:
  - Redundancy in path-label sets? *Sufficient path-label sets*
  - Efficient computation? *Transitivity*



[Jin10] R. Jin et al. Computing label-constraint reachability in graph databases. SIGMOD Conference 2010: 123-134

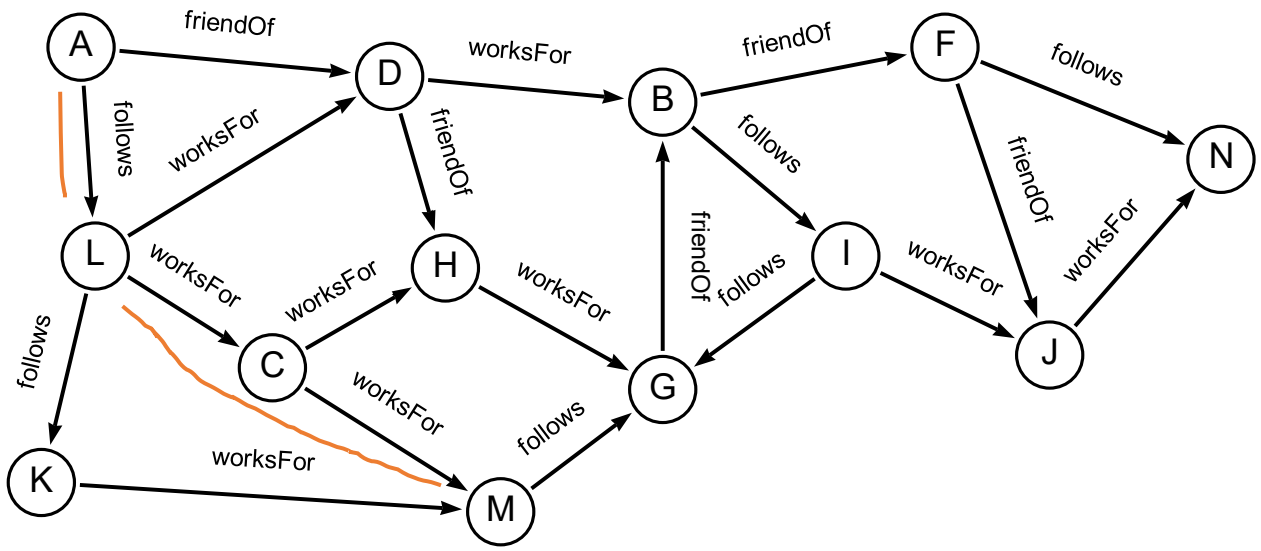
Sufficient Path-Label Sets (SPLS)

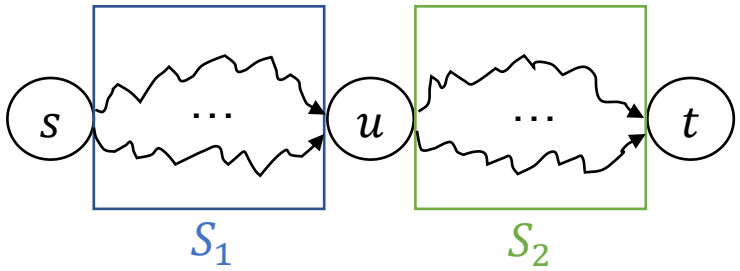
- Intuition:
  - Recording only **minimal sets** of all path-label sets from a source to a target
- Two path-label sets from L to M
  - $\{worksFor, follows\}$
  - $\{worksFor\}$
- $\{worksFor, follows\}$ is **redundant**
  - Given $Q_r(L, M, \alpha)$, and α defines $\mathcal{L}'$
  - If $\{worksFor, follows\} \subseteq \mathcal{L}'$, then $\{worksFor\} \subset \{worksFor, follows\} \subseteq \mathcal{L}'$
- $\{worksFor\}$ is a SPLS from L to M



The Transitivity of SPLSs

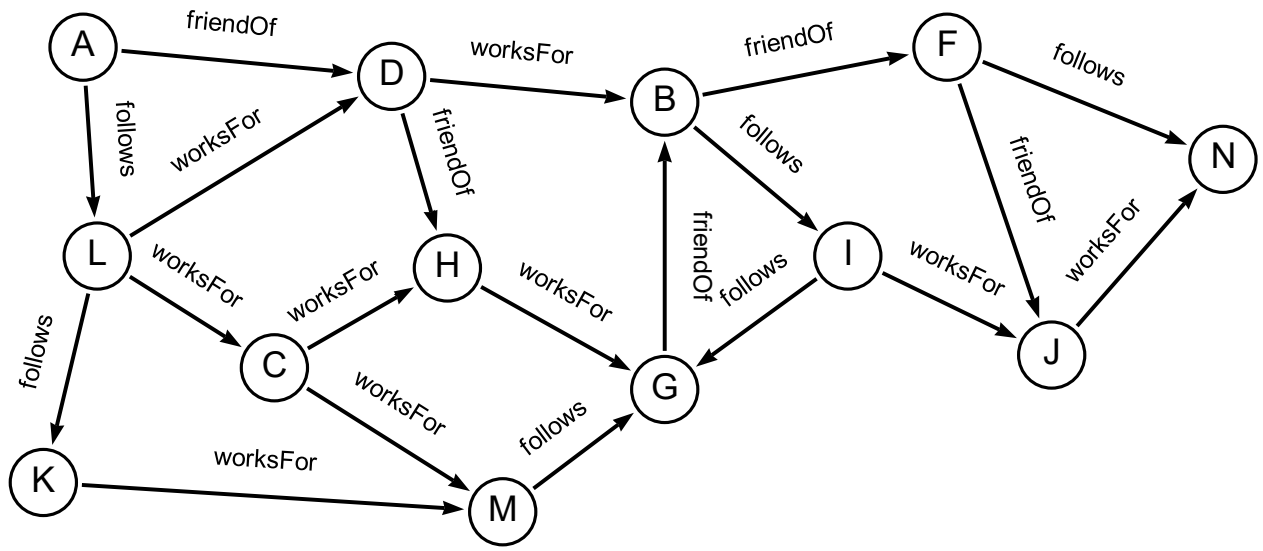
- Given the following SPLSs
  - from A to $\boldsymbol{L}$: $\{follows\}$
  - from $\boldsymbol{L}$ to M: $\{worksFor\}$
- We can compute the SPLSs from A to M:
  - $\{follows, worksFor\}$
- The transitivity of SPLSs
  - Let S_1 and S_2 be the sets of SPLSs from s to u and the set of SPLSs from u to t, resp.
  - Computing SPLSs from s to t via u
    - Computing the cross product $S_1 \times S_2$
    - Computing the minimal sets of $S_1 \times S_2$



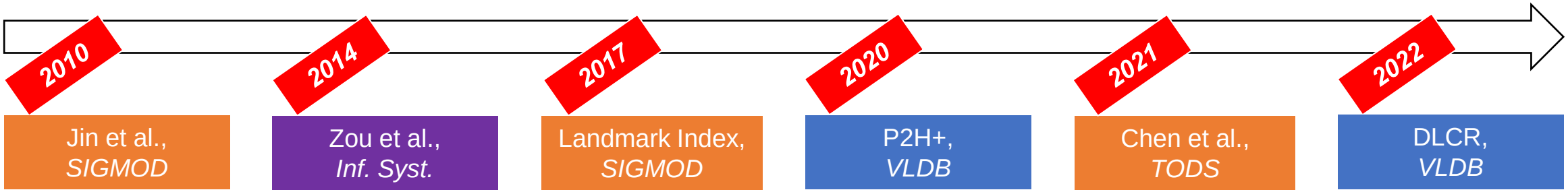


Generalized Transitive Closure (GTC)

- GTC for alternation-based queries
  - For each pair of vertices s and t
    - Recording whether s can reach t
    - Recording all the SPLSs from s to t
- Problems:
  - Computation time
  - Storage space
- How to efficiently compute and effectively compress GTC?



s	t	SPLSs
L	M	$\{worksFor\}$
A	G	$\{follows, worksFor\}$, $\{friendOf, worksFor\}$
…	…	…


2010
Jin et al., SIGMOD
2014
Zou et al., Inf. Syst.
2017
Landmark Index, SIGMOD
2020
P2H+, VLDB
2021
Chen et al., TODS
2022
DLCR, VLDB

Indexing technique	Framework	Index type	Input	Dynamic	References
Jin et al.	Tree cover	Complete	General Graph	No	[Jin10]
Chen et al.	Tree cover	Complete	General Graph	No	[Che21]
Zou et al.	Generalized TC	Complete	General Graph	Yes	[Xu11,Zou14]
Landmark index	Generalized TC	Partial	General Graph	No	[Val17]
P2H+	2-Hop	Complete	General Graph	No	[Pen20]
DLCR	2-Hop	Complete	General Graph	Yes	[Che22]

Three index frameworks:

- Tree cover
- Generalized TC
- 2-Hop

Index Frameworks for Alternation-Based Queries

- Tree Cover
  - *How to deal with non-tree edges*
  - *How to combine the interval labeling with SPLSs*
- Generalized TC
  - *How to efficiently compute generalized TC*
- 2-Hop Labeling
  - *How to combine 2-hop labeling with SPLSs*
  - *Dynamic graphs*

Partial transitive closure: Jin et al.
Recursive graph decomposition: Chen et al.

Augmented DAG transformation: Zou et al.
Partial indexing: Landmark Index

Leveraging the transitive properties: P2H+
Incremental index maintenance: DLCR

Index Frameworks for Alternation-Based Queries

- Tree Cover
  - *How to deal with non-tree edges*
  - *How to combine the interval labeling with SPLSs*
- Generalized TC
  - *How to efficiently compute generalized TC*
- 2-Hop Labeling
  - *How to combine 2-hop labeling with SPLSs*
  - *Dynamic graphs*

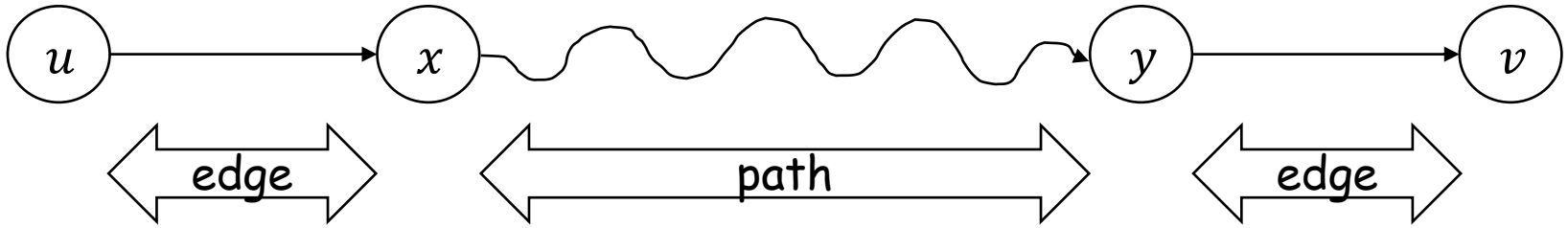
Compressing GTC using Spanning Trees

2010

Jin et al., *SIGMOD*

- Computing spanning trees in G
  - Each edge is either a tree edge or a non-tree edge
- Path characterization [Jin10] for the path from u to v
  - Case 1: either (u, x) or (y, v) is a tree edge
  - Case 2: both (u, x) and (y, v) are non-tree edges

Computing GTC only for a special kind of paths



- Computing a partial GTC to record SPLSs of paths in Case 2

R. Jin et al. Computing label-constraint reachability in graph databases. SIGMOD Conference 2010: 123-134

Example: Path Characterization

2010

Jin et al., *SIGMOD*

- Four paths from L to G

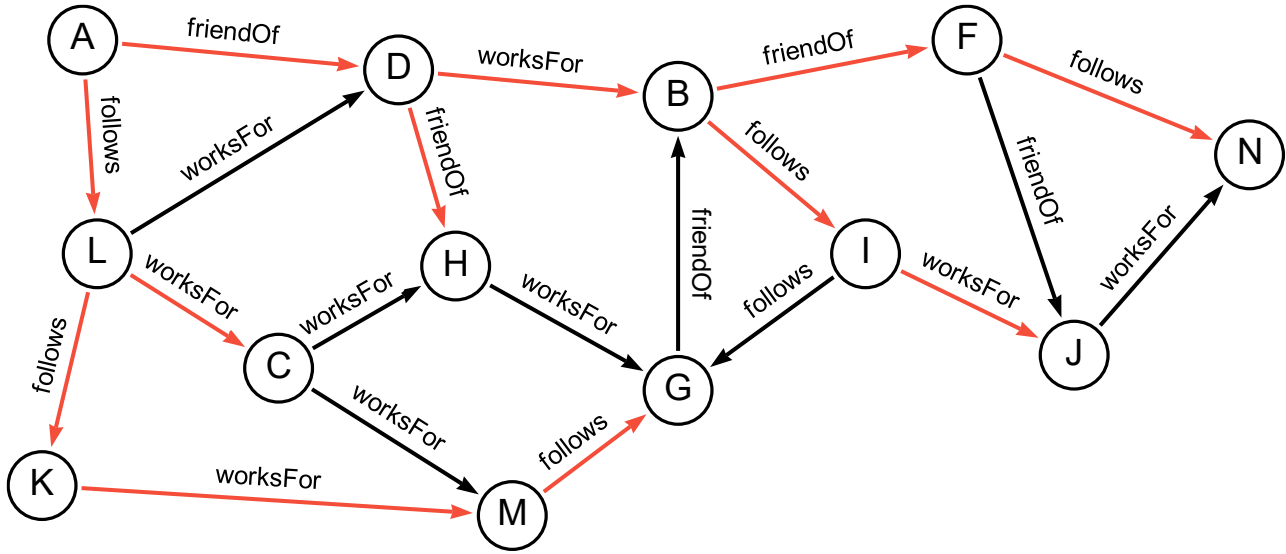
  P1: $L, worksFor, D, friendOf, H, worksFor, G$

  P2: $L, worksFor, C, worksFor, H, worksFor, G$

  P3: $L, worksFor, C, worksFor, M, follows, G$

  P4: $L, follows, K, worksFor, M, follows, G$

- Paths in Case 1: P2, P3, P4
- Paths in Case 2: P1
- The partial GTC records only P1 for (L, G)



Spanning tree in red

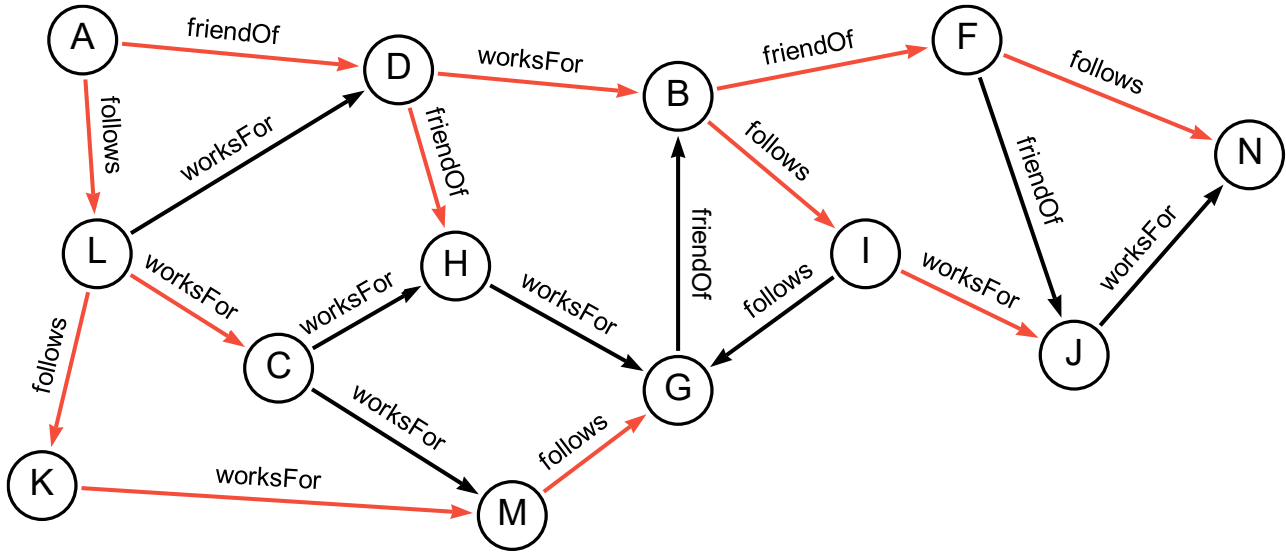
2010

Jin et al., *SIGMOD*

Example: Partial GTC

Partial GTC

Source	Target	SPLSs
…	…	…
L	G	$\{worksFor, friendOf\}$
C	G	$\{worksFor\}$
C	M	$\{worksFor\}$
C	B	$\{worksFor, friendOf\}$, $\{worksFor, follows, friendOf\}$
F	N	$\{friendOf, worksFor\}$
…	…	…



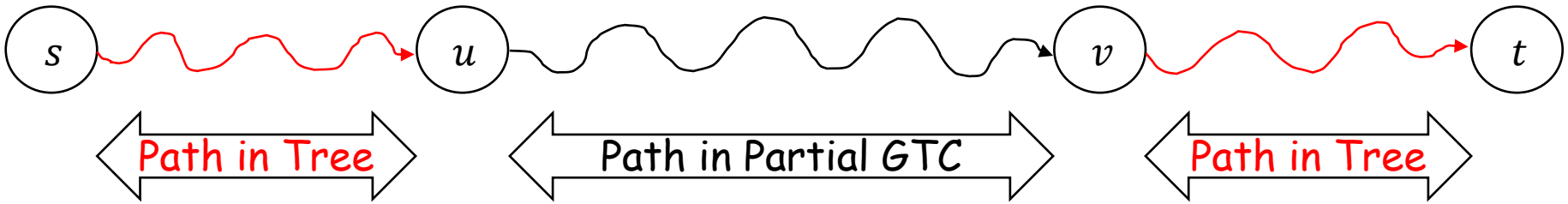
Spanning tree in red

2010

Jin et al., *SIGMOD*

Query Processing

- Query: $Q_r(s, t, \alpha)$
  - Checking whether the partial GTC records the reachability
  - Otherwise, traversing the spanning three, and checking the partial GTC



Example: Query Processing

2010

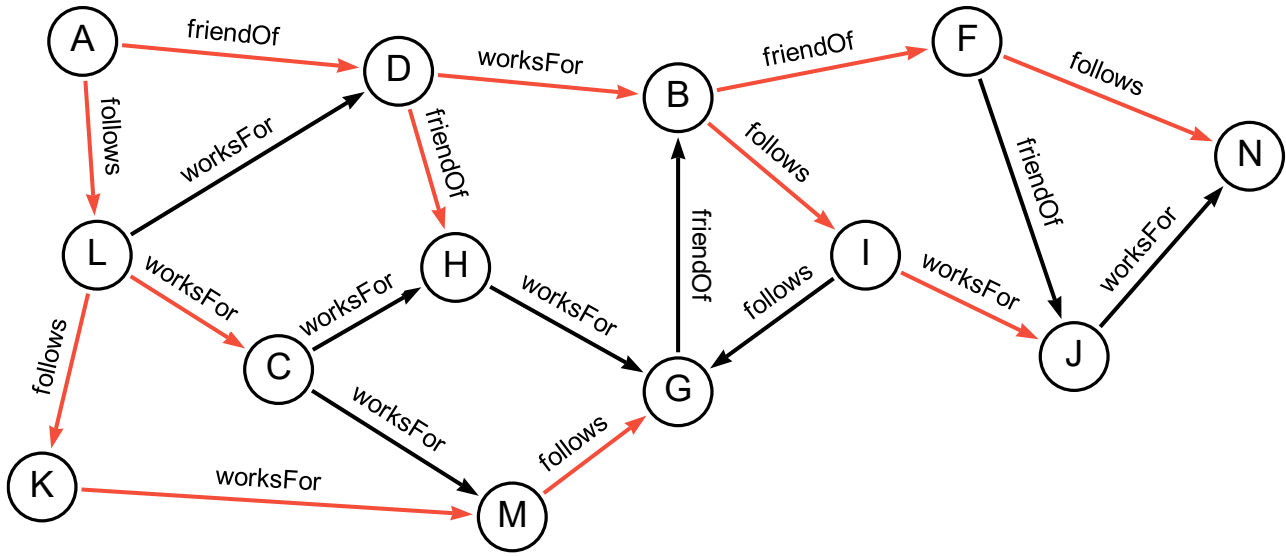
Jin et al., *SIGMOD*

Partial GTC

Source	Target	SPLSs
…	…	…
L	G	$\{worksFor, friendOf\}$
C	G	$\{worksFor\}$
C	M	$\{worksFor\}$
C	B	$\{\{worksFor, friendOf\}, \{worksFor, follows, friendOf\}\}$
F	N	$\{friendOf, worksFor\}$
…	…	…

$Q_r(L, G, (worksFor \cup friendOf)^*)$

- using partial GTC



Spanning tree in red

2010

Jin et al., *SIGMOD*

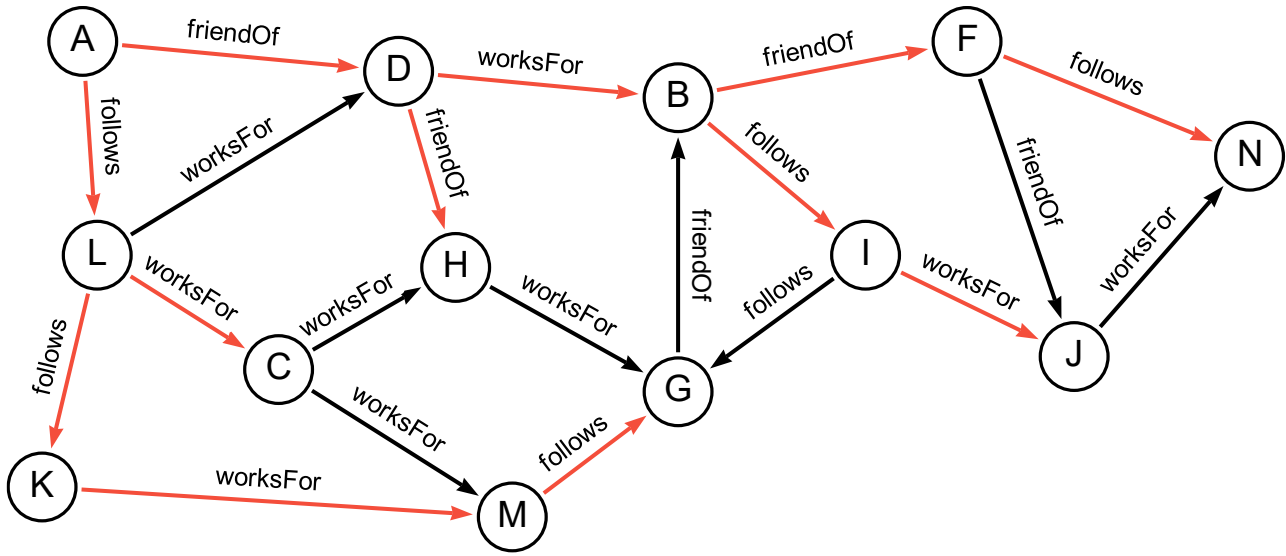
Example: Query Processing

Partial GTC

Source	Target	SPLSs
…	…	…
L	G	$\{worksFor, friendOf\}$
C	G	$\{worksFor\}$
C	M	$\{worksFor\}$
C	B	$\{\{worksFor, friendOf\}, \{worksFor, follows, friendOf\}\}$
F	N	$\{friendOf, worksFor\}$
…	…	…

$Q_r(L, G, (worksFor \cup follows)^*)$

- DFS from L visits C in the spanning tree
- using partial GTC for the reachability from C to M
- DFS from M visits G in the spanning tree



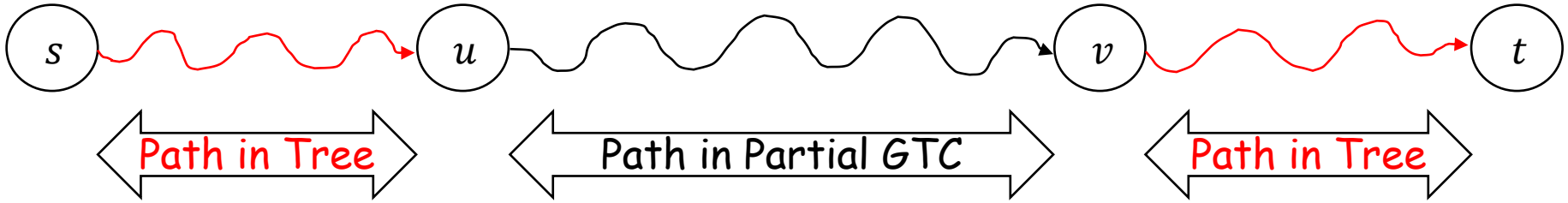
Spanning tree in red

Optimization in Query Processing

2010

Jin et al., *SIGMOD*

- Query: $Q_r(s, t, \alpha)$



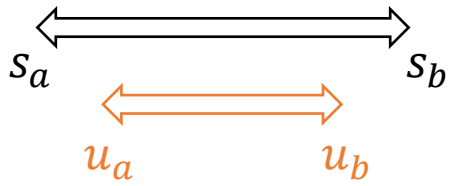
- Problems:
  1. Given (s, t), how to efficiently find (u, v) in the partial GTC
  2. How to efficiently compute the path-label sets for the paths in the tree
     - The SPLS from s to u
     - The SPLS from v to t

2010

Jin et al., *SIGMOD*

Interval-Based Optimization

- **Problem 1**: given (s, t), efficiently finding (u, v) in the partial GTC
- Intuition: using the interval labeling to reduce the problem into a multi-dimensional search
- Searching u that is reachable from s
  - $[u_a, u_b] \subseteq [s_a, s_b]$, which is equivalent to
    - $s_a \leq u_a \leq s_b$, and
    - $s_a \leq u_b \leq s_b$



- Searching v from which t is reachable
  - $[t_a, t_b] \subseteq [v_a, v_b]$, which is equivalent to
    - $0 \leq v_a \leq t_a$, and
    - $t_b \leq v_b \leq |V| - 1$

Interval-Based Optimization

2010

Jin et al., *SIGMOD*

- **Problem 1**: given (s, t), efficiently finding (u, v) in the partial GTC
- Intuition: using the interval labeling to reduce the problem into a multi-dimensional search
- Searching u that is reachable from s
  - $[u_a, u_b] \subseteq [s_a, s_b]$, which is equivalent to

Range 1
  - $s_a \leq u_a \leq s_b$, and

Range 2
  - $s_a \leq u_b \leq s_b$

- Searching v from which t is reachable
  - $[t_a, t_b] \subseteq [v_a, v_b]$, which is equivalent to

Range 3
  - $0 \leq v_a \leq t_a$, and

Range 4
  - $t_b \leq v_b \leq |V| - 1$

Four-dimensional searching problem:

- Each entry (u, v) is mapping to a point (u_a, u_b, v_a, v_b)
- Source s and target t defines the four ranges
- Looking for the points in the four ranges

A k-d tree can be built to accelerate the search

Example: Interval-Based Optimization

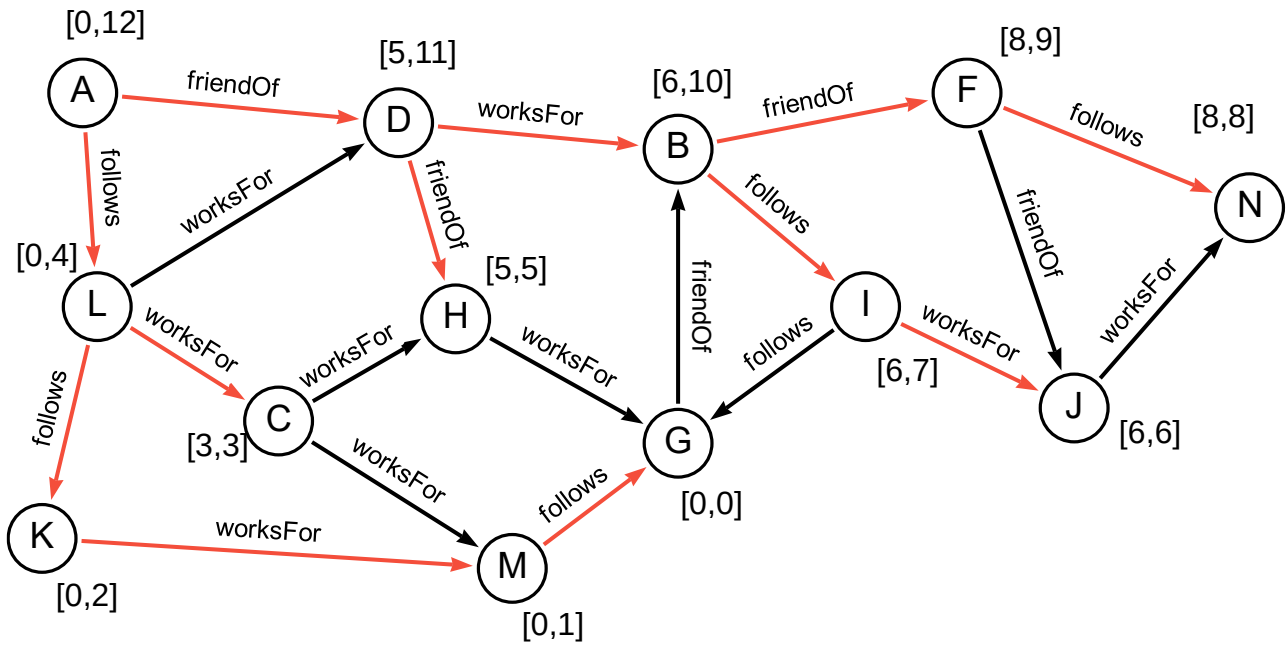
2010

Jin et al., *SIGMOD*

$Q_r(D, N, (worksFor \cup friendOf)^*)$:

- (D, N) defines the ranges: $([5,11], [5,11], [0,8], [8,12])$
- Then, the coordinates of (F, N) in the partial GTC can be located

Source	Target	Coordinates	SPLSs
…	…	…	…
L	G	(0,4,0,0)	$\{worksFor, friendOf\}$
C	G	(3,3,0,0)	$\{worksFor\}$
C	M	(3,3,0,1)	$\{worksFor\}$
C	B	(3,3,6,10)	$\{\{worksFor, friendOf\}, \{worksFor, follows, friendOf\}\}$
F	N	(8,9,8,8)	$\{friendOf, worksFor\}$
…	…		…



Example: Interval-Based Optimization

2010

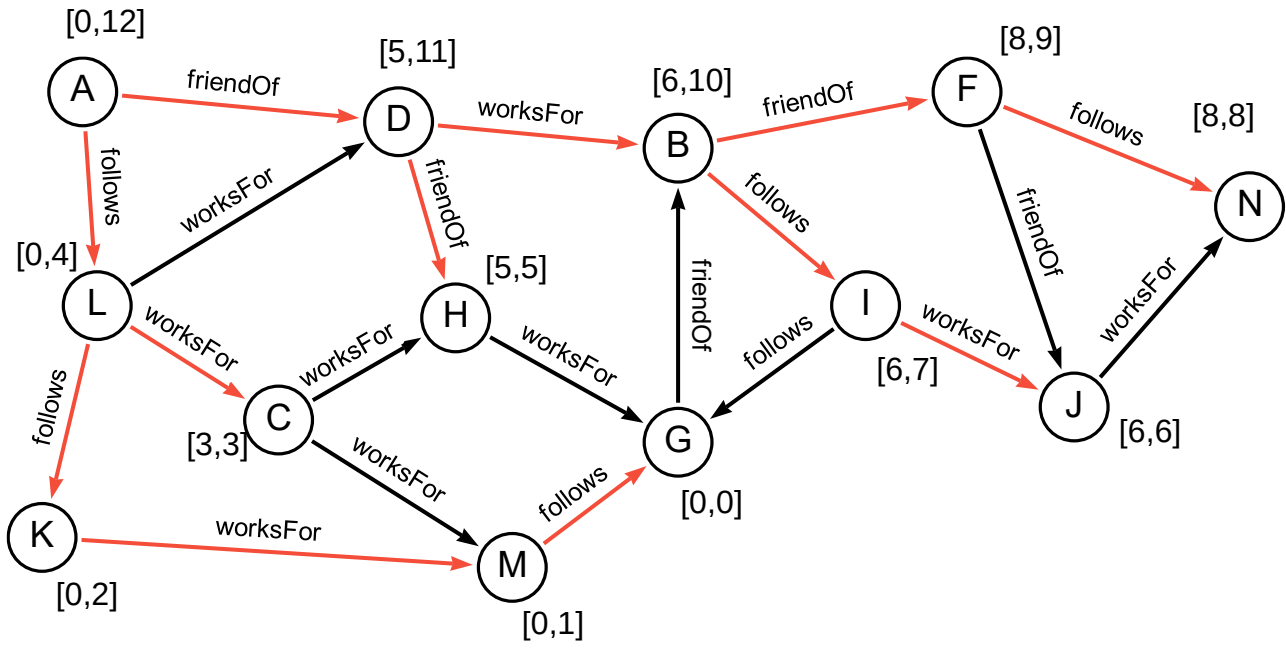
Jin et al., *SIGMOD*

Source	Target	Coordinates	SPLSs
…	…	…	…
L	G	(0,4,0,0)	$\{worksFor, friendOf\}$
C	G	(3,3,0,0)	$\{worksFor\}$
C	M	(3,3,0,1)	$\{worksFor\}$
C	B	(3,3,6,10)	$\{\{worksFor, friendOf\}, \{worksFor, follows, friendOf\}\}$
F	N	(8,9,8,8)	$\{friendOf, worksFor\}$
…	…		…

$Q_r(D, N, (worksFor \cup friendOf)^*)$:

- (D, N) defines the ranges: $([5,11], [5,11], [0,8], [8,12])$
- Then, the coordinates of (F, N) in the partial GTC can be located

How to efficiently compute the path-label set from D to N



2010

Jin et al., *SIGMOD*

Histogram-Based Optimization

- **Problem 2**: how to efficiently compute the path-label sets in the tree
- Histogram:
  - Recording occurrences of each label in the path-label set from the root r to each v
  - Example:
    - The path from r to v: $(friendOf, worksFor, friendOf)$
    - Histogram for v: $\{friendOf : 2, worksFor : 1\}$, denoted as $H(v)$
- Computing the path-label set from u to v in the tree
  - $H(v) - H(u)$
  - Example:
    - $H(v) = \{friendOf : 2, worksFor : 1\}$
    - $H(u) = \{friendOf : 1\}$
    - $H(v) - H(u) = \{friendOf : 1, worksFor : 1\}$

Example: Histogram-Based Optimization

2010

Jin et al., *SIGMOD*

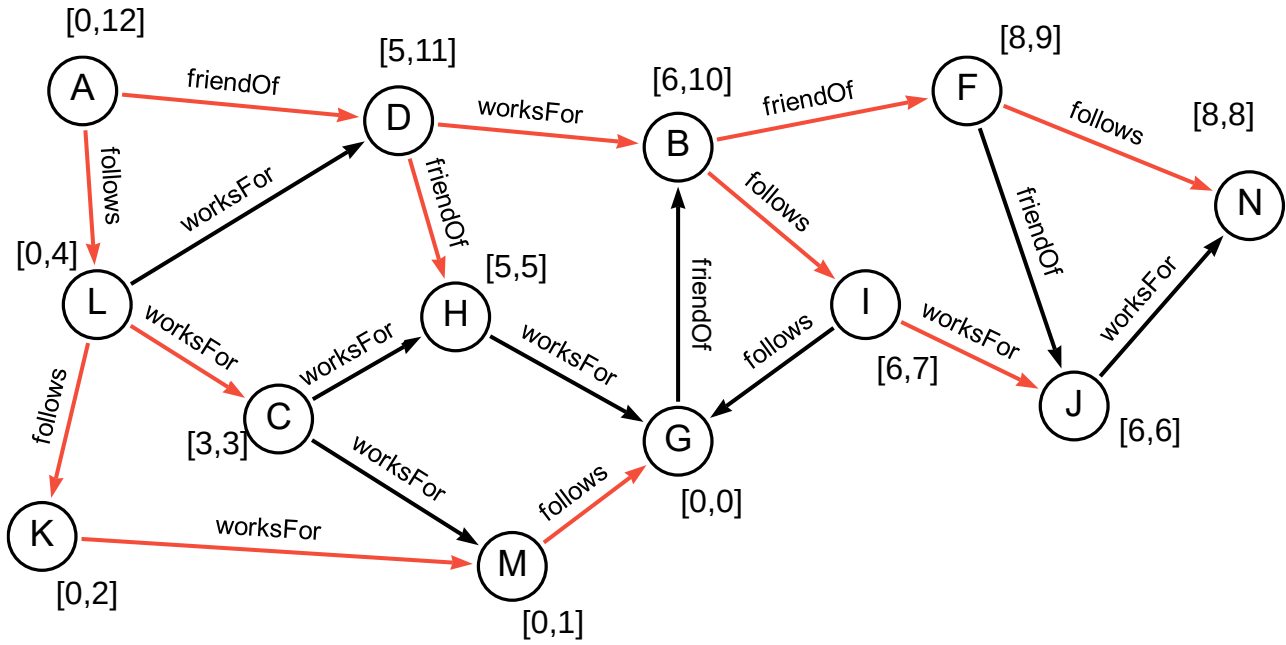
Source	Target	Coordinates	SPLSs
…	…	…	…
L	G	(0,4,0,0)	$\{worksFor, friendOf\}$
C	G	(3,3,0,0)	$\{worksFor\}$
C	M	(3,3,0,1)	$\{worksFor\}$
C	B	(3,3,6,10)	$\{\{worksFor, friendOf\}, \{worksFor, follows, friendOf\}\}$
F	N	(8,9,8,8)	$\{friendOf, worksFor\}$
…	…		…

$Q_r(D, N, (worksFor \cup friendOf)^*)$:

- (D, N) defines the ranges: $([5,11], [5,11], [0,8], [8,12])$
- Then, the coordinate of (F, N) in the partial GTC can be located

Efficiently computing the path-label set from D to F:

- $H(F) - H(D) = \{friendOf : 1, worksFor : 1\}$



Index Construction

2010

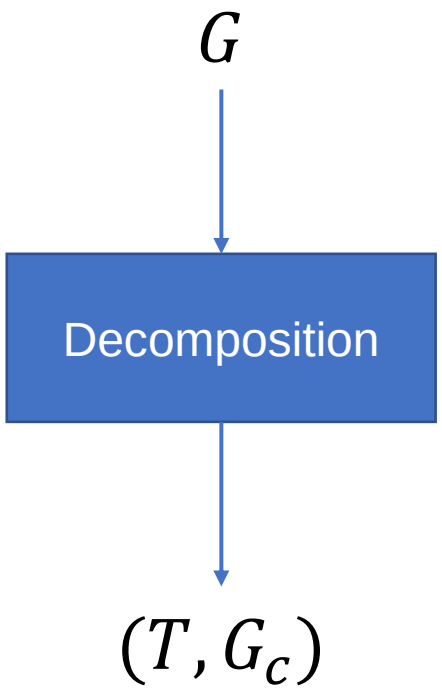
Jin et al., *SIGMOD*

1. Computing a spanning tree T (or a spanning forest)
   - T has an impact on the size of the partial GTC
   - The problem of deriving T that can reduce the size of the partial GTC is **extensively** studied
2. Computing the partial GTC
   - Iterative computation from each vertex

Tree-Based Recursive Graph Decomposition

2021

Chen et al., *TODS*



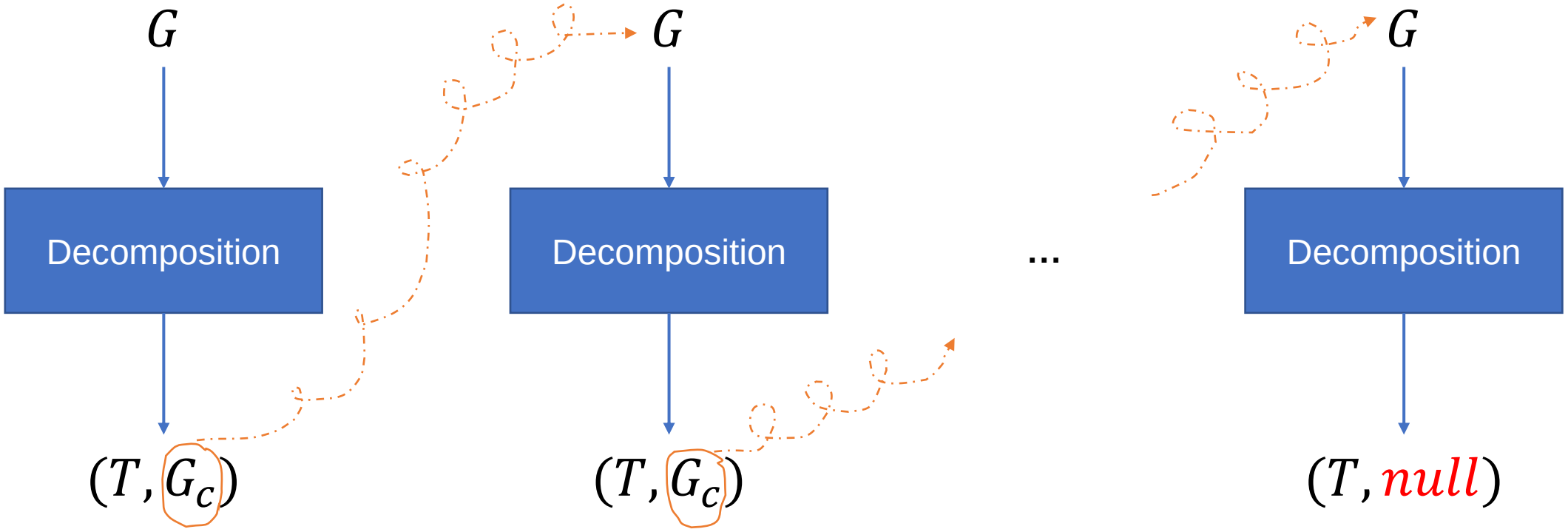
T: an extended spanning tree, reachability on which can be processed by the interval labeling
G_c: a graph summary, only including reachability that cannot be covered by the interval labeling

Y. Chen et al. Graph Indexing for Efficient Evaluation of Label-constrained Reachability Queries. ACM Trans. Database Syst. 46(2): 8:1-8:50 (2021)

Tree-Based Recursive Graph Decomposition

2021

Chen et al., *TODS*



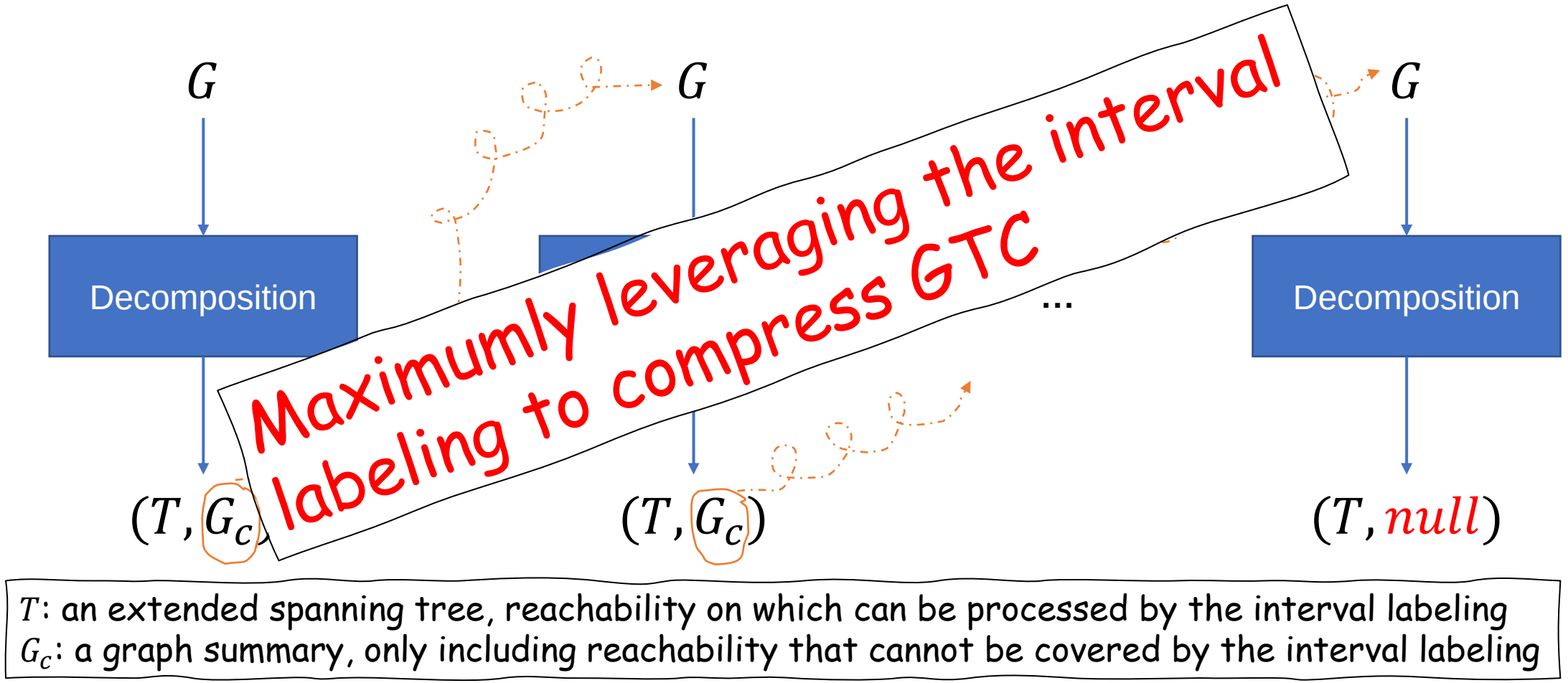
T: an extended spanning tree, reachability on which can be processed by the interval labeling
G_c: a graph summary, only including reachability that cannot be covered by the interval labeling

Y. Chen et al. Graph Indexing for Efficient Evaluation of Label-constrained Reachability Queries. ACM Trans. Database Syst. 46(2): 8:1-8:50 (2021)

Tree-Based Recursive Graph Decomposition

2021

Chen et al., *TODS*



Y. Chen et al. Graph Indexing for Efficient Evaluation of Label-constrained Reachability Queries. ACM Trans. Database Syst. 46(2): 8:1-8:50 (2021)

Edge Classification

2021

Chen et al., *TODS*

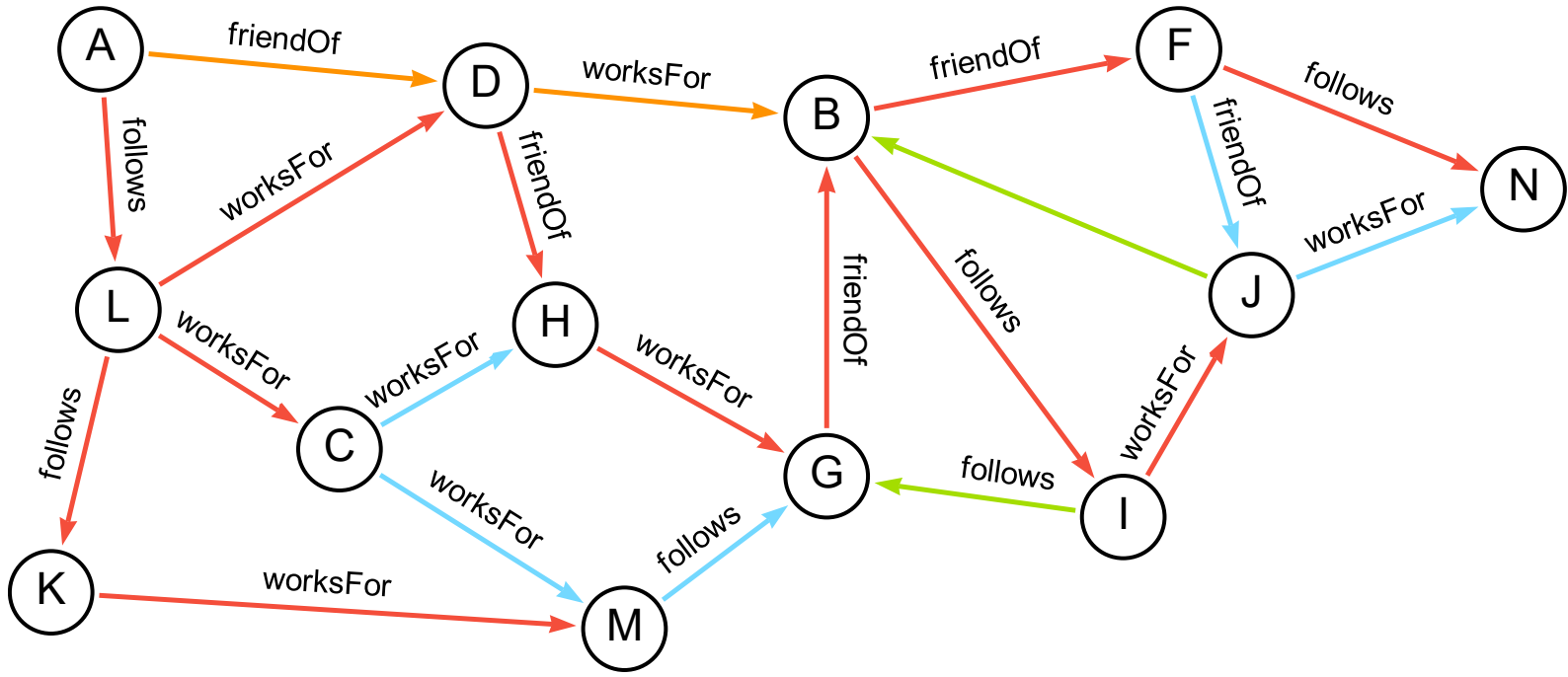
Tree edges (in red)

Forward edges (in orange)

Cross edges (in blue)

Back edges (in green)

Non-tree edges



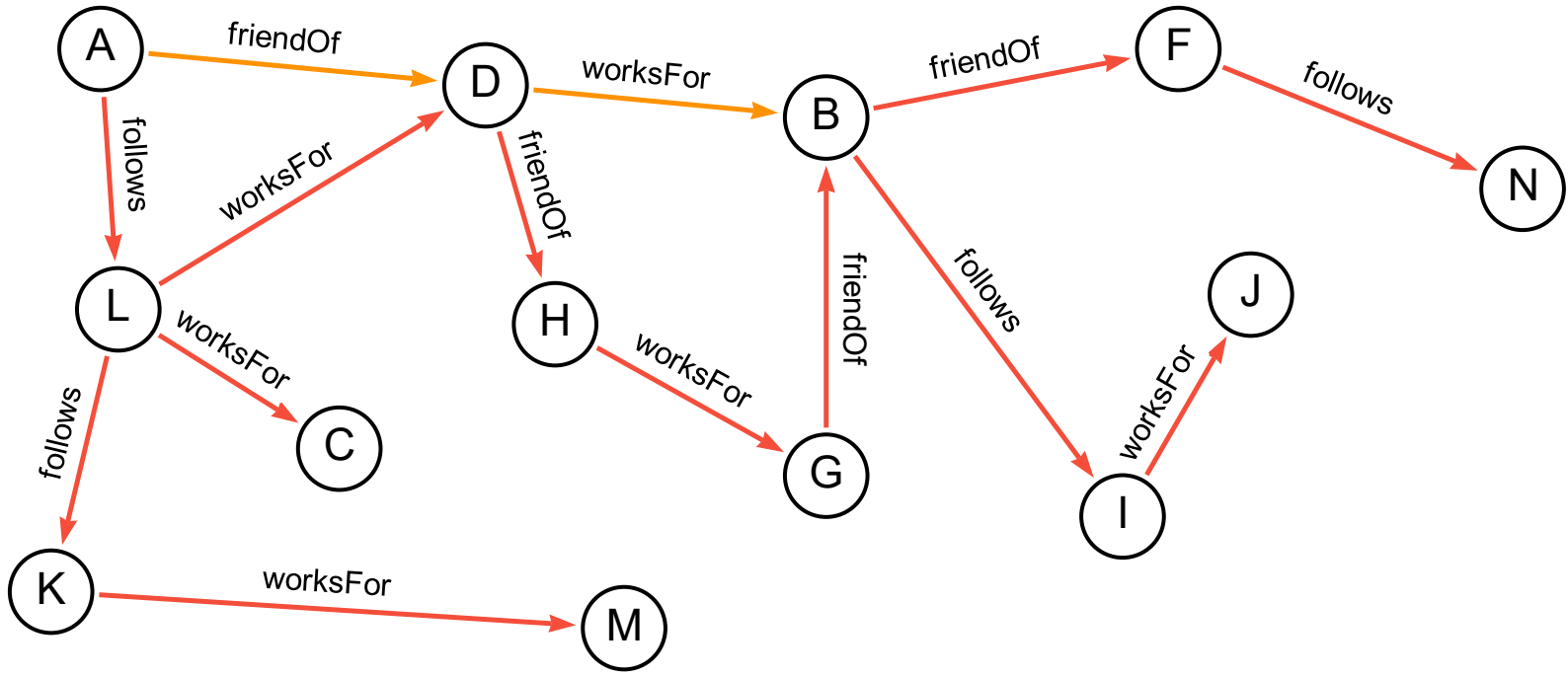
Extended Trees

2021

Chen et al., *TODS*

Tree edges (in red)

Forward edges (in orange)



Extended Trees

2021

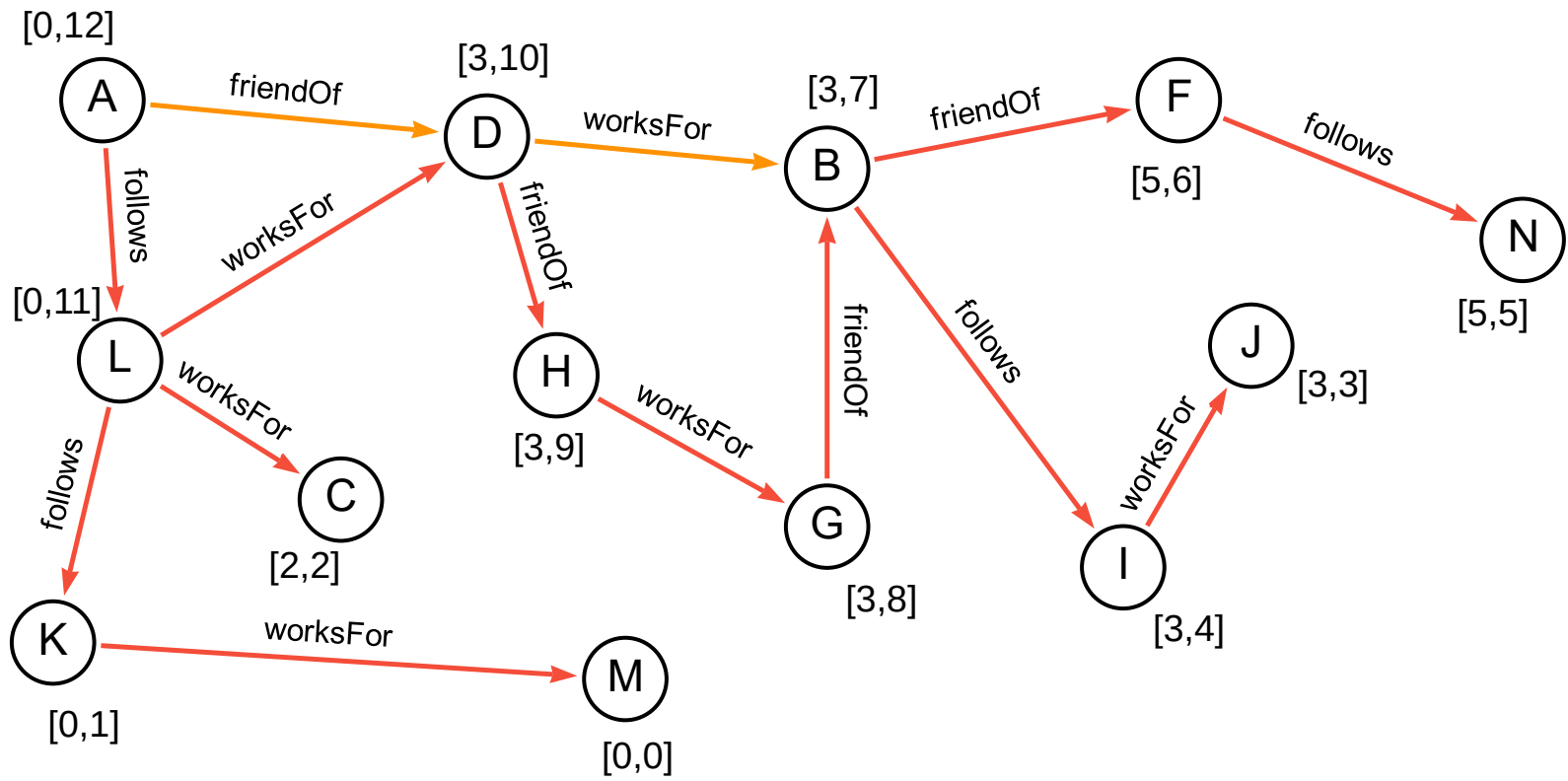
Chen et al., *TODS*

Tree edges (in red)

Forward edges (in orange)

Reachability processing:

- Interval labeling
- Histogram SPLSs



Histogram SPLSs from A to H: $\{friendOf: 2\}$

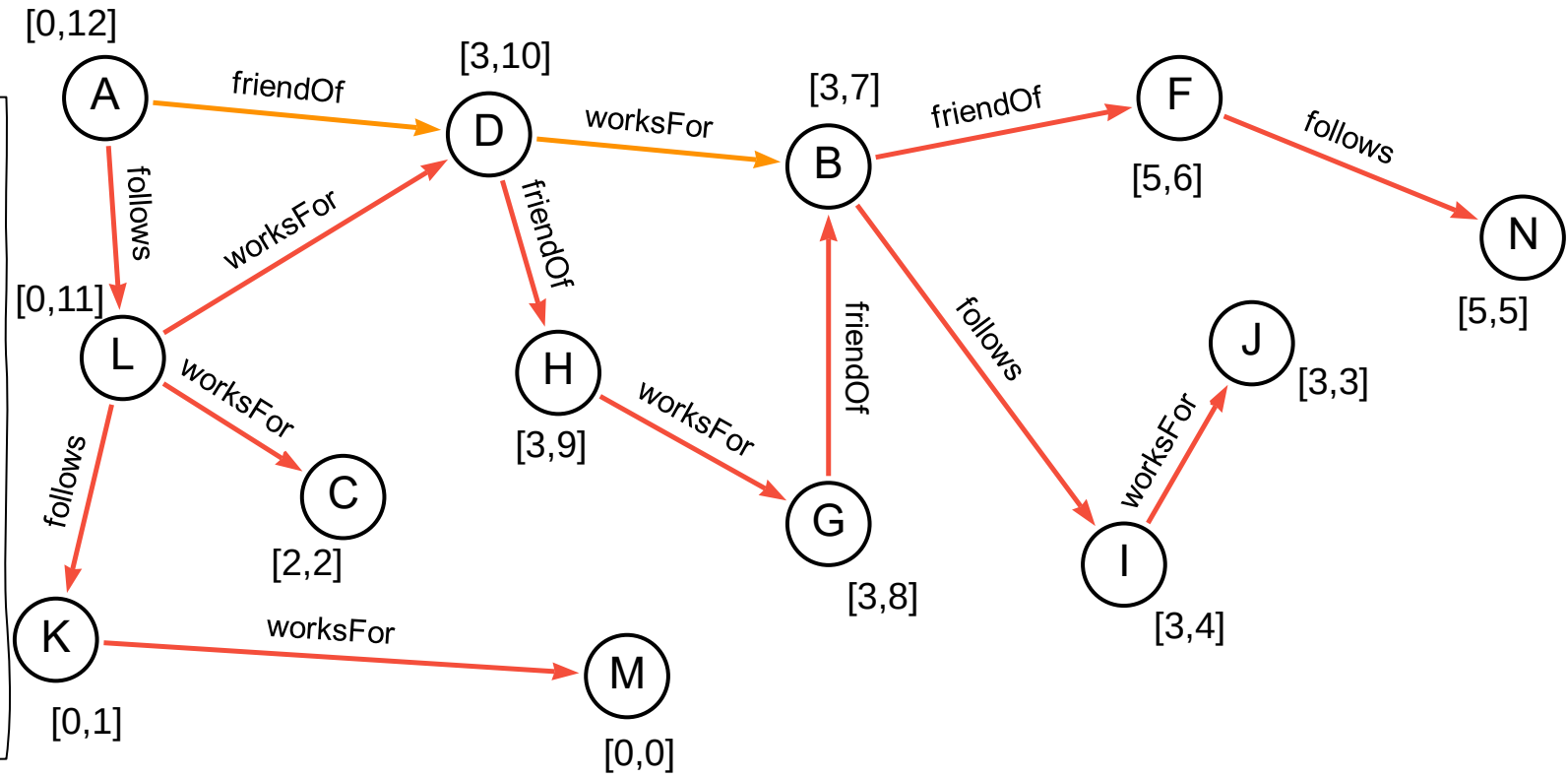
Extended Trees

2021

Chen et al., *TODS*

$Q_r(D, F, (worksFor \cup friendOf)^*)$

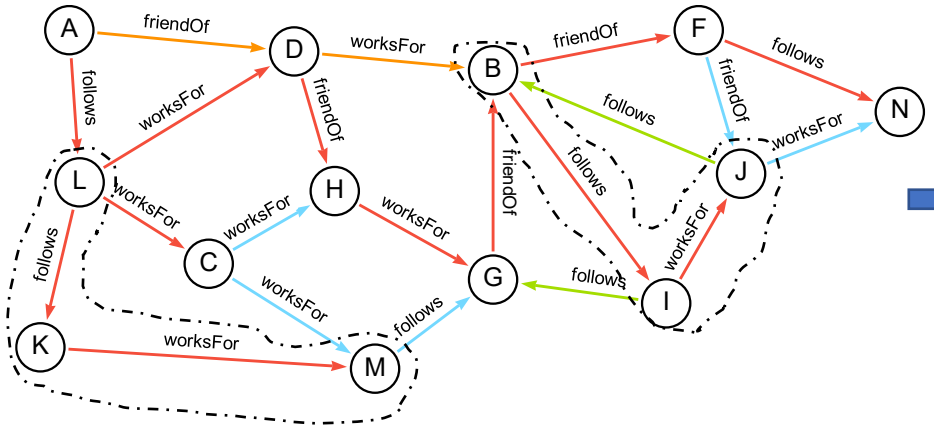
- The interval of D $[3,10]$ contains the interval of F $[5,6]$
- Histogram of D: $\{friendOf: 1\}, \{follows: 1, worksFor: 1\}$
- Histogram of F: $\{friendOf: 2, worksFor: 1\}$
- SPLS from D to F: $\{friendOf: 1, worksFor: 1\}$



Decomposition and Graph Summary

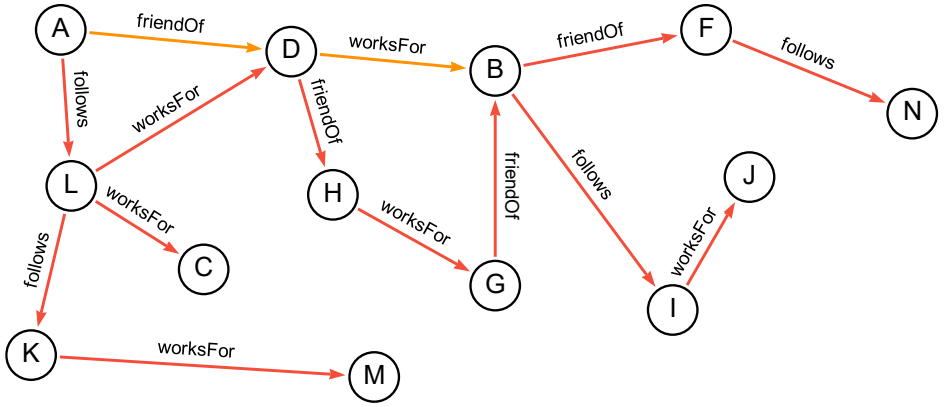
2021

Chen et al., *TODS*



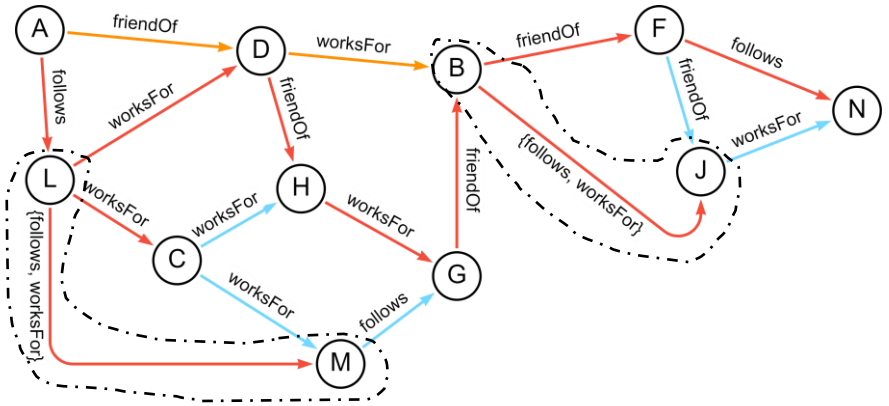
Decomposition

Extended tree



Graph Summary

Containing reachability information through cross edges



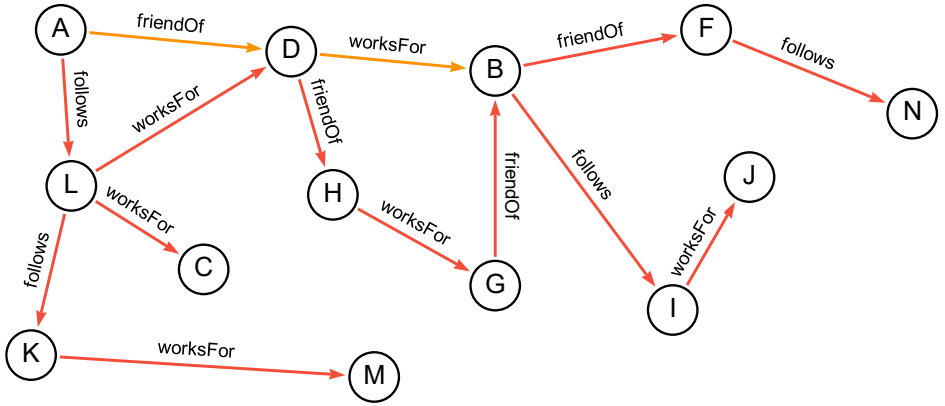
The path from L to M via K and the path from B to J via I are **compressed**

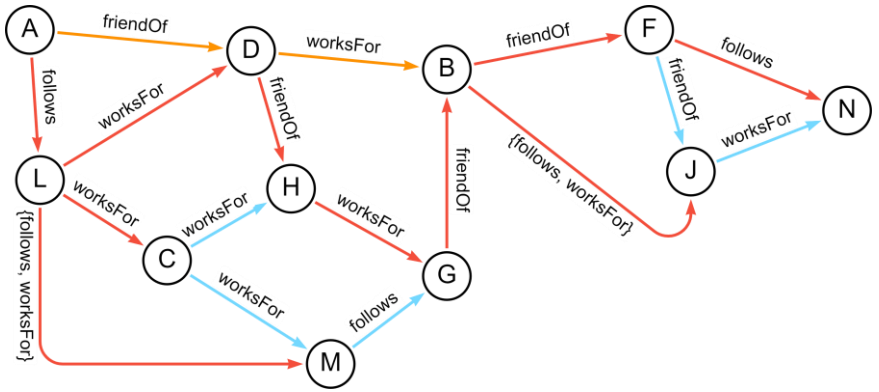
Query Decomposition

2021

Chen et al., *TODS*

- $Q_r(L, F, (worksFor \cup friendOf)^*)$ is decomposed into three sub-queries:
  - Q1: $Q_r(L, C, (worksFor \cup friendOf)^*)$
  - Q2: $Q_r(C, G, (worksFor \cup friendOf)^*)$
  - Q3: $Q_r(G, F, (worksFor \cup friendOf)^*)$
- Subquery processing:
  - Q1 and Q3: extended tree
  - Q2: graph summary (recursively decomposed)



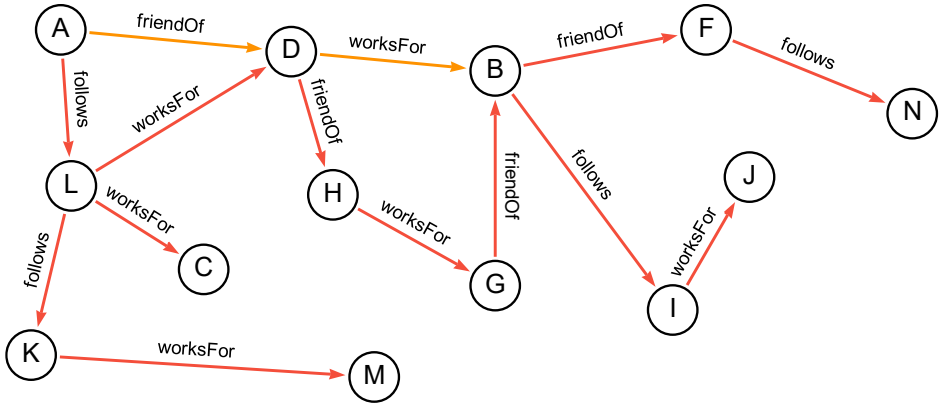


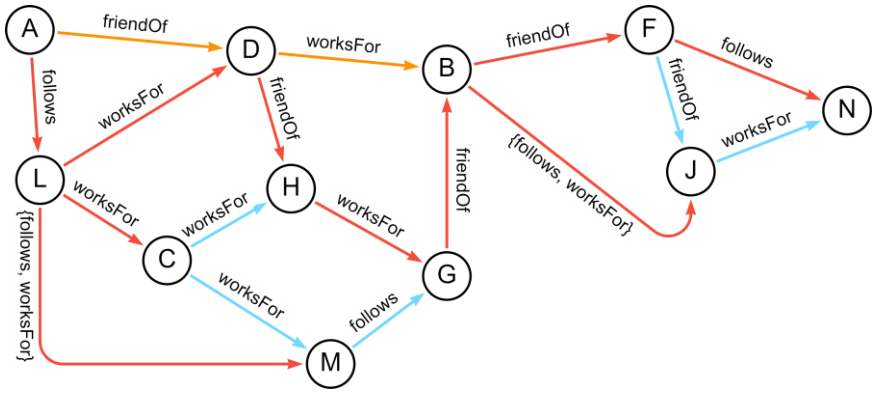
2021

Chen et al., *TODS*

Query Decomposition

- $Q_r(L, F, (worksFor \cup friendOf)^*)$ is decomposed into three sub-queries:
  - Q1: $Q_r(L, C, (worksFor \cup friendOf)^*)$
  - Q2: $Q_r(C, G, (worksFor \cup friendOf)^*)$
  - Q3: $Q_r(G, F, (worksFor \cup friendOf)^*)$
- Subquery processing:
  - Q1 and Q3: extended tree
  - Q2: graph summary (recursively decomposed)





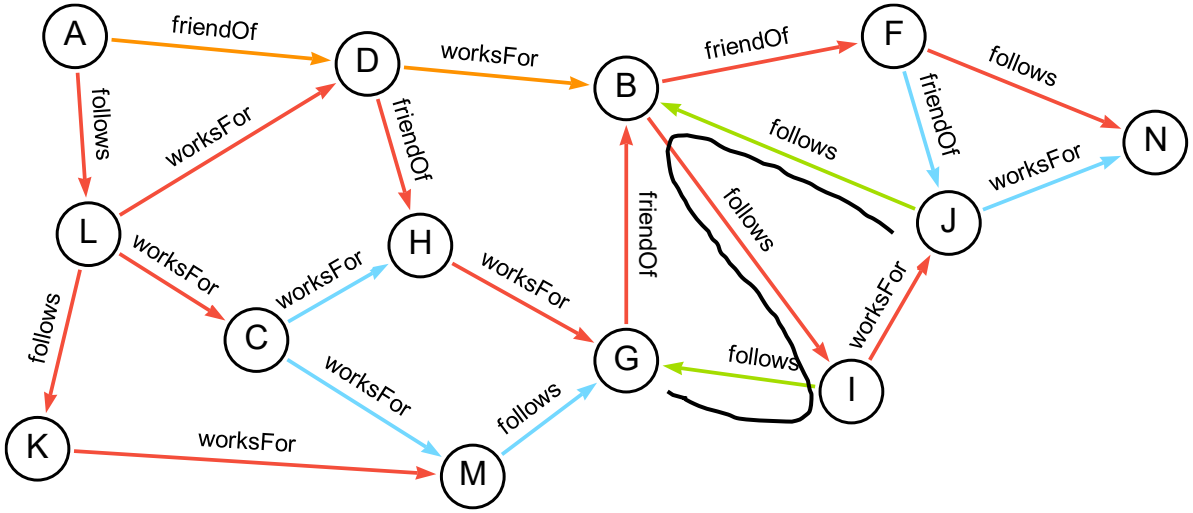
- C is a dominant vertex of L
  - Through C, outgoing reachability with cross edges can be found
- G is a transferring vertex of F
  - Through G, incoming reachability with cross edges can be found

Reachability via Back Edges

2021

Chen et al., *TODS*

- Back edges are compressed to form back chains:
- Example:
  - Back edges: $(J, follows, B)$ and $(I, follows, G)$
  - Back chain: a path from J to G with $\{follows\}$
- Back chains are stored together with extended trees, which will be traversed to process queries



Index Frameworks for Alternation-Based Queries

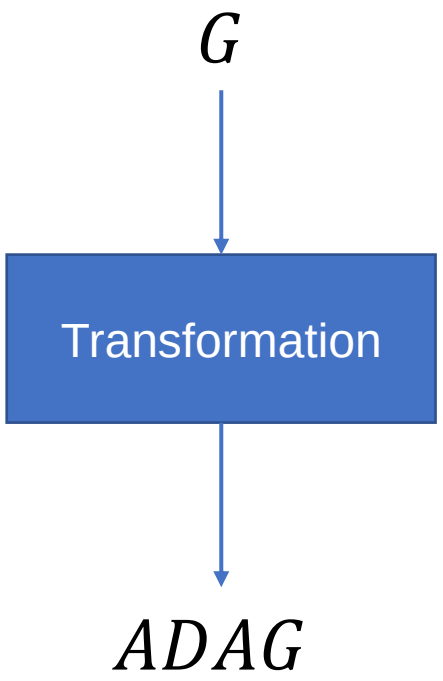
- Tree Cover
  - *How to deal with non-tree edges*
  - *How to combine the interval labeling with SPLSs*
- Generalized TC
  - *How to efficiently compute generalized TC*
- 2-Hop Labeling
  - *How to combine 2-hop labeling with SPLSs*
  - *Dynamic graphs*

Efficient Computation of GTC

2014

Zou et al., Inf. Syst.

1. Transformation from a general graph into an augmented DAG (ADAG)
2. Computing GTC by combining
   a. Reachability in the ADAG
   b. Reachability in each SCC
   c. Reachability between vertices in the ADAG and vertices in the SCCs



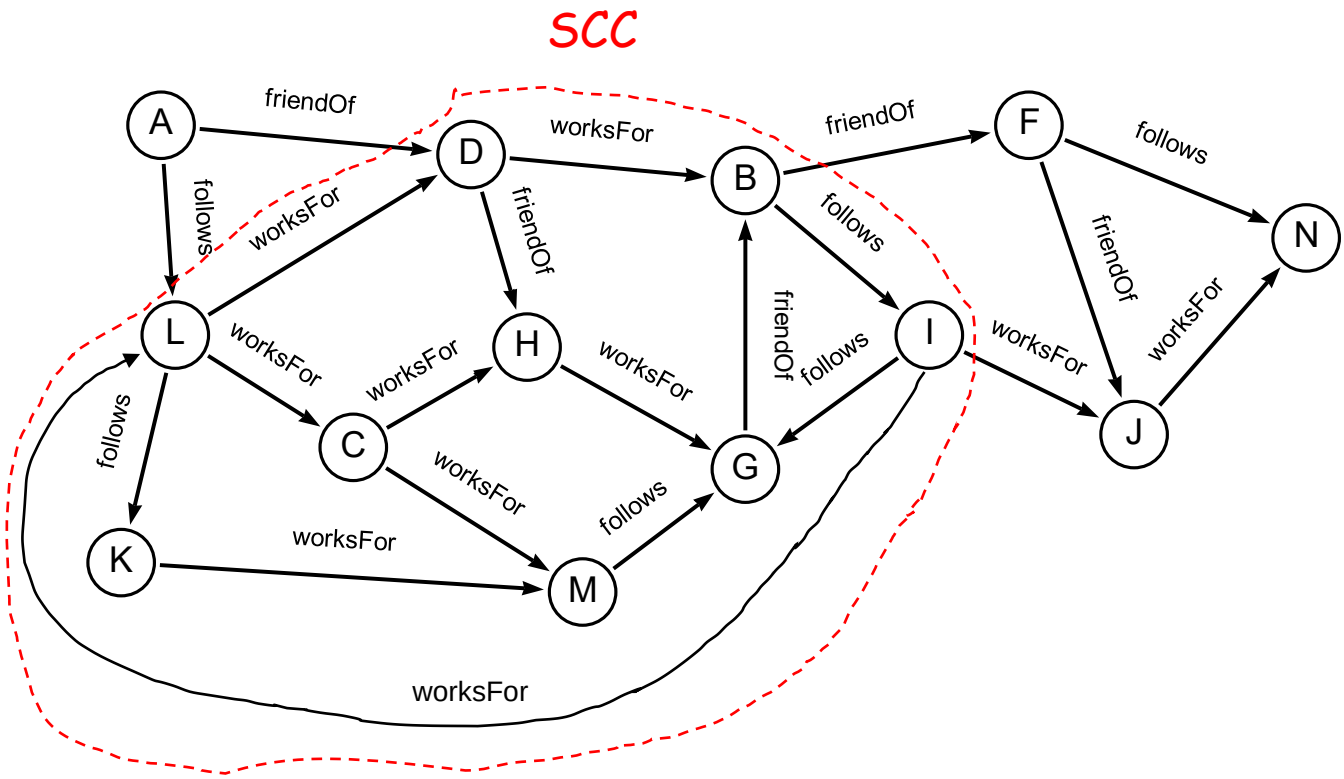
The transformation is more complicated than the one for plain graphs

Transformation into Augmented DAG

2014

Zou et al., Inf. Syst.

- Computing SCCs

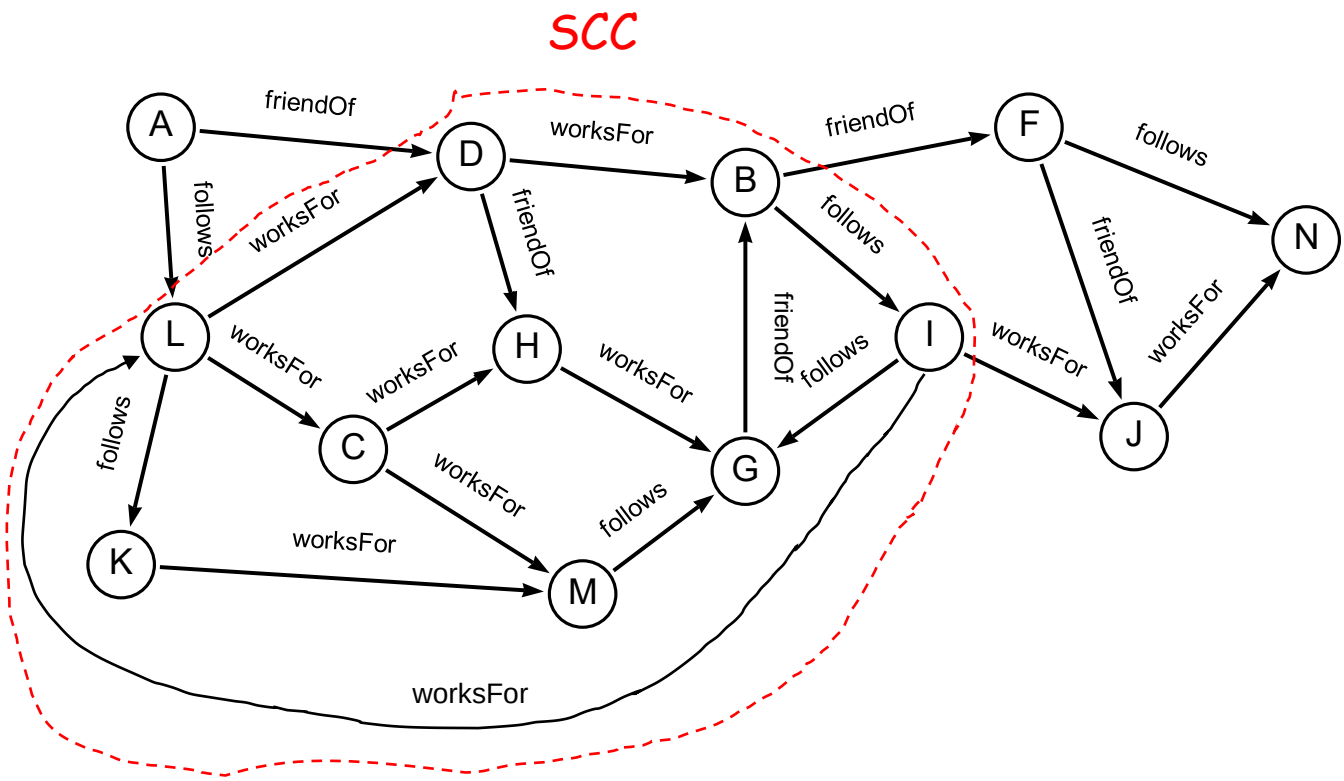


2014

Zou et al., Inf. Syst.

Transformation into Augmented DAG

- Computing SCCs
- Identifying two kinds of vertices in SCCs
  - In-portal vertices: having **incoming** edges from vertices out of the SCC
    - E.g., vertices L and D
  - Out-portal vertices: having **outgoing** edges to vertices out of the SCC
    - E.g., vertices B and I



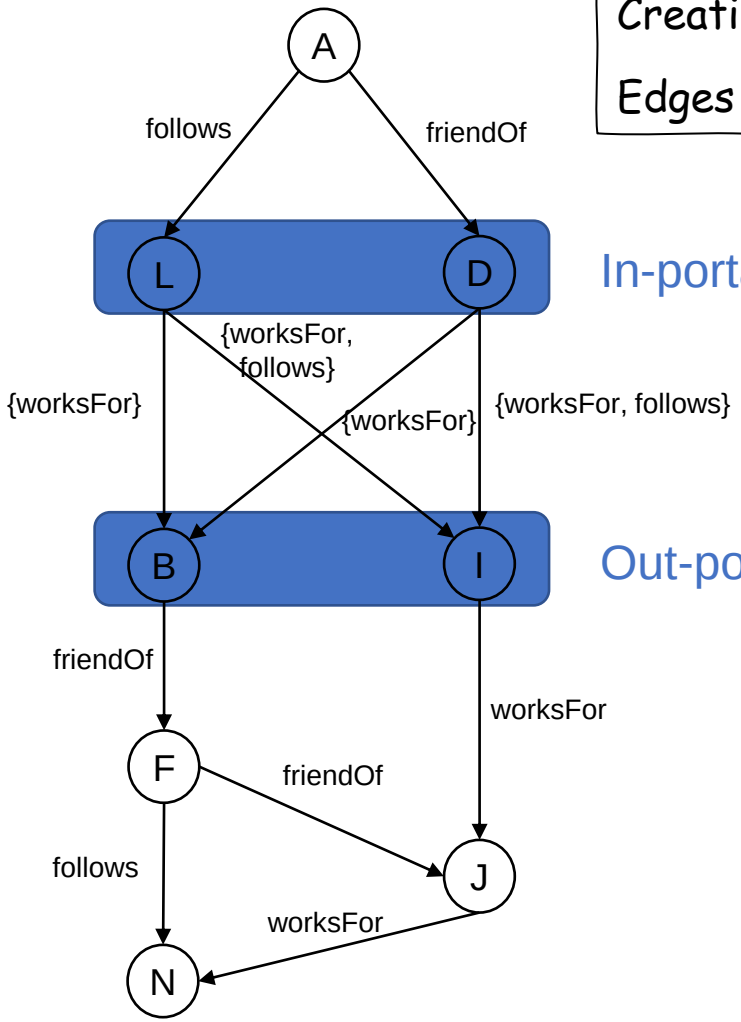
2014

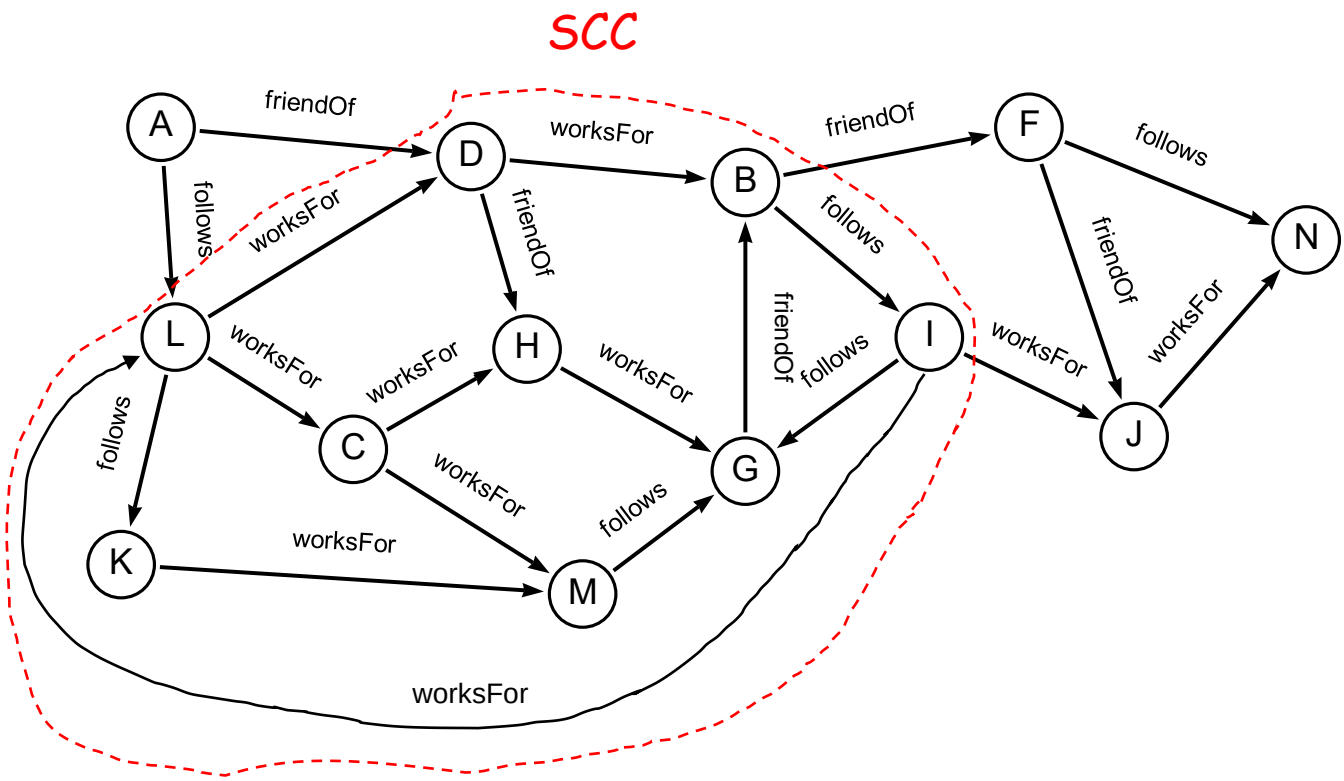
Zou et al., Inf. Syst.

Transformation into Augmented DAG

Creating a bipartite graph for each SCC with the in-portal and out-portal vertices

Edges from in-portal to out-portal vertices are labeled with SPLSs





2014

Zou et al., Inf. Syst.

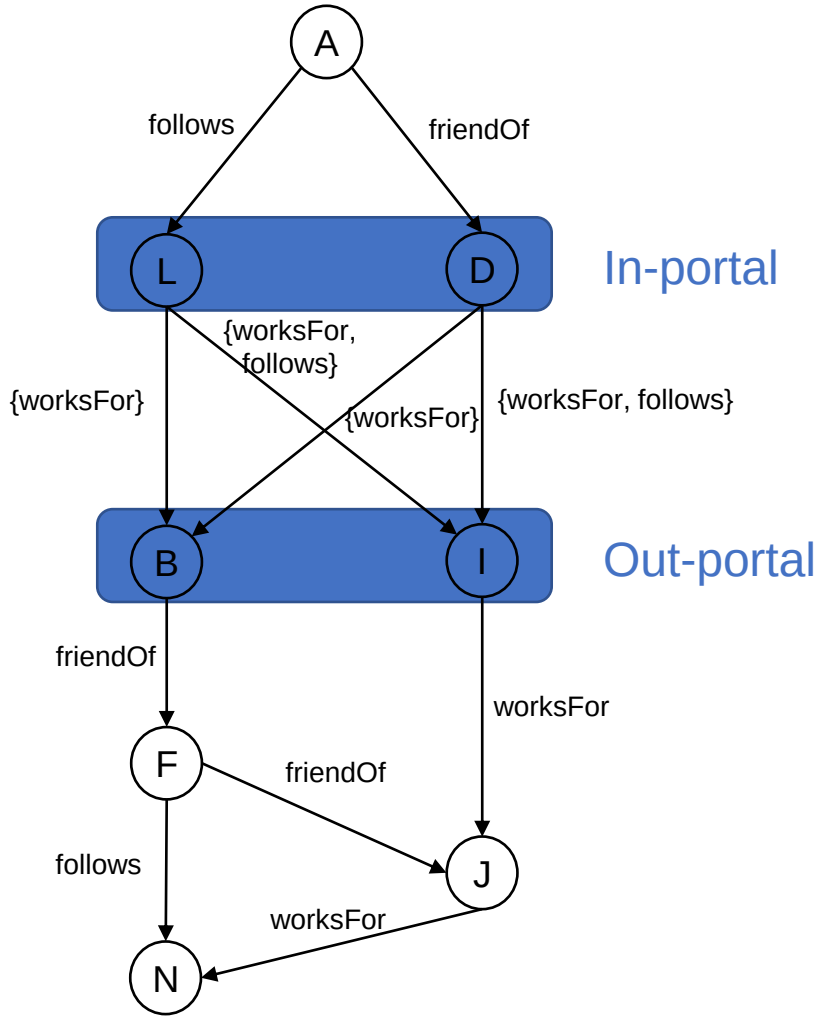
Reachability in ADAG

- Incremental computation
  - Examining vertices in the reverse topological order
  - Sharing the single-source GTC in a bottom-up manner

GTC

Source	Target	SPLSs
…	…	…
F	N	$\{follows\},\{friendOf, worksFor\}$
F	J	$\{friendOf\}$
…	…	…

The single-source GTC of F



Single-source GTC: all the vertices that are reachable from a fixed source and the corresponding SPLSs

Reachability in ADAG

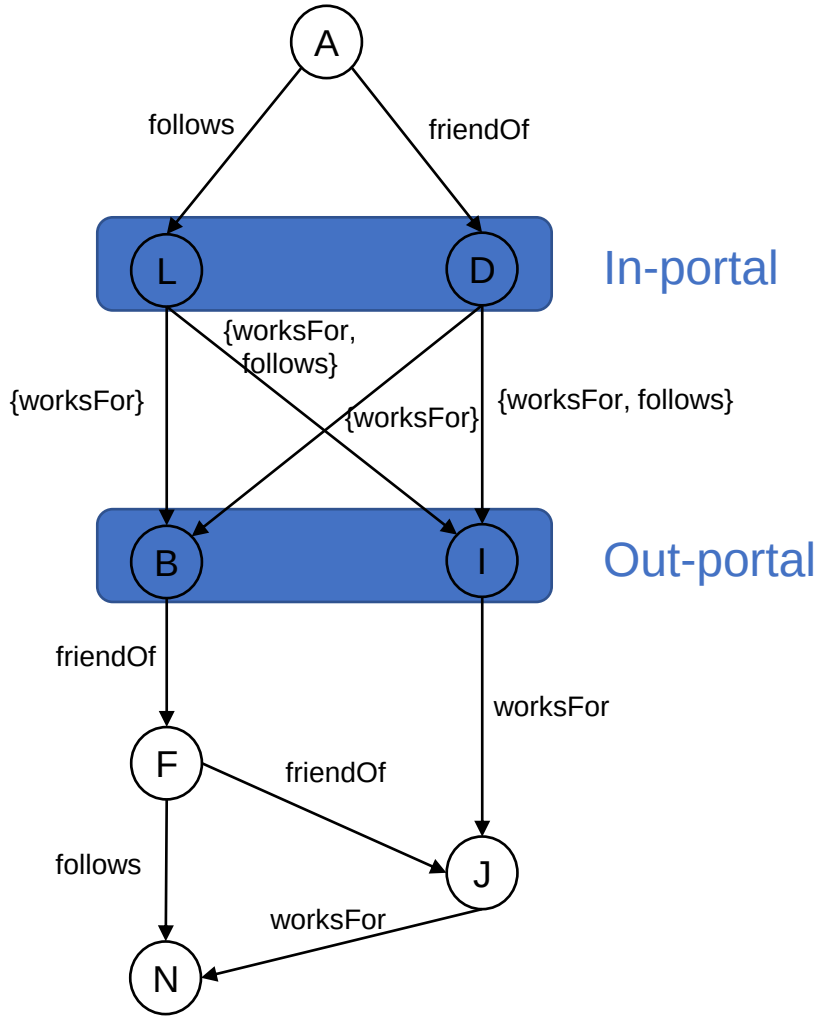
2014

Zou et al., Inf. Syst.

- Incremental computation
  - Examining vertices in the reverse topological order
  - Sharing the single-source GTC in a bottom-up manner

GTC

Source	Target	SPLSs
…	…	$(B, friendOf, F)$ … $+$
F	N	$\{follows\}, \{friendOf, worksFor\}$
F	J	$\{friendOf\}$
B	N	$\{friendOf, follows\}, \{friendOf, worksFor\}$
B	J	$\{friendOf\}$
…	…	…

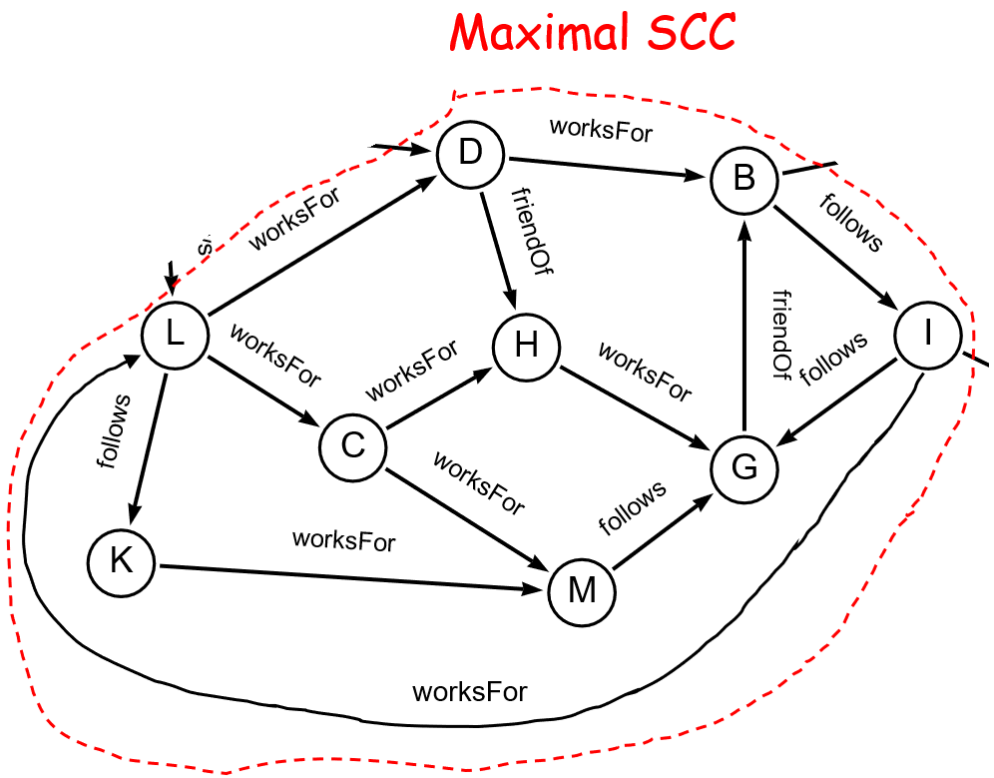


Reachability in Each SCC

2014

Zou et al., Inf. Syst.

- Dijkstra-like algorithm [Zou14]
  - Simulating distance using distinct labels in paths
- Example: two paths from L to H
  - P1: $L, worksFor, D, friendOf, H$
  - P2: $L, worksFor, C, worksFor, H$
  - P2 is considered shorter than P1
  - P1 is pruned for computing the single-source GTC from L



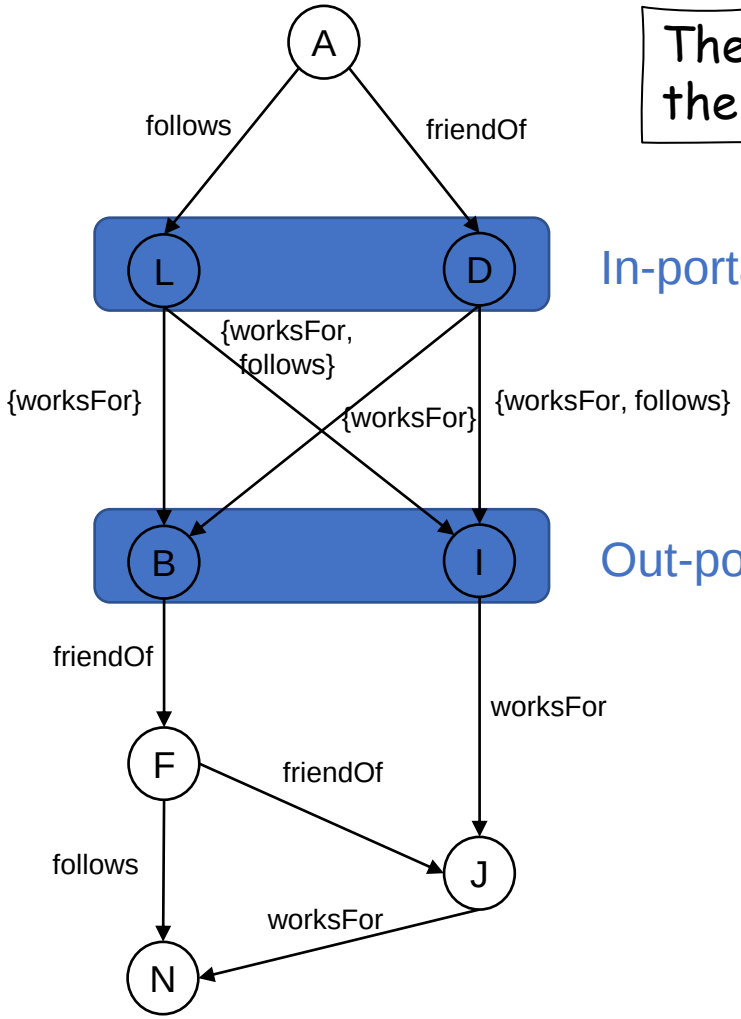
[Zou14] L. Zou et al. Efficient processing of label-constraint reachability queries in large graphs. Inf. Syst. 40: 47-66 (2014)

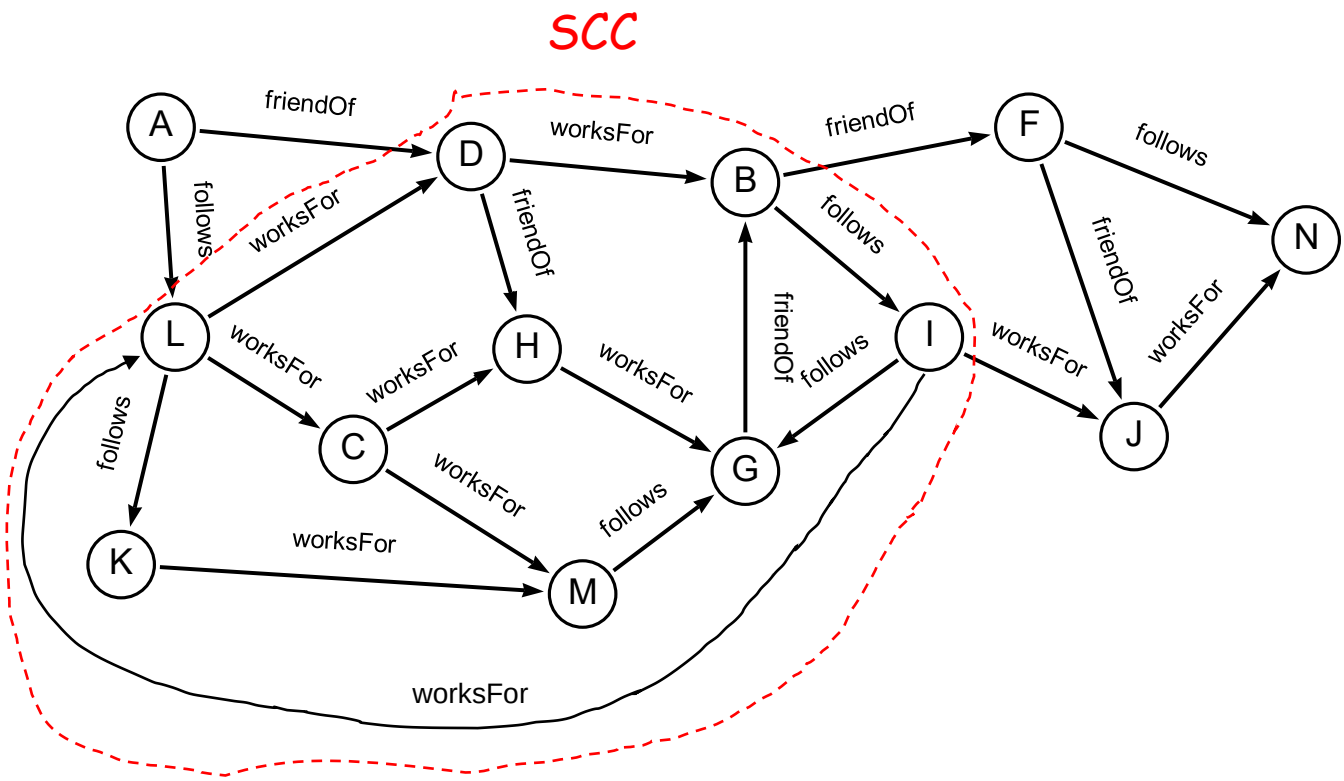
Reachability from SCC to ADAG

2014

Zou et al., Inf. Syst.

The single-source GTC from G can be computed by using the single-source GTC from out-portal vertices, i.e., B and I



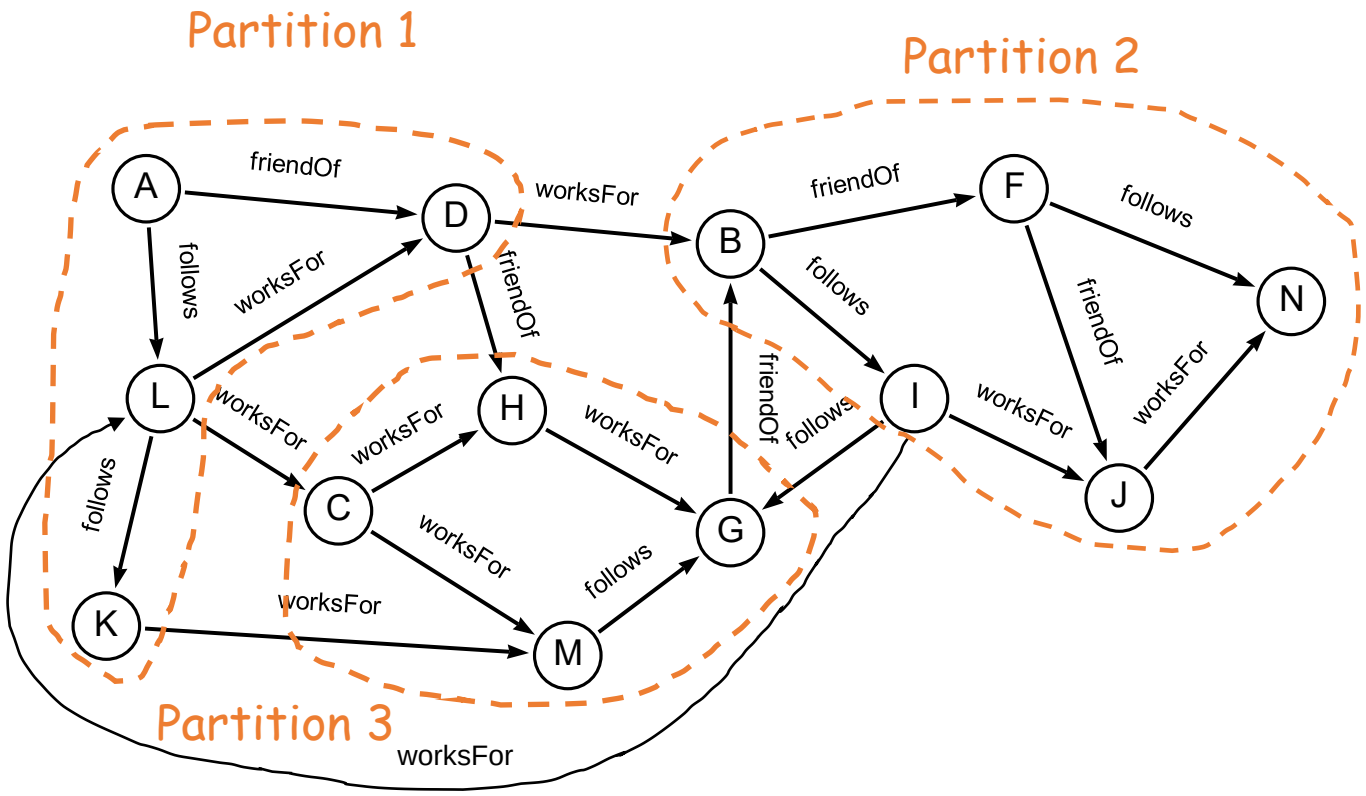


2014

Zou et al., Inf. Syst.

GTC on Large Graphs

- Graph partitioning
  - Computing a local GTC for each partition

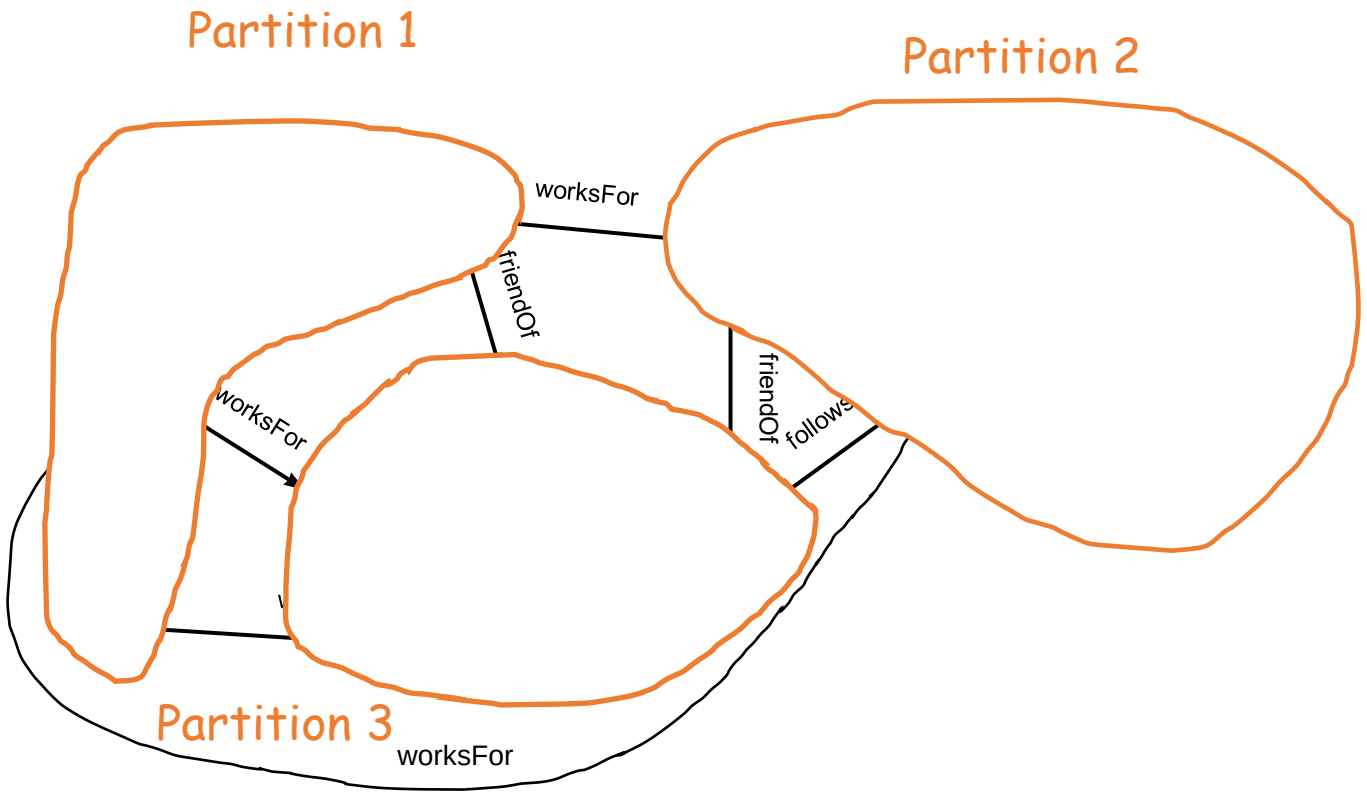


2014

Zou et al., Inf. Syst.

GTC on Large Graphs

- Graph partitioning
  - Computing a local GTC for each partition
- Building a skeleton graph over the partitions
- Query processing:
  - Online traversal on the skeleton graph with index lookups in each local GTC

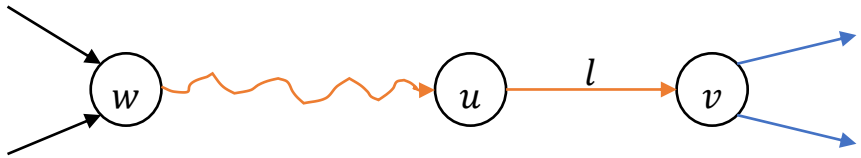


GTC on Dynamic Graphs

2014

Zou et al., Inf. Syst.

- Edge insertion: (u, v) with label l
  - Performing the Dijkstra-like algorithm from v in the **backward** manner
  - For each visited vertex w, updating the single-source GTC of w by using
    - the computed SPLSs from w to v, and
    - the single-source GTC from v



- Edge deletion: (u, v) with label l
  - If u and v are in different SCCs
    - The GTC for each SCC are not changed
    - Reflecting the changes for vertices that can reach v in the ADAG
  - If u and v are in the same SCC
    - Recomputing the vertices in the SCC
    - Finally, reflecting the changes in the ADAG, starting from the lowest changed topological level

2017

Landmark Index, *SIGMOD*

Landmark Index

- Landmark vertices
  - Top-k vertices in the degree ranking
- Computing the single-source GTC for each landmark vertex [Val17]
- Querying processing
  - BFS accelerated by hitting landmark vertices

[Val17] L. Valstar et al. Landmark Indexing for Evaluation of Label-Constrained Reachability Queries. SIGMOD Conference 2017: 345-358

2017

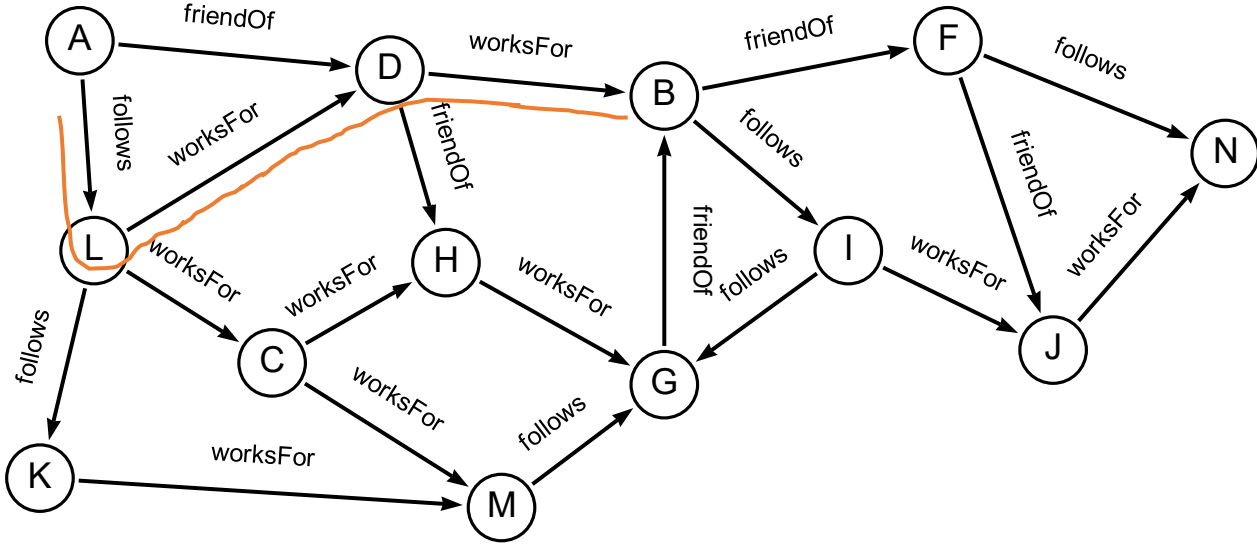
Landmark Index, *SIGMOD*

Landmark Index

- Landmark vertex: B

Single-Source GTC for B

Target	SPLSs
F	$\{friendOf\}$
I	$\{follows\}$
G	$\{follows\}$
J	$\{friendOf\}, \{follows, worksFor\}$
N	$\{follows, worksFor\}$, $\{friendOf, worksFor\}$, $\{friendOf, follows\}$



$Q_r(A, N, (worksFor \cup follows)^*)$:

- BFS from A visits B with $\{follows, worksFor\}$
- Using the single-source GTC for B

2017

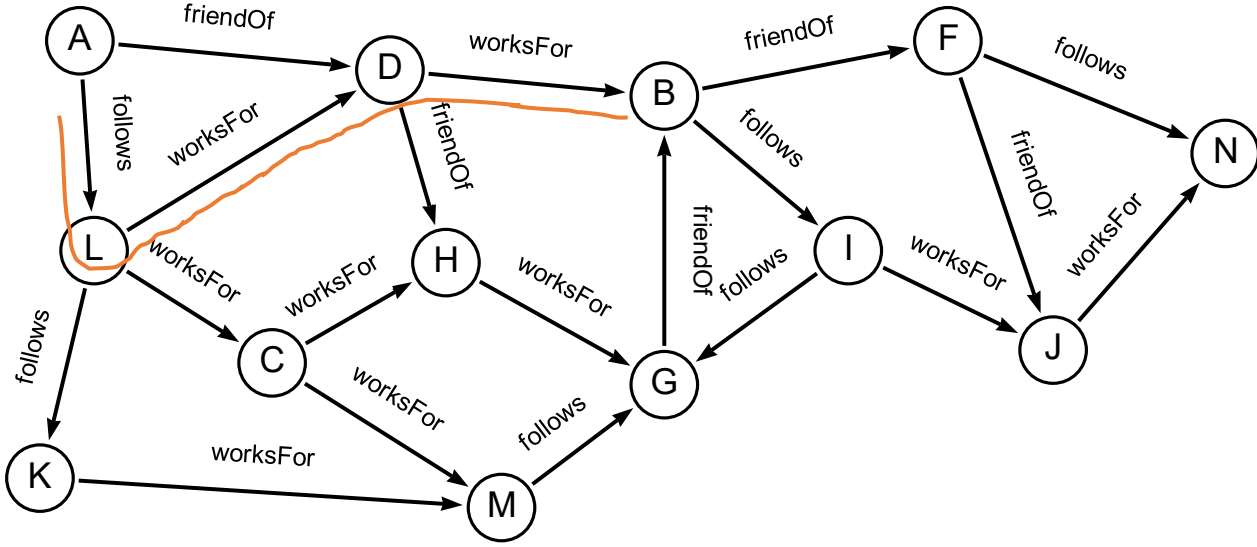
Landmark Index, *SIGMOD*

Optimization for Non-Landmark Vertices

- For each non-landmark v, precomputing a fixed number of SPLSs from v to the landmark vertices

Index Entries for A

Landmark	SPLSs
B	$\{follows, worksFor\}$, $\{friendOf\ worksFor\}$



$Q_r(A, N, (worksFor \cup follows)^*)$:

- BFS from A visits B with $\{follows, worksFor\}$
- Using the single-source GTC for B

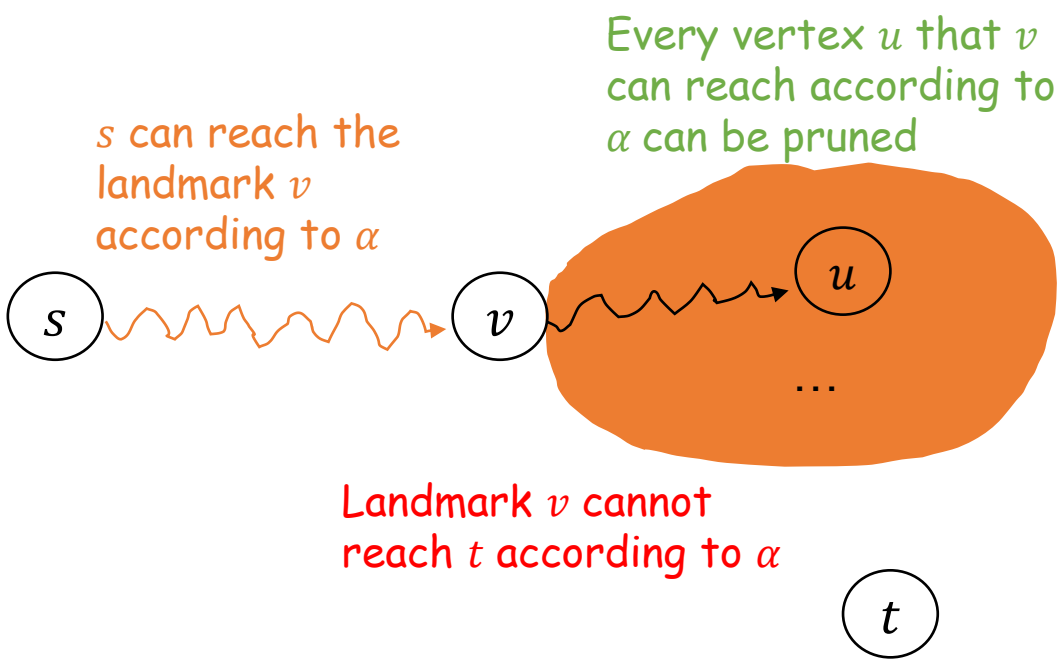
← Optimization

2017

Landmark Index, *SIGMOD*

Optimization for False Queries

- Query: $Q_r(s, t, \alpha),\ \alpha = (l_1 \cup \cdots \cup l_k)^*$
  - For False queries, the index is not helpful in the previous query algorithm
- Pruning:
  - If the landmark cannot reach the target according to α, then all the vertices that are reachable form the landmark according to α can be **pruned**

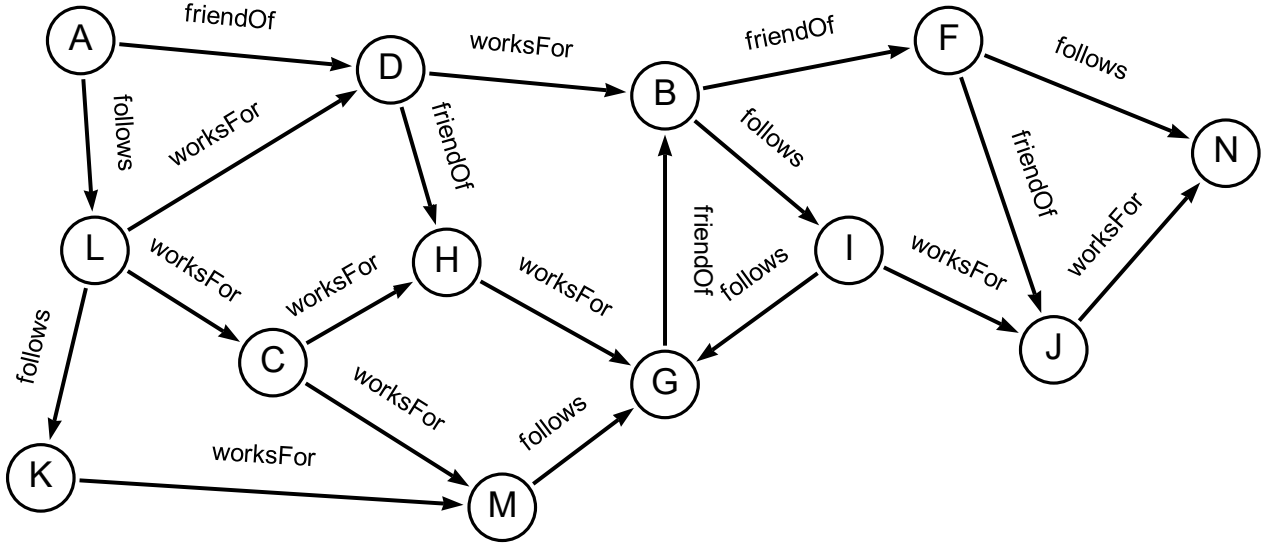


2017

Landmark Index, *SIGMOD*

Optimization for False Queries

- Query: $Q_r(s, t, \alpha),\ \alpha = (l_1 \cup \cdots \cup l_k)^*$
  - For False queries, the index is not helpful in the previous query algorithm
- Pruning:
  - If the landmark cannot reach the target according to α, then all the vertices that are reachable form the landmark according to α can be **pruned**



Single-Source GTC for B

Target	SPLSs
F	$\{friendOf\}$
I	$\{follows\}$
G	$\{follows\}$
J	$\{follows, worksFor\}, \{friendOf\}$
N	$\{follows, worksFor\}, \{friendOf, worksFor\}, \{friendOf, follows\}$

$Q_r(D, F, (worksFor \cup follows)^*)$:

- D can reach B with $\{worksFor\}$
- $Q_r(B, F, (worksFor \cup follows)^*)$ = False
- All the vertices u, such that $Q_r(B, u, (worksFor \cup follows)^*)$ = True, can be pruned
- The BFS only visits I that has been pruned, thus return False

Index Frameworks for Alternation-Based Queries

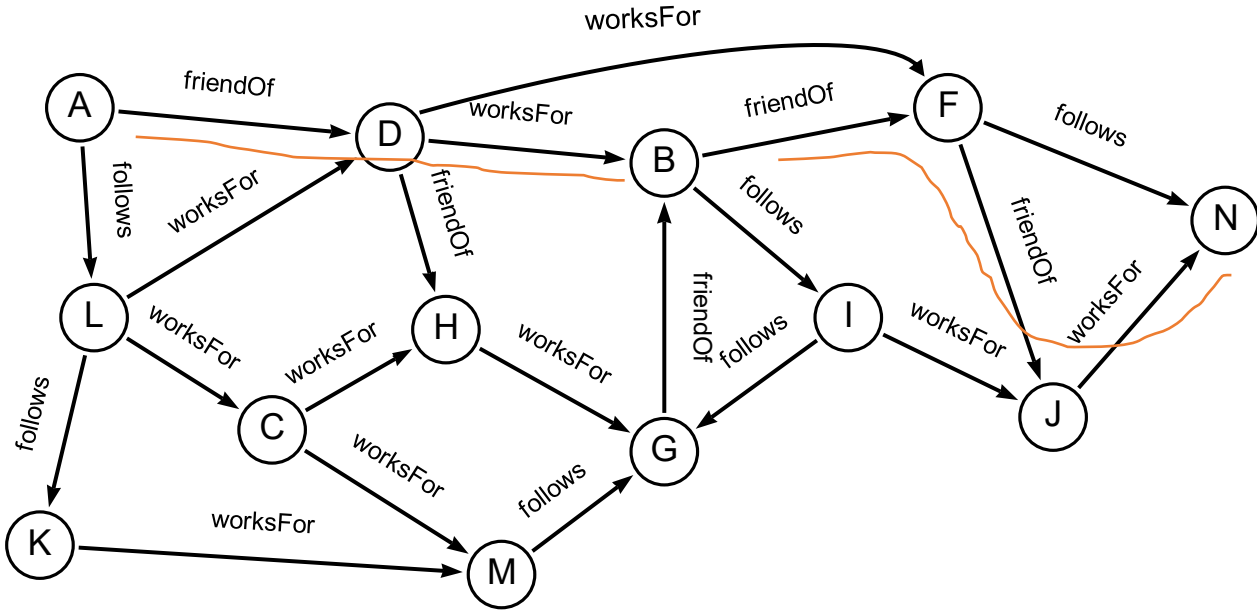
- Tree Cover
  - *How to deal with non-tree edges*
  - *How to combine the interval labeling with SPLSs*
- Generalized TC
  - *How to efficiently compute generalized TC*
- 2-Hop Labeling
  - *How to combine 2-hop labeling with SPLSs*
  - *Dynamic graphs*

Label-Constrained 2-Hop Labeling

2020

P2H+, *VLDB*

- Intuition:
  - Plain reachability is transitive
  - SPLSs are transitive
  - Adding SPLSs into the 2-hop index
- $Q_r(A, N, (worksFor \cup friendOf)^*)$:
  - Plain reachability:
    - A can reach B
    - B can reach N
  - Path constraints:
    - SPLSs from A to B contains $\{worksFor, friendOf\}$
    - SPLSs from B to N contains $\{worksFor, friendOf\}$
  - Thus, return True



Y. Peng et al. Answering billion-scale label-constrained reachability queries within microsecond. Proc. VLDB Endow. 13(6): 812-825 (2020)

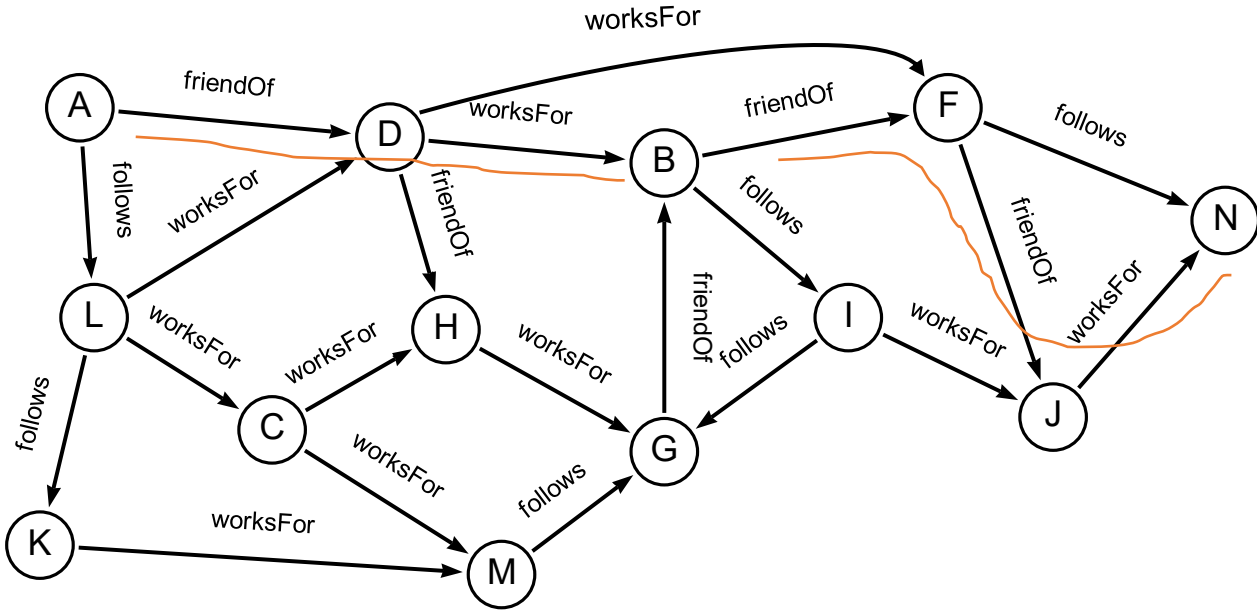
2020

P2H+, *VLDB*

Example: P2H+

P2H+
(*incomplete view*)

v	$L_{in}(v)$	$L_{out}(v)$
A	…	$(B, \{worksFor, friendOf\}), \ldots$
B	…	…
C	…	…
D	…	…
F	$(B, \{friendOf\}), \ldots$	…
G	$(B, \{follows\}), \ldots$	$(B, \{friendOf\}), \ldots$
H	$(D, \{friendOf\}), \ldots$	…
I	$(B, \{friendOf\}), \ldots$	…
J	…	…
K	…	…
L	…	$(D, \{worksFor\}), \ldots$
M	…	…
N	$(B, \{worksFor, friendOf\}), \ldots$	…



$Q_r(A, N, (worksFor \cup follows)^*)$:

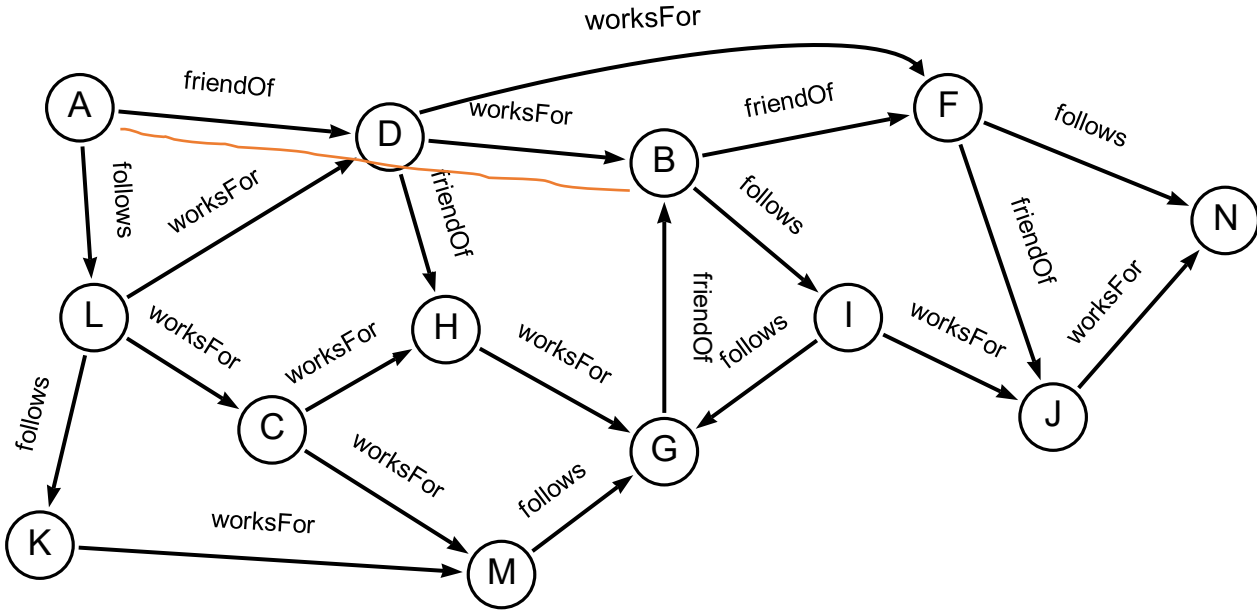
- $(B, \{worksFor, friendOf\}) \in L_{out}(A)$
- $(B, \{worksFor, friendOf\}) \in L_{in}(N)$
- Thus, return True

2020

P2H+, *VLDB*

The Indexing Algorithm of P2H+

- Augmenting PLL [Aki13,Yan13] with SPLSs
  - PLL:
    - *Vertex accessing order: the descending order based on vertex degree*
    - *Backward and forward BFS from each vertex according to the accessing order to create index entries*
    - *Applying pruning rules during BFSs*
  - Augmentation with SPLSs:
    - During the traversal in BFS, computing SPLSs
- Example:
  - Backward BFS from B visits A with $\{worksFor, friendOf\}$, thus inserting $(B, \{worksFor, friendOf\})$ into $L_{out}(A)$



[Aki13] E. Akiba et al. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. SIGMOD Conference 2013: 349-360

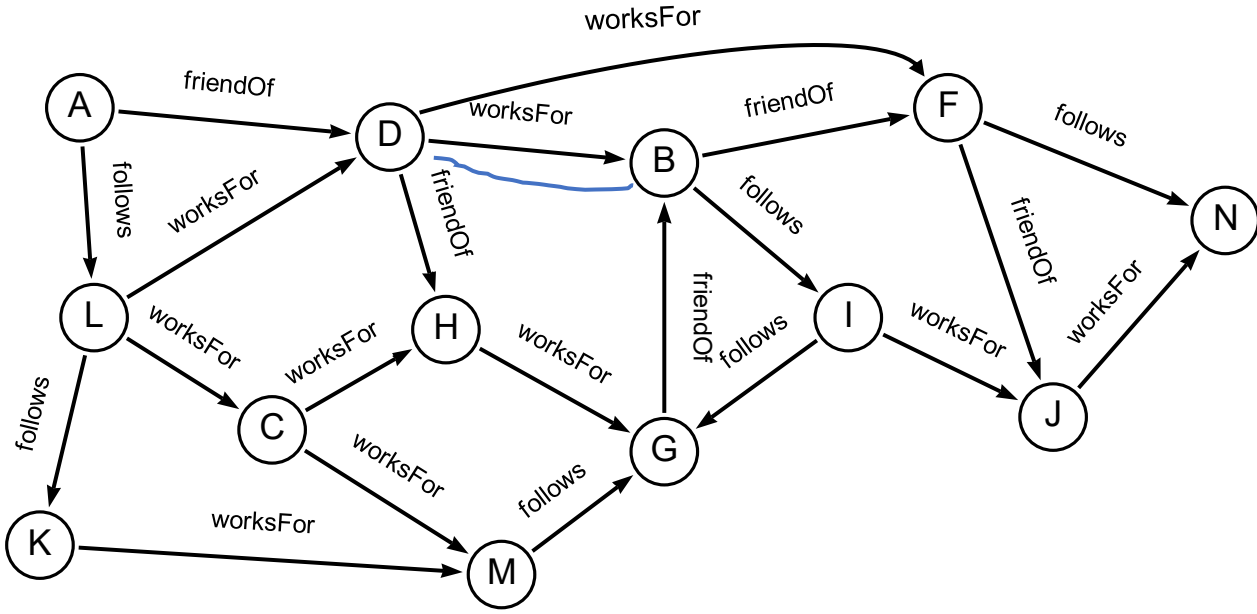
[Yan13] Y. Yano, et al. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In CIKM'13.

2020

P2H+,
VLDB

Pruning Rules in P2H+

- Pruning rule 1:
  - Skipping vertices according to the vertex accessing order
- Vertex accessing order:
  - $(B, D, G, L, C, F, H, I, J, M, K, A, N)$
- Example:
  - Forward BFS from D visits B
  - However, B has been processed
  - Then, this path does not need to be expanded

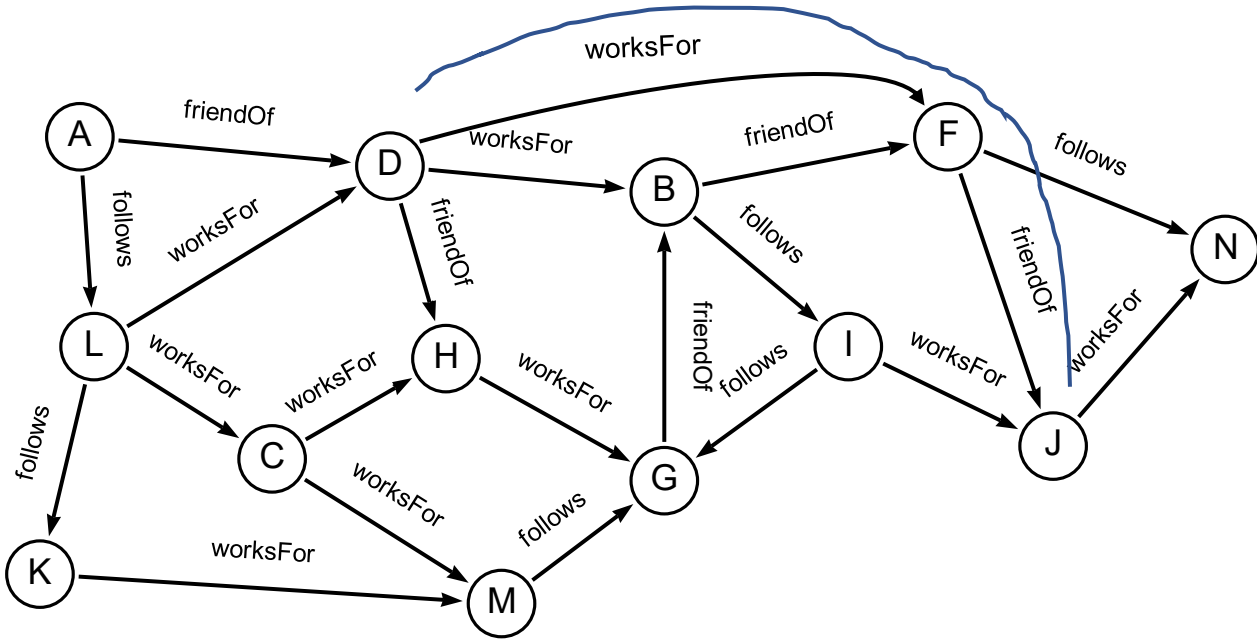


2020

P2H+,
VLDB

Pruning Rules in P2H+

- Pruning rule 2:
  - Skipping index entries if they can be derived from the current index
- Example:
  - Forward BFS from D visits J with path-label set $\{worksFor, friendOf\}$
  - If the following index entries are available in the current snapshot of the index
    - $(B, \{worksFor\}) \in L_{out}(D)$
    - $(B, \{friendOf\}) \in L_{in}(J)$
  - Then, no index entry needs to be recorded, and the path does not need to be expanded

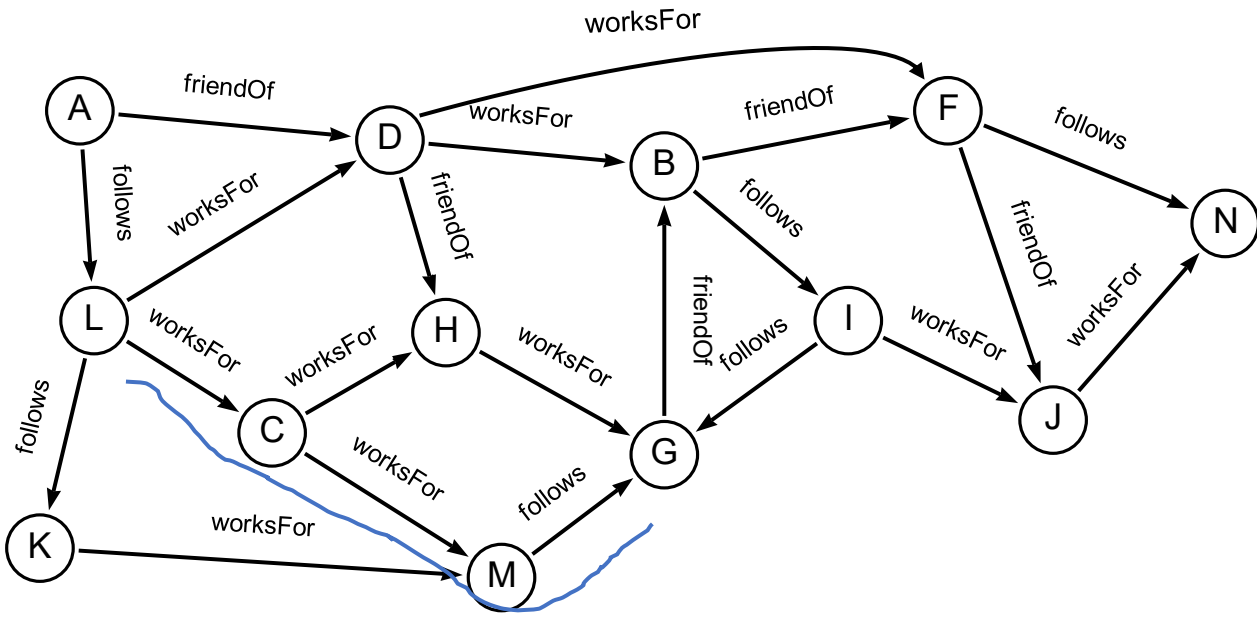


2020

P2H+,
VLDB

Pruning Rules in P2H+

- Pruning rule 3:
  - Removing redundant index entries
- Example:
  - Forward BFS from L visits G via M with $\{worksFor, follows\}$, thus inserting $(L, \{worksFor, follows\})$ into $L_{in}(G)$

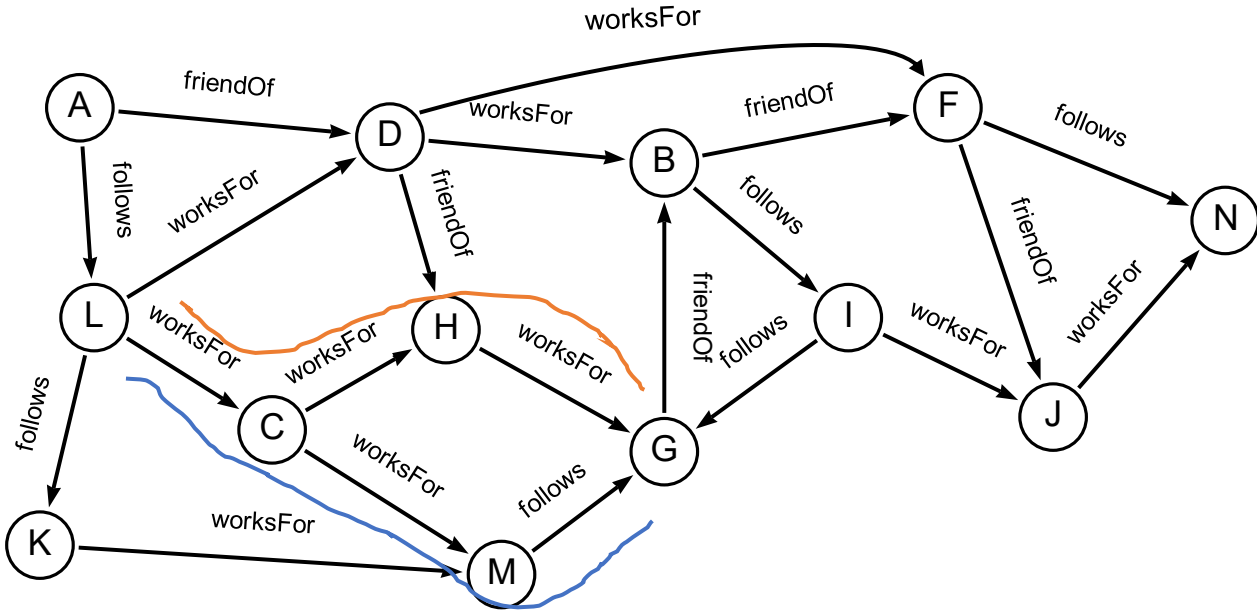


2020

P2H+,
VLDB

Pruning Rules in P2H+

- Pruning rule 3:
  - Removing redundant index entries
- Example:
  - Forward BFS from L visits G via M with $\{worksFor, follows\}$, thus inserting $(L, \{worksFor, follows\})$ into $L_{in}(G)$
  - The BFS also visits G via H with $\{worksFor\}$, thus inserting $(L, \{worksFor\})$ into $L_{in}(G)$
  - As a result, ~~$(L, \{worksFor, follows\})$~~ is redundant, which will be deleted

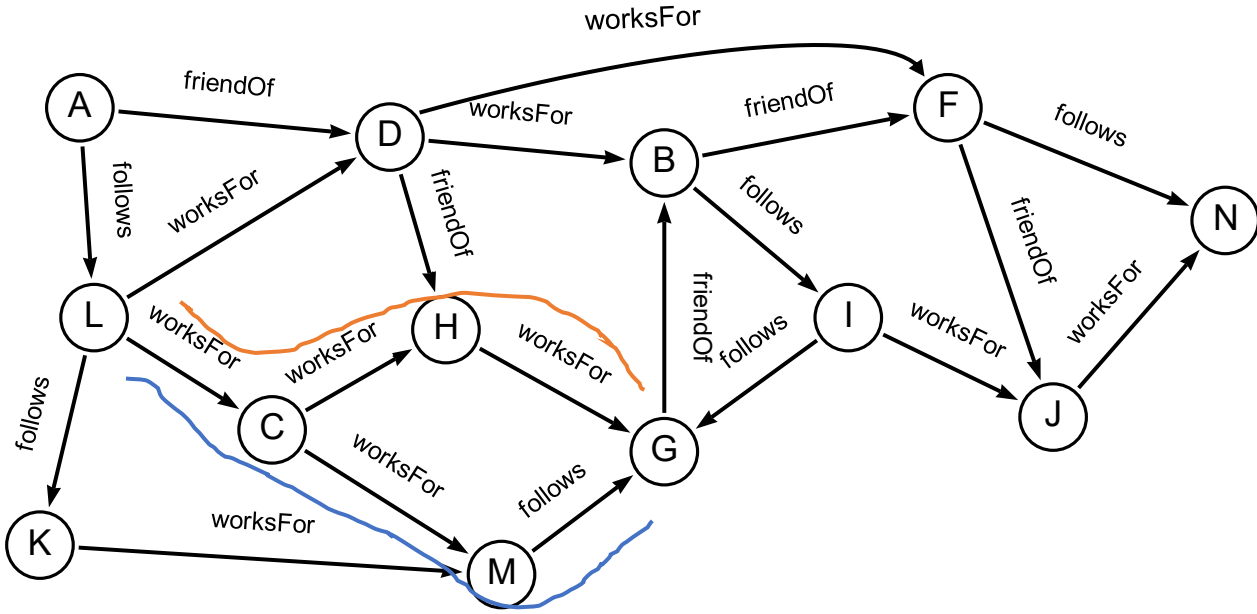


2020

P2H+,
VLDB

Optimization

- Avoiding inserting redundant path-label sets
- Solution:
  - Prioritizing the visiting of the edges with labels that have been included in the BFS
- Example:
  - Previously, the blue path is inserted but deleted later because of the orange path

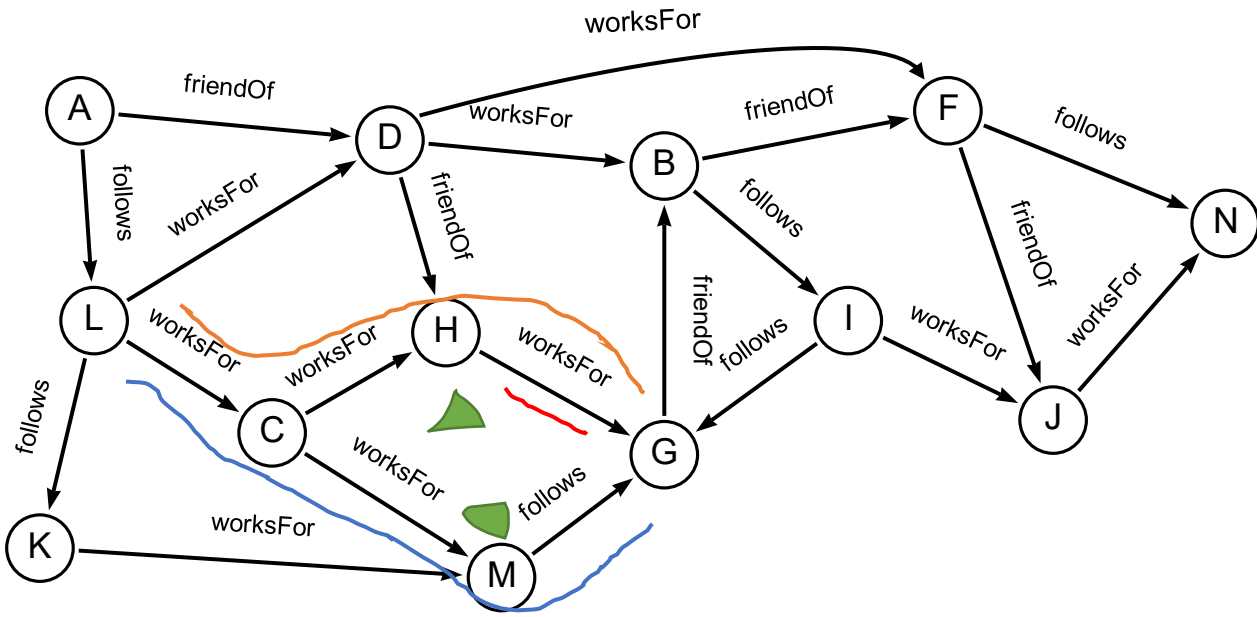


2020

P2H+,
VLDB

Optimization

- Avoiding inserting redundant path-label sets
- Solution:
  - Prioritizing the visiting of the edges with labels that have been included in the BFS
- Example:
  - Previously, the blue path is inserted but deleted later because of the orange path
  - Now, when the BFS from L visits H and M, prioritizing the visiting of $(H, \boldsymbol{worksFor}, G)$

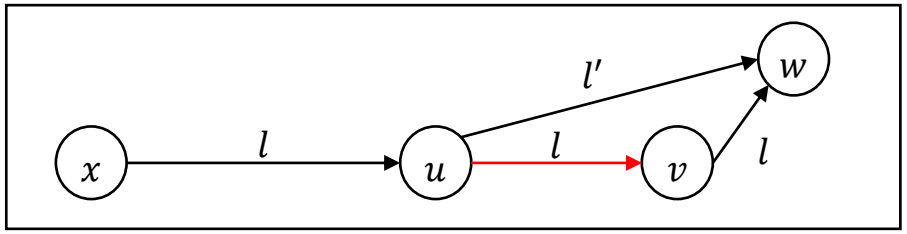


2022

DLCR, *VLDB*

Dynamic Label Constrained Reachability

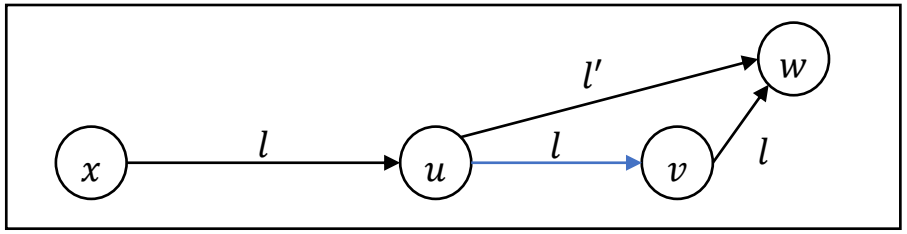
- DLCR: an extension of P2H+ to dynamic graphs
- Inserting (u, v) with label l in DLCR:



Inserting the reachability from u to w with $\{l\}$

Deleting the **redundant** reachability from x to w with $\{l, l'\}$

- Deleting (u, v) with label l in DLCR:



Deleting the reachability from u to w with $\{l\}$

Inserting the **pruned** reachability from x to w with $\{l, l'\}$

X. Chen et al. DLCR: Efficient Indexing for Label-Constrained Reachability Queries on Large Dynamic Graphs. Proc. VLDB Endow. 15(8): 1645-1657 (2022)

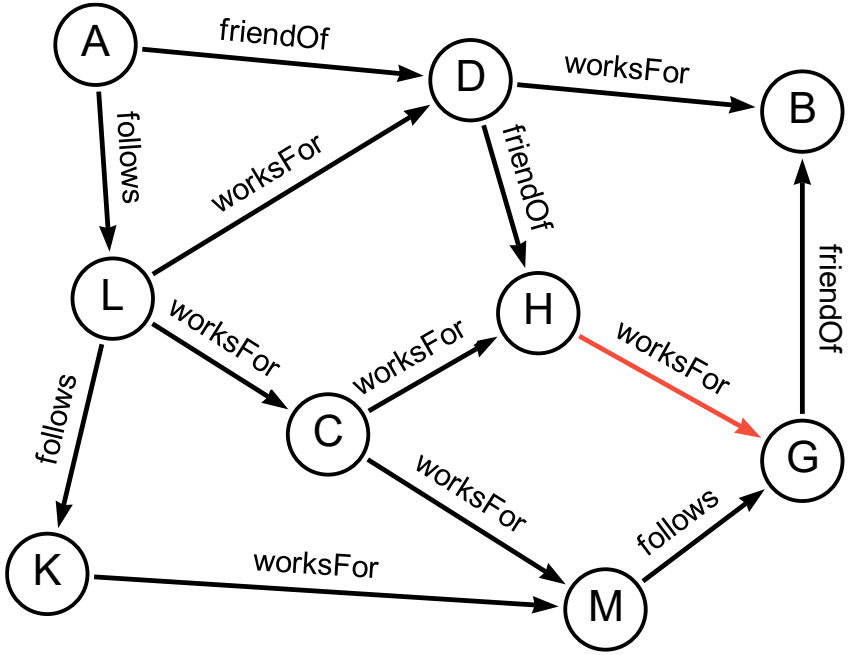
2022

DLCR, *VLDB*

Example: Insertion in DLCR

DLCR
(*incomplete view before the edge insertion*)

v	$L_{in}(v)$	$L_{out}(v)$
A	…	…
B	$(L, \{worksFor\})$, $(C, \{worksFor, follows, friendOf\})$, …	…
C	…	…
D	…	…
G	$(L, \{worksFor, follows\})$, $(C, \{worksFor, follows\})$, …	…
H	$(L, \{worksFor\})$, $(C, \{worksFor\})$, …	…
K	…	…
L	…	…
M	$(L, \{worksFor\})$, $(C, \{worksFor\})$, …	…



Inserting $(H, worksFor, G)$

The forward BFS with source L visited H with $\{worksFor\}$

The BFS launched by the insertion continues from H with $\{worksFor\}$

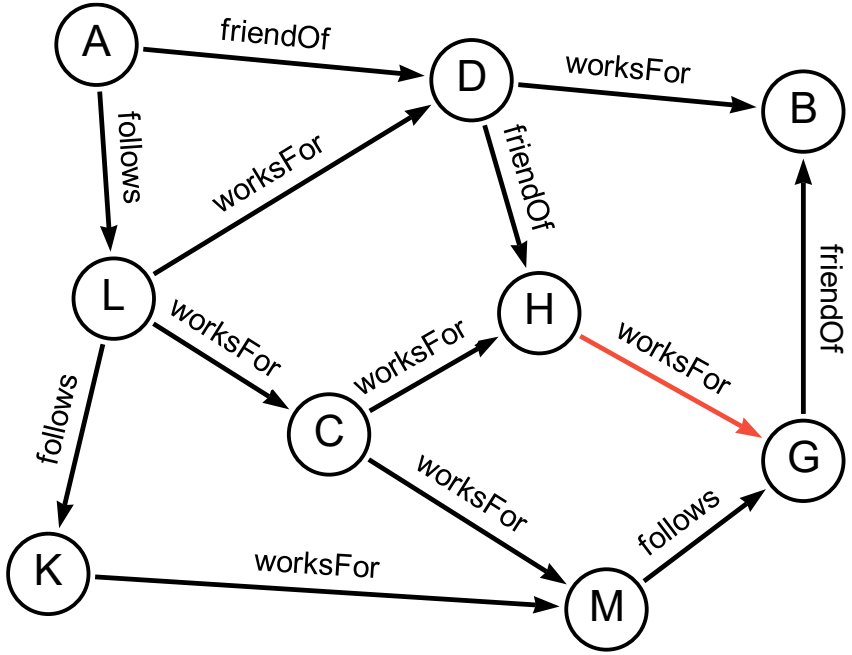
2022

DLCR, *VLDB*

Example: Insertion in DLCR

DLCR
(*incomplete view*)

v	$L_{in}(v)$	$L_{out}(v)$
A	…	…
B	$(L, \{worksFor\})$, $(C, \{worksFor, follows, friendOf\})$, …	…
C	…	…
D	…	…
G	$(L, \{worksFor, follows\})$, $(C, \{worksFor, follows\})$, $(L, \{worksFor\})$, …	…
H	$(L, \{worksFor\})$, $(C, \{worksFor\})$, …	…
K	…	…
L	…	…
M	$(L, \{worksFor\})$, $(C, \{worksFor\})$, …	…



Inserting $(H, worksFor, G)$

Affected vertices: G

The forward BFS with source L visited H with $\{worksFor\}$

The BFS launched by the insertion continues from H with $\{worksFor\}$

The BFS visits G with $\{worksFor\}$ and inserts $(L, \{worksFor\})$ into $L_{in}(G)$

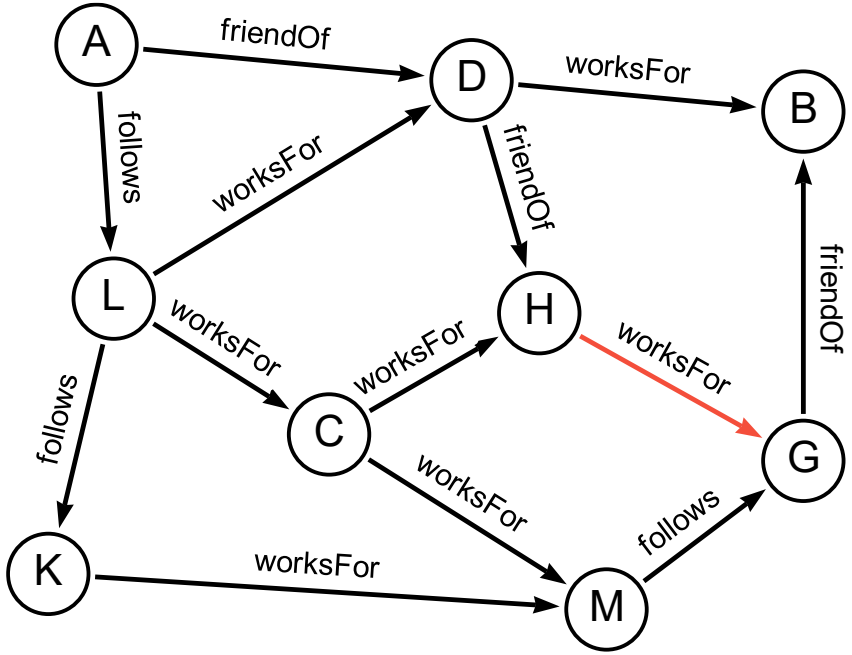
2022

DLCR, *VLDB*

Example: Insertion in DLCR

DLCR
(*incomplete view*)

v	$L_{in}(v)$	$L_{out}(v)$
A	…	…
B	$(L, \{worksFor\})$, $(C, \{worksFor, follows, friendOf\})$, …	…
C	…	…
D	…	…
G	$(L, \{worksFor, follows\})$, $(C, \{worksFor, follows\})$, $(L, \{worksFor\})$, …	…
H	$(L, \{worksFor\})$, $(C, \{worksFor\})$, …	…
K	…	…
L	…	…
M	$(L, \{worksFor\})$, $(C, \{worksFor\})$, …	…



Inserting $(H, worksFor, G)$

Affected vertices: G

The forward BFS with source L visited H with $\{worksFor\}$

The BFS launched by the insertion continues from H with $\{worksFor\}$

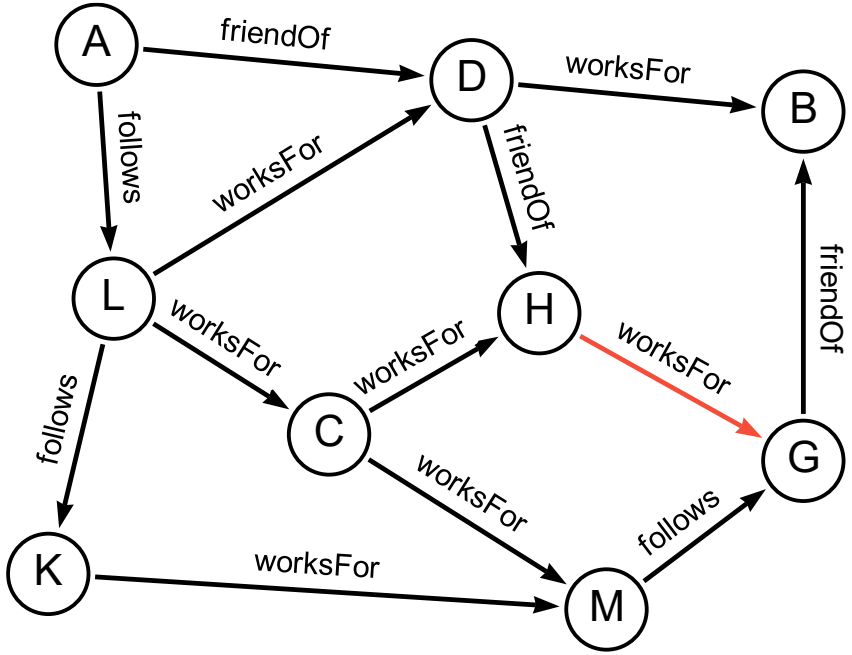
The BFS visits G with $\{worksFor\}$ and inserts $(L, \{worksFor\})$ into $L_{in}(G)$

The BFS can visit B with $\{worksFor, friendOf\}$, but it is pruned by $(L, \{worksFor\})$ in $L_{in}(B)$

Example: Insertion in DLCR

DLCR
(*incomplete view*)

v	$L_{in}(v)$	$L_{out}(v)$
A	…	…
B	$(L,\{worksFor\})$, $(C,\{worksFor, follows, friendOf\})$, $(C,\{worksFor, friendOf\})$, …	…
C	…	…
D	…	…
G	$(L,\{worksFor, follows\})$, $(C,\{worksFor, follows\})$, $(L,\{worksFor\})$, $(C,\{worksFor\})$…	…
H	$(L,\{worksFor\})$, $(C,\{worksFor\})$, …	…
K	…	…
L	…	…
M	$(L,\{worksFor\})$, $(C,\{worksFor\})$, …	…



Inserting $(H, worksFor, G)$

Affected vertices: G, B

The forward BFS with source C visited H with $\{worksFor\}$

The BFS launched by the insertion continues from H with $\{worksFor\}$

The BFS visits G with $\{worksFor\}$ and inserts $(C,\{worksFor\})$ into $L_{in}(G)$

The BFS visits B with $\{worksFor, friendOf\}$ and inserts $(C,\{worksFor, friendOf\})$ in $L_{in}(B)$

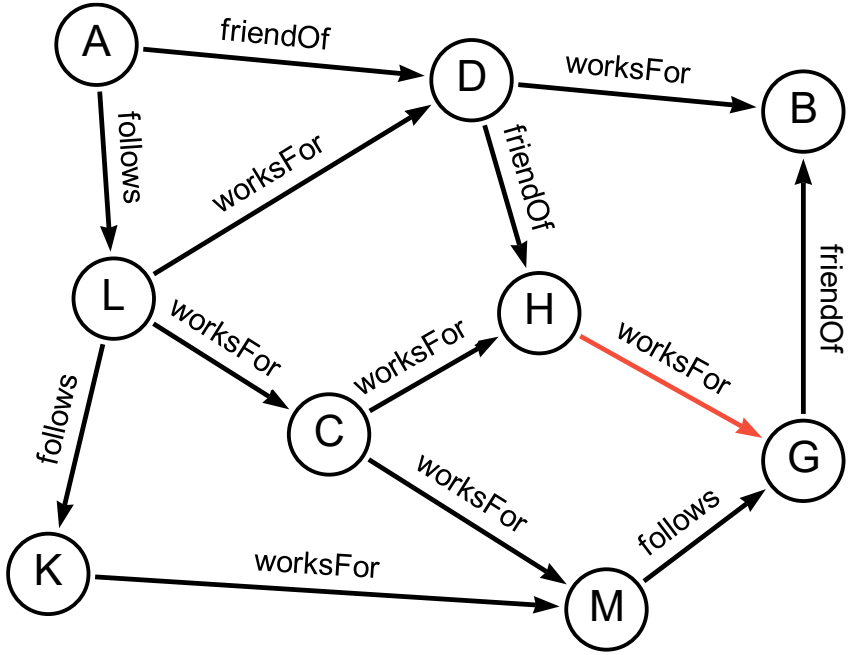
2022

DLCR, *VLDB*

Example: Insertion in DLCR

DLCR
(*incomplete view*)

v	$L_{in}(v)$	$L_{out}(v)$
A	…	…
B	$(L, \{worksFor\})$, ~~$(C, \{worksFor, follows, friendOf\})$~~, $(C, \{worksFor, friendOf\})$, …	…
C	…	…
D	…	…
G	~~$(L, \{worksFor, follows\})$, $(C, \{worksFor, follows\})$,~~ $(L, \{worksFor\})$, $(C, \{worksFor\})$…	…
H	$(L, \{worksFor\})$, $(C, \{worksFor\})$, …	…
K	…	…
L	…	…
M	$(L, \{worksFor\})$, $(C, \{worksFor\})$, …	…



Inserting $(H, worksFor, G)$

Affected vertices: G, B

Redundancy after insertion: paths to the affected vertices

Examining the entries in $L_{in}(G)$ and $L_{in}(B)$ and removing redundant ones

2022

DLCR, *VLDB*

Summary of Edge Insertion in DLCR

- Inserting (u, v) with label l in DLCR
- Intuition:
  - Continue the BFSs as if (u, v) was there in the first place
- BFSs are performed with frontiers from index entries of $L_{in}(u)$ and $L_{out}(v)$
  - For each $(x, SPLS) \in L_{in}(u)$:
    - Computing the **forward** BFS with source x from u to insert new index entries (**not from scratch**)
  - Similar computations for each $(x, SPLS) \in L_{out}(v)$
- Removing redundancy after inserting index entries
  - Require maintaining inverted index entries (*not shown in the running example*)

X. Chen et al. DLCR: Efficient Indexing for Label-Constrained Reachability Queries on Large Dynamic Graphs. Proc. VLDB Endow. 15(8): 1645-1657 (2022)

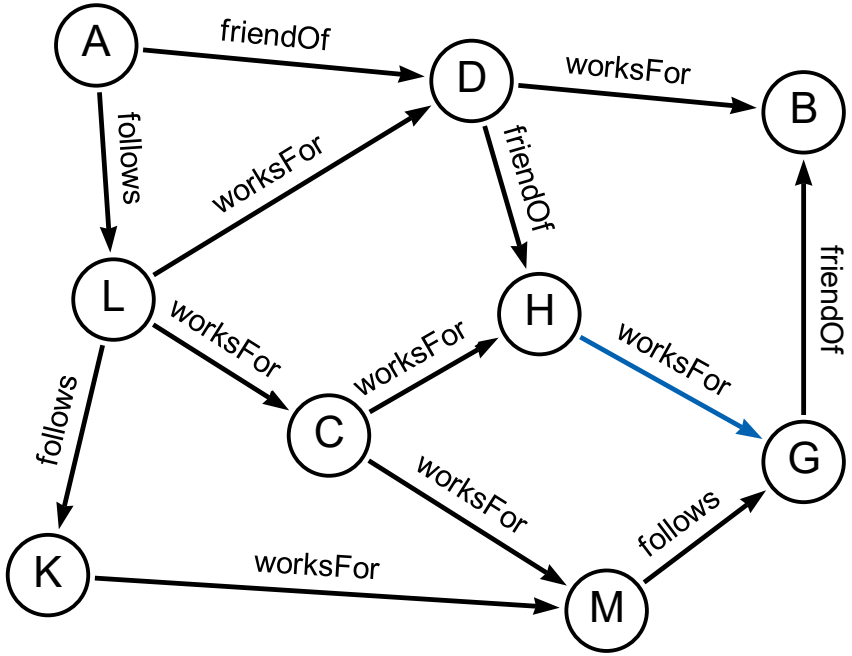
2022

DLCR, *VLDB*

Example: Deletion in DLCR

DLCR
(*incomplete view before the edge deletion*)

v	$L_{in}(v)$	$L_{out}(v)$
A	…	…
B	$(L, \{worksFor\}), (C, \{worksFor, friendOf\}), \ldots$	…
C	…	…
D	…	…
G	~~$(L, \{worksFor\}), (C, \{worksFor\})$~~, …	…
H	$(L, \{worksFor\}), (C, \{worksFor\}), \ldots$	…
K	…	…
L	…	…
M	$(L, \{worksFor\}), (C, \{worksFor\}), \ldots$	…



Deleting $(H, worksFor, G)$

Affected vertices: G

Examining index entries in $L_{in}(G)$: only deleting the index entries representing paths with the deleted edge, which is achieved by maintaining **last edges** in index entries
E.g., $(L, \{worksFor\}, (H, worksFor, G))$, $(C, \{worksFor\}, (H, worksFor, G))$

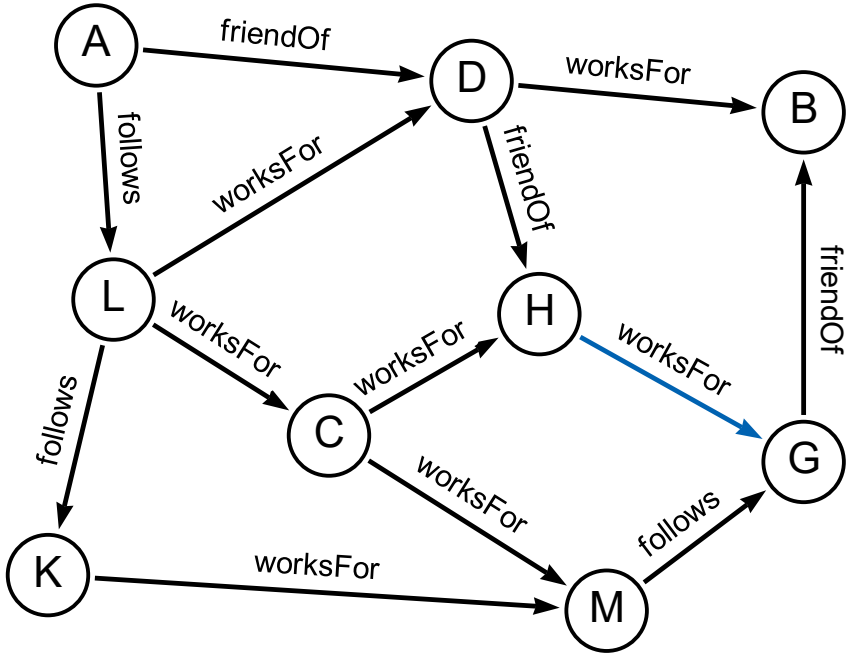
2022

DLCR, *VLDB*

Example: Deletion in DLCR

DLCR
(*incomplete view*)

v	$L_{in}(v)$	$L_{out}(v)$
A	…	…
B	$(L, \{worksFor\})$, ~~$(C, \{worksFor, friendOf\})$~~, …	…
C	…	…
D	…	…
G	~~$(L, \{worksFor\})$, $(C, \{worksFor\})$~~, …	…
H	$(L, \{worksFor\})$, $(C, \{worksFor\})$, …	…
K	…	…
L	…	…
M	$(L, \{worksFor\})$, $(C, \{worksFor\})$, …	…



Deleting $(H, worksFor, G)$

Affected vertices: G, B

Examining index entries in $L_{in}(G)$: only deleting the index entries representing paths with the deleted edge, which is achieved by maintaining **last edges** in index entries

Launch the forward BFS with source L from G, and the forward BFS with source C from G to delete other index entries

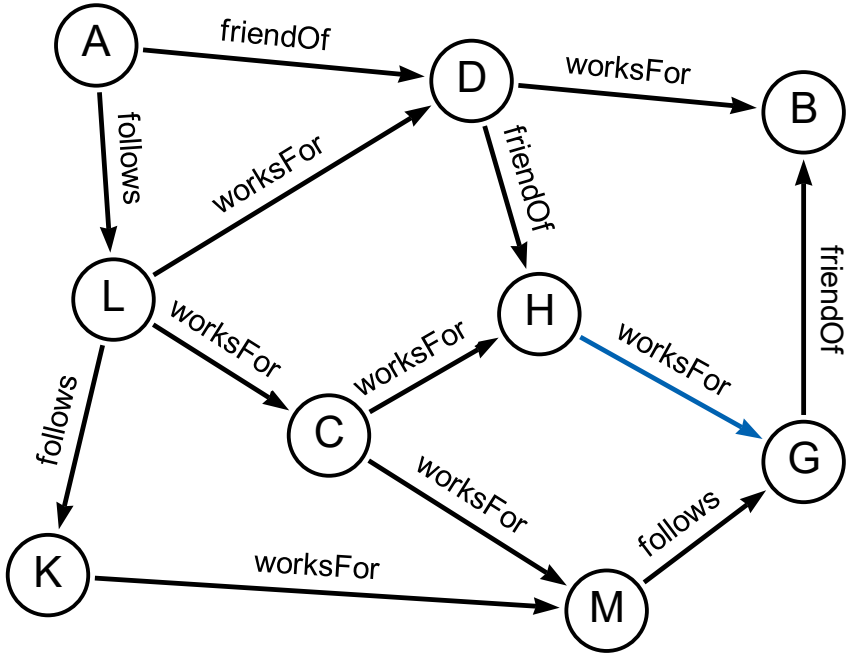
2022

DLCR, *VLDB*

Example: Deletion in DLCR

DLCR
(*incomplete view*)

v	$L_{in}(v)$	$L_{out}(v)$
A	…	…
B	$(L,\{worksFor\})$, ~~$(C,\{worksFor, friendOf\})$~~, …	…
C	…	…
D	…	…
G	~~$(L,\{worksFor\})$, $(C,\{worksFor\})$~~, $(L,\{worksFor, follows\})$, $(C,\{worksFor, follows\})$, …	…
H	$(L,\{worksFor\})$, $(C,\{worksFor\})$, …	…
K	…	…
L	…	…
M	$(L,\{worksFor\})$, $(C,\{worksFor\})$, …	…



Deleting $(H, worksFor, G)$

Affected vertices: G, B

Adding index entries that were pruned because of the deleted edge by using affected vertices

G has an incoming edge from M that has $(L,\{worksFor\})$ and $(C,\{worksFor\})$ in $L_{in}(M)$, thus inserting $(L,\{worksFor, follows\})$ and $(C,\{worksFor, follows\})$ into $L_{in}(G)$

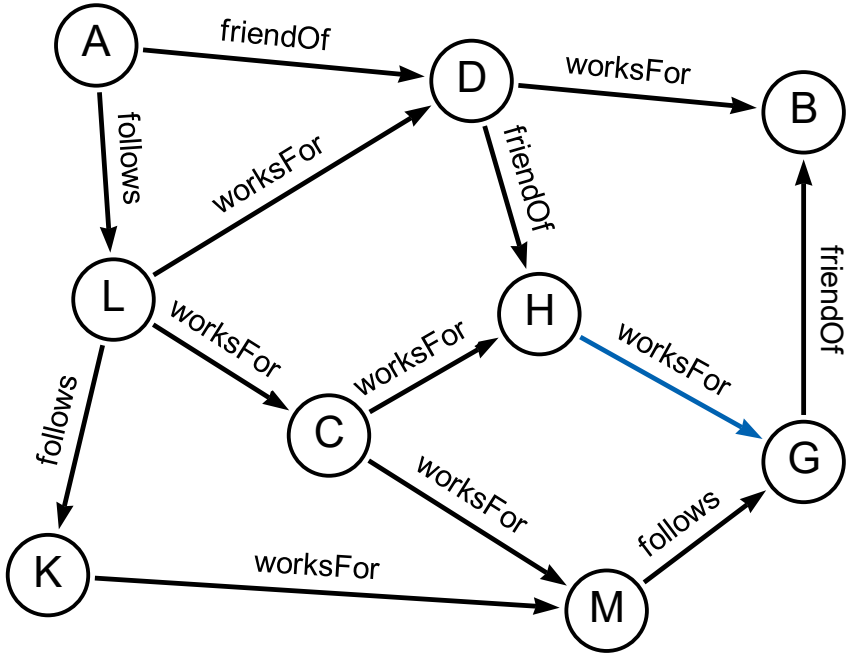
2022

DLCR, *VLDB*

Example: Deletion in DLCR

DLCR
(*incomplete view*)

v	$L_{in}(v)$	$L_{out}(v)$
A	…	…
B	$(L,\{worksFor\})$, ~~$(C,\{worksFor, friendOf\})$~~, $(C,\{worksFor, follows, friendOf\})$, …	…
C	…	…
D	…	…
G	~~$(L,\{worksFor\})$, $(C,\{worksFor\})$~~, $(L,\{worksFor, follows\})$, $(C,\{worksFor, follows\})$, …	…
H	$(L,\{worksFor\})$, $(C,\{worksFor\})$, …	…
K	…	…
L	…	…
M	$(L,\{worksFor\})$, $(C,\{worksFor\})$, …	…



Deleting $(H, worksFor, G)$

Affected vertices: G, B

Adding index entries that were pruned because of the deleted edge by using affected vertices

B has an incoming edge from G that has $(L,\{worksFor, follows\})$ and $(C,\{worksFor, follows\})$ in $L_{in}(G)$, which leads to inserting $(C,\{worksFor, follows, friendOf\})$ into $L_{in}(B)$

2022

DLCR, *VLDB*

Summary of Edge Deletion in DLCR

- Deleting (u, v) with label l in DLCR
- Intuition:
  - Continue the BFSs as if (u, v) was **not** there in the first place
- BFSs are performed with frontiers from index entries of $L_{in}(v)$ and $L_{out}(u)$
  - For each $(x, SPLS) \in L_{in}(v)$:
    - If its last edge is (u, v), delete the index entry and compute the **forward** BFS with source x from v to delete outdated index entries (**not from scratch**)
  - Similar computations for each $(x, SPLS) \in L_{out}(u)$
- Adding index entries that were pruned due to the deleted edge (u, v)
  - Require maintaining inverted index entries (*not shown in the running example*)

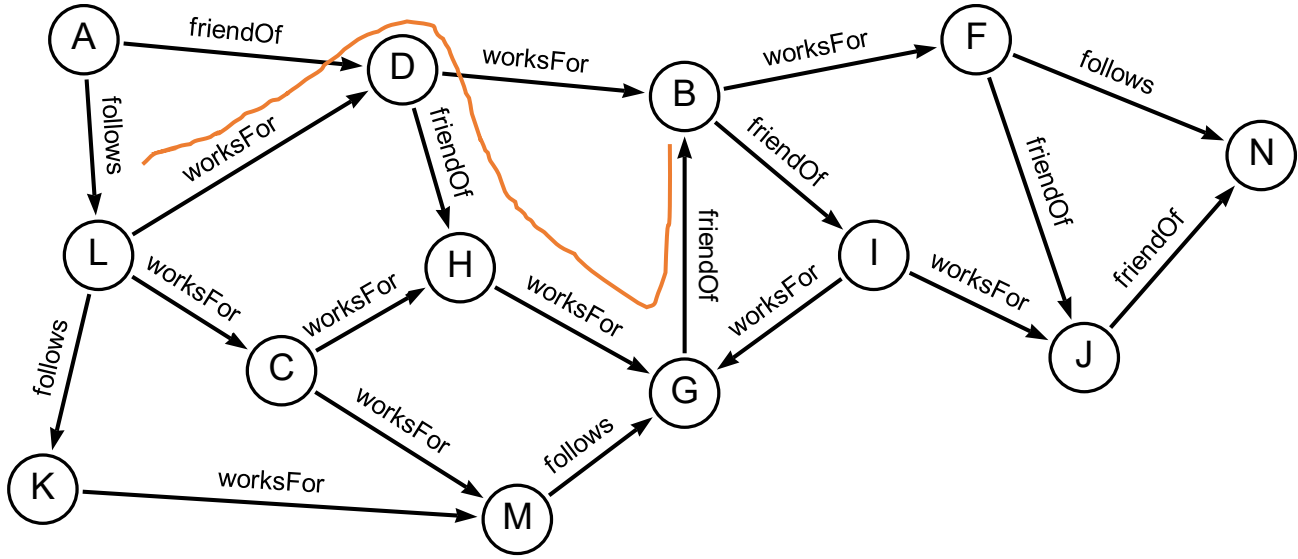
X. Chen et al. DLCR: Efficient Indexing for Label-Constrained Reachability Queries on Large Dynamic Graphs. Proc. VLDB Endow. 15(8): 1645-1657 (2022)

Outline

1. Plain Reachability Indexes
   a) Tree-Cover Indexes
   b) 2-Hop Indexes
   c) Approximate Transitive Closures
   d) Other Techniques
2. Path-Constrained Reachability Indexes
   a) Indexes for Alternation-Based Queries
   b) Indexes for Concatenation-Based Queries
3. Open Challenges

Concatenation-Based Reachability

- $G = (V, E, \mathcal{L})$
- $Q_r(s, t, \alpha)$, $\alpha = (l_1 \cdot \ldots \cdot l_k)^*$
  - α defines a **label sequence LS**, and the edge labels along the path are repeats of **LS**
- Supported languages:
  - SPARQL
  - SQL/PGQ
  - GQL



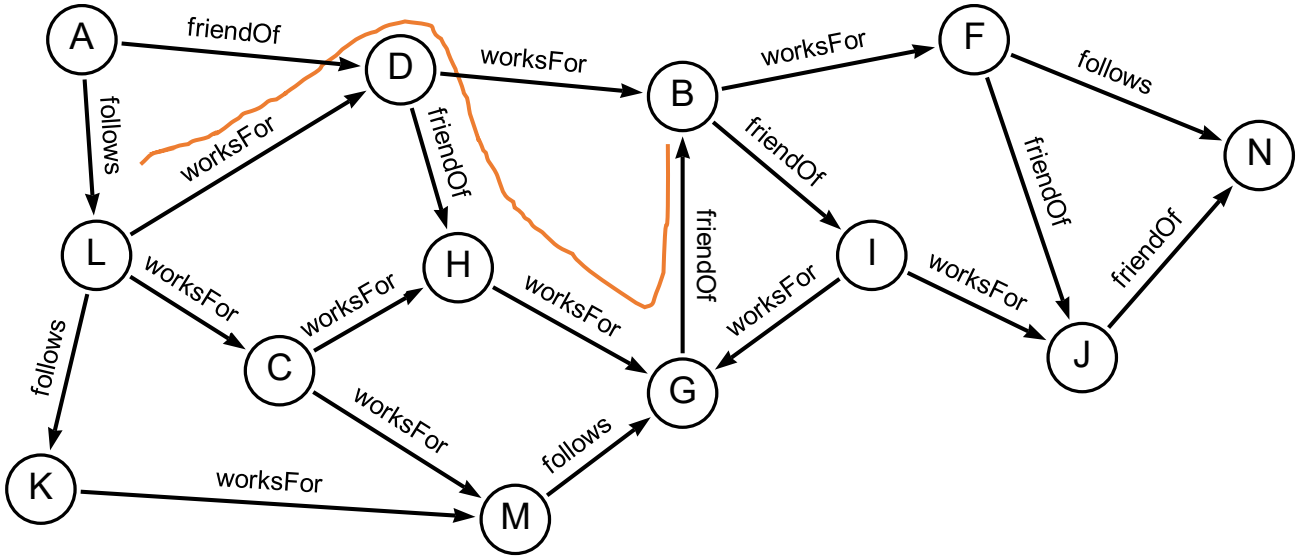
Example:

- $\alpha = (worsFor \cdot friendOf)^*$, $Q_r(L, B, \alpha)$ = True
- SPARQL query:
  ASK WHERE {:L (:friendOf / :follows)* :B}

Indexing for Concatenation-Based Reachability

- Index-based evaluation for $Q_r(s, t, \alpha)$, $\alpha = (l_1 \cdot \ldots \cdot l_k)^*$
  - Path-label sequences need to be recorded
- Efficiently storing path-label sequences
  - Minimum repeats (MR) [Zha23]
- Naïve index: generalized transitive closure

Source	Target	MRs
L	B	$(worksFor, friendOf), \ldots$
…	…	…



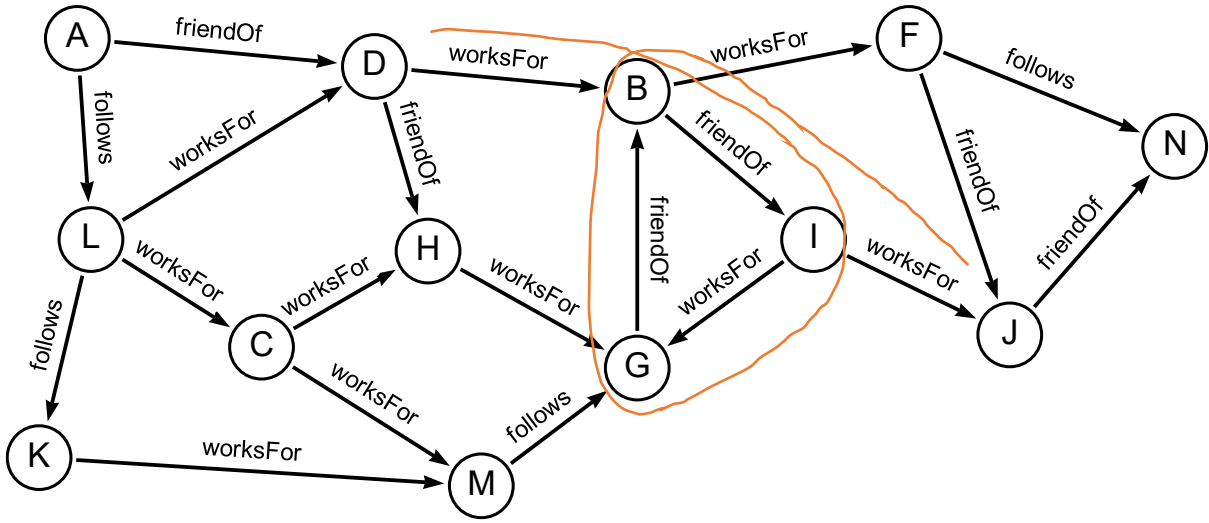
Path-label sequence:
$(worksFor, friendOf, worksFor, friendOf)$

Minimum repeat:
$(worksFor, friendOf)$

[Zha23] C. Zhang et al. A Reachability Index for Recursive Label-Concatenated Graph Queries. ICDE 2023: 66-80

Computing Minimum Repeats

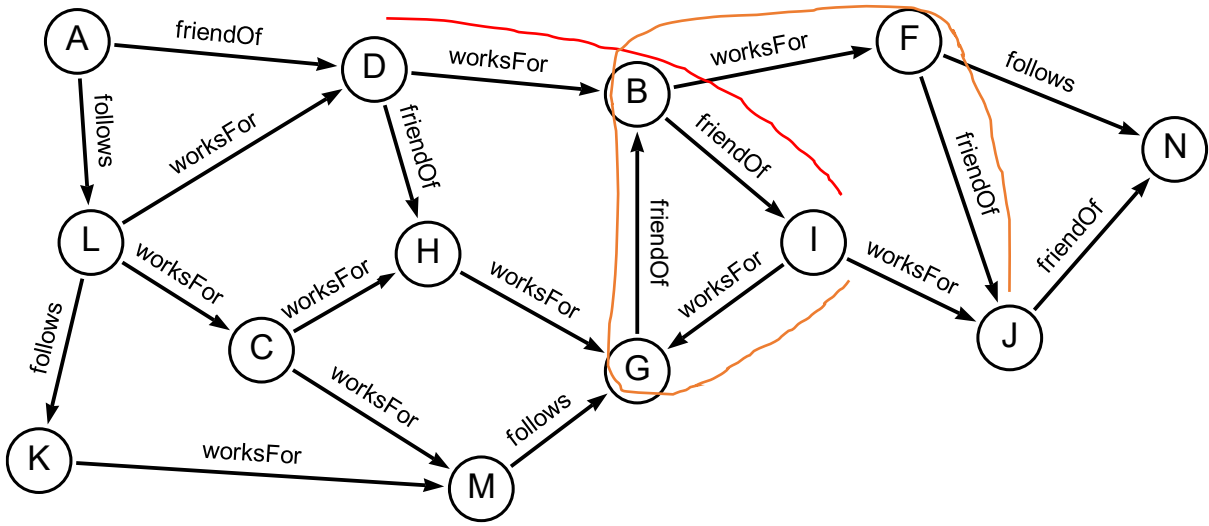
- Potentially an infinite number of minimum repeats
- Example:
  - A cycle exists in the path from D to J
  - The cycle can be traversed any number of times, resulting in an infinite number of minimal repeats
- Problem:
  - *How to compute the minimum repeats from a source vertex*



C. Zhang et al. A Reachability Index for Recursive Label-Concatenated Graph Queries. ICDE 2023: 66-80

Kernel-Based Search

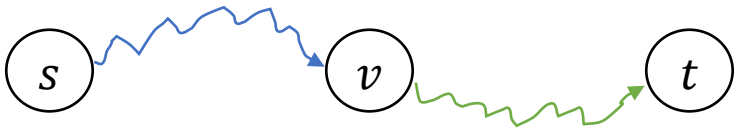
- Practical path constraint: $\alpha = (l_1 \cdot \ldots \cdot l_k)^*, k$ is bounded
- Given a parameter k, computing all the minimum repeats MR, such that $|MR| \leq k$
  1. Computing **kernels** within k hops
  2. Guiding the subsequent search using each kernel
- Example: $k = 2$, computing the minimum repeats from D
  - Traversing the graph from D
  - Label sequence to I: $(worksFor, friendOf)$
  - The subsequent search from I is guided by $(worksFor \cdot friendOf)^*$, which visits J
  - Adding $(worksFor, friendOf)$ as a minimum repeat from D to J



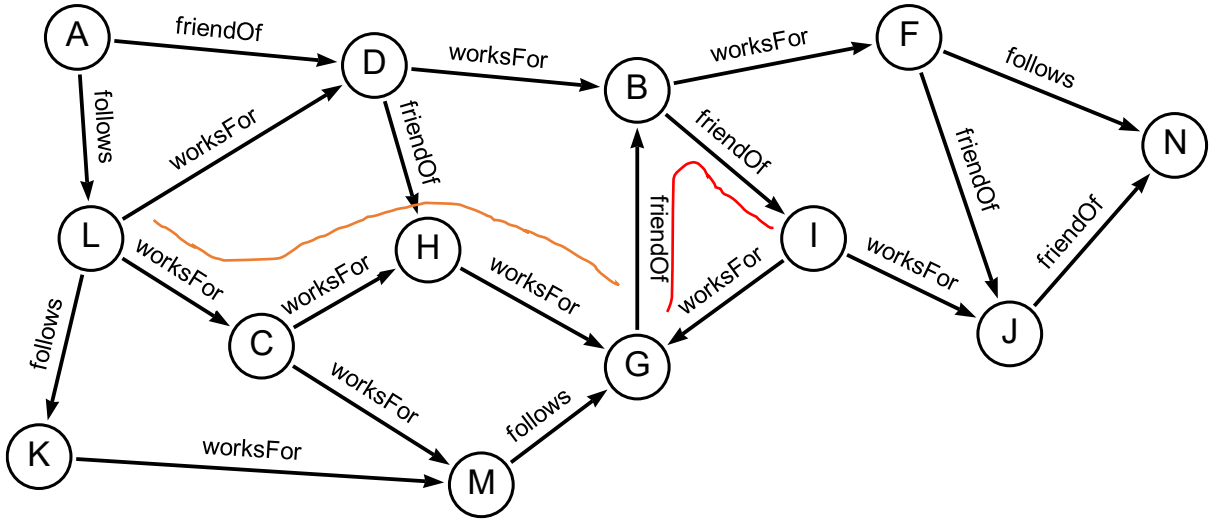
C. Zhang et al. A Reachability Index for Recursive Label-Concatenated Graph Queries. ICDE 2023: 66-80

Transitive Minimum Repeats

- Minimum repeats are not necessarily transitive
- Example:
  - $(worksFor)$ is a minimum repeat from L to G
  - $(friendOf)$ is a minimum repeat from G to I
  - $(worksFor, friendOf)$ is **not** a minimum repeat from L to I
- Transitive minimum repeats



If $MR_1 = MR_2$, then we can derive a minimum repeat from s to t

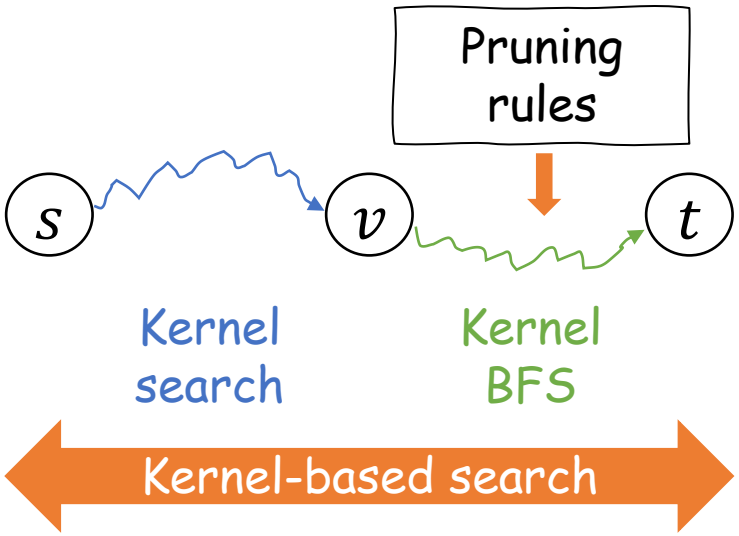


2023

RLC Index, *ICDE*

RLC Index

- A 2-hop index for concatenation-based reachability [Zha23]
  - Computing $L_{in}(v)$ and $L_{out}(v)$ for each vertex
  - Adding minimum repeats into $L_{in}(v)$ and $L_{out}(v)$
- Indexing algorithm: extending PLL [Aki13, Yan13] to compute minimum repeats
  - Performing the backward and forward **kernel-based search** from each vertex
  - Applying pruning rules to prune search space and avoid redundancy



[Zha23] C. Zhang et al. A Reachability Index for Recursive Label-Concatenated Graph Queries. ICDE 2023: 66-80
[Aki13] E. Akiba et al. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. SIGMOD Conference 2013: 349-360
[Yan13] Y. Yano, et al. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In CIKM'13.

2023

RLC Index, *ICDE*

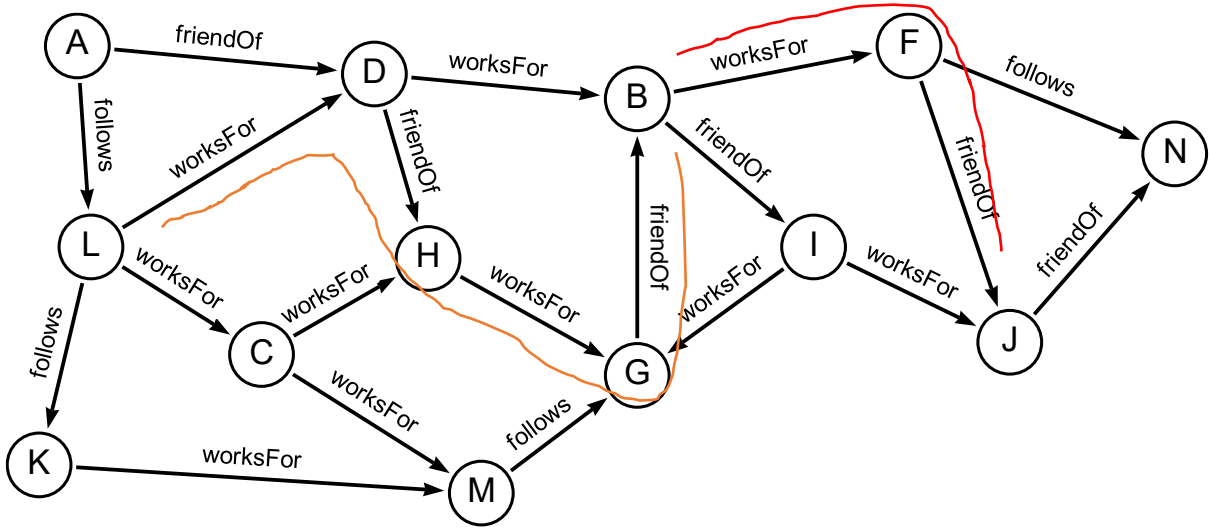
Query Processing using RLC Index

RLC Index ($k = 2$)
(*incomplete view*)

v	$L_{in}(v)$	$L_{out}(v)$
A	…	$(B, (friendOf, worksFor))$, …
B	…	…
C	…	$(G, (worksFor))$, …
D	…	$(B, (worksFor))$, …
F	…	
G	$(B, (friendOf, worksFor))$, …	$(B, (friendOf))$, …
H	$(L, (worksFor))$, …	$(B, (worksFor, friendOf))$, …
I	…	$(B, (worksFor, friendOf))$, …
J	$(B, (worksFor, friendOf))$, $(B, (friendOf, worksFor))$, …	…
K	…	…
L	…	$(B, (worksFor, friendOf))$, $(B, (worksFor))$, …
M	$(L, (follows, worksFor))$, …	$(B, (follows, friendOf))$, …
N	$(B, (worksFor, follows))$, …	…

Example:

- $Q_r(L, J, \alpha)$, $\alpha = (worksFor \cdot friendOf)^*$
  - $(B, (worksFor, friendOf)) \in L_{out}(L)$
  - $(B, (worksFor, friendOf)) \in L_{in}(J)$
  - Return True



Outline

1. Plain Reachability Indexes
    a) Tree-Cover Indexes
    b) 2-Hop Indexes
    c) Approximate Transitive Closures
    d) Other Techniques
2. Path-Constrained Reachability Indexes
    a) Indexes for Alternation-Based Queries
    b) Indexes for Concatenation-Based Queries
3. Open Challenges

Main Problems

- Real-world graphs are
  - large, and
  - fully dynamic
- Desirable features of indexes:
  - Scalability
  - Efficient index updates

Plain Reachability Indexes on Large Graphs

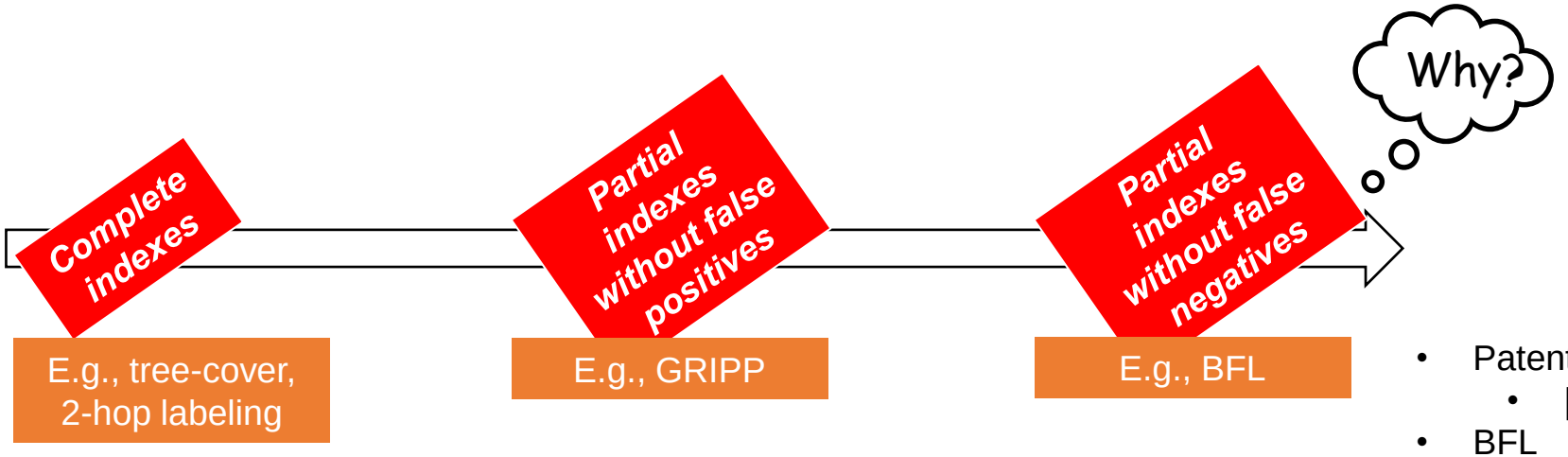
Plain reachability indexes can scale to large graphs

- Patent graph
  - |V|= 3M, |E|=16M
- BFL [Su17]
  - Indexing time: ~ 1.3 s
  - Index size: ~ 132 MB
  - Query time: ~ 0.26 us

[Su17] J. Su et al. Reachability Querying: Can It Be Even Faster? IEEE Trans. Know. Data Eng. 29(3): 683-697 (2017)

Plain Reachability Indexes on Large Graphs

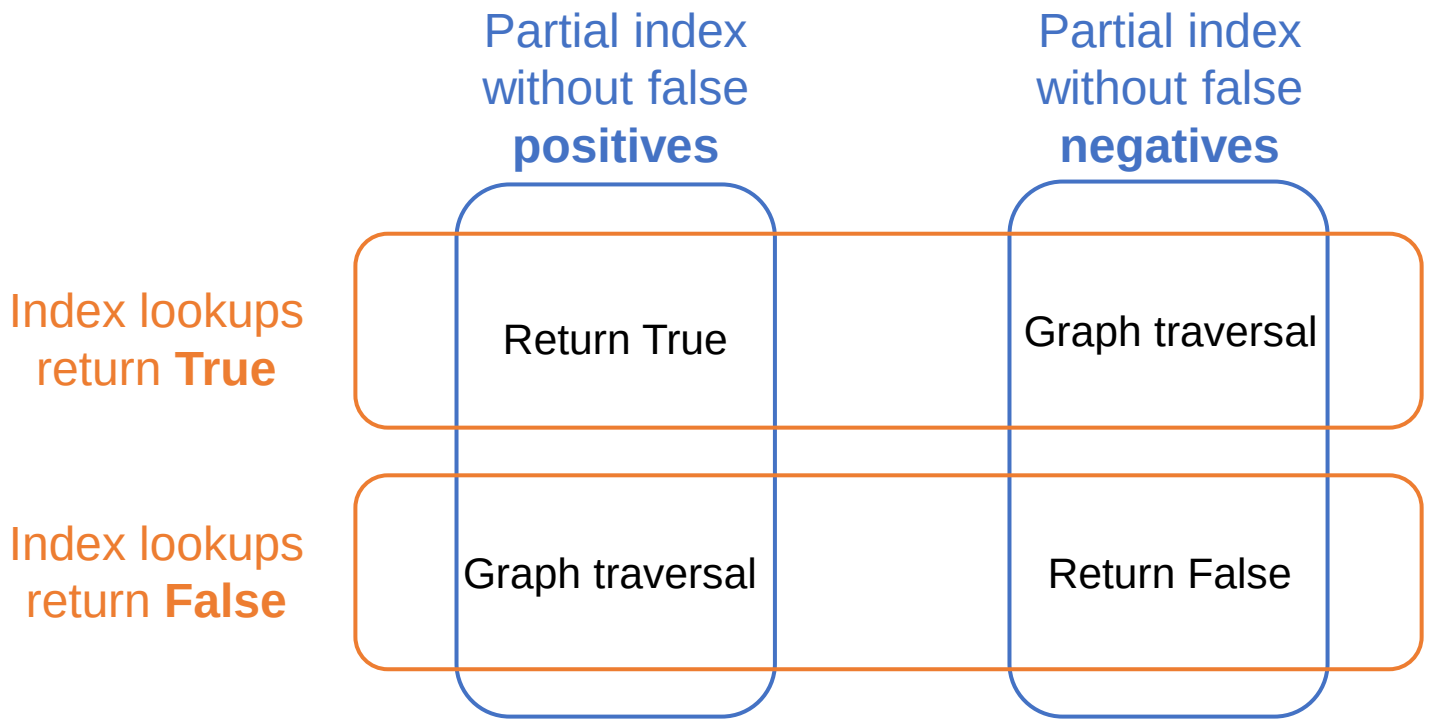
Index type	Indexing each (s,t)	Indexing cost	Query processing
Complete	Yes	High	Index lookups
Partial	No	Low	Index lookups + traversal



The evolution of designing plain reachability indexes

- Patent graph
  - |V|= 3M, |E|=16M
- BFL
  - Indexing time: ~ 1.3 s
  - Index size: ~ 132 MB
  - Query time: ~ 0.26 us

Query Processing Using Partial Indexes



Reachability ratio: $\frac{|reachable\ pairs|}{|V|^2}$

Reachability ratio in real-world graphs is small [Su17]

Partial indexes without false negatives are preferable

[Su17] J. Su et al. Reachability Querying: Can It Be Even Faster? IEEE Trans. Know. Data Eng. 29(3): 683-697 (2017)

Challenge: Plain Reachability Indexes

- State-of-the-art partial indexes without false negatives
  - E.g., BFL is designed for static graphs
- DBL can only support insertion-only graphs
- Research perspective:
  - Partial index without false negatives that can support fully dynamic graphs, including edge insertion and deletion
  - Efficiently indexing for streaming graphs

Challenge: Path-Constrained Reachability Indexes

- For $Q_r(s, t, \alpha)$, the reachability ratio can be even lower
  - A further selection of paths according to α
- The only partial index in literature is Landmark Index [Val17]
  - Only static graphs
  - Without false positives
- Although DLCR [Che22] can support dynamic graphs, it is a complete index
- Research perspective:
  - Designing a **partial** index without false negatives for path-constrained reachability queries supporting **dynamic** graphs
  - Efficiently indexing for streaming graphs

[Val17] L. Valstar et al. Landmark Indexing for Evaluation of Label-Constrained Reachability Queries. SIGMOD Conference 2017: 345-358
[Che22] X. Chen et al. DLCR: Efficient Indexing for Label-Constrained Reachability Queries on Large Dynamic Graphs. Proc. VLDB Endow. 15(8): 1645-1657 (2022)

Challenge: Indexing General Path Constraints

- The path constraints can be general regular expressions, like regular path queries [Ang17, Bon18]
  - $Q_r(s, t, \alpha)$, $\alpha ::= l | \alpha \cdot \alpha | \alpha \cup \alpha | \alpha^+ | \alpha^*$
- Current techniques only focus on the case that
  - either $\alpha = (l_1 \cdot \ldots \cdot l_k)^*$ or
  - $\alpha = (l_1 \cup \cdots \cup l_k)^*$
- Research perspective:
  - Indexing techniques for **general** path constraints

[Ang17] R. Angles et al. Foundations of Modern Query Languages for Graph Databases. ACM Comput. Surv. 50(5): 68:1-68:40 (2017)
[Bon18] A. Bonifati et al. Querying Graphs. Synthesis Lectures on Data Management, Morgan & Claypool Publishers 2018

Challenge: Parallel Indexing

- Existing indexing algorithms are single-threaded
- Research perspective:
  - Parallel computation of reachability indexes
- Recent advance:
  - Parallel computation of PLL for building plain 2-hop indexes [Li19, Jin20]:
    - Intuition: Vertex-centric parallel computation of $L_{out}(v)$ and $L_{in}(v)$
  - The parallel computation can be generalized to path-constrained 2-hop indexes
    - E.g., parallel indexing of P2H+, DLCR, and the RLC index

[Li19] W. Li et al. Scaling Distance Labeling on Small-World Networks. SIGMOD Conference 2019: 1060-1077
[Jin20] R. Jin et al. Parallelizing Pruned Landmark Labeling: Dealing with Dependencies in Graph Algorithms. ICS 2020: 11:1-11:13

Challenge: Indexes for Different Path Semantics

- Path semantics of standard graph query languages (GQL and SQL/PGQ):
  - Restrictors: SIMPLE, ACYCLIC, TRAIL
  - Selectors: ALL SHORTEST, ANY SHORTEST, ANY, ANY K...
- Research perspective:
  - Design indexes for labeled-constrained queries under the above restrictors and selectors
  - All selectors but ANY SHORTEST are non-deterministic
  - If restrictors are present, such as simple (a vertex cannot be traversed twice except first and last node), rethink the indexes with alternation of labels or concatenation of labels
  - Indexes for general path constraints can be combined with the above for full coverage

[De22] A. Deutsch et al. Graph Pattern Matching in GQL and SQL/PGQ. SIGMOD Conference 2022: 2246-2258

Summary of Main Challenges

- Plain reachability indexes
  - Efficient index maintenance for fully dynamic graphs or streaming graphs
- Path-constrained reachability indexes
  - Achieving both scalability and index maintenance for fully dynamic graphs and streaming graphs
  - General path constraints
  - Parallel indexing
  - Different path semantics for standard graph queries

References: 1972 - 2012

[Tar72] R. Tarjan. Depth-First Search and Linear Graph Algorithms. SIAM J. Comput. 1(2): 146-160 (1972)

[Sch83] L. Schubert et al. Determining Type, Part, Color and Time Relationships. IEEE Computer 16(10): 53-60 (1983)

[Oke84] R. O'Keefe. A New Data-Structure for Type Trees. ECAI 1984: 393-402

[Agr89] R. Agrawal et al. Efficient Management of Transitive Relationships in Large Data and Knowledge Bases. SIGMOD Conference 1989: 253-262

[Jag90] H. V. Jagadish. A Compression Technique to Materialize Transitive Closure. ACM Trans. Database Syst. 15(4): 558-598 (1990)

[Coh02] E. Cohen et al. Reachability and distance queries via 2-hop labels. SODA 2002: 937-946

[Che05] L. Chen et al. Stack-based Algorithms for Pattern Matching on DAGs. VLDB 2005: 493-504

[Sch05] R. Schenkel et al. Efficient creation and incremental maintenance of the HOPI index for complex XML document collections. ICDE 2005: 360-371

[Wan06] H. Wang et al. Dual Labeling: Answering Graph Reachability Queries in Constant Time. ICDE 2006: 75

[Tri07] S. Tril et al. Fast and practical indexing and querying of very large graphs. SIGMOD Conference 2007: 845-856

[Che08] Y. Chen et al. An Efficient Algorithm for Answering Graph Reachability Queries. ICDE 2008: 893-902

[Jin08] R. Jin et al. Efficiently answering reachability queries on very large directed graphs. SIGMOD Conference 2008: 595-608

[Jin09] R. Jin et al. 3-HOP: a high-compression indexing scheme for reachability query. SIGMOD Conference 2009: 813-826

[Bra10] R. Bramandia et al. Incremental Maintenance of 2-Hop Labeling of Large Graphs. IEEE Trans. Knowl. Data Eng. 22(5): 682-698 (2010)

[Cai10] J. Cai et al. Path-hop: efficiently indexing large graphs for reachability queries. CIKM 2010: 119-128

[Jin10] R. Jin et al. Computing label-constraint reachability in graph databases. SIGMOD Conference 2010: 123-134

[Yil10] H. Yildirim et al. GRAIL: Scalable Reachability Index for Large Graphs. Proc. VLDB Endow. 3(1): 276-284 (2010)

[Jin11] R. Jin et al. Path-tree: An efficient reachability indexing scheme for large directed graphs. ACM Trans. Database Syst. 36(1): 7:1-7:44 (2011)

[Xu11] K. Xu et al. Answering label-constraint reachability in large graphs. CIKM 2011: 1595-1600

[Bar12] P. Barceló et al. Expressive Languages for Path Queries over Graph-Structured Data. ACM Trans. Database Syst. 37(4): 31:1-31:46 (2012)

References: 2013 - 2023

[Yil13] H. Yildirim et al. DAGGER: A Scalable Index for Reachability Queries in Large Dynamic Graphs. CoRR abs/1301.0977 (2013)

[Che13] J. Cheng et al. TF-Label: a topological-folding labeling scheme for reachability querying in a large graph. SIGMOD Conference 2013: 193-204

[Jin13] R. Jin et al. Simple, Fast, and Scalable Reachability Oracle. Proc. VLDB Endow. 6(14): 1978-1989 (2013)

[Seu13] S. Seufert et al. FERRARI: Flexible and efficient reachability range assignment for graph indexing. ICDE 2013: 1009-1020

[Yan13] Y. Yano et al. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. CIKM 2013: 1601-1606

[Zou14] L. Zou et al. Efficient processing of label-constraint reachability queries in large graphs. Inf. Syst. 40: 47-66 (2014)

[Zhu14] A. Zhu et al. Reachability queries on large dynamic graphs: a total order approach. SIGMOD Conference 2014: 1323-1334

[Mer14] F. Merz et al. PReaCH: A Fast Lightweight Reachability Index Using Pruning and Contraction Hierarchies. ESA 2014: 701-712

[Vel14] R. Veloso et al. Reachability Queries in Very Large Graphs: A Fast Refined Online Search Approach. EDBT 2014: 511-522

[Wei14] H. Wei et al. Reachability Querying: An Independent Permutation Labeling Approach. Proc. VLDB Endow. 7(12): 1191-1202 (2014)

[Ang17] R. Angles et al. Foundations of Modern Query Languages for Graph Databases. ACM Comput. Surv. 50(5): 68:1-68:40 (2017)

[Val17] L. Valstar et al. Landmark Indexing for Evaluation of Label-Constrained Reachability Queries. SIGMOD Conference 2017: 345-358

[Su17] J. Su et al. Reachability Querying: Can It Be Even Faster? IEEE Trans. Know. Data Eng. 29(3): 683-697 (2017)

[Bon18] A. Bonifati et al. Querying Graphs. Synthesis Lectures on Data Management, Morgan & Claypool Publishers 2018

[Wei18] H. Wei et al. Reachability querying: an independent permutation labeling approach. VLDB J. 27(1): 1-26 (2018)

[Li19] W. Li et al. Scaling Distance Labeling on Small-World Networks. SIGMOD Conference 2019: 1060-1077

[Pen20] Y. Peng et al. Answering billion-scale label-constrained reachability queries within microsecond. Proc. VLDB Endow. 13(6): 812-825 (2020)

[Che21] Y. Chen et al. Graph Indexing for Efficient Evaluation of Label-constrained Reachability Queries. ACM Trans. Database Syst. 46(2): 8:1-8:50 (2021)

[Han21] K. Han et. O'Reach: Even Faster Reachability in Large Graphs. SEA 2021: 13:1-13:24

[Lyu21] Q. Lyu et al. DBL: Efficient Reachability Queries on Dynamic Graphs. DASFAA (2) 2021: 761-777

[Che22] X. Chen et al. DLCR: Efficient Indexing for Label-Constrained Reachability Queries on Large Dynamic Graphs. Proc. VLDB Endow. 15(8): 1645-1657 (2022)

[Zha23] C. Zhang et al. A Reachability Index for Recursive Label-Concatenated Graph Queries. ICDE 2023: 66-80