Bisimulation partitioning and partition maintenance

on very large directed acyclic graphs

Jelle Hellings

Department of Mathematics and Computer Science
Eindhoven University of Technology

Supervisor
dr. G. H. L. Fletcher

Committee members
prof. dr. P.M.E. de Bra
dr. G. H. L. Fletcher
dr. H. J. Haverkort

July 19, 2011

Overview

Why bisimulation?

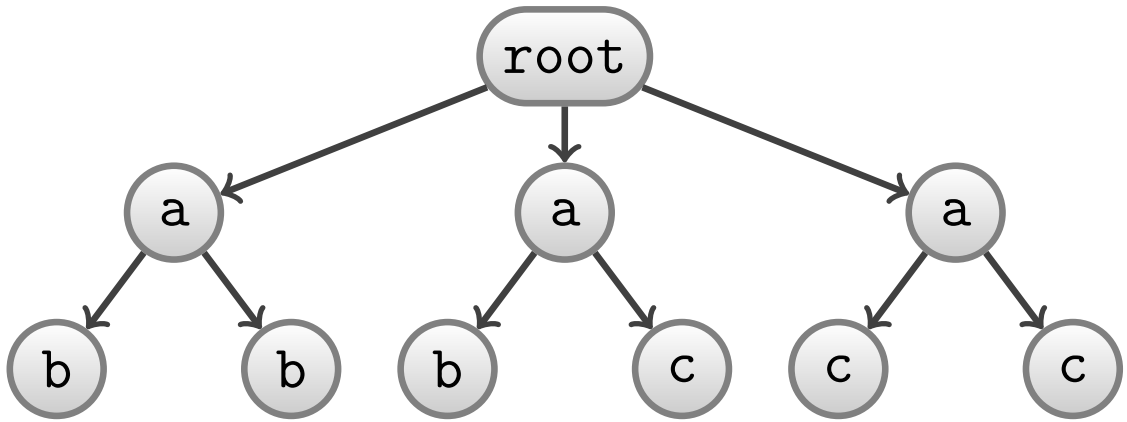
- For my Master: Introduced formally in 6 courses
- Played a role in 4 other Master courses

Usage

- Some data represented by graph
- Operation retrieve useful information from this data
- Operation is slow on large graphs
- Group together all data in the graph which are equivalent with respect to the operation

Example: Querying XML data

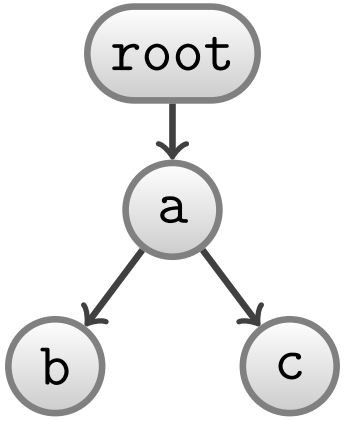
```
<root>
   <a>
      <b/> <b/>
   </a>
   <a>
      <b/> <c/>
   </a>
   <a>
      <c/> <c/>
   </a>
</root>
```



Query 1 is the path `root/a/b` reachable?

Query 2 give all nodes reachable by path `root/a/b`.

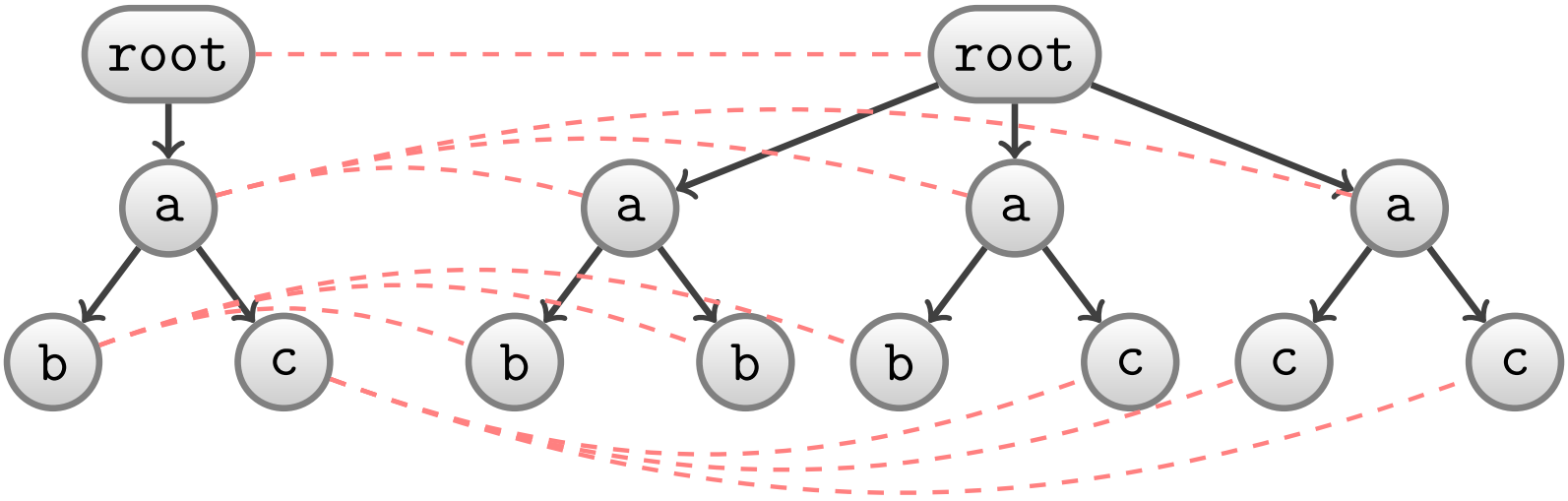
Grouping nodes using backward bisimulation



Query 1 is the path `root/a/b` reachable?

Query 2 give all nodes reachable by path `root/a/b`.

Graph index



Query 1 is the path `root/a/b` reachable?

Query 2 give all nodes reachable by path `root/a/b`.

Problem statement

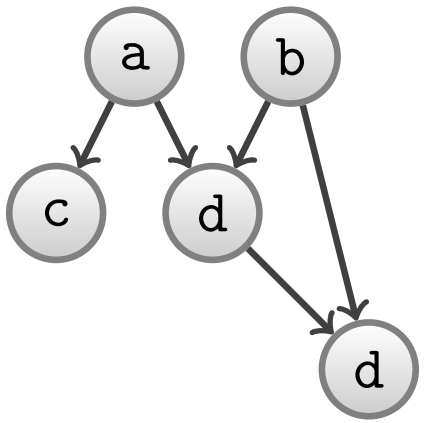
1. Bisimulation partitioning
2. Partition maintenance

Graph

Definition

A (node labeled) graph is represented by a triple $G =< N, E, l >$:

1. N is a set of nodes,
2. $E \subseteq N \times N$ is a directed edge relation,
3. l is a node-label function

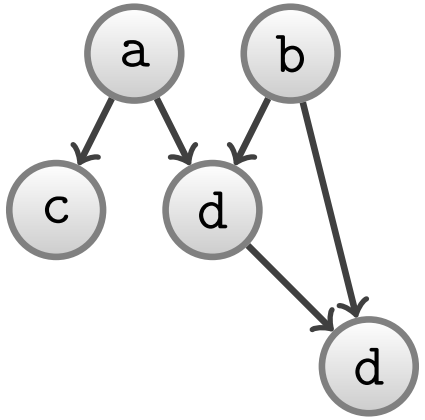


Graph

Definition

A (node labeled) graph is represented by a triple $G =< N, E, l >$:

1. N is a set of nodes,
2. $E \subseteq N \times N$ is a directed edge relation,
3. l is a node-label function



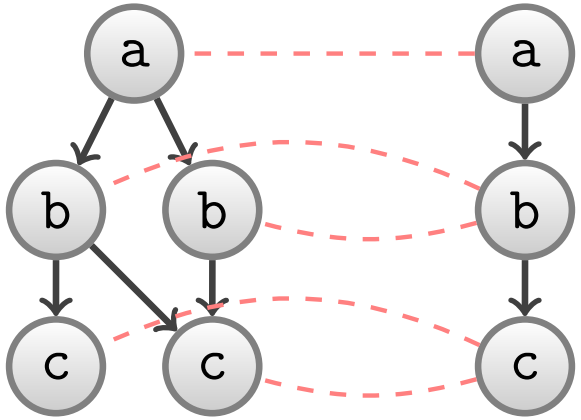
- We restrict ourselves to acyclic graphs

Bisimulation

Definition

Node n bisimulates node m; denoted as $n \approx m$; if and only if:

1. The nodes have the same label,
2. Every child n' of n is bisimulated by a child m' of m, and
3. Every child m' of m is bisimulated by a child n' of n.



Bisimulation partitioning

- Already very well studied
- Fast general algorithm by Paige and Tarjan [1987]
  - Runtime complexity: $O(|E| \log(|N|))$
  - Memory usage: $O(|N| + |E|)$

Internal vs External

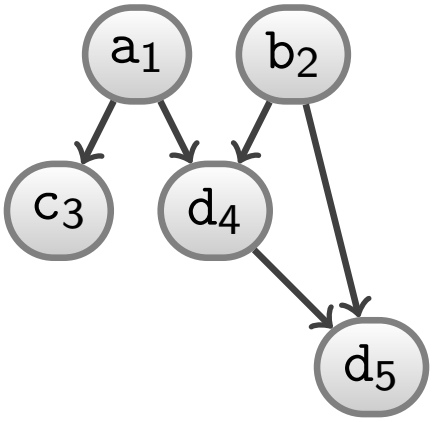
Cost for reading or writing data from hard disk drive:

Seek move to the correct position on the hard disk drive
Slow: 15ms

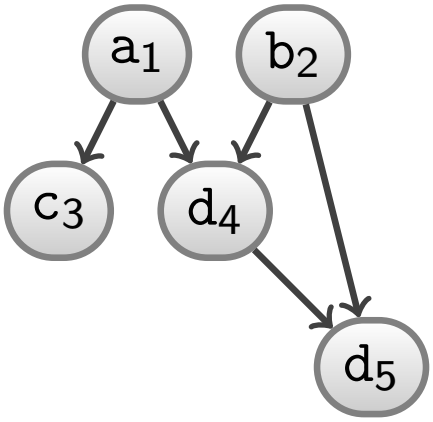
Transfer from the correct position read or write the data
'Fast', small blocks in less than 1μs

Goal: minimize seeks by transferring large blocks

Challenge: graphs



Challenge: graphs



[a, 2, 16, b, 2, 18, c, 0, 20, d, 1, 20, d, 0, 20, 3, 4, 4, 5, 5]

Algorithm

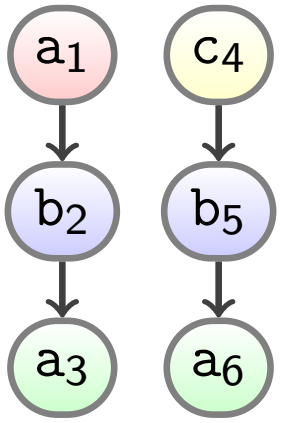
Input: Directed acyclic graph $G = \langle N, E, l \rangle$.
Output: Bisimulation partition of N.

1: P is a partition decision structure
2: **for all** $n \in N$, in reverse-topological order **do**
3: $\quad k \leftarrow \text{Key}(n)$
4: $\quad p \leftarrow \text{Query}(P, k)$
5: $\quad$ **print** (p, n)

Bisimulation key

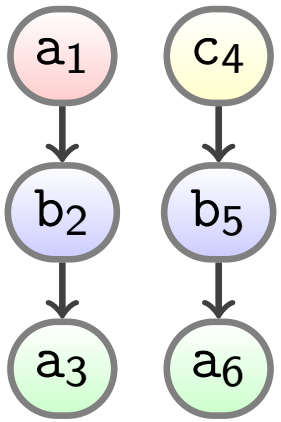
- Nodes with the same key are bisimular equivalent
- Key is a pair (l, S) with
  - l the label of the node
  - S the set of keys of child nodes
- Key is large: replace keys in S by identifiers

Optimize input



Random input a_3; b_2; a_6; a_1; b_5; c_4

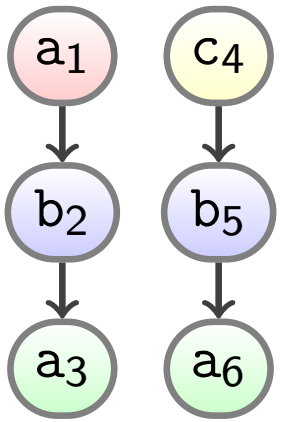
Optimize input



Random input a_3; b_2; a_6; a_1; b_5; c_4

Group on label a_3; a_1; a_6; b_2; b_5; c_4

Optimize input



Random input a_3; b_2; a_6; a_1; b_5; c_4

Group on label a_3; a_1; a_6; b_2; b_5; c_4

Grouping on rank a_3; a_6; b_2; b_5; c_4; a_1

Structural summary

Group nodes on key?

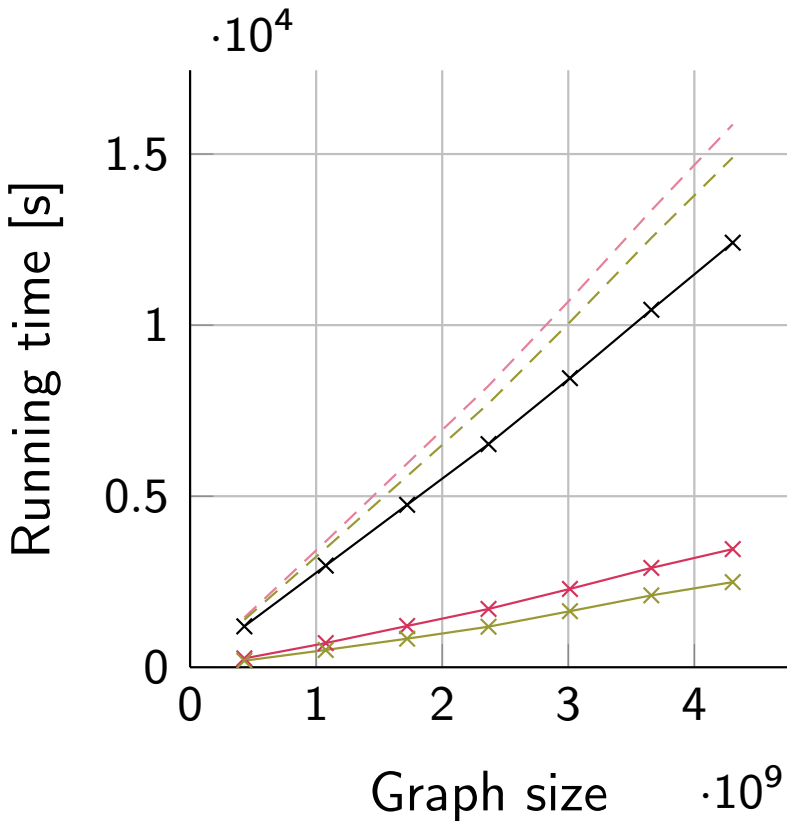
- Therefore we need large partition decision structure P
- Approximate P with hash function

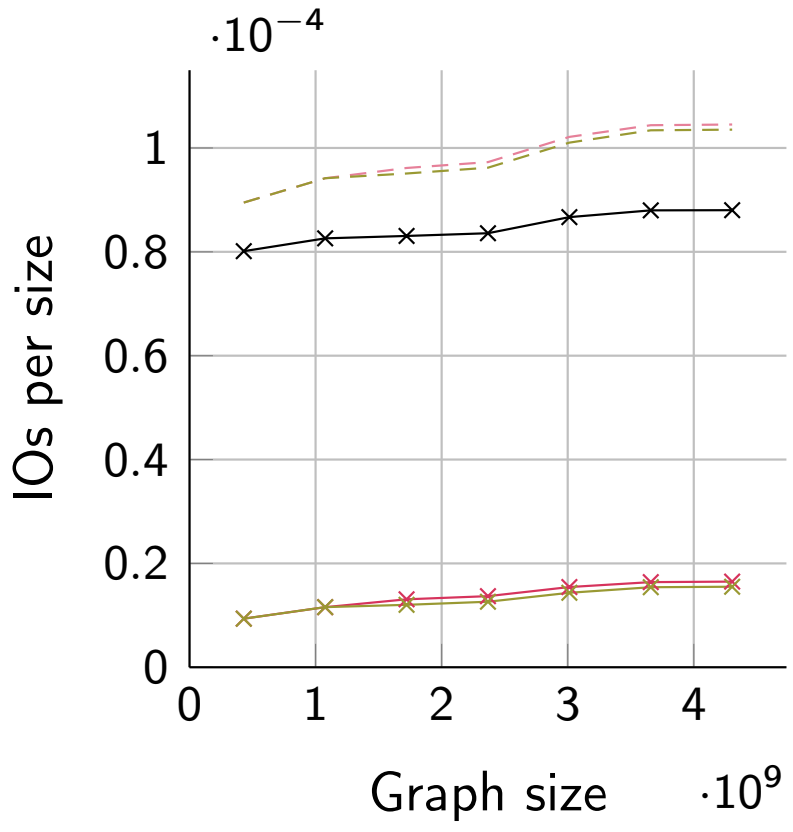
Bisimulation partitioning

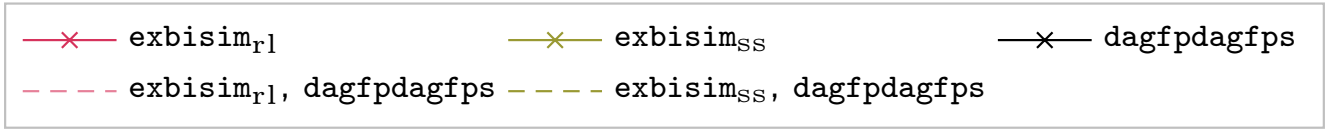
- General purpose algorithm
- Reverse-topological ordered directed acyclic graphics
- Expected complexity $O(\text{Sort}(|N|) + \text{Sort}(|E|) + \text{PQ}(|E|))$

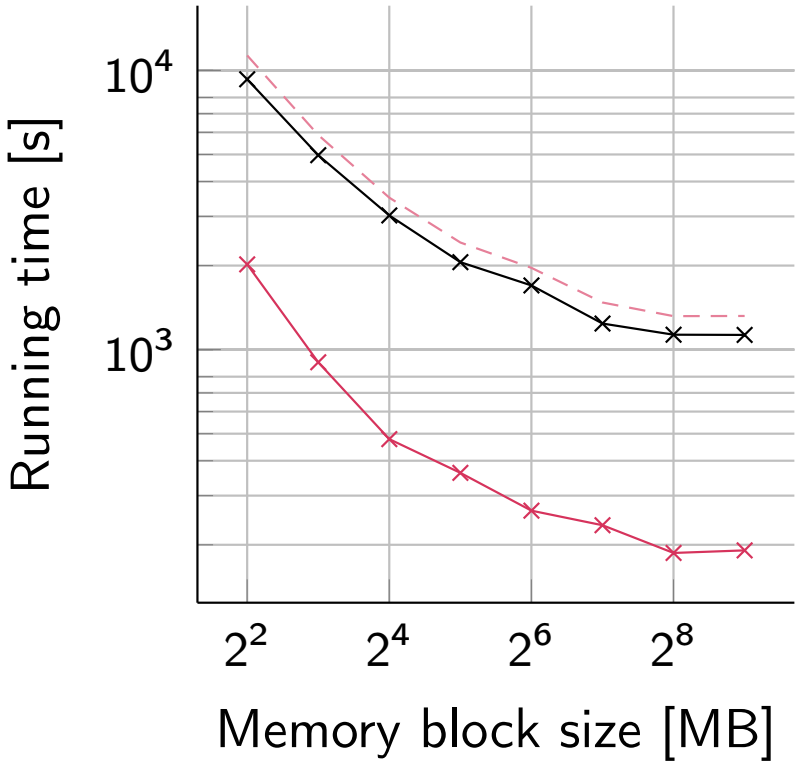
Bisimulation partitioning

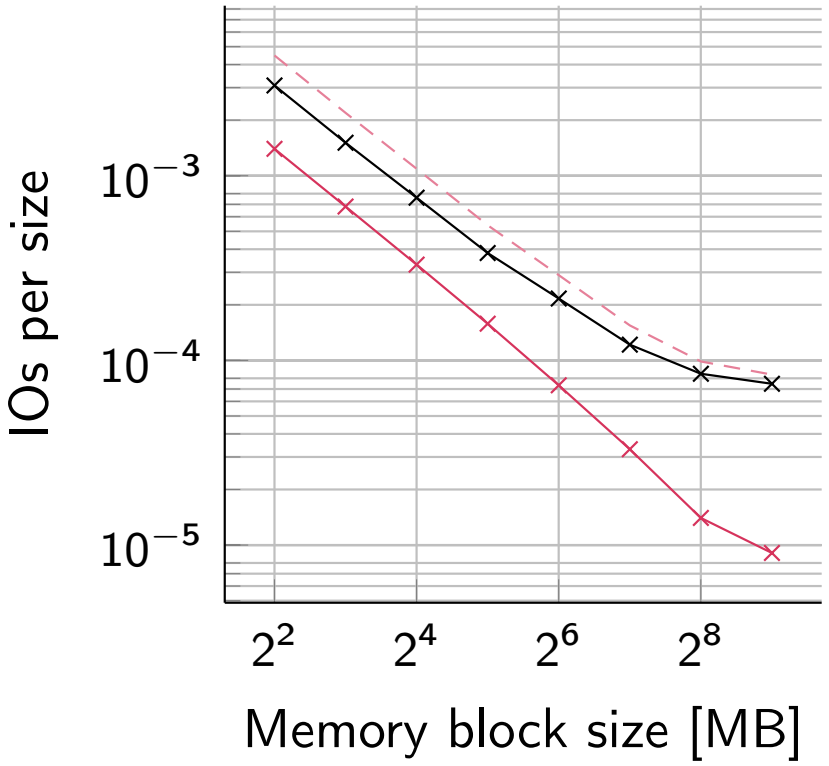
- General purpose algorithm
- Reverse-topological ordered directed acyclic graphics
- Expected complexity $O(\text{Sort}(|N|) + \text{Sort}(|E|) + \text{PQ}(|E|))$
  - Worst case complexity $O(\cdots + \text{Scan}(|N||E|))$


·10^4
1.5
1
0.5
0
Running time [s]
0
1
2
3
4
Graph size
·10^9


·10^−4
1
0.8
0.6
0.4
0.2
0
IOs per size
0
1
2
3
4
Graph size
·10^9


exbisim_rl
exbisim_ss
dagfpdagfps
exbisim_rl, dagfpdagfps
exbisim_ss, dagfpdagfps

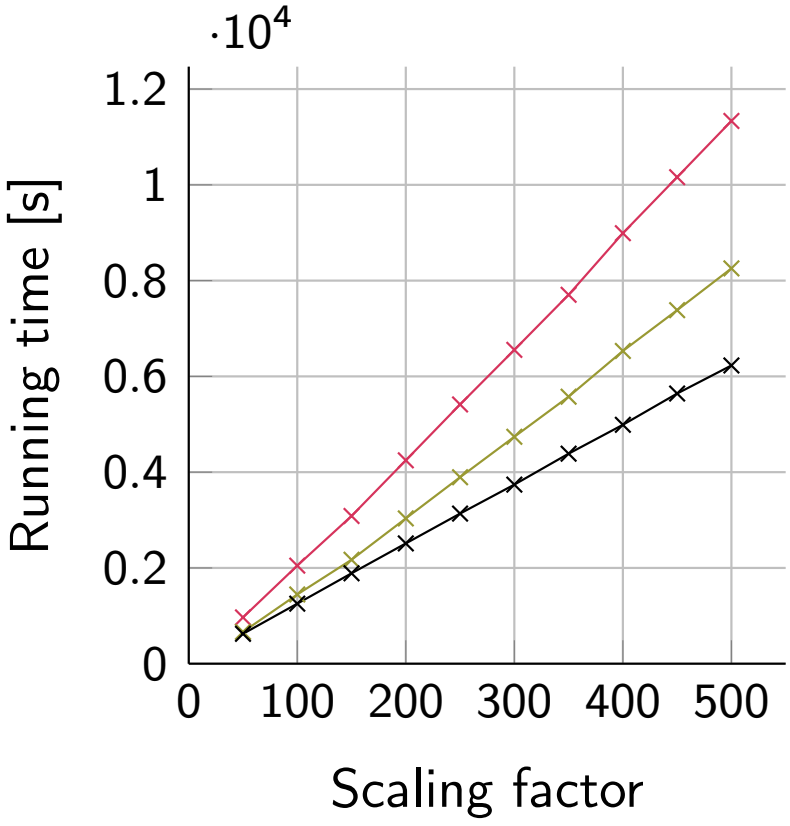

Running time [s]
10^4
10^3
2^2
2^4
2^6
2^8
Memory block size [MB]

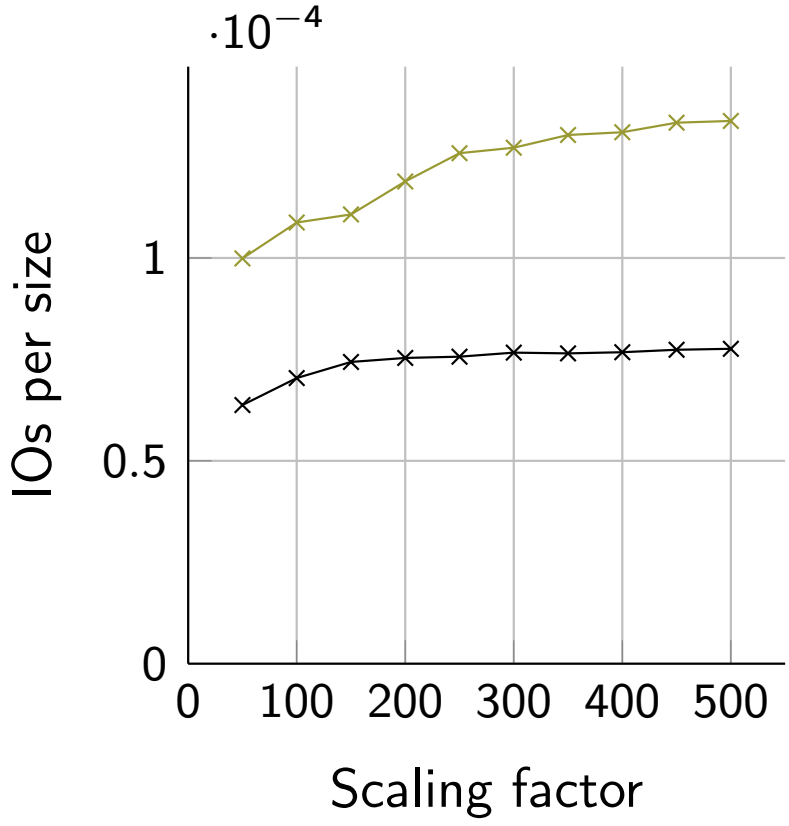

IOs per size
10^{-3}
10^{-4}
10^{-5}
2^2
2^4
2^6
2^8
Memory block size [MB]


exbisim
dagfpdagfps
exbisim, dagfpdagfps

XML index construction

- 1-Index in $O(\text{Sort}(|N|) + \text{PQ}(|N|))$
- A(k)-Index in $O(\text{Sort}(k|N|))$
- F&B-Index by using general algorithm


·10^4
1.2
1
0.8
0.6
0.4
0.2
0
Running time [s]
0
100
200
300
400
500
Scaling factor


·10^−4
1
0.5
0
IOs per size
0
100
200
300
400
500
Scaling factor


exbisim, dagfpdagfps, xmldagfpr
exbisim, dagfpdagfps
xmlbbisim

Why maintenance?

We perform small update to graph; we want an updates partition

- Bisimulation partitioning the entire graph is expensive
- Try to update existing partition

Lower bounds

- Subgraph updates in $O(|N_s| + |E_s|)$

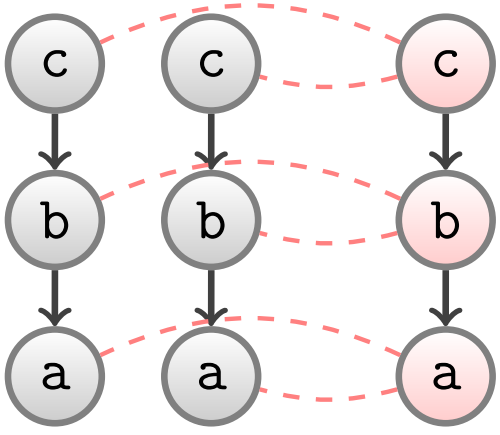
Lower bounds

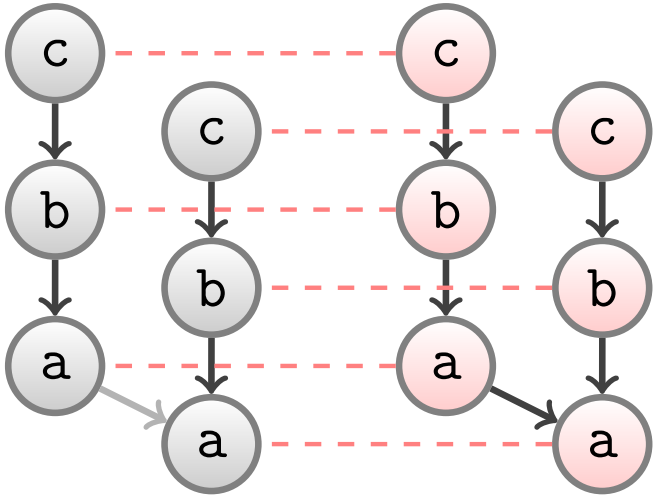
- Subgraph updates in $O(|N_s| + |E_s|)$
  - Useful: collection of graphs
  - Only possible under 'unreasonable' assumptions

Lower bounds

- Subgraph updates in $O(|N_s| + |E_s|)$
  - Useful: collection of graphs
  - Only possible under 'unreasonable' assumptions
- Edge updates in $O(|N| + |E|)$

Edge updates





Conclusion

- Partitioning
  - Fast general algorithm for directed acyclic graphs
  - Fast specializations for practical XML indices

Conclusion

- Partitioning
  - Fast general algorithm for directed acyclic graphs
  - Fast specializations for practical XML indices
- Maintenance
  - Shown limits

Conclusion

- Partitioning
  - Fast general algorithm for directed acyclic graphs
  - Fast specializations for practical XML indices
- Maintenance
  - Shown limits
- Future work?