# Maintenance of Minimal Bisimulation of Cyclic Graphs

Jintian DENG

11th Postgraduate Research Symposium

March 14, 2010

## Outline

1 Introduction
2 Preliminaries
- Data Graph
- Index Graph
- Bisimilarity
- Paige-Tarjan Algorithm

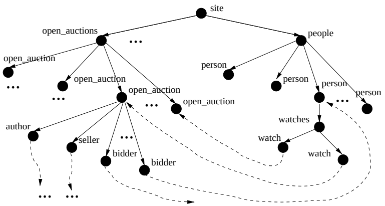
3 Related Work
- Update using P.T. Algorithm
- Update using Split/Merge Algorithm

4 Our Solution
- SCC-Based Approach
- SCC Bisimulation
- Construction Algorithm Outline
- Update Algorithm Outline

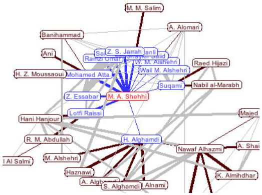
5 End

## Graph Data

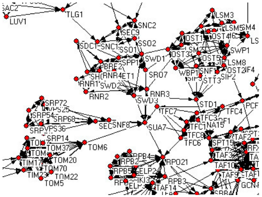
What's *graph data*? XML tree, social network, protein, Internet structure etc. In practice, data graphs are often *cyclic*.



XML tree



social network



protein

Internet structure

## Graph Query

What we want to know in a graph-structured data?

- Is vertex $v_1$ reachable from vertex $v_2$? *Reachability Query*
- Is a graph patten exits in a large graph database? *Graph matching Query*
- Given a path patten, return all the nodes that can be reached via a specific starting nodes? *Path Query*
- ......

No data structures are general enough for answering all these different queries efficiently.

We focus on solving *Path Query* over *Cyclic Graph*.

## Problem?

- What's the data structure supposed to be?
- How to construct such data structure?
- How to maintain the data structure properly when the data graph is being updated(edge insertion/deletion, subgraph addition/deletion etc)?
- What's the efficiency of such data structure, e.g. construction cost, query cost, update cost, memory cost?

## Outline

1 Introduction

2 Preliminaries
- Data Graph
- Index Graph
- Bisimilarity
- Paige-Tarjan Algorithm

3 Related Work
- Update using P.T. Algorithm
- Update using Split/Merge Algorithm

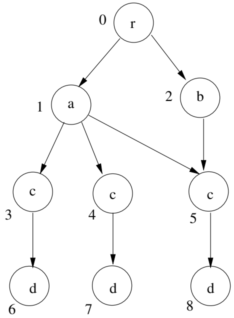
4 Our Solution
- SCC-Based Approach
- SCC Bisimulation
- Construction Algorithm Outline
- Update Algorithm Outline

5 End

## Data Graph

Directed, labeled graph $G = (V, E, r, \Sigma, \rho)$:

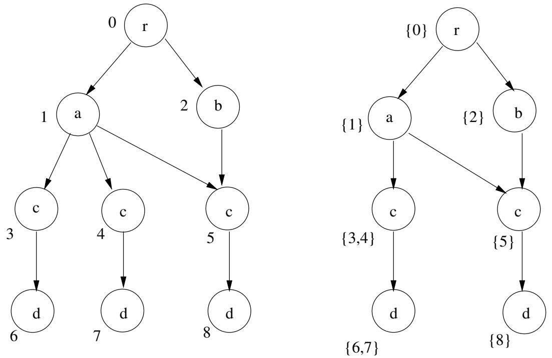
- $V$ Vertices set
- $E$ Edges set
- $r$ Root vertex
- $\Sigma$ Finite set of labels
- $\rho$ A function maps a vertex to a label: $V \rightarrow \Sigma$



DataGraph

## Index Graph

A index graph(structural index) takes the form of another labeled, directed graph.



DataGraph IndexGraph

6,7 are returned as the result of path Query *r/a/c/d*.

## Index Graph Construction

Following is the general procedure to build an index graph:

- Partition the data vertices into classes according to some equivalence relation;
- Make an index vertex for each equivalence class, with all data vertices in this class being its extent;
- Add an index edge from index vertex $I$ to index vertex $J$ if there is an edge from some data vertices in the extent of $I$ to some data vertices in the extent of $J$.

Which equivalence relation? *Bisimilarity*

## Bisimilarity

There are two different ways to define bisimilarity, one is based on *upward bisimulation*, another is based on *stability*.

Upward Bisimulation
if $v_1 \sim v_2$ then

- If $v_1$ is root vertex, then $v_2$ is root vertex and vise versa.
- $\rho(v_1) = \rho(v_2)$, they have the same label.
- For each edge $(v_1', v_1) \in E$ , there is an edge $(v_2', v_2) \in E$ such that $v_1' \sim v_2'$ and vise versa.

From this definition, it seems very easy to construct the index graph:first try to bisimilar the children of the root, and then following a top down manner to bisimilar other vertices. Merging based algorithm

## Bisimilarity

Stability

- $Succ(u) = \{v|(u,v) \in E\}$.
- $Succ(I) = \cup_{u \in I} Succ(u)$.
- An index vertex $I$ is stable with respect to $J$ if either $I \subseteq Succ(J)$ or $I \cap Succ(J) = \theta$.
- If $I$ is not stable w.r.t. $J$, we can make it stable by splitting $I$ into $I \cap Succ(J)$ and $I - I \cap Succ(J)$.
- After stabilization, if $v_1$ and $v_2$ are in the same index vertex, then $v_1 \sim v_2$.

We first put all the vertices into the same index node(the big cake), and then we cut the 'cake' into small part according to stablity. Partition refinement based algorithm

## Bisimilarity

The definition of *stability* is quite complex while *upward bisimulation* is very straightforward. We can build the index graph based on either definition.

Criteria of a good construction algorithm.

- minimum index size(number of index vertices)
- small construction time

What we know:

| Algorithm | Index Size | Construction Time | Based on |
|---|---|---|---|
| Paige-Tarjan Algo. | minimum | $O(n \log m)$[1] | Stability |
| ? | minimum | ? | Upward bisim. |
| ... | not-minimum | ... | ... |

[1] n is the number of vertices and m is the number of edges

## Paige-Tarjan Algorithm

Paige-Tarjan Algorithm is based on the definition of *stability*.
It can construct the minimum index graph in $O(n \log m)$[2].
It's based on a technique called *Partition refinement*.

Partition:
A partition $p$ divide the data vertices set into different disjoint classes.
*$v_1$ and $v_2$ are in the same class in partition $p \rightleftharpoons v_1 \sim v_2$ in partition $p$*

Refinement:
Partition $p_1$ is a refinement of partition $p_2$ iff. $v_1 \sim v_2$ in $p_1$ then $v_1 \sim v_2$ in $p_2$.

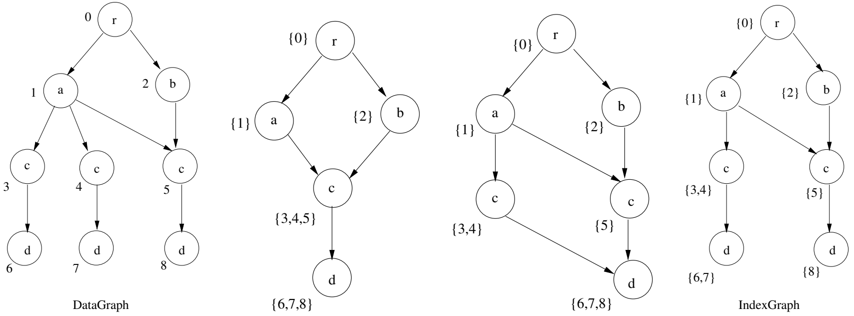
[2] n is the number of vertices and m is the number of edges

## Paige-Tarjan Algorithm Cont.

*Outline of P.T. Algorithm*

- Initialize the index vertices by partition the data vertices by label
- Stabilize every unstable index vertex by refinement(splitting)
- Add edges

P.T. essentially turn an unstable partition into stable by splitting index vertices.



## Outline

1 Introduction

2 Preliminaries
- Data Graph
- Index Graph
- Bisimilarity
- Paige-Tarjan Algorithm

3 Related Work
- Update using P.T. Algorithm
- Update using Split/Merge Algorithm

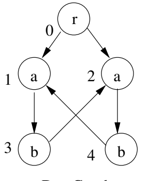
4 Our Solution
- SCC-Based Approach
- SCC Bisimulation
- Construction Algorithm Outline
- Update Algorithm Outline

5 End

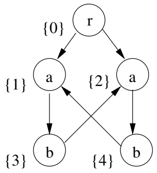
## Update

- P.T. Algorithm can also be directly applied to update, which requires reconstruction the index graph after every insertion/deletion.
- There are two incremental maintenance algorithms which can at most maintain a minimal index graph over directed acyclic graph.
- Difference between minimal and minimum?

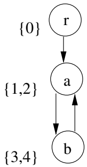
  *In a minimal index graph, it's impossible to merge two index nodes into one and keep the partition stable.*



DataGraph



minimal(but not minimum)



minimum

## (1)Propagate Algorithm [3]

Main idea:

After edge is inserted, the original partition is no longer stable. Run P.T. algorithm over this partition will make it stable again.

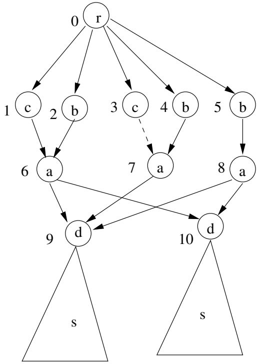
Disadvantage:

This method can't guarantee the minimal index size even in a tree. After the partition is stable by running P.T. algorithm, there are some index vertices can be merged.

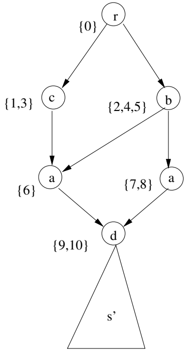
[3] R. Kaushik, P. Bohannon, J. F. Naughton, and P. Shenoy. Updates for structure indexes. In *VLDB 2002*, pages 239–250, 2002.

## (1)Propagate Algorithm Cont.

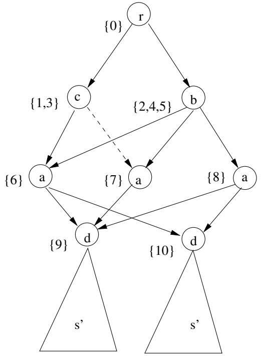
Handling edge insertion using Propagate algorithm.



DataGraph



IndexGraph



IndexGraph (after edge insertion)

## (2)Split/Merge Algorithm [4]

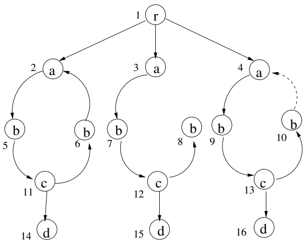
Main idea:
Split the index vertices to make the partition stable(Same as running P.T. algorithm). After that, merge nearby index vertices without violating any constraint, one pair at a time, until no more merges can be made.

Disadvantage:
This method can only guarantee minimum index size over DAG(Directed Acyclic Graph). It miss some bisimulation between index vertices in a cyclic graph.

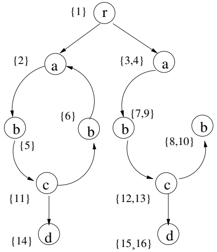
[4] Y. Ke, H. Hao, S. Ioana, and Y. Jun. Incremental maintenance of xml structural indexes. In *SIGMOD*, pages 491–502, 2004.

## (2)Split/Merge Cont.

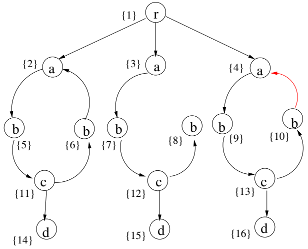
Handling edge insertion using Split/Merge algorithm.



DataGraph



IndexGraph



IndexGraph(after edge insertion)

## Outline

1 Introduction

2 Preliminaries
- Data Graph
- Index Graph
- Bisimilarity
- Paige-Tarjan Algorithm

3 Related Work
- Update using P.T. Algorithm
- Update using Split/Merge Algorithm

4 Our Solution
- SCC-Based Approach
- SCC Bisimulation
- Construction Algorithm Outline
- Update Algorithm Outline

5 End

## Our Solution: SCC-based Approach

Key observation:

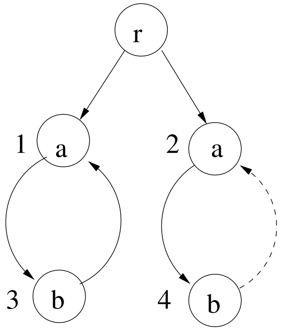
- The key concept in cyclic graph is Strong Connected Component(SCC). As the graph being updated, SCCs will be generated or destroyed. They can be bisimilared to each other too.

Merging, in previous approach, can only happen to two index vertices iff. their parent index vertices is bisimilar.

When comes to bisimilar two index vertices inside two different but bisimilar SCC, this will introduce cyclic-proof and end up un-bisimilar.

## Cyclic-Proof

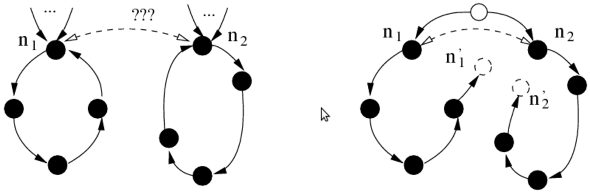
We want to bisimilar 1 and 2 $\longrightarrow$ we must first bisimilar 3 and 4 $\longrightarrow$ we must first bisimilar 1 and 2 ...



*Solution*: Consider one pair of SCCs instead of one pair of vertices. Bisimilar the whole SCCs first and then we can get the bisimulation between vertices.

## SCC Bisimulation

We take the approach of breaking cycle to bisimilar two SCCs. It's a recursive procedure.



## SCC Bisimulation Cont.

SCC bisimulation is a costly procedure(recursive) and many SCCs are not bisimilar to each other in the real data set. In order to avoid unnecessary bisimulation, we introduce some features.

- Label-based feature
- Edge-based feature
- Path-based feature
- Circuit-based feature

All above features can be constructed and maintained efficiently.

## Construction Algorithm Outline

We propose a new construction algorithm based on SCC bisimulation which can achieve the *minimal* index graph.

- Scan the data graph once and reduce each SCC to a super vertex
- The graph is acyclic now, take a top-down approach to do bisimulation(based on *upward bisimulation*)
- Adapt the SCC bisimulation procedure when bisimilaring two super vertices

## Update Algorithm Outline

We also propose a new update algorithm which can maintain the *minimal* index graph over cyclic data graph.

The key point in our approach is maintaining the *live* SCCs all the time. We also follow the split/merge pattern, but adapt our SCC bisimulation algorithm when trying to bisimilar two super vertices.

- Split– make the index graph stable
- Scan– maintain the *live* SCCs
- Merge– two case:
  - merging normal data vertices is easy
  - we 'merge' two SCCs if they are bisimilar

## Outline

1 Introduction
2 Preliminaries
- Data Graph
- Index Graph
- Bisimilarity
- Paige-Tarjan Algorithm

3 Related Work
- Update using P.T. Algorithm
- Update using Split/Merge Algorithm

4 Our Solution
- SCC-Based Approach
- SCC Bisimulation
- Construction Algorithm Outline
- Update Algorithm Outline

5 End

Thank You!

Question?