Search on Graphs
Theory meets Engineering

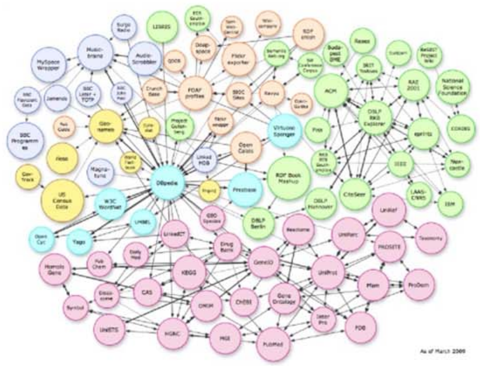
Yuqing Wu
Indiana University
U.S.A.

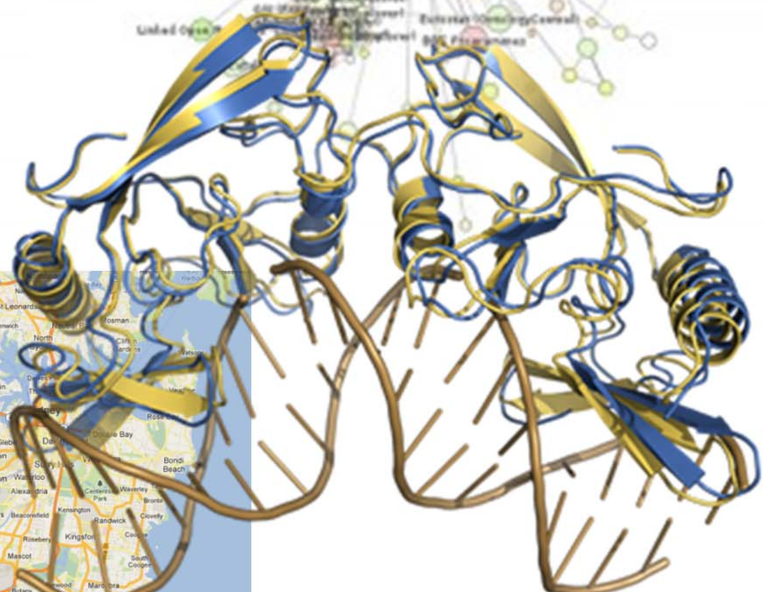
George Fletcher
Eindhoven Univ. of Tech.
The Netherlands

Tutorial Outline

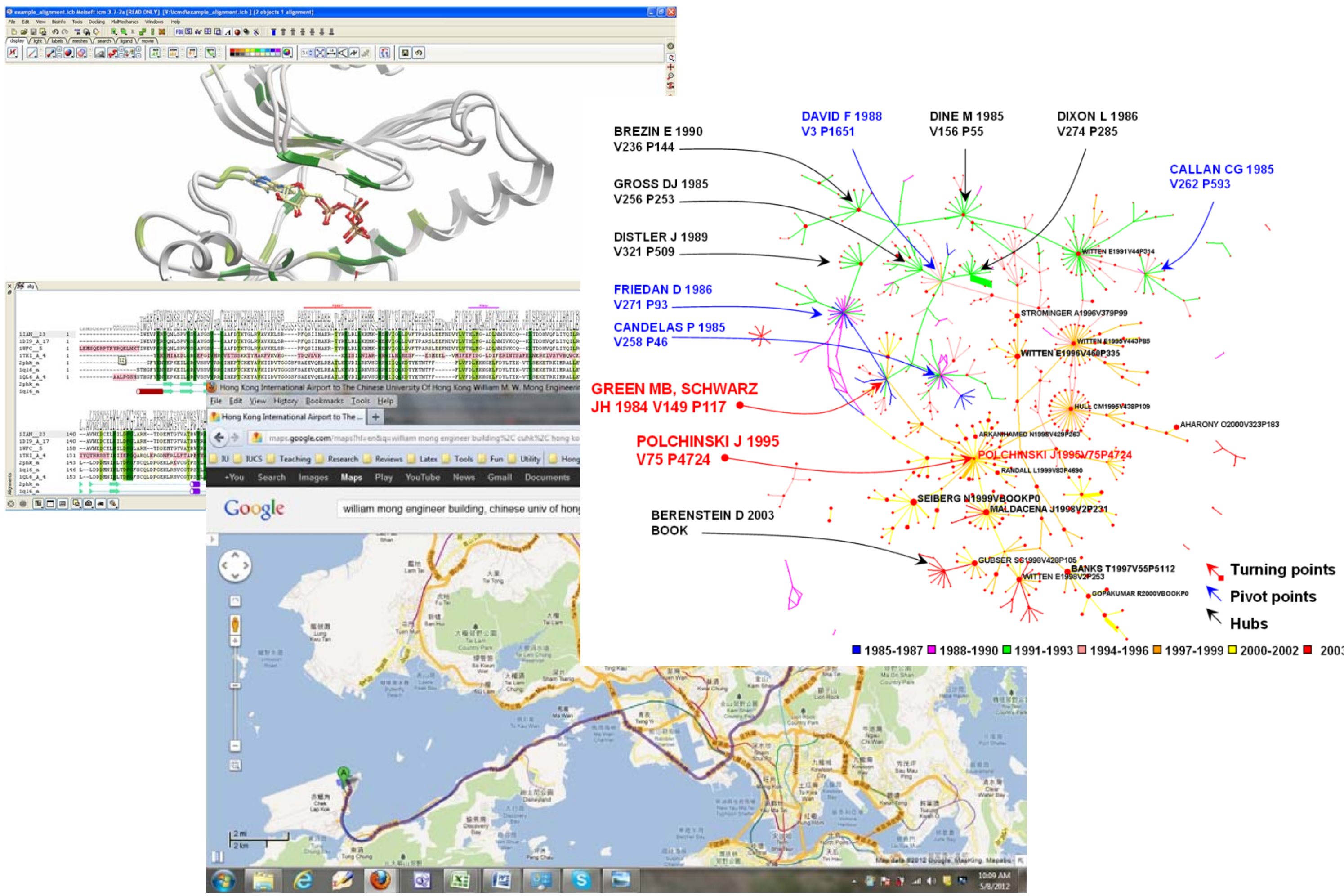
- **Graph Data and Search**
- The Theory
- The Engineering
- Summary & Future Directions

Graph Data



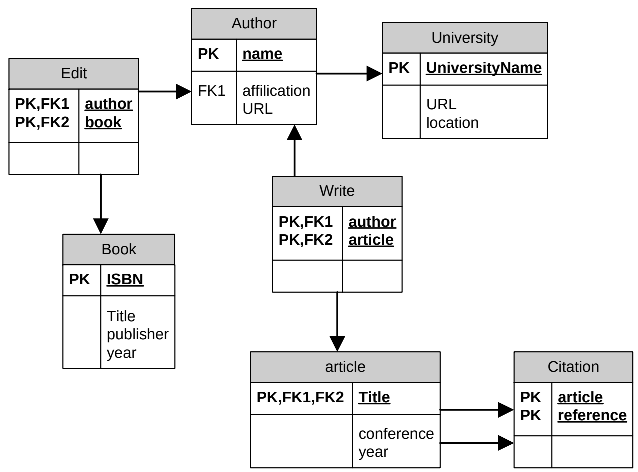


Search on Graphs

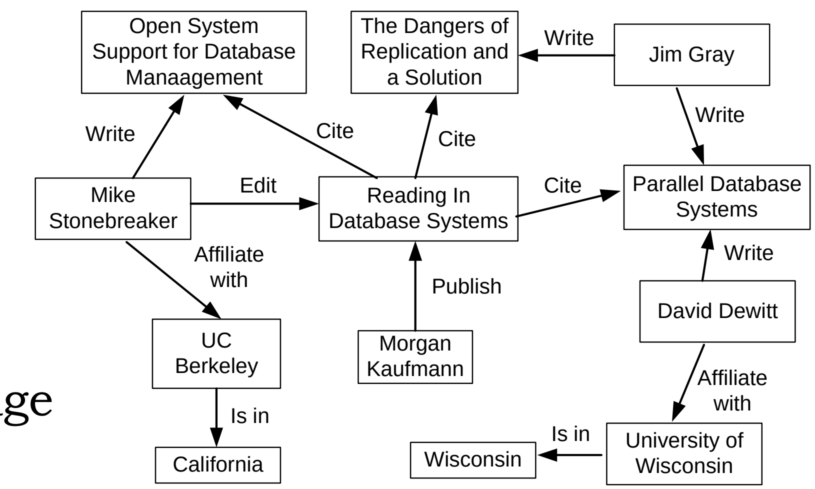


Data – in the Eyes of DB Researchers

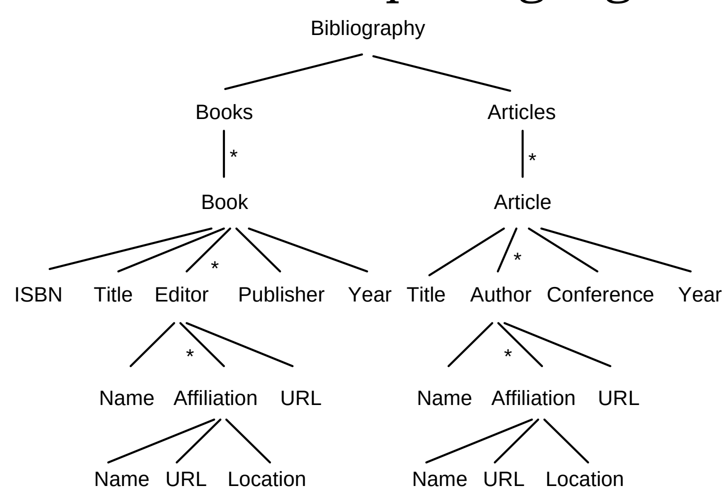
RDB – Relational Database



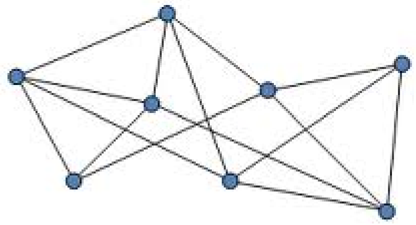
RDF – Resource Description Framework



XML – eXtensible Markup Language



Graph – an Abstract View



- The Major Ingredients
  - Nodes
  - Edges
- Variants
  - Labeled vs. unlabeled
  - Directed vs. undirected
  - Connected?
  - Weight
  - One big graph vs. many smaller graphs
  - Certain vs. uncertain
  - ……

Search on Graph – in the Eyes of DB Researchers

- SPARQL query answering
  - Triple store, vertical partition, property tables, …
  - MAP, Hexastore, TripleT, …
  - ……
- Keyword search
  - BANKS, BANKS2, BLINK,…
  - Community, r-clique,…
  - ……
- Other types of search problems
  - PSPARQL, CPSPARQL, SPARQL2L, SPARQLeR, …
  - GraphQL, GADDI, SUMMA, SAPPER, SAGA, TALE, …
  - ……

Search on Graph – an Abstract View

- Input
  - A node
  - A pair of nodes
  - A set of nodes
  - A sub-graph
  - A pattern
  - A set of keywords
  - ……
- Constraints
  - Size
  - Distance
  - Weight
  - *k* (for top-*k*)
  - Similarity threshold
  - ……
- Output
  - Connectivity
  - Distance
  - Path
  - Sub-structure
    - Node set
    - Sub-tree
    - Sub-graph
    - ……
- Variants
  - One / any / all / top-*k*
  - Shortest / smallest /most meaningful
  - Precise vs. approximate
  - ……

The Question

How to evaluate search queries efficiently??

- Challenges
  - Data access
  - Computation
  - ……
- Techniques
  - Data modeling and storage
  - Filtering and optimization
  - Indexing
  - Using statistical summary
  - Pre-compute partial results
  - …….

The Common Theme

- How to efficiently
  - Pick up the smallest superset of the results
  - Remove as many non-promising candidates as possible.

The Common Theme

- How to efficiently
  - Pick up the **smallest superset** of the results
  - Remove as many non-promising candidates as possible.

The Common Theme

- How to efficiently
  - Pick up the smallest superset of the results
  - Remove **as many** non-promising candidates **as possible**.

The Common Theme

- How to efficiently
  - Pick up the smallest superset of the results
  - Remove as many **non-promising candidates** as possible.

Tutorial Outline

- Graph Data and Search
- **The Theory**
  - **Methodology**
  - Relation Algebra on Tree-Structured Data
  - Relation Algebra on Graph
- The Engineering
- Summary & Future Direction

Theoretical Methodology

General methodology

- Coupling of expressive power of query languages with appropriate structural notions
  - *Fletcher, Van Gucht, Wu, Gyssens, Brenes, Paredaens. Inf. Syst. 2009*

Theoretical Methodology

Basic idea

- Restrict focus on a fixed query language L and (an arbitrary) fixed database instance I
- Define an equivalence relation $\approx_I$ on objects in the instance, purely in terms of the structure of the instance
  - For example, nodes m and n of graph I are structurally equivalent (i.e., $m \approx_I n$) iff m and n have the same node-labels
- Define an equivalence relation $\approx_L$ on objects in the instance, in terms of their indistinguishability by queries in L, i.e., for every query Q in L, either both of the objects are in $Q(I)$ or both are not in $Q(I)$
  - For example, $m \approx_L n$ for the language L of simple keyword queries
- Finally, establish the relationship between $m \approx_I n$ and $m \approx_L n$, preferably *iff*.
  - In this way, the behavior of an infinite object (i.e., the query language L) is reduced to a finite object (i.e., $\approx_I$)
  - Ideally, $\approx_I$ is tractable

Theoretical Applications

- Tarski's "relation algebra" (RA)
  - Proposed by Alfred Tarski in the 1940s
  - Simple navigational query language at the core of many standards
  - As a basic query language for reasoning about paths in trees/graphs

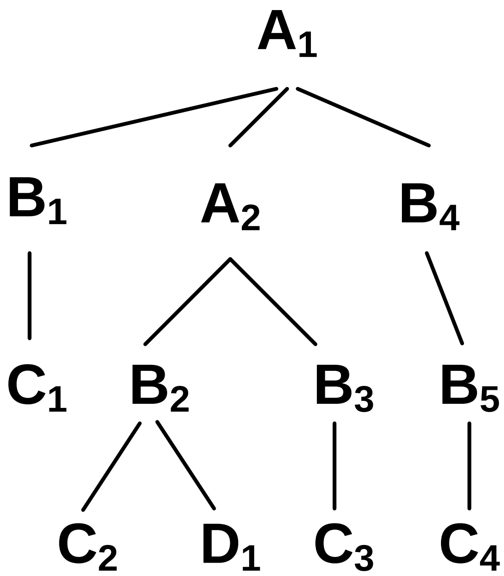
Theoretical Applications

- Study of the RA on trees
  - *Gyssens, Paredaens, Van Gucht, Fletcher. PODS 2006.*
  - *Wu, Van Gucht, Gyssens, Paredaens. Computer Journal, 2011.*
- Study of the RA on graphs
  - *Fletcher, Gyssens, Leinders, Van den Bussche, Van Gucht, Vansummeren, Wu. ICDT 2011.*

Tutorial Outline

- Graph Data and Search
- **The Theory**
  - Methodology
  - **Relation Algebra on Tree-Structured Data**
  - Relation Algebra on Graph
- The Engineering
- Summary & Future Directions

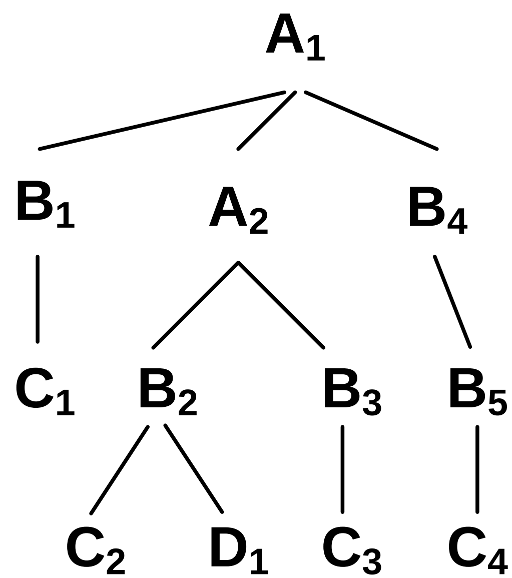
XML and Queries – An Example



- Query 1: //A/B/C
- Query 2: //B/C
- Query 3: //A/B[./D]/C
- Query 4: //A[./B[./D]]/B/C

Tree Data Model

- Represent XML document D as a finite unordered node-labeled tree
- D = (V, Ed, r, λ)
- Nodes: V
- Edges: Ed
- Root: r
- Labels: $\lambda : V \rightarrow \mathcal{L}$



Tarski's Relation Algebra on Trees

- Path semantics

XPath Algebra

$$\varepsilon(D) = \{(m,m) \mid m \in V\}$$

$$\phi(D) = \phi$$

$$l(D) = \{(m,m) \mid m \in V \wedge \lambda(m) = l\}$$

$$\downarrow(D) = Ed$$

$$\uparrow(D) = Ed^{-1}$$

$$\pi_1(E_1)(D) = \{(m,m) \mid \exists n : (m,n) \in E_1(D)\}$$

$$E_1 \circ E_2(D) = \{(m,n) \mid \exists w : (m,w) \in E_1(D) \wedge (w.n) \in E_2(D)\}$$

- Node semantics

$$E(D)[nodes] = \{n \mid \exists m : (m,n) \in E(D)\}$$

XPath Algebra – Examples

- Query 1: //A/B/C

$$A \circ \downarrow \circ B \circ \downarrow \circ C$$

- Query 3: //A/B[./D]/C

$$A \circ \downarrow \circ B \circ \pi_1 \left(\downarrow \circ D \right) \circ \downarrow \circ C$$

- Query 4: //A[./B[./D]]/B/C

$$A \circ \pi_1 \left(\downarrow \circ B \circ \pi_1 \left(\downarrow \circ D \right) \right) \circ \downarrow \circ B \circ \downarrow \circ C$$

Fragments of XPath Algebra

- $\mathcal{U}$ algebra   XPath algebra - $\downarrow$, π_1
- $\mathcal{D}$ algebra   XPath algebra - $\uparrow$, π_1
- $\mathcal{D}^{[\,]}$ algebra   XPath algebra - $\uparrow$

- $\mathcal{U}[k]$ algebra $\mathcal{U}$ algebra up to length k
- $\mathcal{D}[k]$ algebra $\mathcal{D}$ algebra up to length k
- $\mathcal{D}^{[\,]}[k]$ algebra $\mathcal{D}^{[\,]}$ algebra up to length k

$\mathcal{D}$ [k] Equivalence

Given an XML document and value k and (m_1, n_1), (m_2, n_2) in *DownPairs($\mathcal{D}$)*

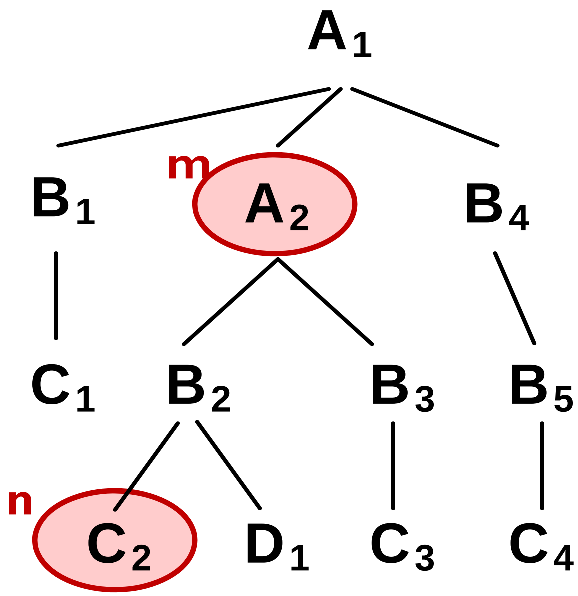
$$(m_1, n_1) =_{\mathcal{D}[k]} (m_2, n_2)$$

$$\Updownarrow$$

For any E in $\mathcal{D}$ [k]

$$(m_1, n_1) \in E(D) \Leftrightarrow (m_2, n_2) \in E(D)$$

Label Path



- $DownPair(D,k)$
- $LP(m,n)$
  - $LP(m,n)=$ (A,B,C)
- $LP(n,k)$
  - $LP(n,0)=$ (C)
  - $LP(n,1)=$ (B,C)
  - $LP(n,4)=$ (A,A,B,C)
  - $LP(n,7)=$ (A,A,B,C)

$\mathcal{N}[k]$ Equivalence

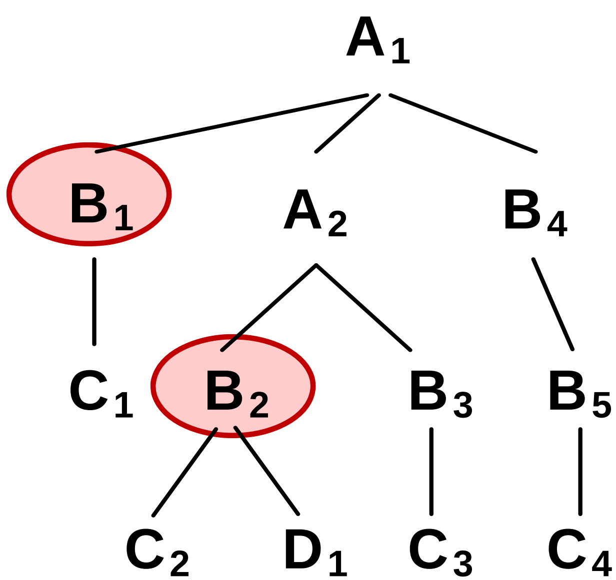
- Given an XML document and value k

$$n_1 \equiv_{\mathcal{N}[k]} n_2$$
$$\Updownarrow$$
$$\mathcal{LP}(n_1, k) = \mathcal{LP}(n_2, k)$$

$\mathcal{N}[k]$ Equivalence

- Given an XML document and value k

$$n_1 \equiv_{\mathcal{N}[k]} n_2 \quad \Leftrightarrow \quad \mathcal{LP}(n_1, k) = \mathcal{LP}(n_2, k)$$



$$B_1 \equiv_{\mathcal{N}[1]} B_2$$

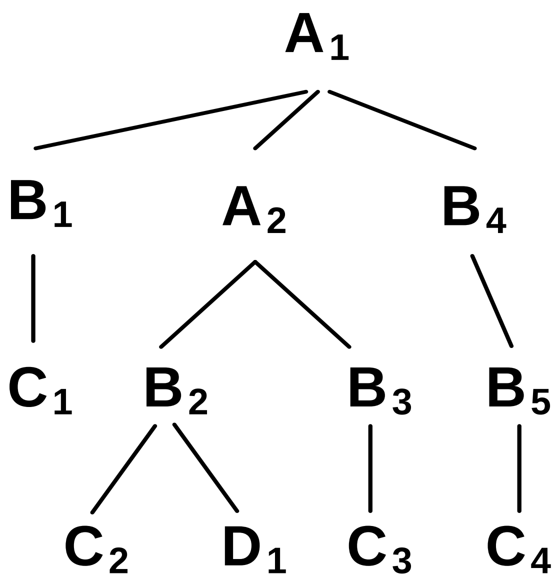
$$B_1 \neq_{\mathcal{N}[2]} B_2$$

$\mathcal{N}[k]$ Partition

- Partition induced by the $\mathcal{N}[k]$-equivalence relationship.
- $\mathcal{N}[k]$-partition block $\leftrightarrow$ label path

$\mathcal{N}[k]$ Partition

$$n_1 \equiv_{\mathcal{N}[k]} n_2 \quad \Leftrightarrow \quad \mathcal{LP}(n_1, k) = \mathcal{LP}(n_2, k)$$

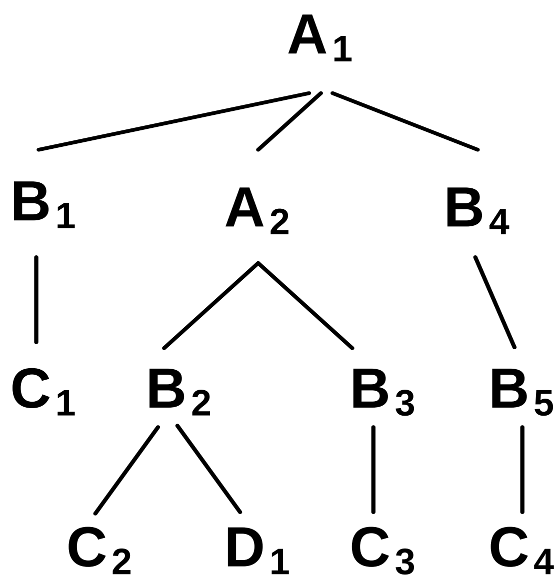


$\mathcal{N}[1]$

(A)	$\{A_1\}$
(A,A)	$\{A_2\}$
(A,B)	$\{B_1, B_2, B_3, B_4\}$
(B,B)	$\{B_5\}$
(B,C)	$\{C_1, C_2, C_3, C_4\}$
(B,D)	$\{D_1\}$

$\mathcal{N}[1][(A,B)] = \{B_1, B_2, B_3, B_4\}$

Partition Refinement



$\mathcal{N}[1]$	(A)	$\{A_1,\}$
	(A,A)	$\{A_2\}$
	(A,B)	$\{B_1, B_2, B_3, B_4\}$
	(B,B)	$\{B_5\}$
	(B,C)	$\{C_1, C_2, C_3, C_4\}$
	(B,D)	$\{D_1\}$

$\mathcal{N}[2]$	(A)	$\{A_1,\}$
	(A,A)	$\{A_2\}$
	(A,B)	$\{B_1, B_4\}$
	(A,A,B)	$\{B_2, B_3,\}$
	(A,B,B)	$\{B_5\}$
	(A,B,C)	$\{C_1, C_2, C_3\}$
	(B,B,C)	$\{C_4\}$
	(A,B,D)	$\{D_1\}$

$$\mathcal{A} \prec \mathcal{B}$$

$$\left.\begin{array}{l} \mathcal{A} \prec \mathcal{B} \\ \mathcal{B} \prec \mathcal{A} \end{array}\right\} \Rightarrow \quad \mathcal{A} \cong \mathcal{B}$$

$$\mathcal{N}[2] \prec \mathcal{N}[1]$$

$$\mathcal{N}[8] \cong \mathcal{N}[10]$$

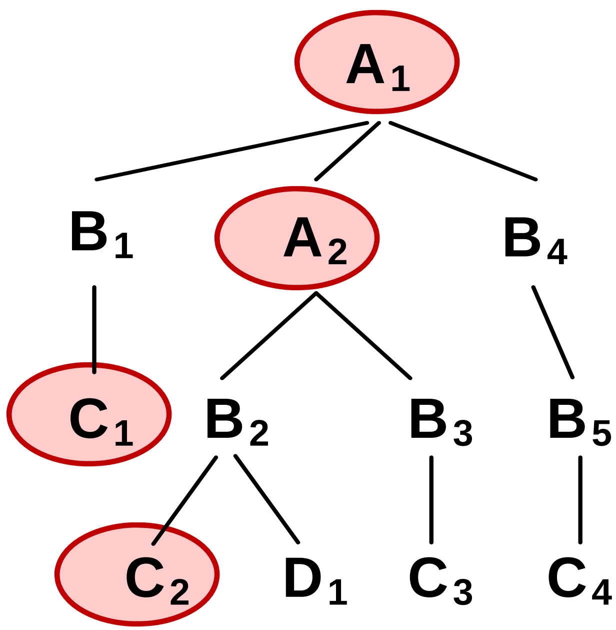
$\mathcal{P}[k]$ Equivalence

- Given an XML document and value k

$$(m_1, n_1) \equiv_{\mathcal{P}[k]} (m_2, n_2)$$

$$\Updownarrow$$

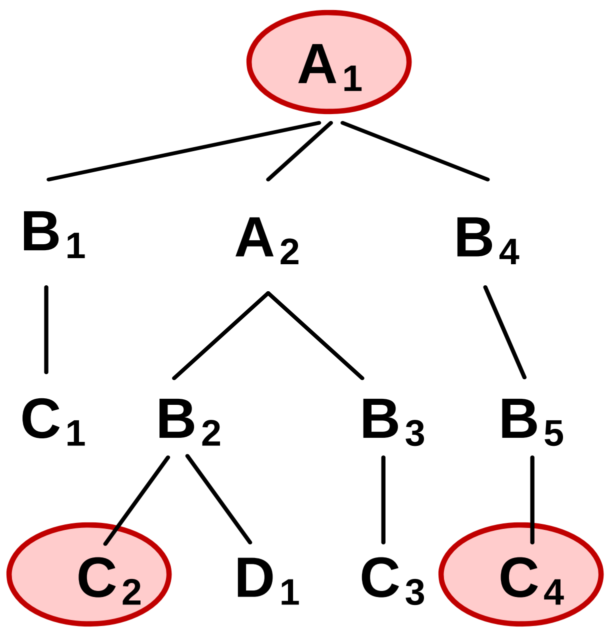
$$\mathcal{LP}(m_1, n_1) = \mathcal{LP}(m_2, n_2)$$



$$(A_1, C_1) \equiv_{\mathcal{P}[2]} (A_2, C_2)$$

$\mathcal{P}[k]$ Equivalence

- Given an XML document and value k



$$(m_1, n_1) \equiv_{\mathcal{P}[k]} (m_2, n_2)$$

$$\Updownarrow$$

$$\mathcal{LP}(m_1, n_1) = \mathcal{LP}(m_2, n_2)$$

$$(A_1, C_1) \equiv_{\mathcal{P}[2]} (A_2, C_2)$$

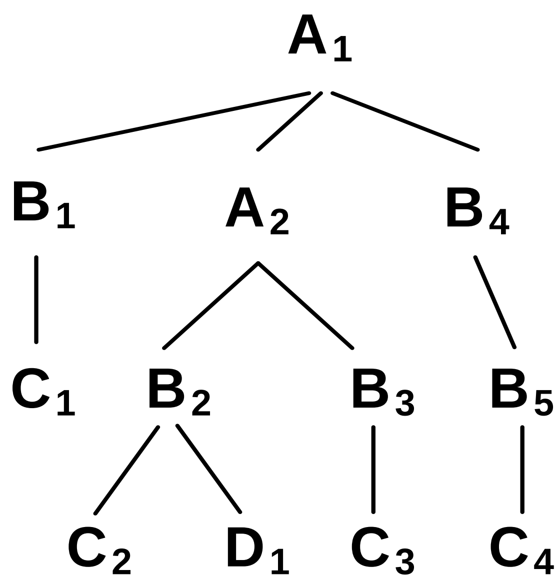
$$(A_1, C_2) \neq_{\mathcal{P}[3]} (A_1, C_4)$$

$\mathcal{P}[k]$ Partition

- Partition induced by the $\mathcal{P}[k]$-equivalence relationship.

- $\mathcal{P}[k]$-partition block $\leftrightarrow$ label path

$\mathcal{P}[k]$ Partition

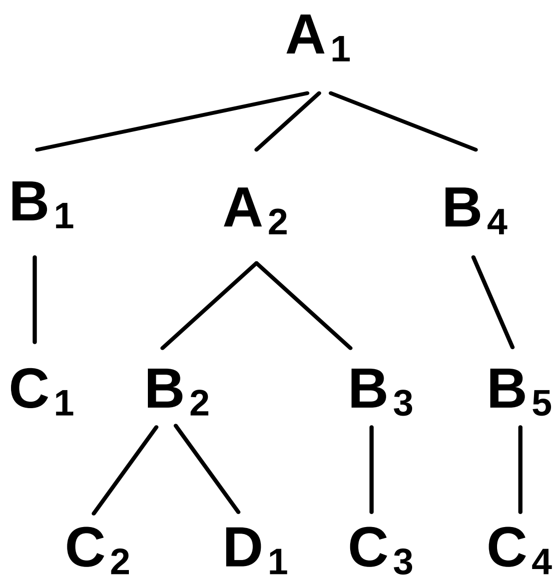


$\mathcal{P}[1]$	(A)	$\{(A_1,A_1), (A_2,A_2)\}$
	(B)	$\{(B_1, B_1), (B_2, B_2), (B_3, B_3), (B_4, B_4), (B_5, B_5)\}$
	(C)	$\{(C_1, C_1), (C_2, C_2), (C_3, C_3), (C_4, C_4)\}$
	(D)	$\{(D_1, D_1)\}$
	(A,A)	$\{(A_1,A_2)\}$
	(A,B)	$\{(A_1, B_1), (A_2, B_2), (A_2, B_3), (A_1, B_4)\}$
	(B,B)	$\{(B_4, B_5)\}$
	(B,C)	$\{(B_1, C_1), (B_2, C_2), (B_3, C_3), (B_5, C_4)\}$
	(B,D)	$\{(B_2, D_1)\}$

Partition refinement?

$$\mathcal{P}[1][(A,A)] = \{(A_1,A_2)\}$$

$\mathcal{P}[k]$ Partition



$\mathcal{P}[2]$	(A)	$\{(A_1,A_1), (A_2,A_2)\}$
	(B)	$\{(B_1, B_1), (B_2, B_2), (B_3, B_3), (B_4, B_4), (B_5, B_5)\}$
	(C)	$\{(C_1, C_1), (C_2, C_2), (C_3, C_3), (C_4, C_4)\}$
	(D)	$\{(D_1, D_1)\}$
	(A,A)	$\{(A_1,A_2)\}$
	(A,B)	$\{(A_1, B_1), (A_2, B_2), (A_2, B_3), (A_1, B_4)\}$
	(B,B)	$\{(B_4, B_5)\}$
	(B,C)	$\{(B_1, C_1), (B_2, C_2), (B_3, C_3), (B_5, C_4)\}$
	(B,D)	$\{(B_2, D_1)\}$
	(A,A,B)	$\{(A_1, B_2), (A_1, B_3)\}$
	(A,B,B)	$\{(A_1, B_5)\}$
	(A,B,C)	$\{(A_1, C_1), (A_2, C_2), (A_2, C_3)\}$
	(A,B,D)	$\{(A_2, D_1)\}$
	(B,B,C)	$\{(B_4, C_4)\}$

$$\mathcal{P}[2][(A,B,C)] = \{(A_1, C_1), (A_2, C_2), (A_2, C_3)\}$$

Coupling Theorem

Let *D be a document and k is an integer.*

- *The $\mathcal{P}[k]$-partition of D* and the *$\mathcal{D}[k]$- partition of D are the same under the path semantics*
- *The $\mathcal{N}[k]$-partition of D* and the *$\mathcal{D}[k]$-partition of D are the same under the node semantics*

$$\mathcal{D}[k] \cong \mathcal{N}[k] \qquad \mathcal{D} \cong \mathcal{N}[\infty]$$

$$\mathcal{D}[k] \cong \mathcal{P}[k] \qquad \mathcal{D} \cong \mathcal{P}[\infty]$$

Label-Union Theorem

Let *D be a document, k an integer, and E is an* $\mathcal{D}[k]$ *expression. Then there* exists a class of *partition blocks of the* $\mathcal{P}[k]$*-partition (*$\mathcal{N}[k]$*-partition)* of *D such that*

$$E(D)[nodes] = \bigcup_{lp \in LPS(E,k)} \mathcal{N}[k][lp]$$

$$E(D) = \bigcup_{lp \in LPS(E,k)} \mathcal{P}[k][lp]$$

Tutorial Outline

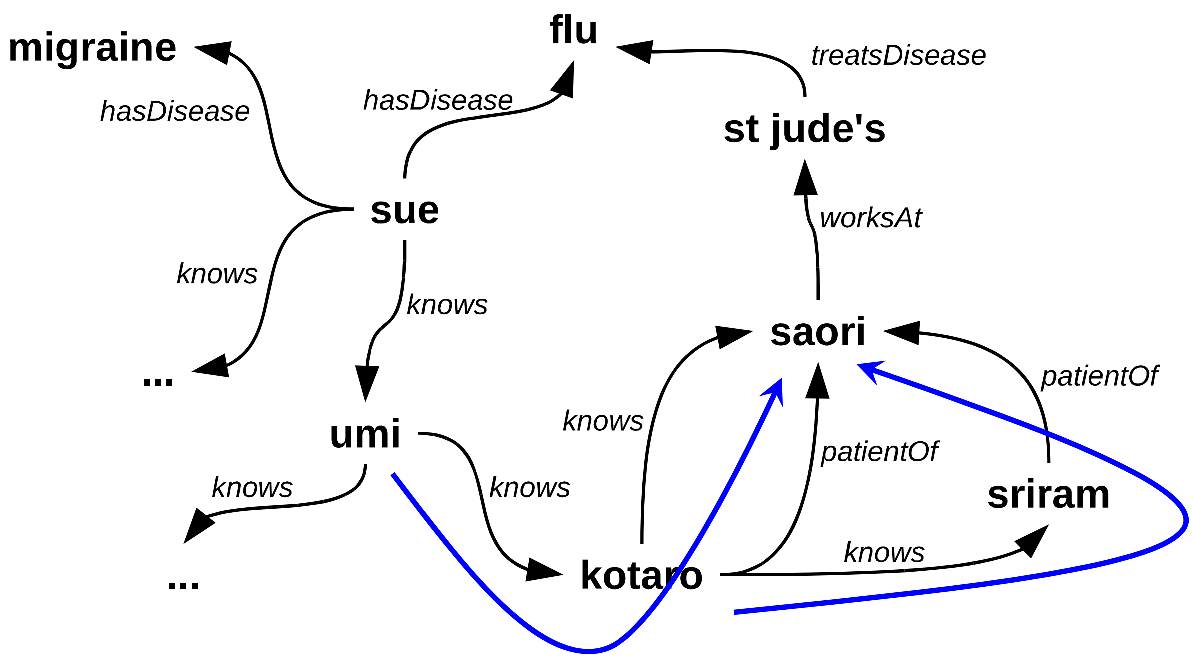
- Graph Data and Search
- **The Theory**
  - Methodology
  - Relation Algebra on Tree-Structured Data
  - **Relation Algebra on Graph**
- The Engineering
- Summary & Future Directions

Tarski's Relation Algebra on Graphs

- Path semantics, for graph G with edge labels R

$$
\begin{aligned}
R(G) &= G(R)\,; \\
\emptyset(G) &= \emptyset\,; \\
id(G) &= \{(m,m) \mid m \in \mathrm{adom}(G)\}\,; \\
e_1 \circ e_2(G) &= \{(m,n) \mid \exists p\,((m,p) \in e_1(G) \;\&\; (p,n) \in e_2(G))\}\,; \\
e_1 \cup e_2(G) &= e_1(G) \cup e_2(G)\,.
\end{aligned}
$$

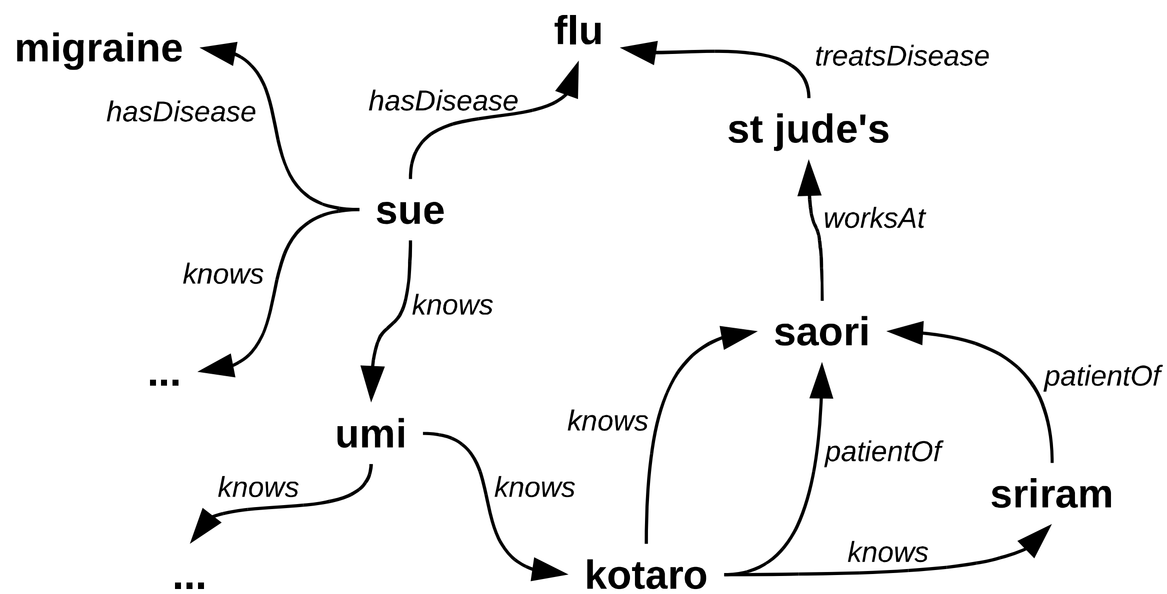
Tarski's Relation Algebra on Graphs



Example: doctors of friends, by person

$$\text{knows} \circ \text{patientOf}(G) = \{(umi, saori), (kotaro, saori), \ldots\}$$

Tarski's Relation Algebra on Graphs



Example: friends and friends of friends

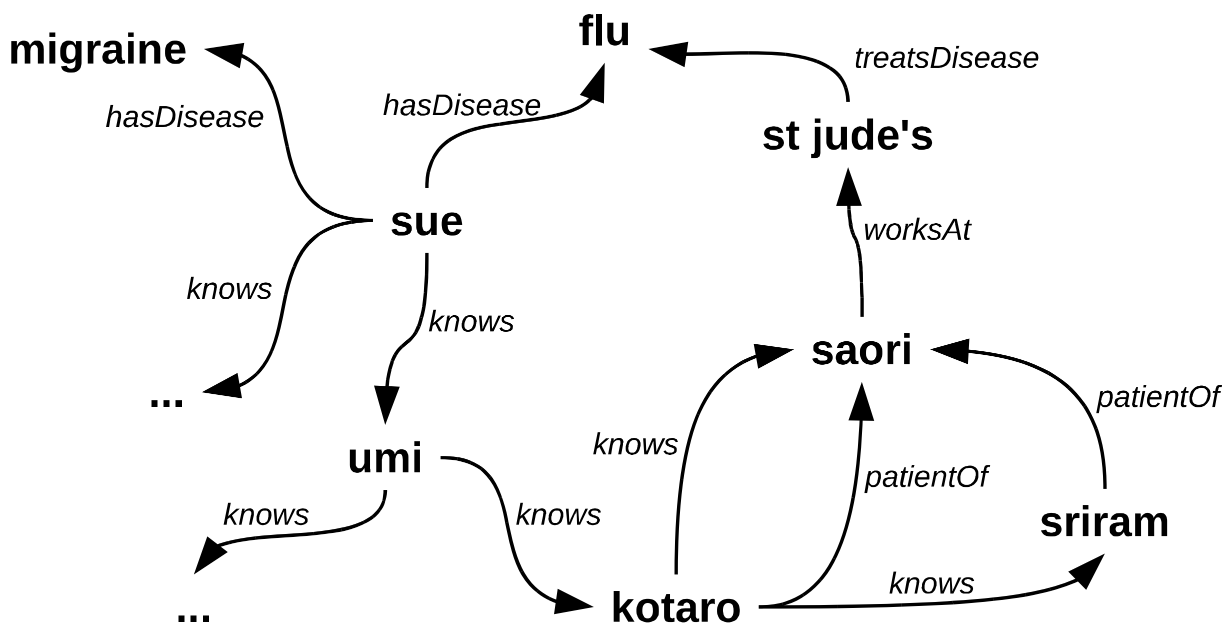
$$\text{knows} \cup \text{knows} \circ \text{knows}(G) = \{(sue, umi), (sue, kotaro), (umi, kotaro), (umi, saori), (kotaro, saori), \ldots\}$$

Tarski's Relation Algebra on Graphs

- Path semantics, cont.

$$
\begin{aligned}
di(G) &= \{(m,n) \mid m,n \in \mathrm{adom}(G) \;\&\; m \neq n\}\,; \\
e^{-1}(G) &= \{(m,n) \mid (n,m) \in e(G)\}\,; \\
e_1 \cap e_2(G) &= e_1(G) \cap e_2(G)\,; \\
e_1 \setminus e_2(G) &= e_1(G) \setminus e_2(G)\,; \\
\pi_1(e)(G) &= \{(m,m) \mid m \in \mathrm{adom}(G) \;\&\; \exists n\,(m,n) \in e(G)\}\,; \\
\pi_2(e)(G) &= \{(m,m) \mid m \in \mathrm{adom}(G) \;\&\; \exists n\,(n,m) \in e(G)\}\,; \\
\overline{\pi}_1(e)(G) &= \{(m,m) \mid m \in \mathrm{adom}(G) \;\&\; \neg\exists n\,(m,n) \in e(G)\}\,; \\
\overline{\pi}_2(e)(G) &= \{(m,m) \mid m \in \mathrm{adom}(G) \;\&\; \neg\exists n\,(n,m) \in e(G)\}\,; \\
e^{+}(G) &= \bigcup_{k \geq 1} e^{k}(G)\,.
\end{aligned}
$$

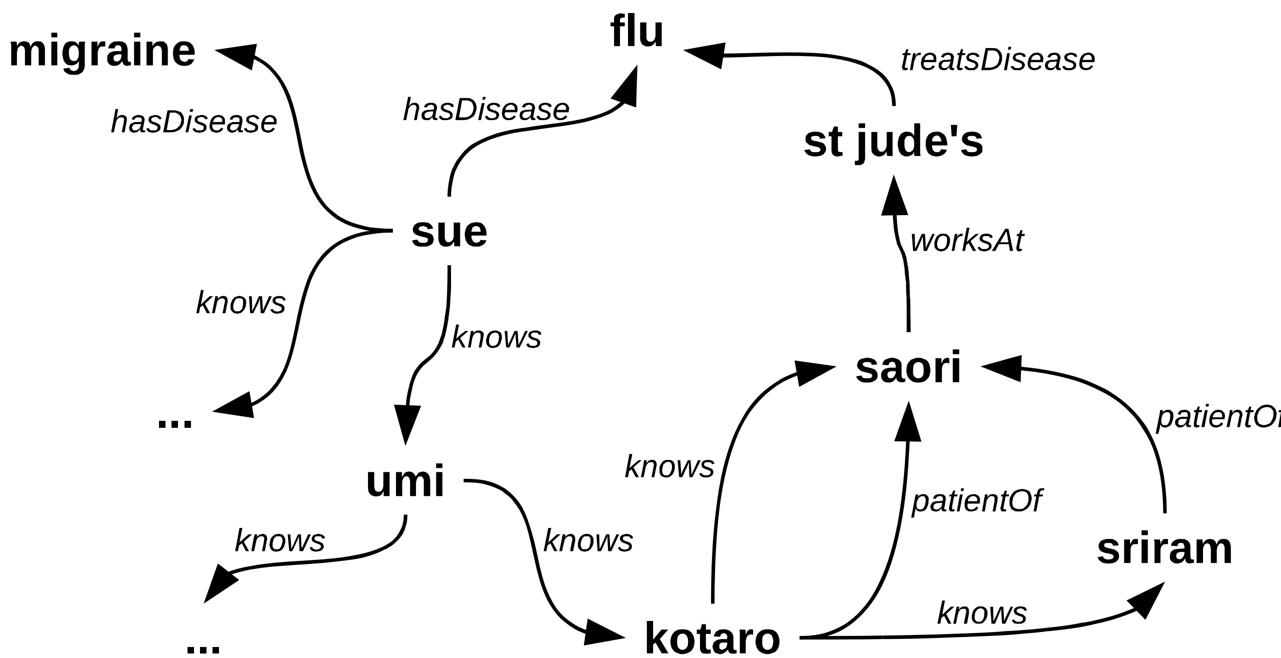
Tarski's Relation Algebra on Graphs



Example: people with untreatable diseases

$$\mathsf{hasDisease} \setminus (\mathsf{hasDisease} \circ \pi_2(\mathsf{treatsDisease}))(G) = \{(\mathit{sue}, \mathit{migraine}), \ldots\}$$

Tarski's Relation Algebra on Graphs



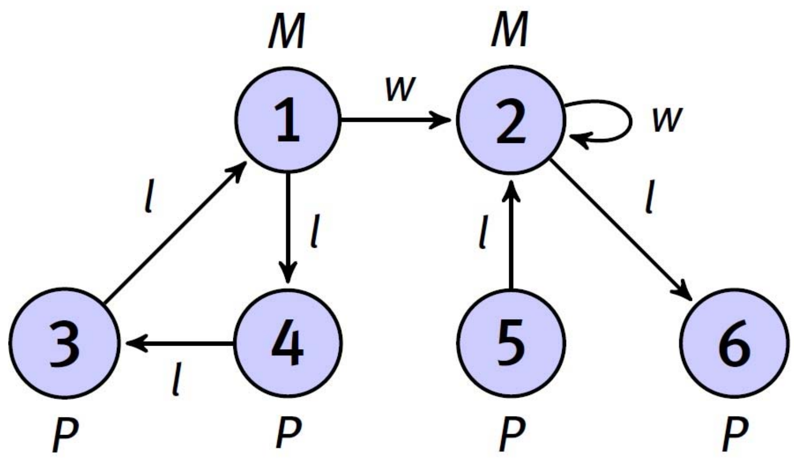
Example: the social network

$$\text{knows}^{+}(G) = \{(sue, umi), (sue, kotaro), (sue, saori), \ldots\}$$

k-Bisimilarity on Graphs

- Let $G = <N,E,\lambda_N,\lambda_E>$ be a graph and k be a nonnegative integer.
- Node u, v $\in$ N are called **k-bimsimilar** ($u \approx^{\kappa} v$), iff the following holds
  - $\lambda_N(u) = \lambda_N(v)$
  - If k>0, then, $\forall u' \in N\ [(u,u') \in E \Rightarrow \exists v' \in N\ [(v,v') \in E,\ u' \approx^{\kappa-1} v'$ and $\lambda_E(u,u') = \lambda_E(v,v')]]$
  - If k>0, then, $\forall v' \in N\ [(v,v') \in E \Rightarrow \exists u' \in N\ [(u,u') \in E,\ v' \approx^{\kappa-1} u'$ and $\lambda_E(v,v') = \lambda_E(u,u')]]$

k-Bisimilarity on Graphs – An Example



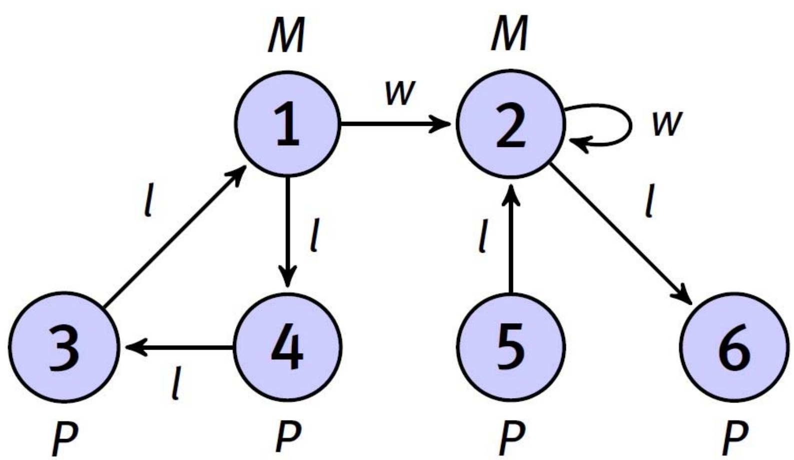
$n_1 \approx^0 n_2$ √

$n_1 \approx^1 n_2$ √

$n_1 \approx^2 n_2$ ✕

k-Partition

A partition of N based on k-bisimilarity



k=0	$\{n_1, n_2\}$
	$\{n_3, n_4, n_5, n_6\}$
k=1	$\{n_1, n_2\}$
	$\{n_3, n_5\}$
	$\{n_4\}$
	$\{n_6\}$
k=2	$\{n_1\}$
	$\{n_2\}$
	$\{n_3, n_5\}$
	$\{n_4\}$
	$\{n_6\}$

Partition refinement?

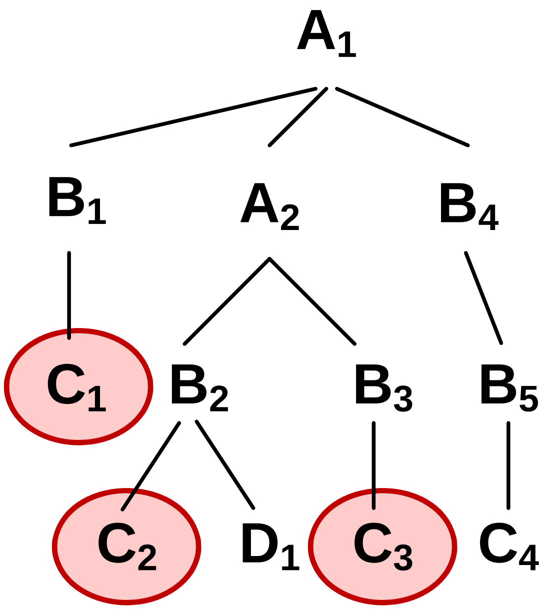
Variations and Applications

- Appropriate variations of bisimilarity can be defined for fragments of the RA
  - Positive fragments (i.e., those without difference) correspond to a weaker notion of "similarity"
  - Both bisimilarity and similarity are tractable structural equivalence notions
- Applications
  - establish coupling of language and structural equivalence, as with trees
  - use these couplings to separate many basic fragments of the RA

Tutorial Outline

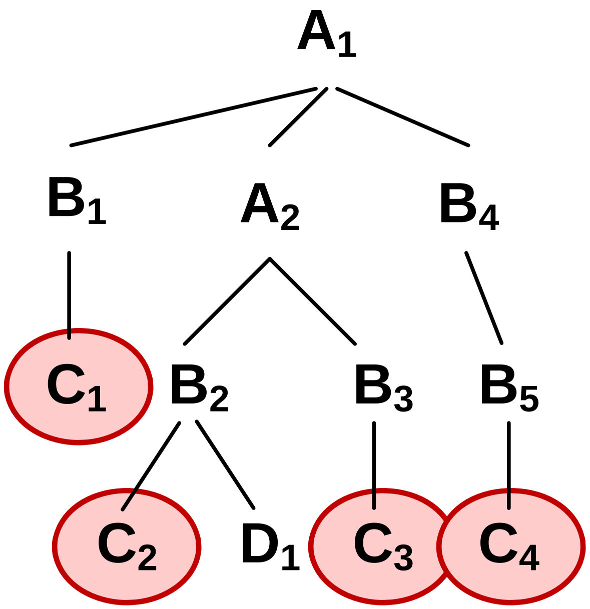
- Graph Data and Search
- The Theory
- **The Engineering**
  - **Indexing Tree Structured Data**
  - Bi-Similarity Partition on Graphs
- Summary & Future Directions

XML and Queries – An Example



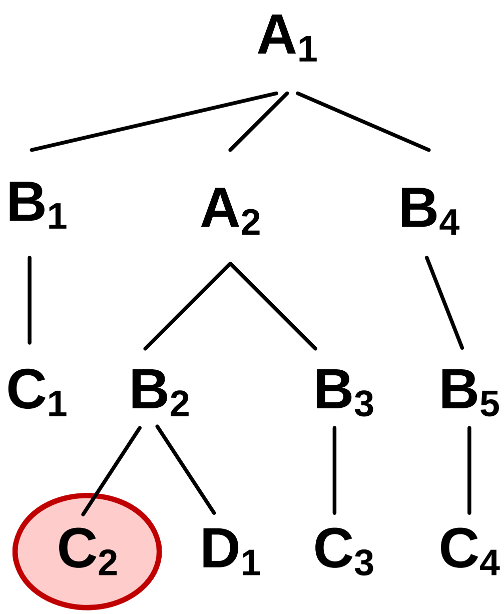
- Query 1: //A/B/C
- Query 2: //B/C
- Query 3: //A/B[./D]/C
- Query 4: //A[./B[./D]]/B/C

XML and Queries – An Example



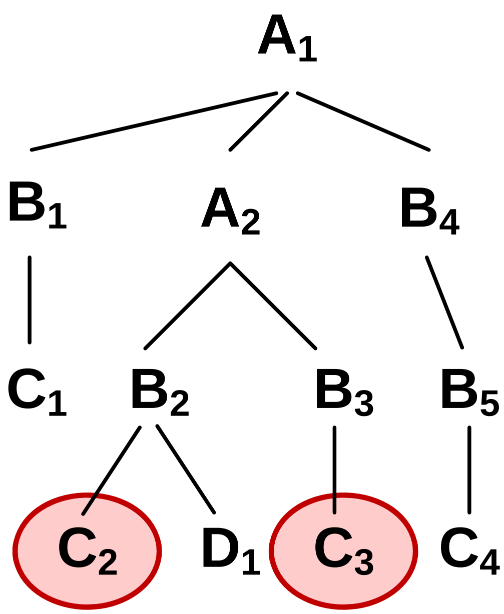
- Query 1: //A/B/C
- Query 2: //B/C
- Query 3: //A/B[./D]/C
- Query 4: //A[./B[./D]]/B/C

XML and Queries – An Example



- Query 1: //A/B/C
- Query 2: //B/C
- Query 3: //A/B[./D]/C
- Query 4: //A[./B[./D]]/B/C

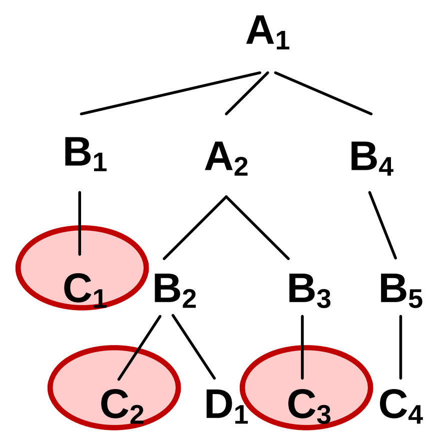
XML and Queries – An Example



- Query 1: //A/B/C
- Query 2: //B/C
- Query 3: //A/B[./D]/C
- Query 4: //A[./B[./D]]/B/C

Query Evaluation Using Label-Union Theorem

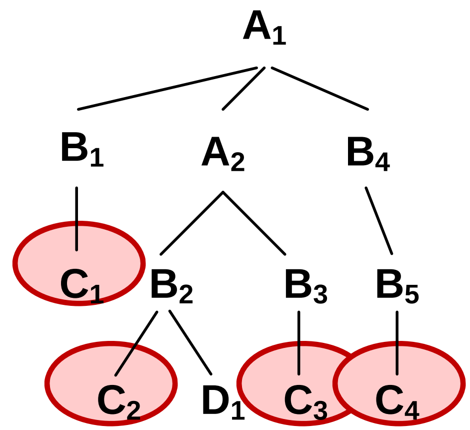
- Query 1: //A/B/C
- LPS(E,2) = {(A,B,C)}



$\mathcal{N}[2]$	(A)	$\{A_1,\}$
	(A,A)	$\{A_2\}$
	(A,B)	$\{B_1, B_4\}$
	(A,A,B)	$\{B_2, B_3,\}$
	(A,B,B)	$\{B_5\}$
	(A,B,C)	$\{C_1, C_2, C_3\}$
	(B,B,C)	$\{C_4\}$
	(A,B,D)	$\{D_1\}$

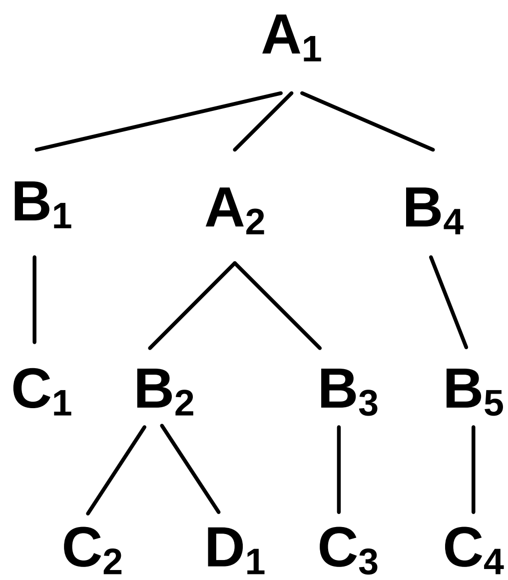
Query Evaluation Using Label-Union Theorem

- Query 2: //B/C
- LPS(E,2) = {(A,B,C), (B,B,C)}

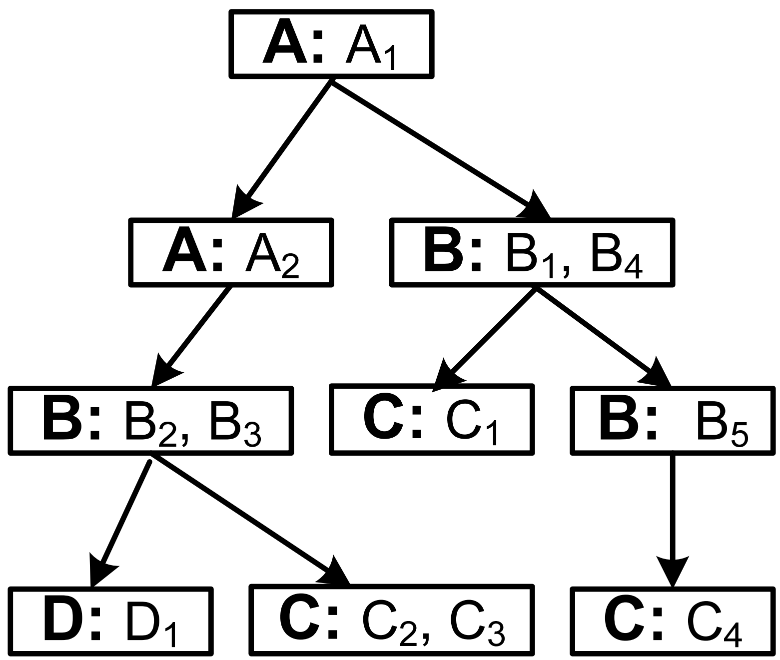


$\mathcal{N}[2]$	(A)	$\{A_1,\}$
	(A,A)	$\{A_2\}$
	(A,B)	$\{B_1, B_4\}$
	(A,A,B)	$\{B_2, B_3,\}$
	(A,B,B)	$\{B_5\}$
	(A,B,C)	$\{C_1, C_2, C_3\}$
	(B,B,C)	$\{C_4\}$
	(A,B,D)	$\{D_1\}$

I-Index – An Example



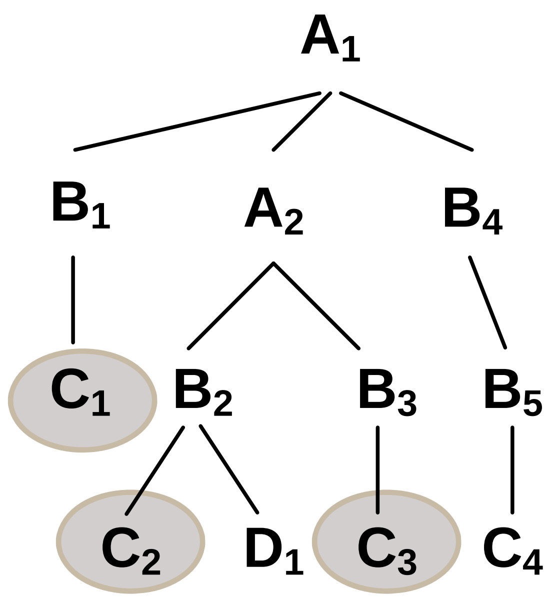
An index based on $\mathcal{N}[d]$ partition.
(d: height of XML tree)



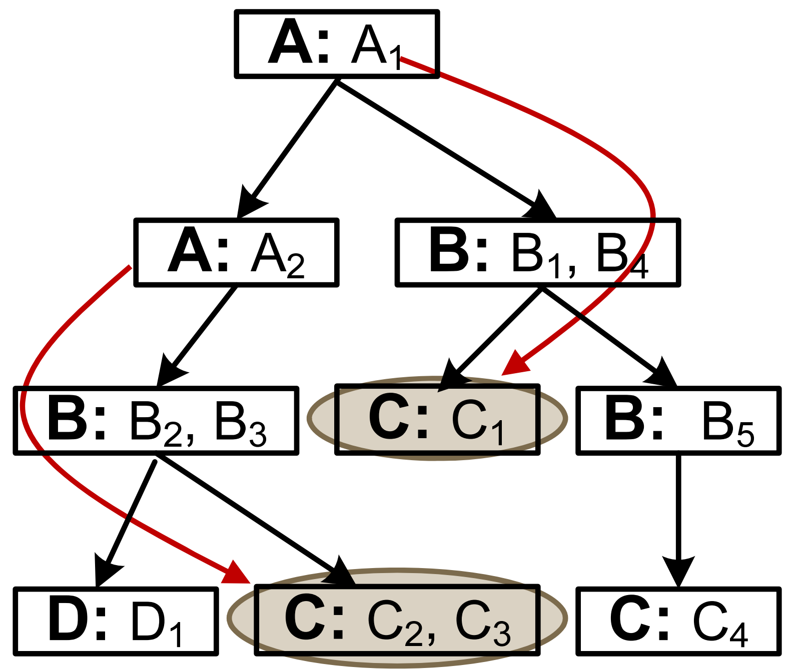
I-index

Tova Milo, Dan Suciu: Index Structures for Path Expressions. ICDT 1999.

Answering Queries with 1-Index

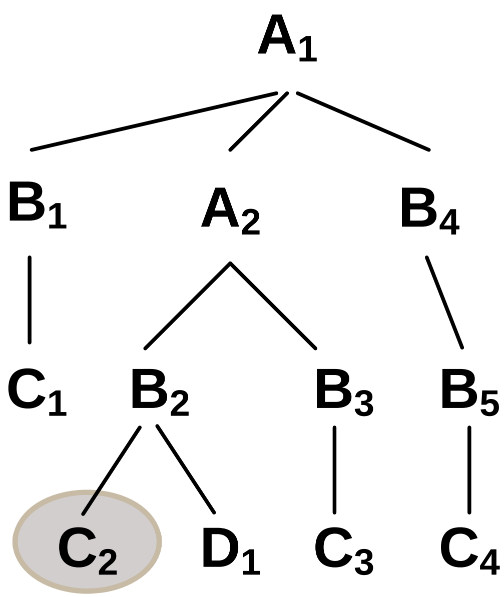


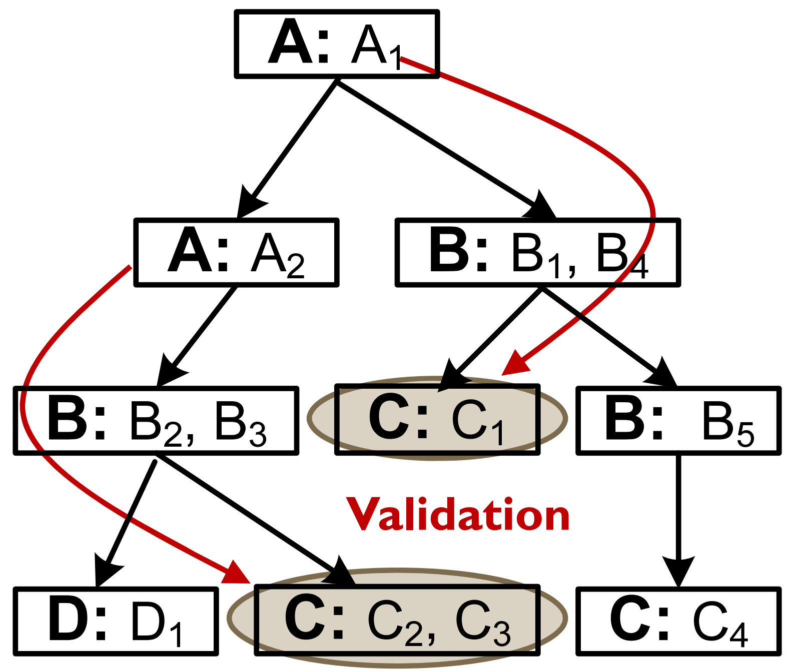
- Query 1: //A/B/C
- Query 2: //B/C
- Query 3: //A/B[./D]/C



1-Index

Answering Queries with 1-Index

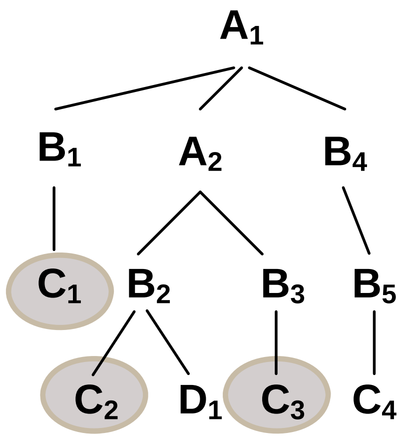




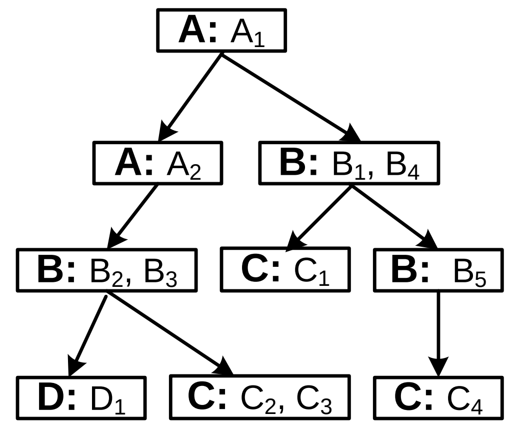
1-Index

- Query 1: //A/B/C
- Query 2: //B/C
- Query 3: //A/B[./D]/C

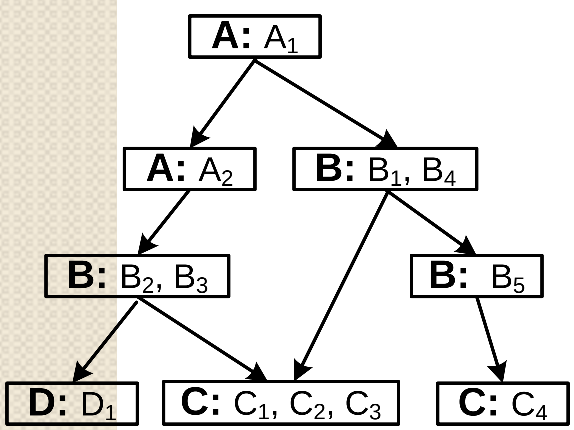
A(k)-Index – Examples



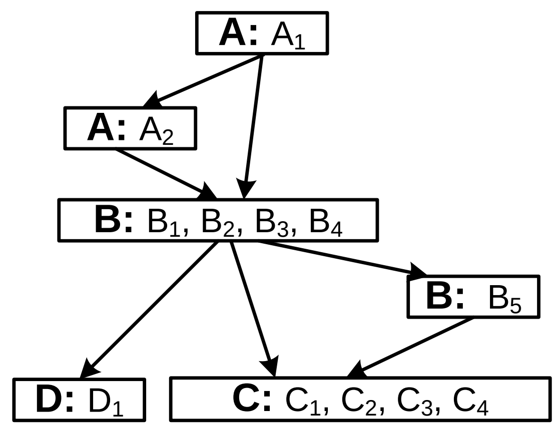
An index based on $\mathcal{N}[k]$ partition.



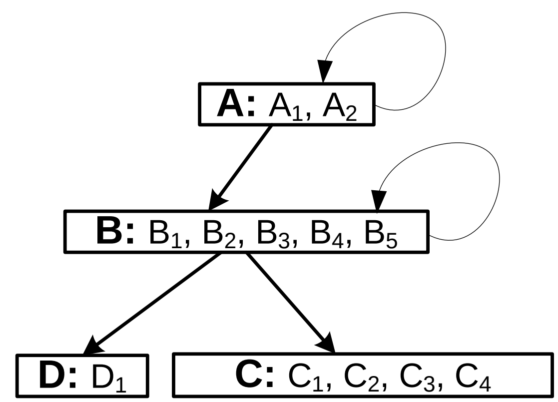
A(3)-Index (1-Index)



A(2)-Index



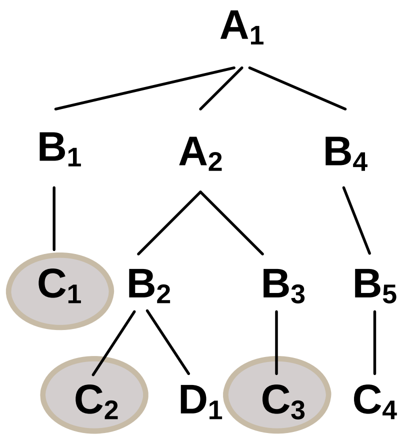
A(1)-Index



A(0)-Index

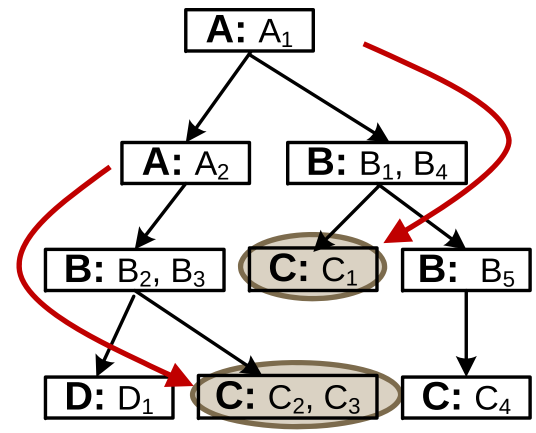
Raghav Kaushik et al.: Exploiting Local Similarity for Indexing Paths in Graph-Structured Data. ICDE 2002.

A(k)-Index – Examples

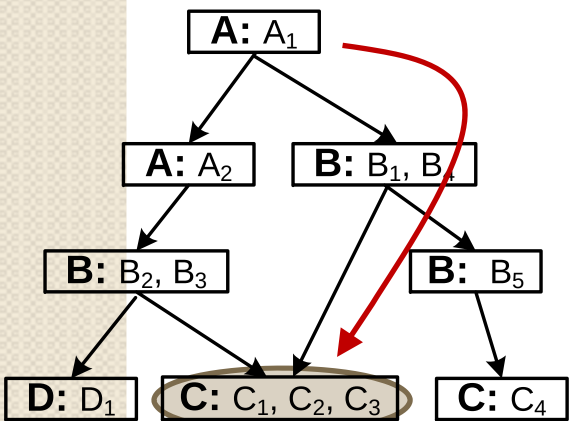


- Query 1: //A/B/C
- Query 2: //B/C
- Query 3: //A/B[./D]/C

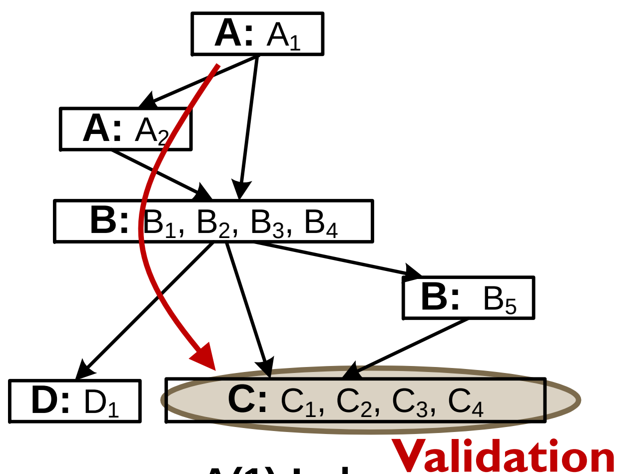
$\mathcal{D}$ [2] algebra



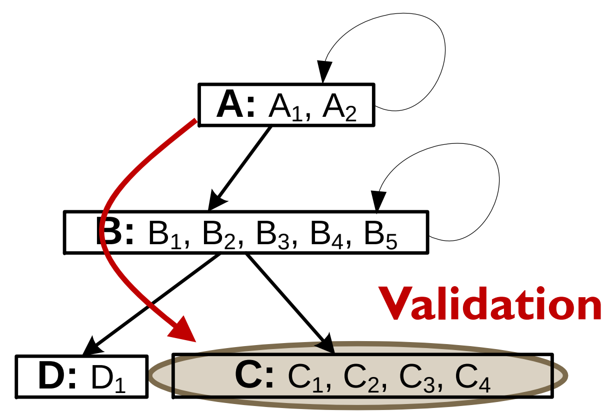
A(3)-Index (DataGuide)



A(2)-Index



A(1)-Index



A(0)-Index

Back to Theory

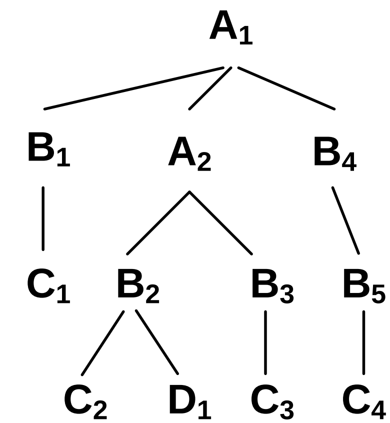
- Observation
  - These indices are based on node partition.

$$n_1 \equiv_{\mathcal{N}[k]} n_2$$
$$\Updownarrow$$
$$\mathcal{LP}(n_1, k) = \mathcal{LP}(n_2, k)$$

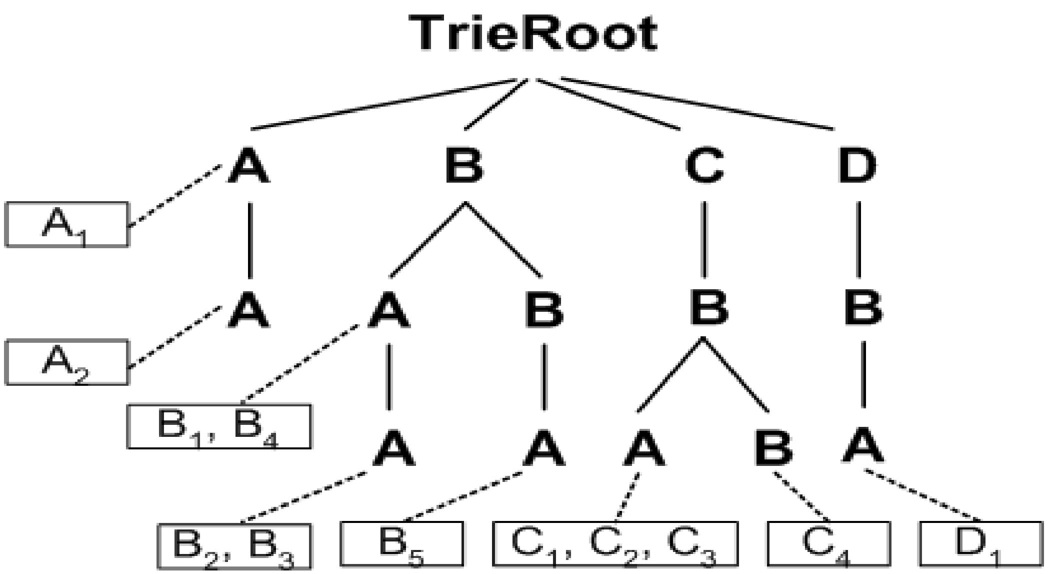
    - Label paths are remembered, the head of the paths are forgotten.
  - Partition blocks are organized in a graph
    - Searches start at different nodes when evaluating $\mathcal{D}[k]$- query on index based on $\mathcal{N}[k]$- partition, $k < k'$.
- Proposal
  - Alternative organization of $\mathcal{N}[k]$- partitions → $\mathcal{N}[k]$- index
  - Partition node pairs → $\mathcal{P}[k]$- index

$\mathcal{N}[k]$-Trie Index

- Keep track of the $\mathcal{N}[k]$- partitions
- Use the reverse label path as key



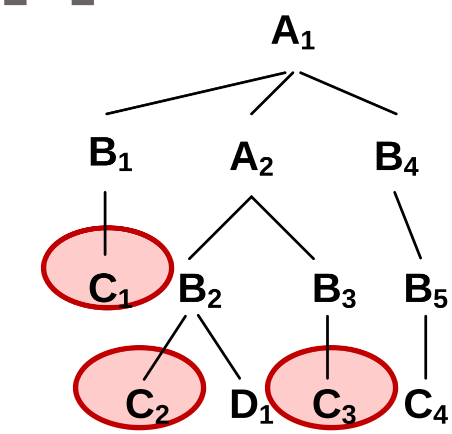
$\mathcal{N}[2]$	(A)	$\{A_1,\}$
	(A,A)	$\{A_2\}$
	(A,B)	$\{B_1, B_4\}$
	(A,A,B)	$\{B_2, B_3,\}$
	(A,B,B)	$\{B_5\}$
	(A,B,C)	$\{C_1, C_2, C_3\}$
	(B,B,C)	$\{C_4\}$
	(A,B,D)	$\{D_1\}$

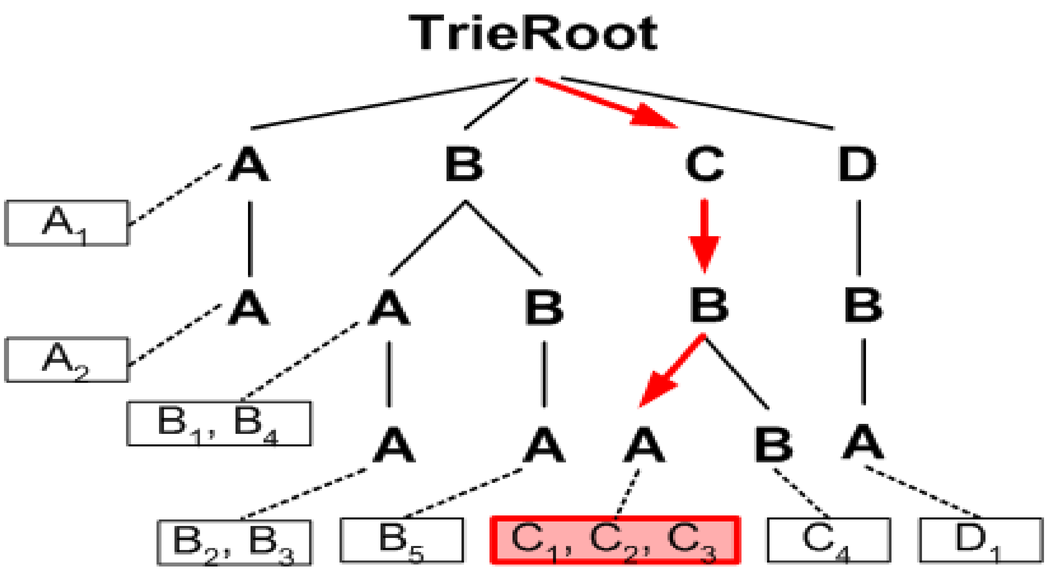


$\mathcal{N}[2]$-Trie

Query Evaluation with $\mathcal{N}[k]$-Trie Index

- Query 1: //A/B/C
- LPS(E,2) = {(A,B,C)}



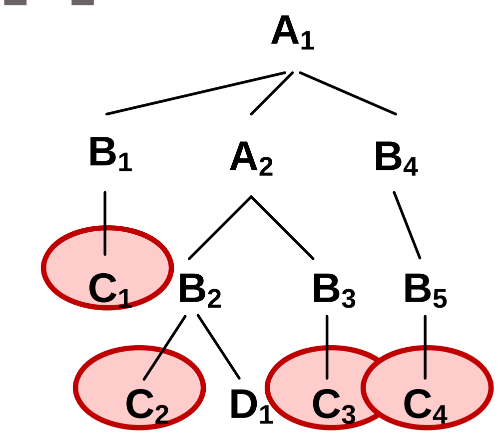


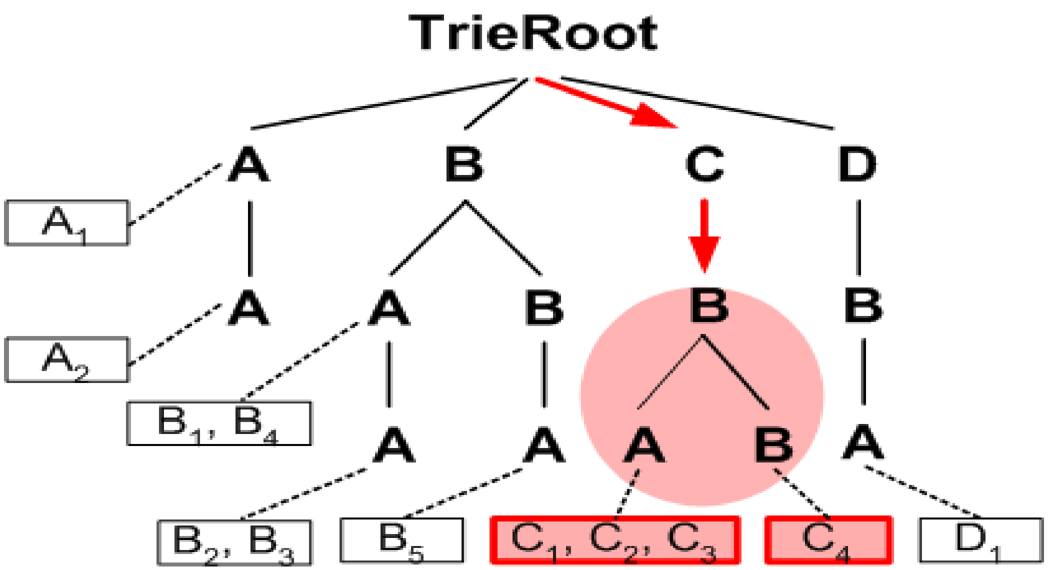
$\mathcal{N}$[2]-Trie

$\mathcal{N}[2]$	(A)	$\{A_1,\}$
	(A,A)	$\{A_2\}$
	(A,B)	$\{B_1, B_4\}$
	(A,A,B)	$\{B_2, B_3,\}$
	(A,B,B)	$\{B_5\}$
	(A,B,C)	$\{C_1, C_2, C_3\}$
	(B,B,C)	$\{C_4\}$
	(A,B,D)	$\{D_1\}$

Query Evaluation with $\mathcal{N}[k]$-Trie Index

- Query 2: //B/C
- LPS(E,2) = {(A,B,C), (B,B,C)}



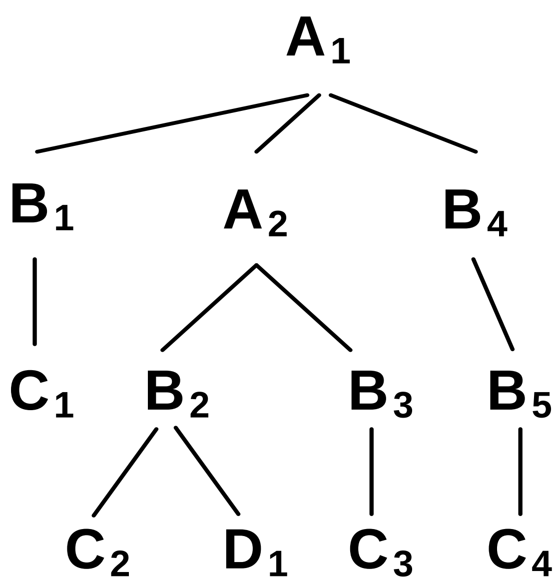


$\mathcal{N}$[2]-Trie

$\mathcal{N}[2]$		
	(A)	$\{A_1,\}$
	(A,A)	$\{A_2\}$
	(A,B)	$\{B_1, B_4\}$
	(A,A,B)	$\{B_2, B_3,\}$
	(A,B,B)	$\{B_5\}$
	(A,B,C)	$\{C_1, C_2, C_3\}$
	(B,B,C)	$\{C_4\}$
	(A,B,D)	$\{D_1\}$

Recall $\mathcal{P}[k]$ Equivalence

- Given an XML document and value k

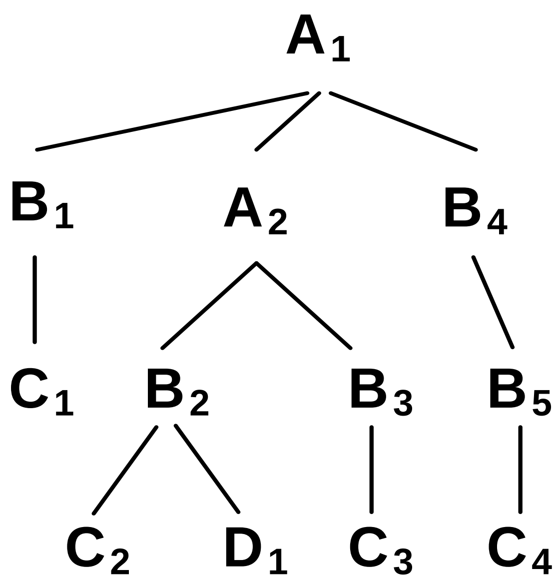


$$(m_1, n_1) \equiv_{\mathcal{P}[k]} (m_2, n_2)$$

$$\Updownarrow$$

$$\mathcal{LP}(m_1, n_1) = \mathcal{LP}(m_2, n_2)$$

Recall $\mathcal{P}[k]$ Partition



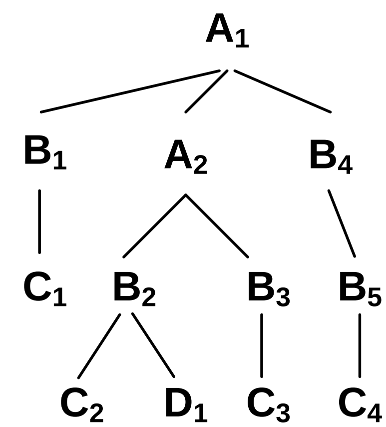
$\mathcal{P}[2]$

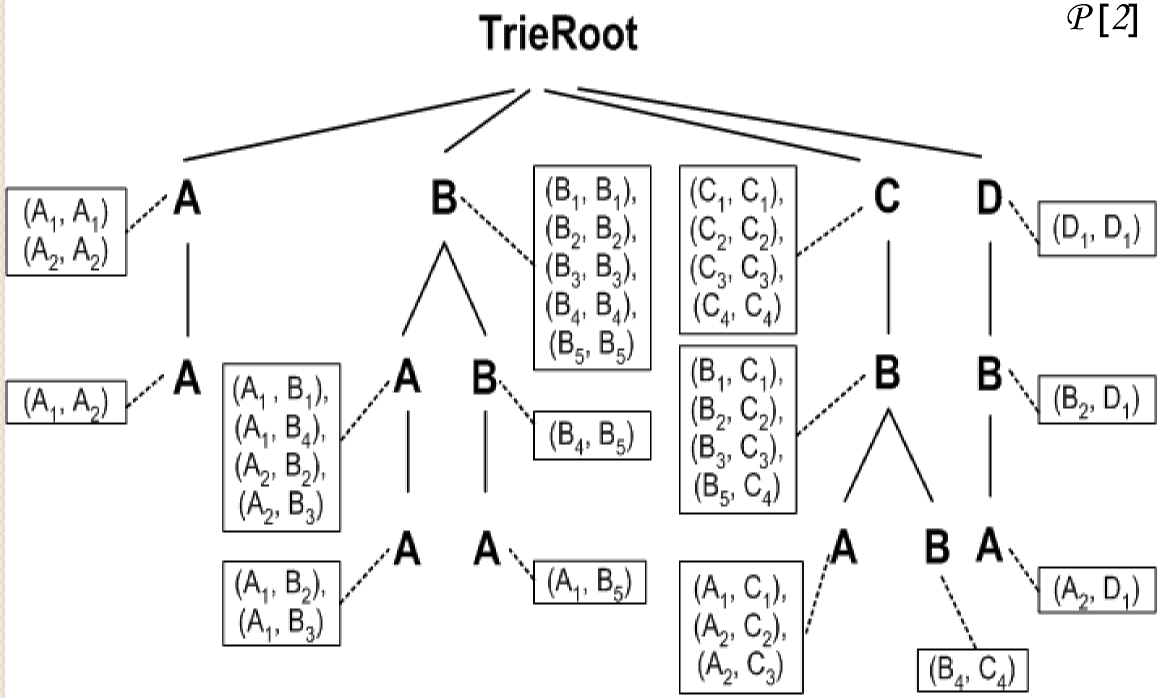
(A)	$\{(A_1,A_1), (A_2,A_2)\}$
(B)	$\{(B_1, B_1), (B_2, B_2), (B_3, B_3), (B_4, B_4), (B_5, B_5)\}$
(C)	$\{(C_1, C_1), (C_2, C_2), (C_3, C_3), (C_4, C_4)\}$
(D)	$\{(D_1, D_1)\}$
(A,A)	$\{(A_1,A_2)\}$
(A,B)	$\{(A_1, B_1), (A_2, B_2), (A_2, B_3), (A_1, B_4)\}$
(B,B)	$\{(B_4, B_5)\}$
(B,C)	$\{(B_1, C_1), (B_2, C_2), (B_3, C_3), (B_5, C_4)\}$
(B,D)	$\{(B_2, D_1)\}$
(A,A,B)	$\{(A_1, B_2), (A_1, B_3)\}$
(A,B,B)	$\{(A_1, B_5)\}$
(A,B,C)	$\{(A_1, C_1), (A_2, C_2), (A_2, C_3)\}$
(A,B,D)	$\{(A_2, D_1)\}$
(B,B,C)	$\{(B_4, C_4)\}$

$$\mathcal{P}[2][(A,B,C)] = \{(A_1, C_1), (A_2, C_2), (A_2, C_3)\}$$

$\mathcal{P}[k]$-Trie Index

- Keep track of the $\mathcal{P}[k]$- partitions
- Use the reverse label path as key





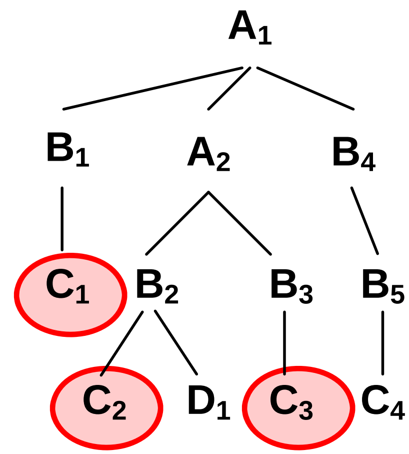
$\mathcal{P}[2]$-Trie

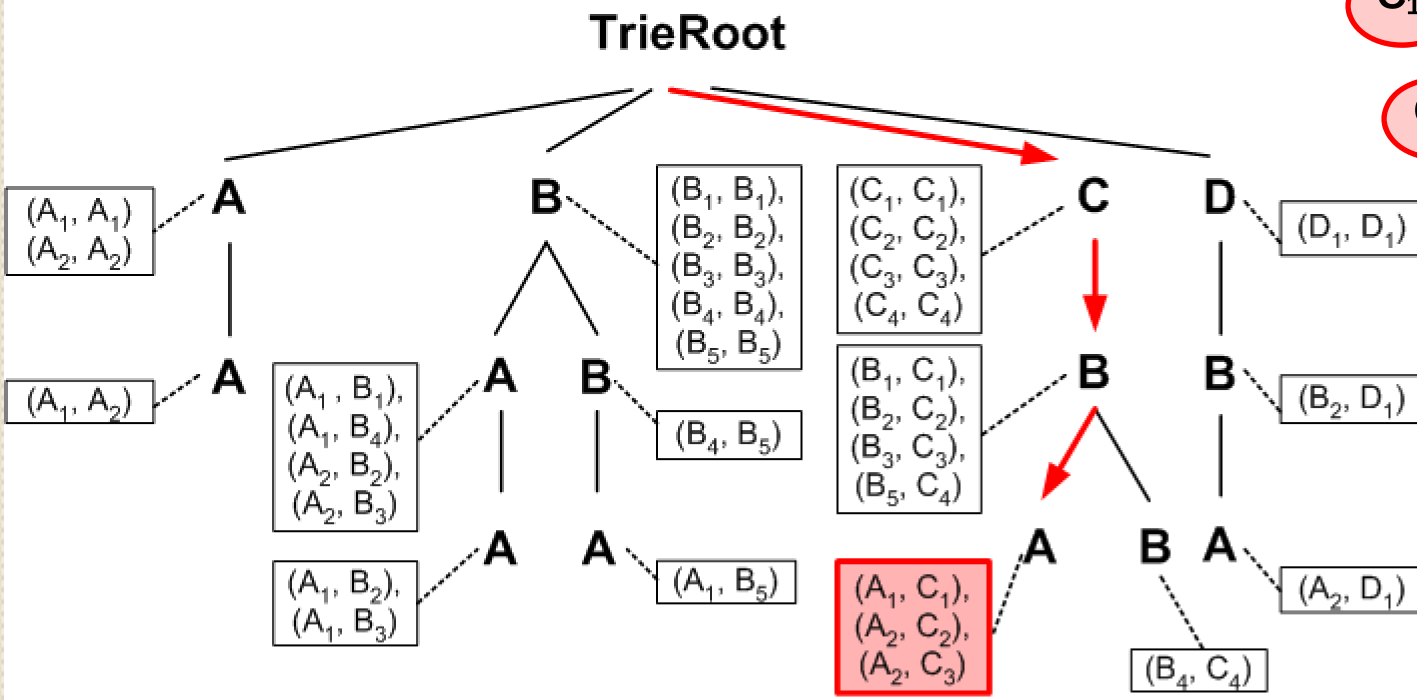
$\mathcal{P}[2]$

(A)	$\{(A_1,A_1), (A_2,A_2)\}$
(B)	$\{(B_1,B_1), (B_2,B_2), (B_3,B_3), (B_4,B_4), (B_5,B_5)\}$
(C)	$\{(C_1,C_1), (C_2,C_2), (C_3,C_3), (C_4,C_4)\}$
(D)	$\{(D_1,D_1)\}$
(A,A)	$\{(A_1,A_2)\}$
(A,B)	$\{(A_1,B_1), (A_2,B_2), (A_2,B_3), (A_1,B_4)\}$
(B,B)	$\{(B_4,B_5)\}$
(B,C)	$\{(B_1,C_1), (B_2,C_2), (B_3,C_3), (B_5,C_4)\}$
(B,D)	$\{(B_2,D_1)\}$
(A,A,B)	$\{(A_1,B_2), (A_1,B_3)\}$
(A,B,B)	$\{(A_1,B_5)\}$
(A,B,C)	$\{(A_1,C_1), (A_2,C_2), (A_2,C_3)\}$
(A,B,D)	$\{(A_2,D_1)\}$
(B,B,C)	$\{(B_4,C_4)\}$

Query Evaluation with $\mathcal{P}[k]$-Trie Index

- Query 1: //A/B/C

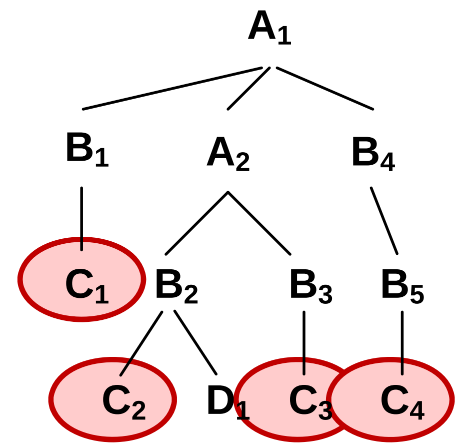


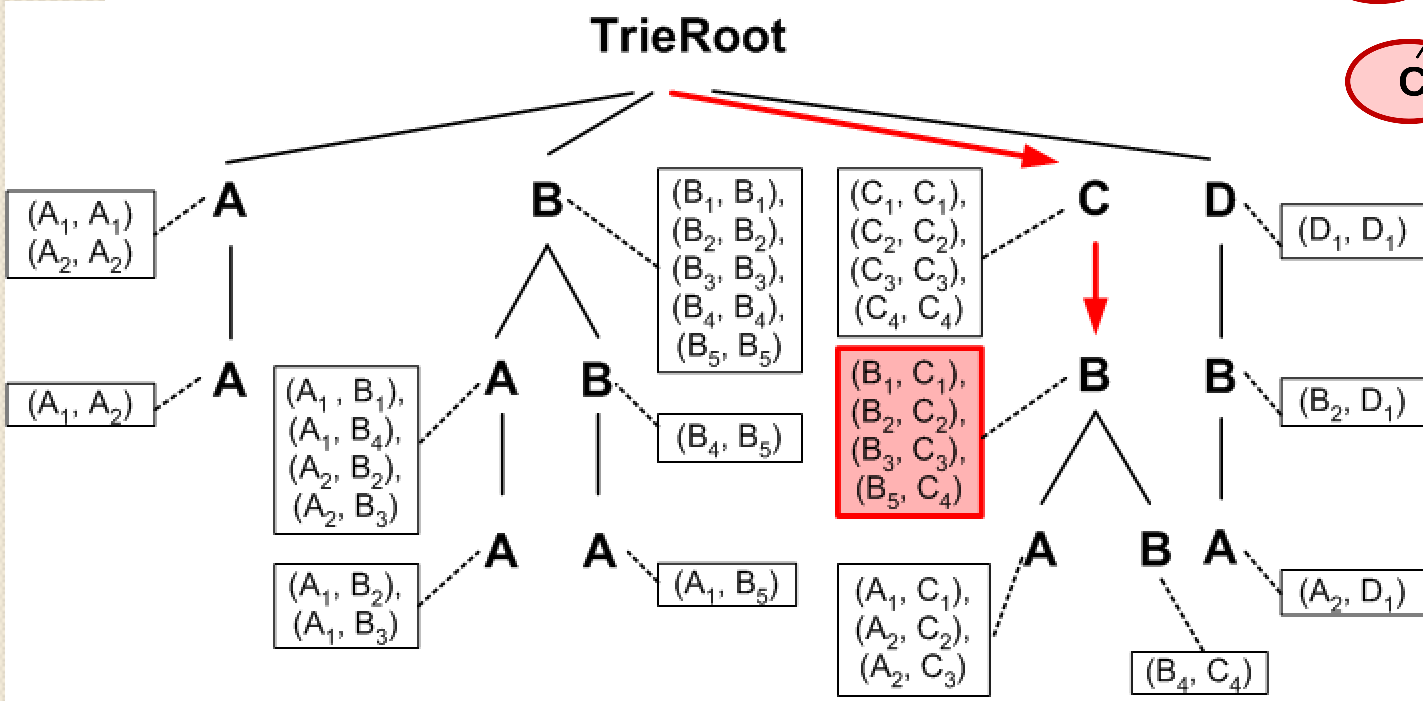


$\mathcal{P}$[2]-Trie

Query Evaluation with $\mathcal{P}[k]$-Trie Index

- Query 2: //B/C



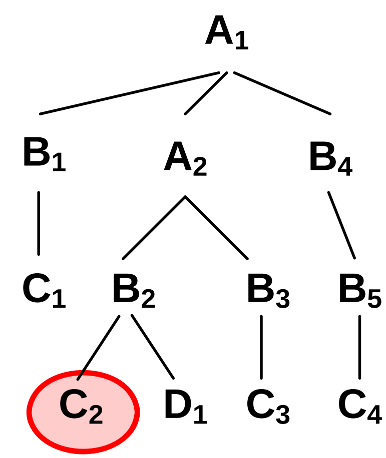


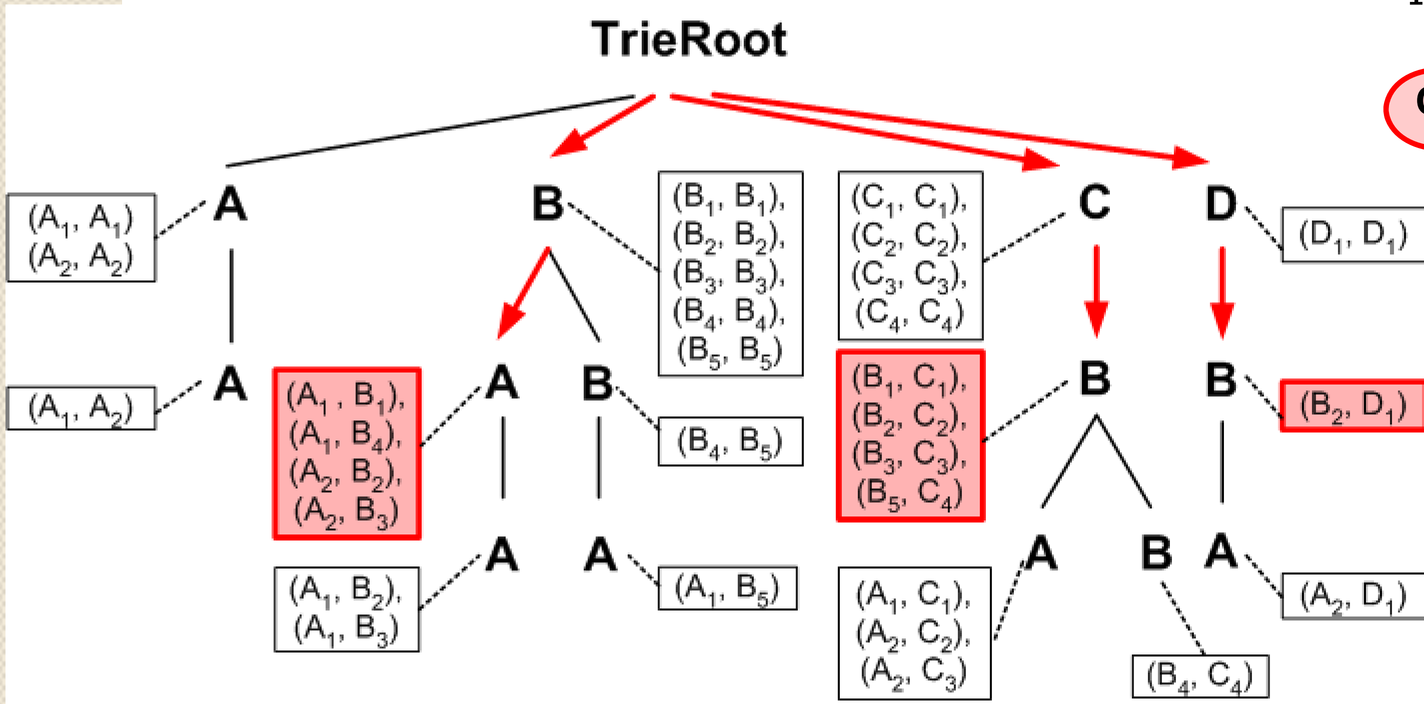
$\mathcal{P}$[2]-Trie

Query Evaluation with $\mathcal{P}[k]$-Trie Index

- Query 3: //A/B[./D]/C

$$E_1(D) \bowtie \pi(E_2(D)) \bowtie E_3(D) \quad \begin{aligned} E_1 &= A \circ\!\downarrow\!\circ B \\ E_2 &= B \circ\!\downarrow\!\circ D \\ E_3 &= B \circ\!\downarrow\!\circ C \end{aligned}$$



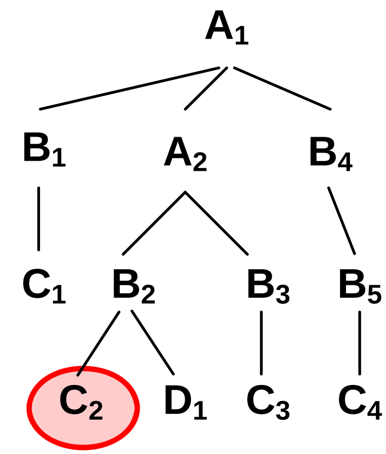


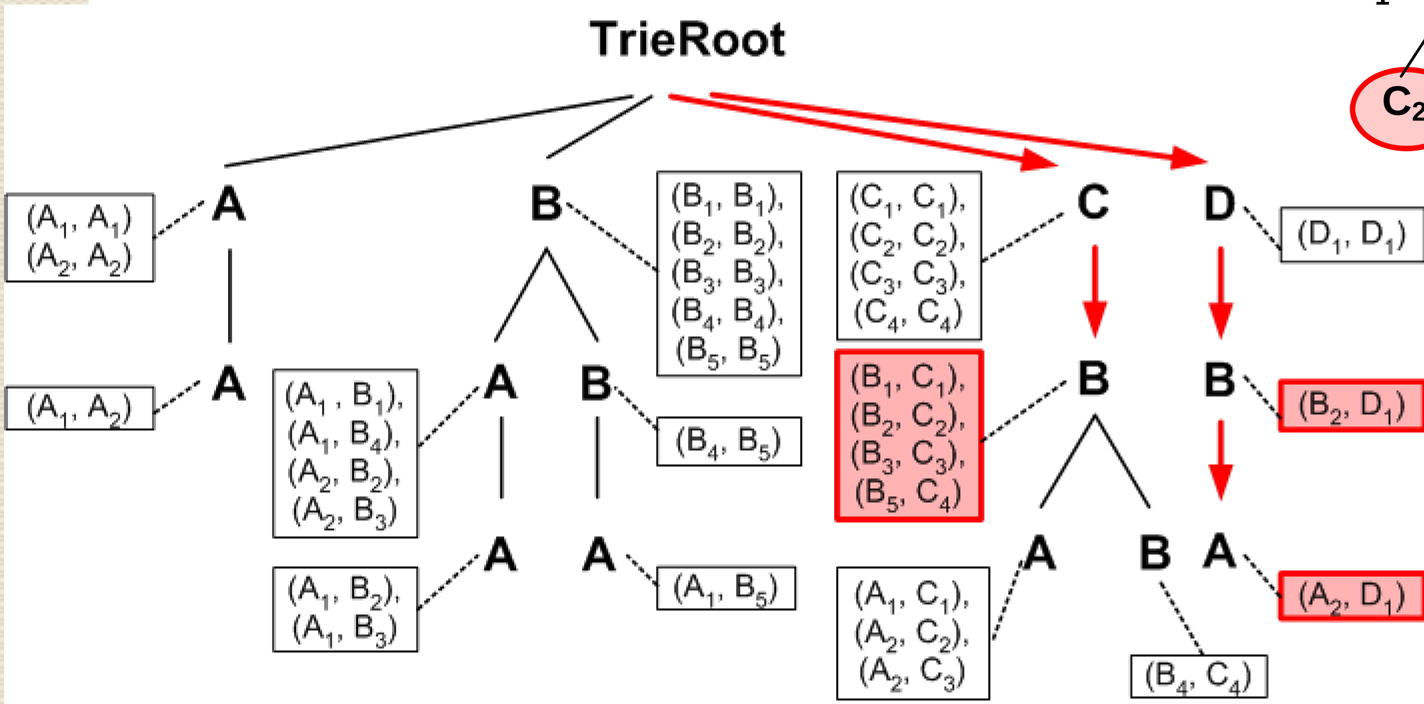
$\mathcal{P}$[2]-Trie

Query Evaluation with $\mathcal{P}[k]$-Trie Index

- Query 3: //A/B[./D]/C

$$E_1(D) \bowtie (E_2(D))^{-1} \bowtie E_3(D) \qquad \begin{aligned} E_1 &= A \circ\!\downarrow\!\circ B \circ\!\downarrow\!\circ D \\ E_2 &= B \circ\!\downarrow\!\circ D \\ E_3 &= B \circ\!\downarrow\!\circ C \end{aligned}$$





$\mathcal{P}$[2]-Trie

Other Applications

- Workload-aware Trie indices for XML
  - Wu, Brenes, Yi. Workload-aware Trie Indexes for XML. In CIKM 2009.
- Algebra-based index comparison
  - Wu, Brenes, Totade, Damani, Joshua, Salim. ASIC: Algebra-based Structural Indices Comparison. In CIKM, 2009
- XML Benchmarking
  - Wu, Lele, Aroskar, Chinnusamy. XQGen - An Algebra-based XPath Query Generator For Micro-Benchmarking. In CIKM, 2009.

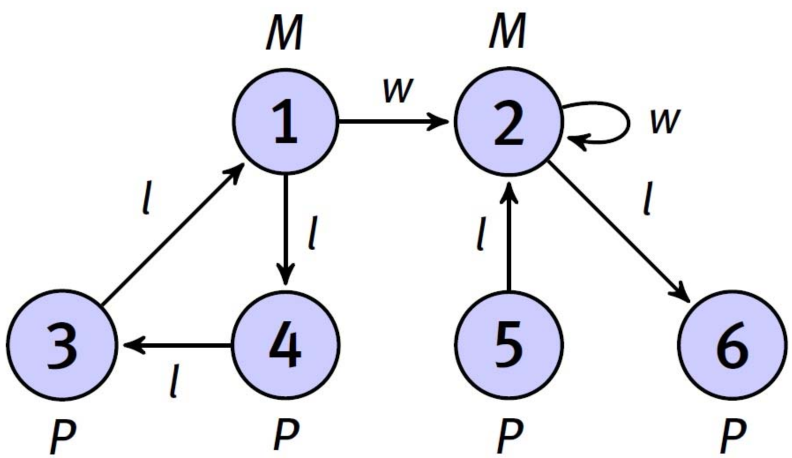
Tutorial Outline

- Graph Data and Search
- The Theory
- **The Engineering**
  - Indexing Tree Structured Data
  - **Bi-Similarity on Graphs**
- Summary & Future Directions

Recall k-bisimilarity on Graphs

- Let k be a nonnegative integer and $G = \langle N,E,\lambda_N,\lambda_E \rangle$ be a graph. Node u, v $\in$ N are called k-bimsiminar (u $\approx^k$ v), iff the following holds
  - $\lambda_N(u) = \lambda_N(v)$
  - If k>0, then, $\forall u' \in N\ [(u,u') \in E \Rightarrow \exists v' \in N\ [(v,v') \in E,\ u' \approx^{k-1} v'$ and $\lambda_E(u,u') = \lambda_E(v,v')]]$
  - If k>0, then, $\forall v' \in N\ [(v,v') \in E \Rightarrow \exists u' \in N\ [(u,u') \in E,\ v' \approx^{k-1} u'$ and $\lambda_E(v,v') = \lambda_E(u,u')]]$

K-Bisimilarity and k-Partition



$n_1 \approx^0 n_2$ √

$n_1 \approx^1 n_2$ √

$n_1 \approx^2 n_2$ ×

k=0	$\{n_1, n_2\}$
	$\{n_3, n_4, n_5, n_6\}$
k=1	$\{n_1, n_2\}$
	$\{n_3, n_5\}$
	$\{n_4\}$
	$\{n_6\}$
k=2	$\{n_1\}$
	$\{n_2\}$
	$\{n_3, n_5\}$
	$\{n_4\}$
	$\{n_6\}$

k-Partition Signature

Signature:

$\text{sig}_k(u) = (\text{pID}_0(u), L)$

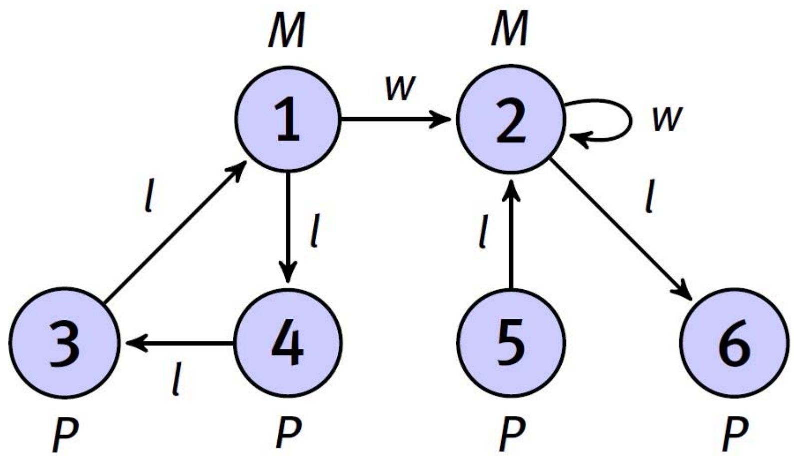
$$L = \begin{cases} \emptyset & if\ k = 0 \\ \{(\lambda_E(u,u'), pID_{k-1}(u')) \mid (u,u') \in E\} & if\ k > 0 \end{cases}$$

Proposition:

$\forall u,v \in N,$

$\text{pID}_k(u)=\text{pID}_k(v) \Leftrightarrow \text{sig}_k(u) = \text{sig}_k(v)$

k-Partition Signature – Example



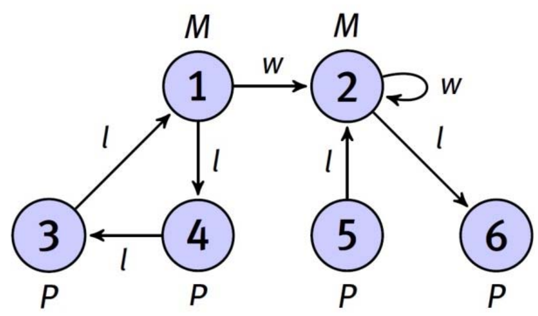
nID	$pID_0(nID)$	$sig_1(nID)$	$pID_1(nID)$	$sig_2(nID)$	$pID_2(nID)$
1	1	1, {(w,1),(l,2)}	3	1, {(w,3),(l,5)}	7
2	1	1, {(w,1),(l,2)}	3	1, {(w,3),(l,6)}	8
3	2	2, {(l,1)}	4	2, {(l,3)}	9
4	2	2, {(l,2)}	5	2, {(l,4)}	10
5	2	2, {(l,1)}	4	2, {(l,3)}	9
6	2	2, {}	6	2, {}	11

k-Partition Construction

1. Sort N_t on nID
2. Sort E_t on tID
3. Merge join N_t and E_t on nID and tID, fill in $E_t.pID_{old\_tid}$
4. Sort E_t on sID, project on (sID, eLabel, pID_{old_tid}), remove duplicates, get F
5. Merge join N_t and F on nID and sID, get signature
6. Assigning new pID based on signature.

k-Partition Construction – Example

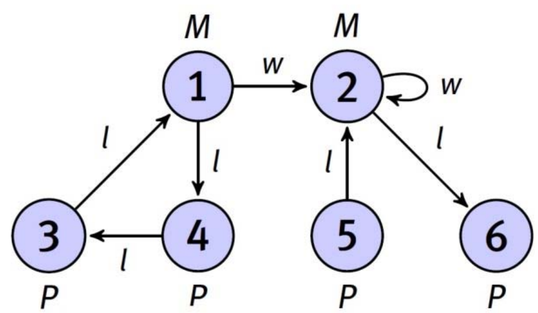
1. Sort N_t on nID
2. Sort E_t on tID
3. Merge join N_t and E_t on nID and tID, fill in $E_t.pID_{old\_tid}$
4. Sort E_t on sID, project on (sID, eLabel, pID_{old_tid}), remove duplicates, get F
5. Merge join N_t and F on nID and sID, get signature
6. Assigning new pID based on signature.



nID	nLabel	pID_0(nID)	sig_1(nID)	pID_1(nID)
1	M	1		
2	M	1		
3	P	2		
4	P	2		
5	P	2		
6	P	2		

k-Partition Construction – Example

1. Sort N_t on nID
2. Sort E_t on tID
3. Merge join N_t and E_t on nID and tID, fill in E_t.pID_{old_tid}
4. Sort E_t on sID, project on (sID, eLabel, pID_{old_tid}), remove duplicates, get F
5. Merge join N_t and F on nID and sID, get signature
6. Assigning new pID based on signature.

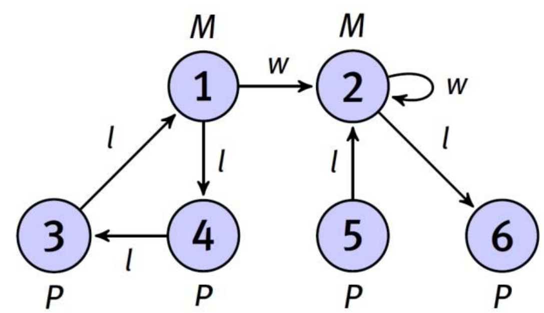


nID	nLabel	pID_0(nID)	sig_I(nID)	pID_I(nID)
1	M	1		
2	M	1		
3	P	2		
4	P	2		
5	P	2		
6	P	2		

sID	eLabel	tID	pID_{old_tID}
3	l	1	
1	w	2	
2	w	2	
5	l	2	
4	l	3	
1	l	4	
2	l	6	

k-Partition Construction – Example

1. Sort N_t on nID
2. Sort E_t on tID
3. Merge join N_t and E_t on nID and tID, fill in $E_t.pID_{old\_tid}$
4. Sort E_t on sID, project on (sID, eLabel, pID_{old_tid}), remove duplicates, get F
5. Merge join N_t and F on nID and sID, get signature
6. Assigning new pID based on signature.

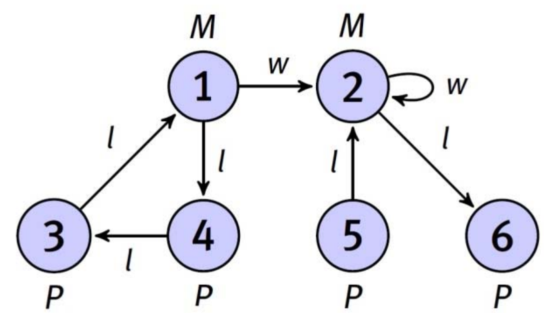


nID	nLabel	pID_0(nID)	sig_I(nID)	pID_I(nID)
1	M	1		
2	M	1		
3	P	2		
4	P	2		
5	P	2		
6	P	2		

sID	eLabel	tID	pID_{old_tID}
3	l	1	1
1	w	2	1
2	w	2	1
5	l	2	1
4	l	3	2
1	l	4	2
2	l	6	2

k-Partition Construction – Example

1. Sort N_t on nID
2. Sort E_t on tID
3. Merge join N_t and E_t on nID and tID, fill in $E_t.pID_{old\_tid}$
4. Sort E_t on sID, project on (sID, eLabel, pID_{old_tid}), remove duplicates, get F
5. Merge join N_t and F on nID and sID, get signature
6. Assigning new pID based on signature.

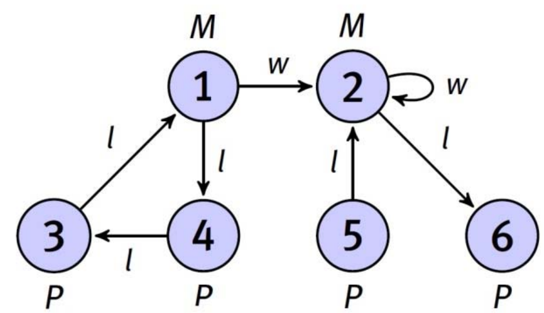


nID	nLabel	pID_0(nID)	sig_1(nID)	pID_1(nID)
1	M	1		
2	M	1		
3	P	2		
4	P	2		
5	P	2		
6	P	2		

sID	eLabel	pID_{old_tID}
1	w	1
1	l	2
2	w	1
2	l	2
3	l	1
4	l	2
5	l	1

k-Partition Construction – Example

1. Sort N_t on nID
2. Sort E_t on tID
3. Merge join N_t and E_t on nID and tID, fill in $E_t.pID_{old\_tid}$
4. Sort E_t on sID, project on (sID, eLabel, pID_{old_tid}), remove duplicates, get F
5. Merge join N_t and F on nID and sID, get signature
6. Assigning new pID based on signature.

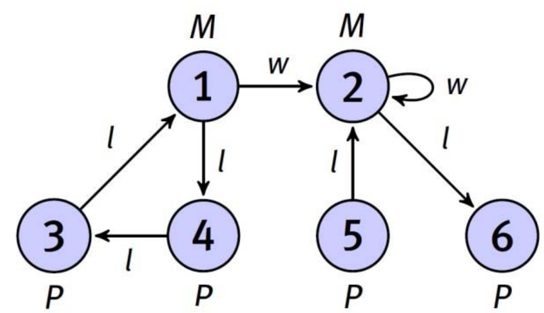


nID	nLabel	pID_0(nID)	sig_1(nID)	pID_1(nID)
1	M	1	1, {(w,1),(l,2)}	
2	M	1	1, {(w,1),(l,2)}	
3	P	2	2, {(l,1)}	
4	P	2	2, {(l,2)}	
5	P	2	2, {(l,1)}	
6	P	2	2, {}	

sID	eLabel	pID_{old_tID}
1	w	1
1	l	2
2	w	1
2	l	2
3	l	1
4	l	2
5	l	1

k-Partition Construction – Example

1. Sort N_t on nID
2. Sort E_t on tID
3. Merge join N_t and E_t on nID and tID, fill in $E_t.pID_{old\_tid}$
4. Sort E_t on sID, project on (sID, eLabel, pID_{old_tid}), remove duplicates, get F
5. Merge join N_t and F on nID and sID, get signature
6. Assigning new pID based on signature.



nID	nLabel	$pID_0(nID)$	$sig_l(nID)$	$pID_l(nID)$
1	M	1	1, {(w,1),(l,2)}	3
2	M	1	1, {(w,1),(l,2)}	3
3	P	2	2, {(l,1)}	4
4	P	2	2, {(l,2)}	5
5	P	2	2, {(l,1)}	4
6	P	2	2, {}	6

sID	eLabel	pID_{old_tID}
1	w	1
1	l	2
2	w	1
2	l	2
3	l	1
4	l	2
5	l	1

k-Partition Construction – Complexity

1. Sort N_t on nID
2. Sort E_t on tID
3. Merge join N_t and E_t on nID and tID, fill in $E_t.pID_{old\_tid}$
4. Sort E_t on sID, project on (sID, eLabel, pID_{old_tid}), remove duplicates, get F
5. Merge join N_t and F on nID and sID, get signature
6. Assigning new pID based on signature.

I/O complexity

$$O(k \cdot sort(|E_t|) + k \cdot scan(|N_t|) + sort(|N_t|))$$

Space Complexity

$$O(|N_t| + |E_t|)$$

Other Applications and Results

- Triple indexing
  - Picalausa, Luo, Fletcher, Hidders, Vansummeren. A structural approach to indexing triples. ESWC 2012.
- DAG and tree indexing
  - Hellings, Fletcher, Haverkort. Efficient external-memory bisimulation on DAGs. SIGMOD 2012.

Tutorial Outline

- Graph Data and Search
- The Theory
- The Engineering
- **Summary and Future Directions**

Summary

In this tutorial we have

- motivated the study of graphs and search languages
- introduced a general methodology for studying the design and implementation of graph languages
- demonstrated the application of this methodology to Tarski's RA
- shown the practical impact of the methodology in graph indexing for efficient query processing

Future Directions

- Open problems in theory
  - Structural characterizations for uncertain or imprecise data
  - Relationships of instance expressivity characterizations to work in logics
  - …
- Open problems in practice
  - Design of index data structures for path languages
  - Design and study of practical applications of the methodology for more flexible types of search and data (e.g., keyword search and similarity search)
  - …

References

- Yuqing Wu, Dirk Van Gucht, Marc Gyssens and Jan Paredaens: A Study of a Positive Fragment of Path Queries: Expressiveness, Normal Form and Minimization. In the *Computer Journal* 54(7): 1091-1118, 2011.
- George H. L. Fletcher, Marc Gyssens, Dirk Leinders, Jan Van den Bussche, Dirk Van Gucht, Stijn Vansummeren and Yuqing Wu: Relative expressive power of navigational querying on graphs. In *the 14th International Conference on Database Theory*. March, 2011.
- Yuqing Wu, Sofia Brenes, and Hyungdae Yi. Workload-aware Trie Indexes for XML. In the *18th ACM Conference on Information and Knowledge Management.* November, 2009.
- George H. L. Fletcher, Dirk Van Gucht, Yuqing Wu, Marc Gyssens, Sofia Brenes, and Jan Paredaens. A Methodology for Coupling Fragments of XPath with Structural Indexes for XML Documents. In *Information Systems*, 2009.
- Sofia Brenes, Yuqing Wu, Dirk Van Gucht, and Pablo Santa Cruz. Trie Indexes for Efficient XML Query Evaluation. In the *11th International Workshop on the Web and Databases.* June, 2008
- Marc Gyssens, Jan Paredaens, Dirk Van Gucht, George H. L. Fletcher: Structural characterizations of the semantics of XPath as navigation tool on a document. PODS 2006: 318-327
- Yongming Luo, G. H. L. Fletcher, J. Hidders, Y. Wu, and P. De Bra. I/O-efficient algorithms for localized bisimulation partition construction and maintenance on massive graphs. CoRR, abs/1210.0748, 2012.
- Y. Luo, Y. de Lange, G. H. L. Fletcher, P. De Bra, J. Hidders, and Y. Wu. Bisimulation reduction of big graphs on MapReduce. To appear in BNCOD, Oxford, UK, 2013.
- G. H. L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. The impact of transitive closure on the boolean expressiveness of navigational query languages on graphs. In FoIKS, pages 124-143, Kiel, Germany, 2012.

Collaborators

- Colleagues:
  - Marc Gyssens
  - Jan Hidders
  - Jan Paredaens
  - Jan Van den Bussche
  - Dirk Van Gucht
  - Stijn Vansummere
  - Dirk Leinder
  - Paul De Bra
- Supported by FWO

- Ph.D. Students:
  - Indiana Univ: Sofia Brenes, Vahid Jalali, Yifan Pan, Mo Zhou, …
  - Eindhoven: Yongming Luo
- MS Students:
  - Indiana Univ: Hyungdae Yi, Pablo Santa Cruz, Namrata Lele, Rashmi Aroskar, Sharanya Chinnusamy, …
  - Eindhoven: Yannick de Lange

Thanks!

Questions, Please.