Bisimulation partitioning and partition maintenance

On very large directed acyclic graphs

Jelle Hellings

April 28, 2011

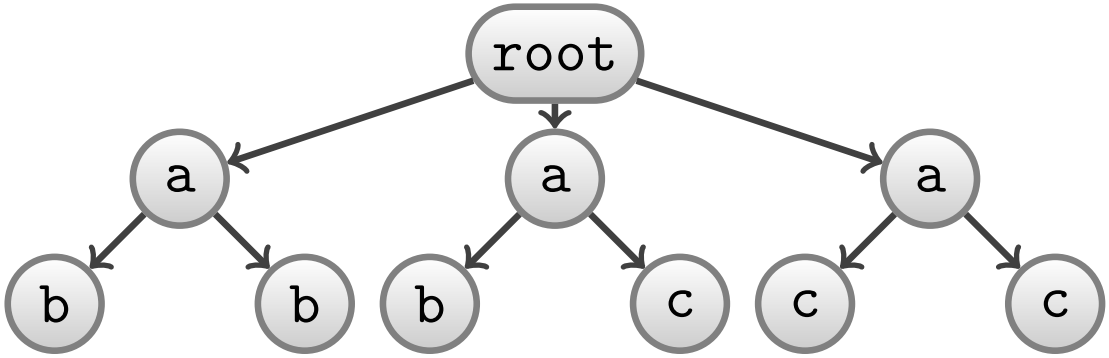
Overview

The objectives

- Directed acyclic graphs
- External memory
- Bisimulation
  - Partition refinement algorithm
  - Partition maintenance

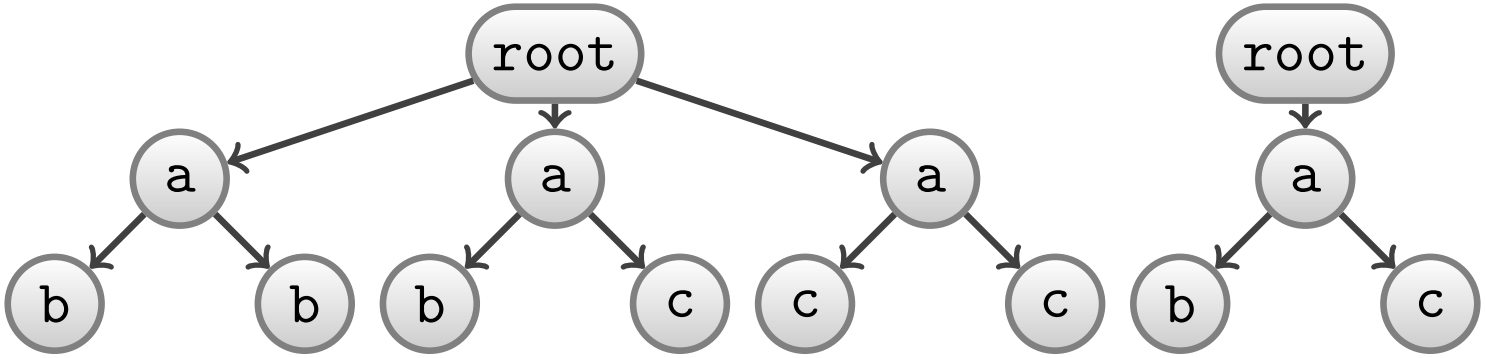
Example

- Query: is the path `root/a/b` reachable?
- Query: give all nodes reachable by path `root/a/b`



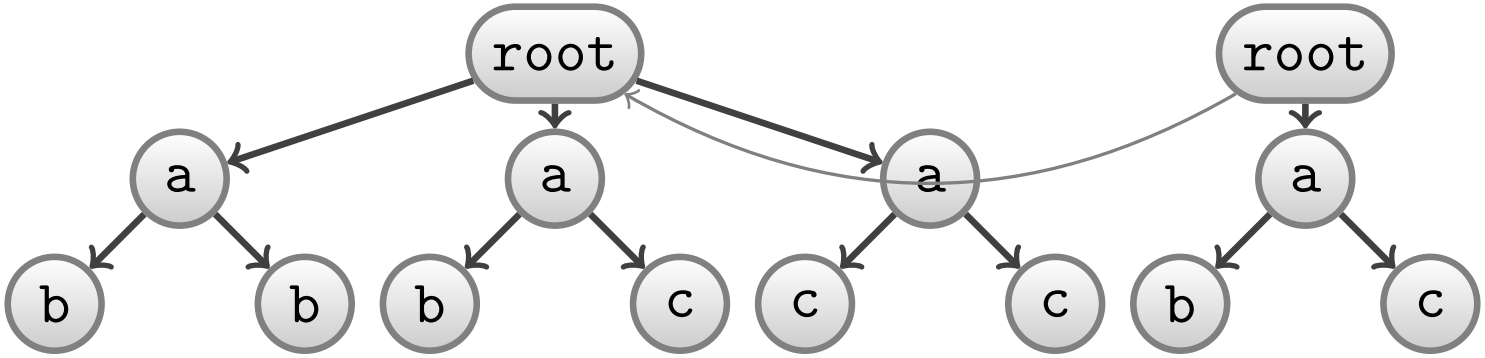
Example

- Query: is the path `root/a/b` reachable?
- Query: give all nodes reachable by path `root/a/b`
- Answer queries using an 1-index



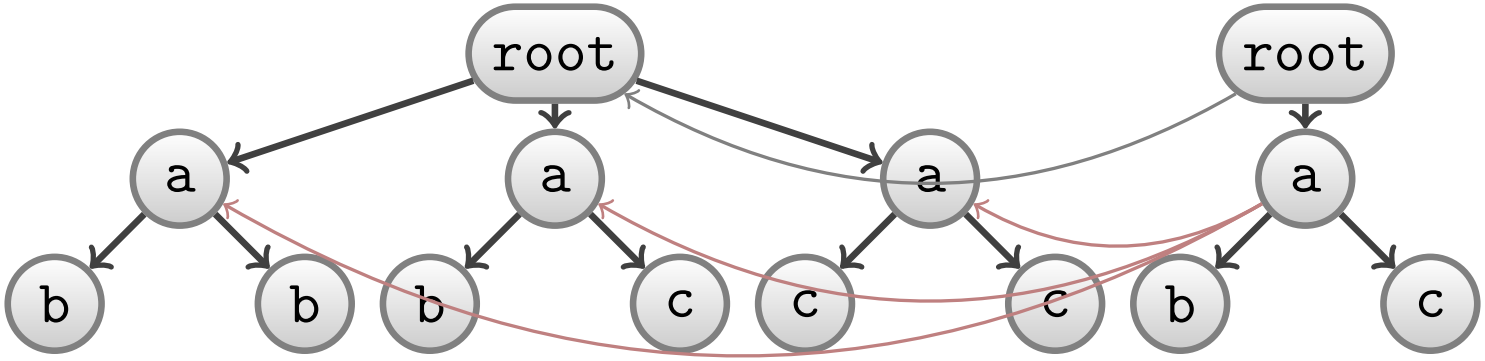
Example

- Query: is the path `root/a/b` reachable?
- Query: give all nodes reachable by path `root/a/b`
- Answer queries using an 1-index



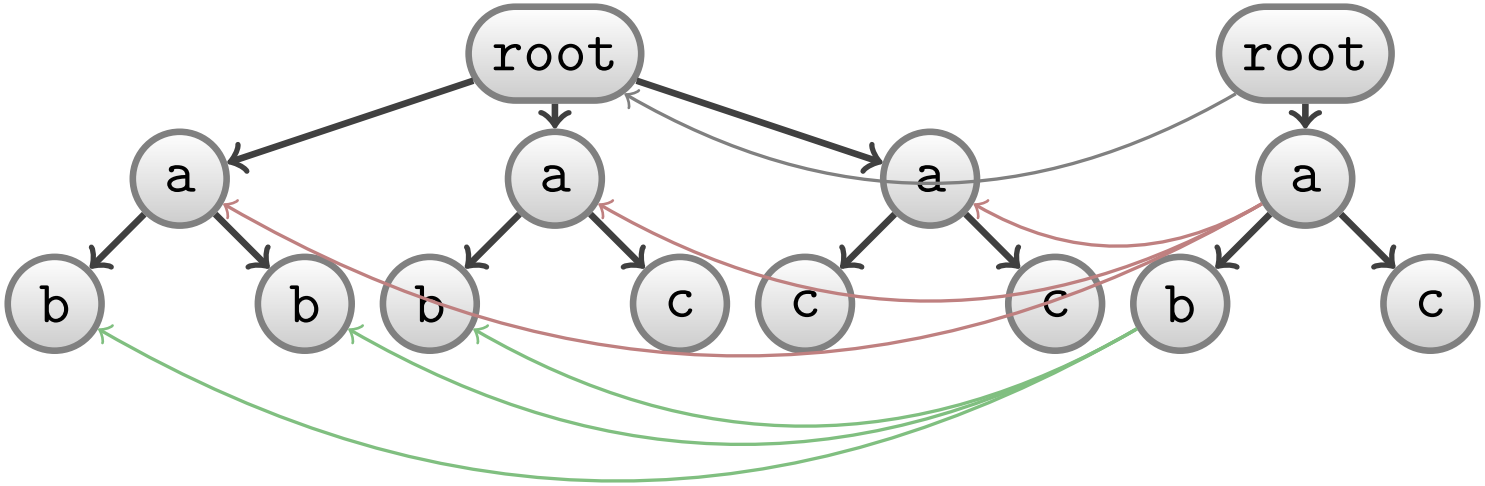
Example

- Query: is the path `root/a/b` reachable?
- Query: give all nodes reachable by path `root/a/b`
- Answer queries using an 1-index



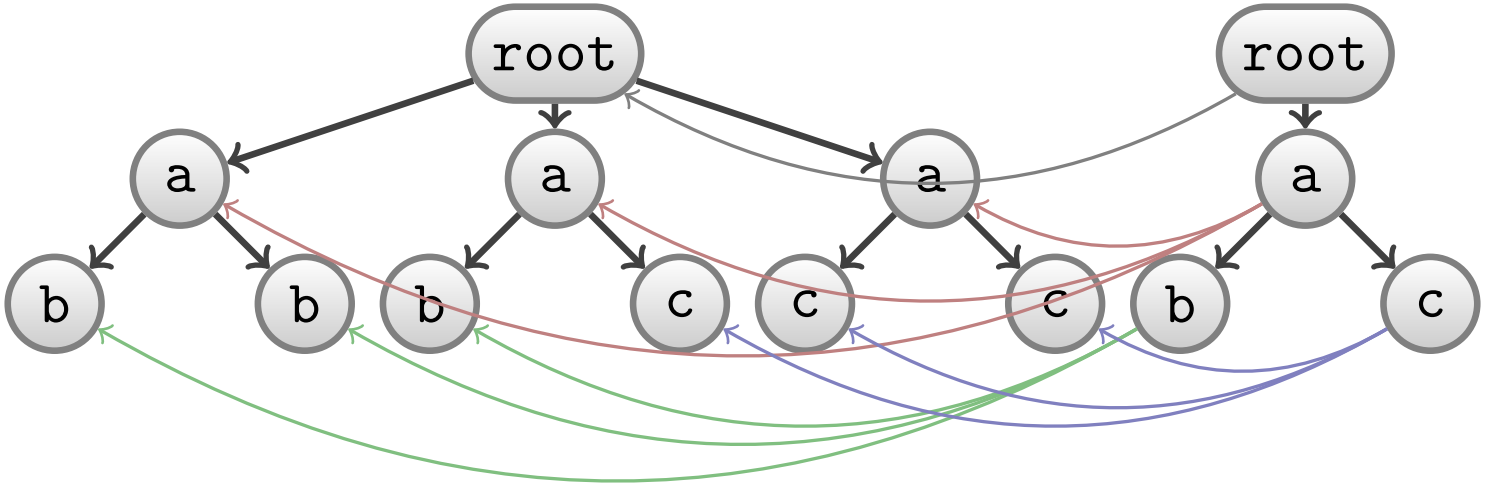
Example

- Query: is the path `root/a/b` reachable?
- Query: give all nodes reachable by path `root/a/b`
- Answer queries using an 1-index



Example

- Query: is the path `root/a/b` reachable?
- Query: give all nodes reachable by path `root/a/b`
- Answer queries using an 1-index



Overview

Node labeled graph

Definition

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

- N is a set of nodes,
- $E \subseteq N \times N$ is a directed edge relation,
- $l : N \rightarrow \mathcal{D}$ is a node-label function

Node bisimilarity

Definition

Two nodes n, m are bisimilar; denoted as $n \approx m$; if and only if:

- The nodes have the same label; $l(n) = l(m)$,
- For every child $n' \in E(n)$: $\exists m' \in E(m)$ such that $n' \approx m'$,
- For every child $m' \in E(m)$: $\exists n' \in E(n)$ such that $m' \approx n'$

Graph bisimularity

Definition

Two graphs $G_1 =< N_1, E_1, l_1 >, G_2 =< N_2, E_2, l_2 >$ are bisimilar; denoted as $G_1 \approx_G G_2$; if and only if:

- For every node $n \in N_1$: $\exists m \in N_2$ such that $n \approx m$,
- For every node $m \in N_2$: $\exists n \in N_1$ such that $n \approx m$

Maximum bisimulation graph

Definition

Graph $G_{\downarrow} =< N_{\downarrow}, E_{\downarrow}, l_{\downarrow} >$ is a maximum bisimulation graph of $G =< N, E, l >$ if

- $G \approx_G G_{\downarrow}$,
- For every $G' =< N', E', l' >$, $G' \approx_G G$ we have $|N_{\downarrow}| \leq |N'|$

Partition block

Definition

A partition block is a set of nodes

Bisimulation partition blocks

All nodes in the partition block are bisimilar equivalent

Partition

Definition

A partition $P = \{p_1, \ldots, p_n\}$ of a set of nodes N is a set of partition blocks such that:

- The partition blocks contain all nodes from the set N: $N = \bigcup_{1 \leq i \leq n} p_i$,
- Each node from the set N is present in only one partition block: $\forall_{i,j}(p_i \cap p_j = \emptyset)$

Bisimulation partition

All bisimilar nodes in the same partition block

Partition refinement

Definition

If we have two different partitions P_1 and P_2 of a set of nodes N then P_1 is a refinement of P_2 if and only if:

- For every $p \in P_1$ there is a $p' \in P_2$ such that $p \subseteq p'$,
- For every $p' \in P_2$ there is a set of partition blocks $p_1 \in P_1, \ldots, p_n \in P_1$ such that $p' = \bigcup_{1 \leq i \leq n} p_i$

Index graph

Definition

An index graph of graph $G =< N, E, l >$ with maximum bisimulation graph $G_{\downarrow} =< N_{\downarrow}, E_{\downarrow}, l_{\downarrow} >$ is represented by a quadruple $\mathcal{I} =< N_{\downarrow}, E_{\downarrow}, l_{\downarrow}, p >$ where:

- $p : N_{\downarrow} \rightarrow \mathcal{P}(N)$, a bisimulation partition function,
- For every $m \in P(n)$ we have $m \approx n$

External Memory Algorithms

- Internal memory size M
- Block size B

External Memory Algorithms

- Internal memory size M
- Block size B
- $O(\textsc{scan}(N)) = O(\frac{N}{B})$ IOs
- $O(\textsc{sort}(N)) = O(\frac{N}{B} \log_{\frac{M}{B}}(\frac{N}{B}))$ IOs
- $O(\textsc{pq}(N)) = O(\frac{N}{B} \log_{\frac{M}{B}}(\frac{N}{B}))$ IOs

Overview

Adapting existing algorithms

Problem

IO efficient bisimulation partitioning algorithm

Adapting existing algorithms

Problem

IO efficient bisimulation partitioning algorithm

Given

- Runtime efficient algorithms for Directed (Acyclic) Graphs
  - $O(|E|\log(|N|))$ by Robert Paige and Robert E. Tarjan (1987)
  - $O(|N| + |E|)$ 'refinement' for directed acyclic graphs
- These algorithms 'access' all parts of the graph continuously

Adapting existing algorithms

Problem

IO efficient bisimulation partitioning algorithm

Given

- Runtime efficient algorithms for Directed (Acyclic) Graphs
  - $O(|E|\log(|N|))$ by Robert Paige and Robert E. Tarjan (1987)
  - $O(|N| + |E|)$ 'refinement' for directed acyclic graphs
- These algorithms 'access' all parts of the graph continuously

Solution

- No straight forward adaption of internal memory algorithms
- We start with designing a basic 'online' approach

Oracle

Assume we have an oracle

- We can present nodes to this oracle
- Oracle responds with a bisimulation partition block

Oracle

Assume we have an oracle

- We can present nodes to this oracle
- Oracle responds with a bisimulation partition block

Oracle is implementable if

- It knows the label of the node
- For every child of the node:
  - The partition block wherein this child is placed

Partition decision structure

Definition

A partition decision structure *pds* is a mapping $\mathcal{D} \times \mathcal{P}(N_{\downarrow}) \rightarrow N_{\downarrow}$ providing a single operation $\textsc{query}(pds, (l, S))$

Partition decision structure

Definition

A partition decision structure *pds* is a mapping $\mathcal{D} \times \mathcal{P}(N_{\downarrow}) \rightarrow N_{\downarrow}$ providing a single operation $\textsc{query}(pds, (l, S))$

- The *pds* represents $G_{\downarrow}$
- $\textsc{query}$ is non-trivial to implement

Basic algorithm

1: $pds \leftarrow$ empty partition decision structure
2: $M \leftarrow$ empty mapping between nodes and partition blocks
3: **for each** $n \in N$, in reverse-topological order **do**
4: $\quad k \leftarrow (l(n), \{p : \exists_{m \in E(n)} p = M[m]\})$
5: $\quad M[n] \leftarrow \textsc{query}(pds, k)$
6: $\quad$ **print** $(M[n], n)$
7: **end for**

Analysis

- Reverse-topological order
- Runtime cost is $O(|N| + |E| + \textsc{query}(|N|))$
- Storage cost is $O(|N|)$ for M
- Storage cost is at least $O(|N_{\downarrow}| + |E_{\downarrow}|)$ for *pds*
- Algorithm is not IO efficient

Analysis

- Reverse-topological order
- Runtime cost is $O(|N| + |E| + \textsc{query}(|N|))$
- Storage cost is $O(|N|)$ for M
- Storage cost is at least $O(|N_{\downarrow}| + |E_{\downarrow}|)$ for *pds*
- Algorithm is not IO efficient
  - Implementation details *pds*
  - Structure M is randomly accessed

Time-forward processing

Idea: send partition blocks from child node to parents

- Nodes are reverse-topological ordered
- Give nodes numeric identifiers i
- Give partition blocks numeric identifiers p
- Per node n: add (i, p) to a queue for every parent i
- Order queue on identifier: use priority queue

Time-forward processing algorithm

$P \leftarrow$ empty partition decision structure
$Q \leftarrow$ empty minimum-priority queue
for each $n \in N$, in reverse-topological order **do**
  $k \leftarrow (l(n), \{c : \textsc{top}(Q) = (n, c)\})$
  $p \leftarrow \textsc{query}(P, k)$
  print (p, n)
  for each $m \in E'(n)$ **do**
    $\textsc{add}(Q, (m, p))$
  end for
end for

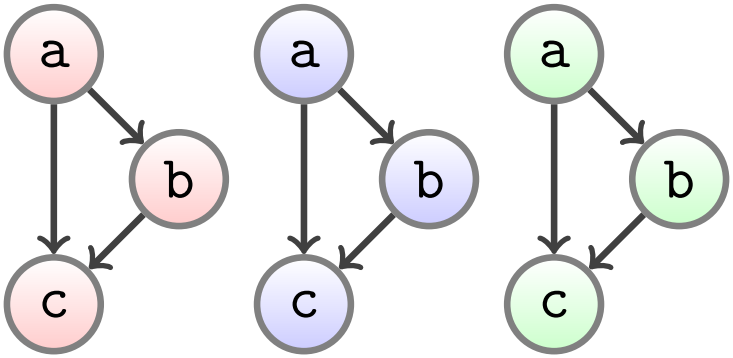
Time-forward processing algorithm

$P \leftarrow$ empty partition decision structure
$Q \leftarrow$ empty minimum-priority queue
for each $n \in N$, in reverse-topological order **do**
  $k \leftarrow (l(n), \{c : \textsc{top}(Q) = (n, c)\})$
  $p \leftarrow \textsc{query}(P, k)$
  print (p, n)
  for each $m \in E'(n)$ **do**
    $\textsc{add}(Q, (m, p))$
  end for
end for

Analysis

IO cost is $O(\textsc{scan}(|N| + |E|) + \textsc{pq}(|E|) + \textsc{query}(|N|))$

pds: queries in unpredictable order



Reverse-topological orderings

- $\mathtt{c} <_{rt} \mathtt{b} <_{rt} \mathtt{a} <_{rt} \mathtt{c} <_{rt} \mathtt{b} <_{rt} \mathtt{a} <_{rt} \mathtt{c} <_{rt} \mathtt{b} <_{rt} \mathtt{a}$
- $\mathtt{c} <_{rt} \mathtt{c} <_{rt} \mathtt{c} <_{rt} \mathtt{b} <_{rt} \mathtt{b} <_{rt} \mathtt{b} <_{rt} \mathtt{a} <_{rt} \mathtt{a} <_{rt} \mathtt{a}$

Initial partition

Idea

Use initial partition P_i of N; bisimulation partition of N refines P_i

Initial partition

Idea

Use initial partition P_i of N; bisimulation partition of N refines P_i

- Label-equivalence partition
  - How to maintain reverse-topological order?

Initial partition

Idea

Use initial partition P_i of N; bisimulation partition of N refines P_i

- Label-equivalence partition
  - How to maintain reverse-topological order?
- Rank-label equivalence partition
  - Rank of node n is length of longest path from n to leaf
  - Order blocks on increasing rank: reverse-topologically sorted

Initial partition: bisimulation partition

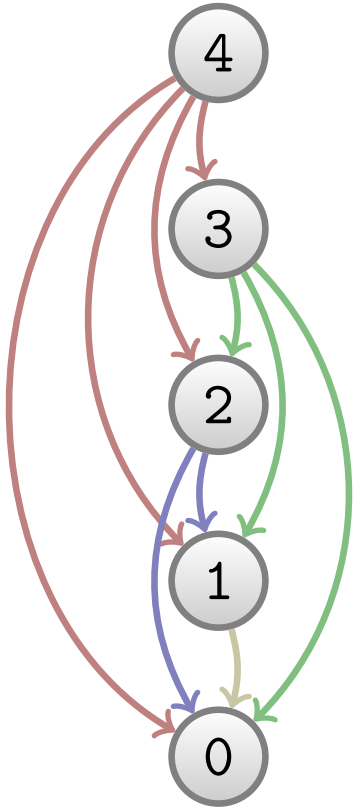
Definition

Express bisimulation equivalence as equivalence of a recursively defined node value v

- For leaf node n: value $v(n) = (l(n), \emptyset)$
- For non-leaf node n: value $v(n) = (l(n), \{v(m) : m \in E(n)\})$

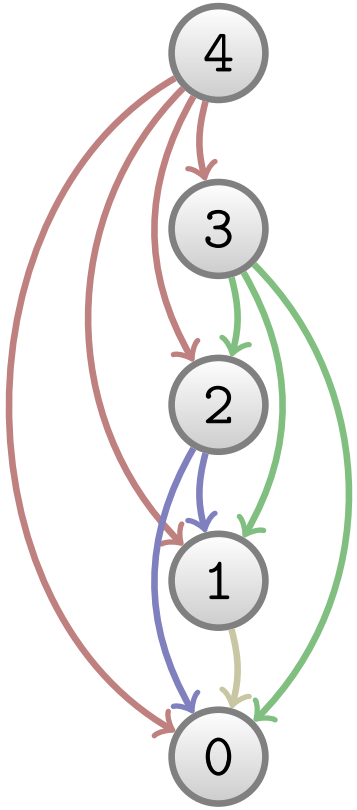
Structural summary

- Node value v is 'arbitrary large'



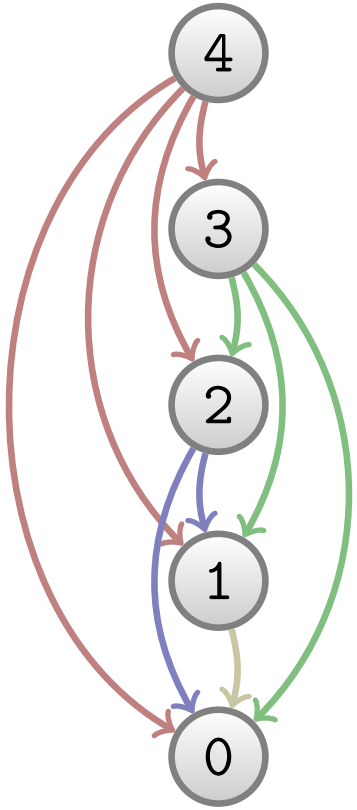
Structural summary

- Node value v is 'arbitrary large'
- Map $v = (l, S)$ to value:
  - Depth of nested sets: rank
  - (Perfect) hash function: structural summary



Structural summary

- Node value v is 'arbitrary large'
- Map $v = (l, S)$ to value:
  - Depth of nested sets: rank
  - (Perfect) hash function: structural summary
- Partition on (rank, label, structural summary)
  - We need rank for reverse-topological ordering



Annotation algorithm

$N', E' \leftarrow$ empty list, empty list
$Q \leftarrow$ empty minimum priority queue
for each $n \in N$, in reverse-topological order **do**
  $S \leftarrow \{(r+1, h) : \textsc{top}(Q) = (n, r, h)\}$
  $r, h \leftarrow \max(\{r : (r, h) \in S\}), \textsc{hash}(\{h : (r, h) \in S\})$
  $\textsc{add}(N', (r, l(n), h, n))$
  for each $m \in E(n)$ **do**
    $\textsc{add}(E', (n, m))$
    $\textsc{add}(Q, (m, r, h))$
  end for
end for
$\textsc{sort}(N')$
$\textsc{reorder}(E')$

Analysis

- $O(\textsc{scan}(|N| + |E|) + \textsc{sort}(|N|) + \textsc{sort}(|E|) + \textsc{pq}(|E|))$
- Incremental hash function
- Do we need bisimulation after annotation?

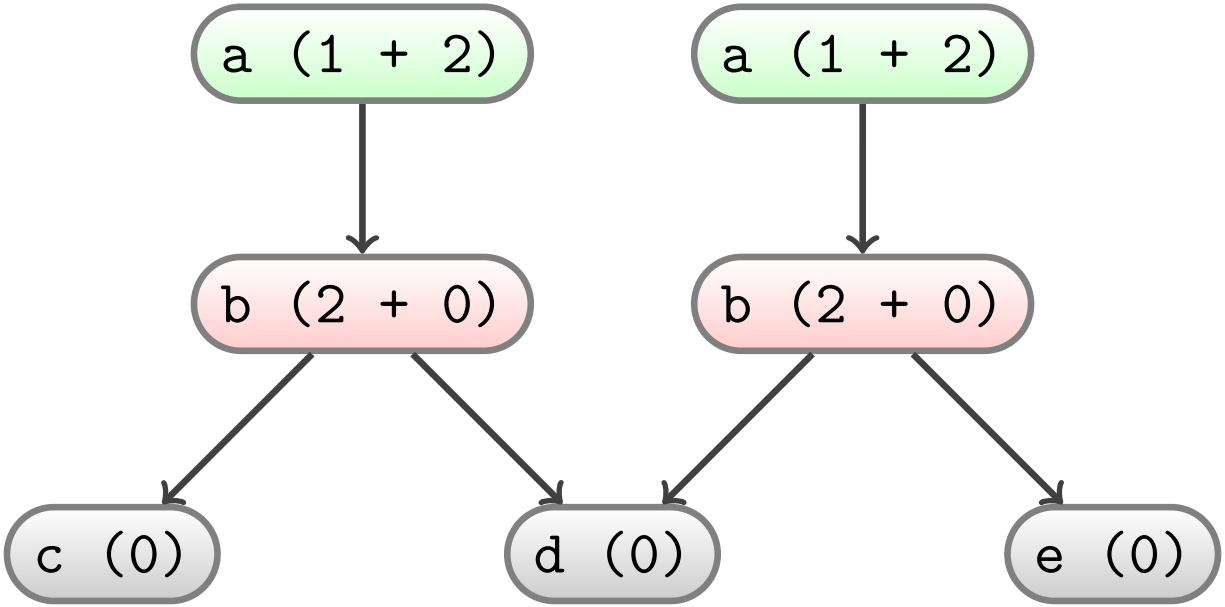
Analysis

- $O(\textsc{scan}(|N| + |E|) + \textsc{sort}(|N|) + \textsc{sort}(|E|) + \textsc{pq}(|E|))$
- Incremental hash function
- Do we need bisimulation after annotation?
  - Hash introduces probability on collisions
    - Different values $v \neq v'$ get same hash value $h(v) = h(v')$

Analysis

- $O(\textsc{scan}(|N| + |E|) + \textsc{sort}(|N|) + \textsc{sort}(|E|) + \textsc{pq}(|E|))$
- Incremental hash function
- Do we need bisimulation after annotation?
  - Hash introduces probability on collisions
    - Different values $v \neq v'$ get same hash value $h(v) = h(v')$
  - Thus: structural summaries is not full bisimulation
  - Structural summaries expected to provide good initial partition

Collision propagation



Handling collisions and collision propagation

- Problem:
  - Partitions can have collisions
  - Collisions propagate
- When processing partition block p:
  - Children of nodes in p are processed

Handling collisions and collision propagation

- Problem:
  - Partitions can have collisions
  - Collisions propagate
- When processing partition block p:
  - Children of nodes in p are processed
- Locally refine each initial partition block
  - We have full *pds* keys to refine on

Handling collisions and collision propagation

- Problem:
  - Partitions can have collisions
  - Collisions propagate
- When processing partition block p:
  - Children of nodes in p are processed
- Locally refine each initial partition block
  - We have full *pds* keys to refine on
  - We can refine on hash of this key

Handling collisions and collision propagation

- Problem:
  - Partitions can have collisions
  - Collisions propagate
- When processing partition block p:
  - Children of nodes in p are processed
- Locally refine each initial partition block
  - We have full *pds* keys to refine on
  - We can refine on hash of this key
  - Refine on size of key

External memory bisimulation

- Initial partition: using structural summaries
- Refine each initial partition; gives local partitions

External memory bisimulation

- Initial partition: using structural summaries
- Refine each initial partition; gives local partitions
- Local collisions still possible
  - Low probability: 'constant factor'

External memory bisimulation

- Initial partition: using structural summaries
- Refine each initial partition; gives local partitions
- Local collisions still possible
  - Low probability: 'constant factor'
- Implement locally needed *pds* as list of (key, partition block)
  - Keys have a fixed size
  - 'constant number of keys per local partition'

Analysis

- $O(\textsc{scan}(|N| + |E|) + \textsc{sort}(|N|) + \textsc{sort}(|E|) + \textsc{pq}(|E|))$
- Do we need structural summaries?

Analysis

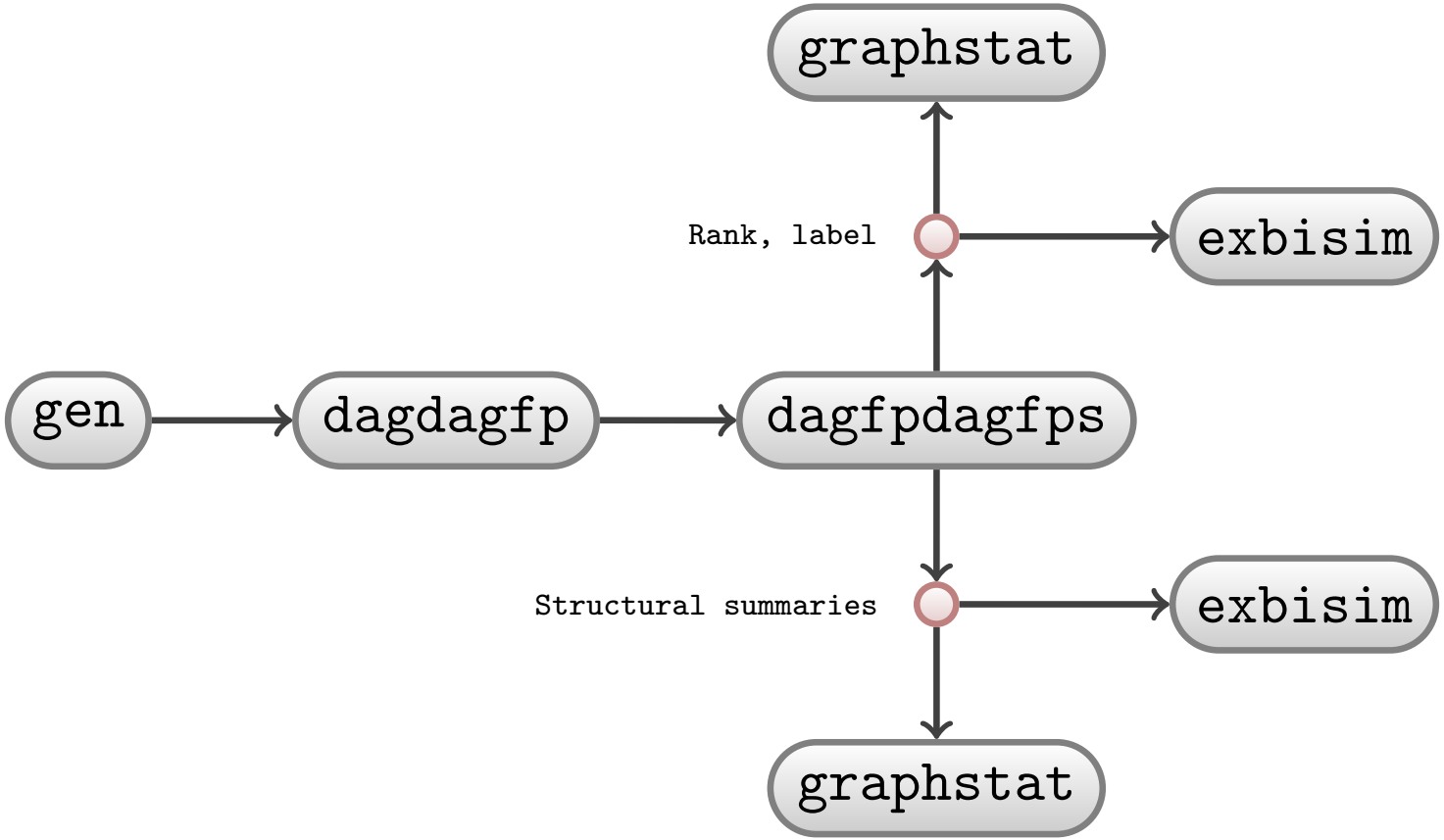
- $O(\textsc{scan}(|N| + |E|) + \textsc{sort}(|N|) + \textsc{sort}(|E|) + \textsc{pq}(|E|))$
- Do we need structural summaries?
  - Worst case: $O(\cdots + \frac{\max_{p \in P} |p||E(p)|}{B})$
  - p is a local partition block, $|p| \leq |N|$

Analysis

- $O(\textsc{scan}(|N| + |E|) + \textsc{sort}(|N|) + \textsc{sort}(|E|) + \textsc{pq}(|E|))$
- Do we need structural summaries?
  - Worst case: $O(\cdots + \frac{\max_{p \in P} |p||E(p)|}{B})$
  - p is a local partition block, $|p| \leq |N|$
  - Structural summaries: more initial partition blocks

Overview

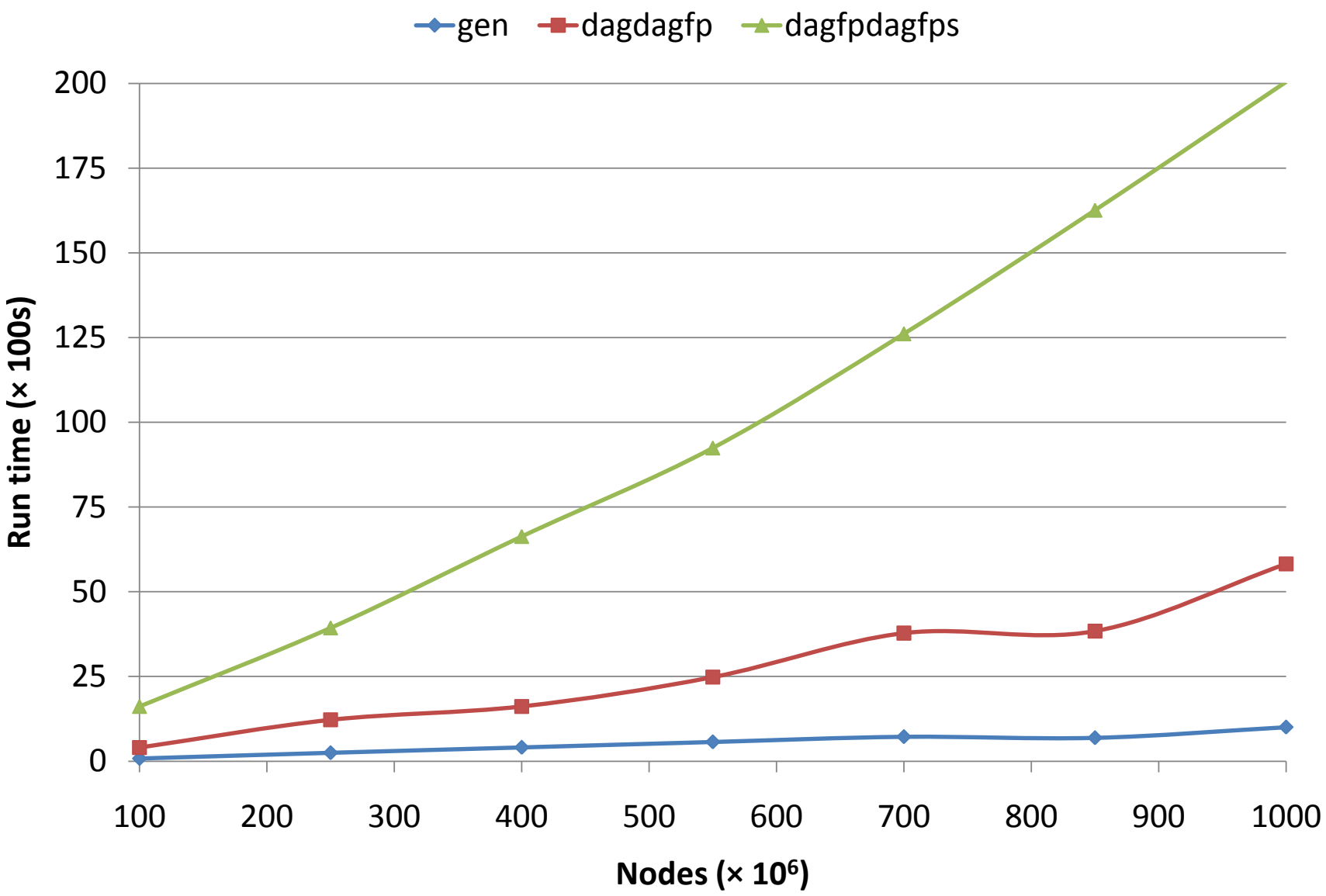
Programs


graphstat
Rank, label
exbisim
gen
dagdagfp
dagfpdagfps
Structural summaries
exbisim
graphstat

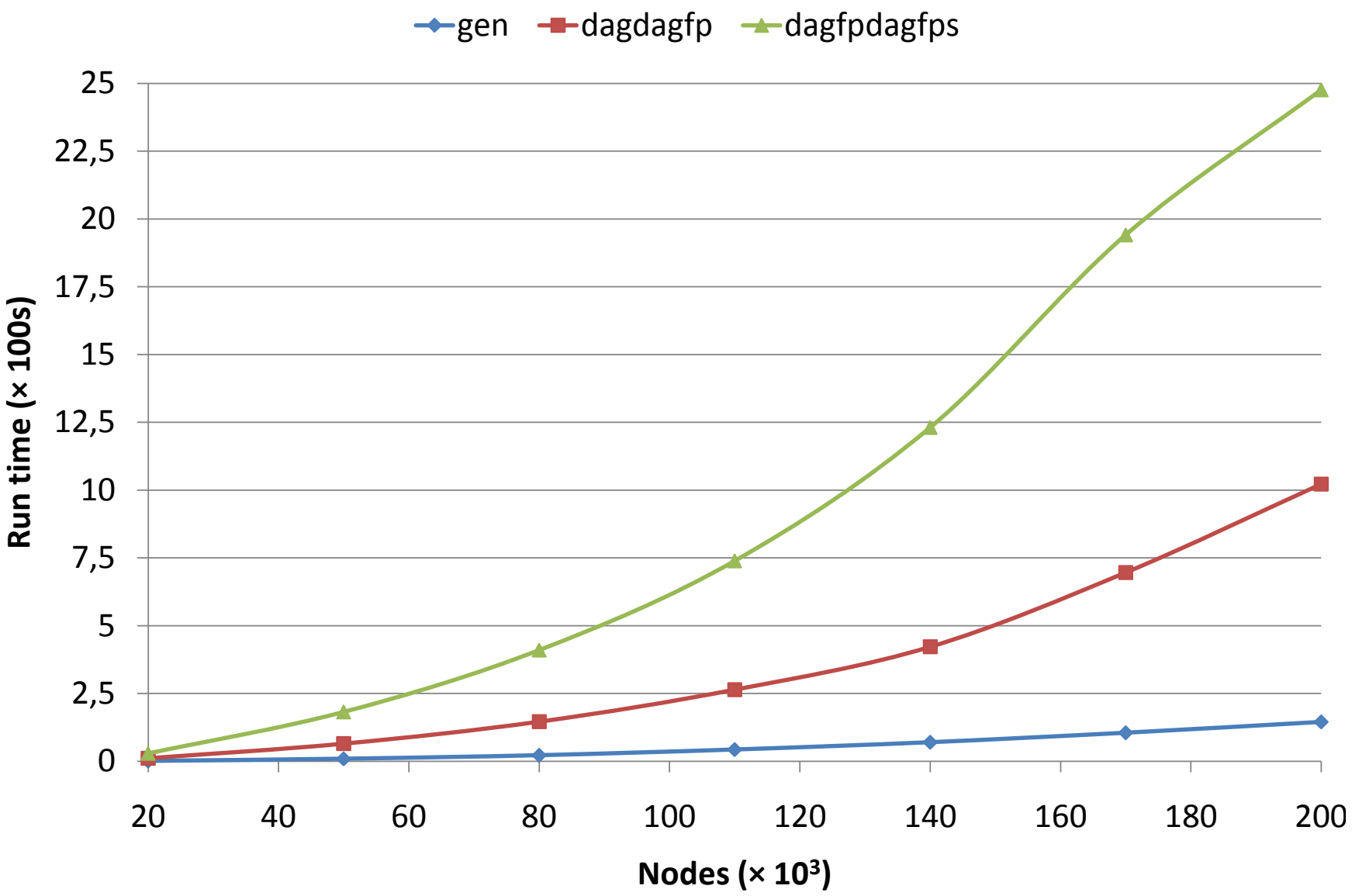
Datasets

- Random generated graphs
- Transitive closure graphs

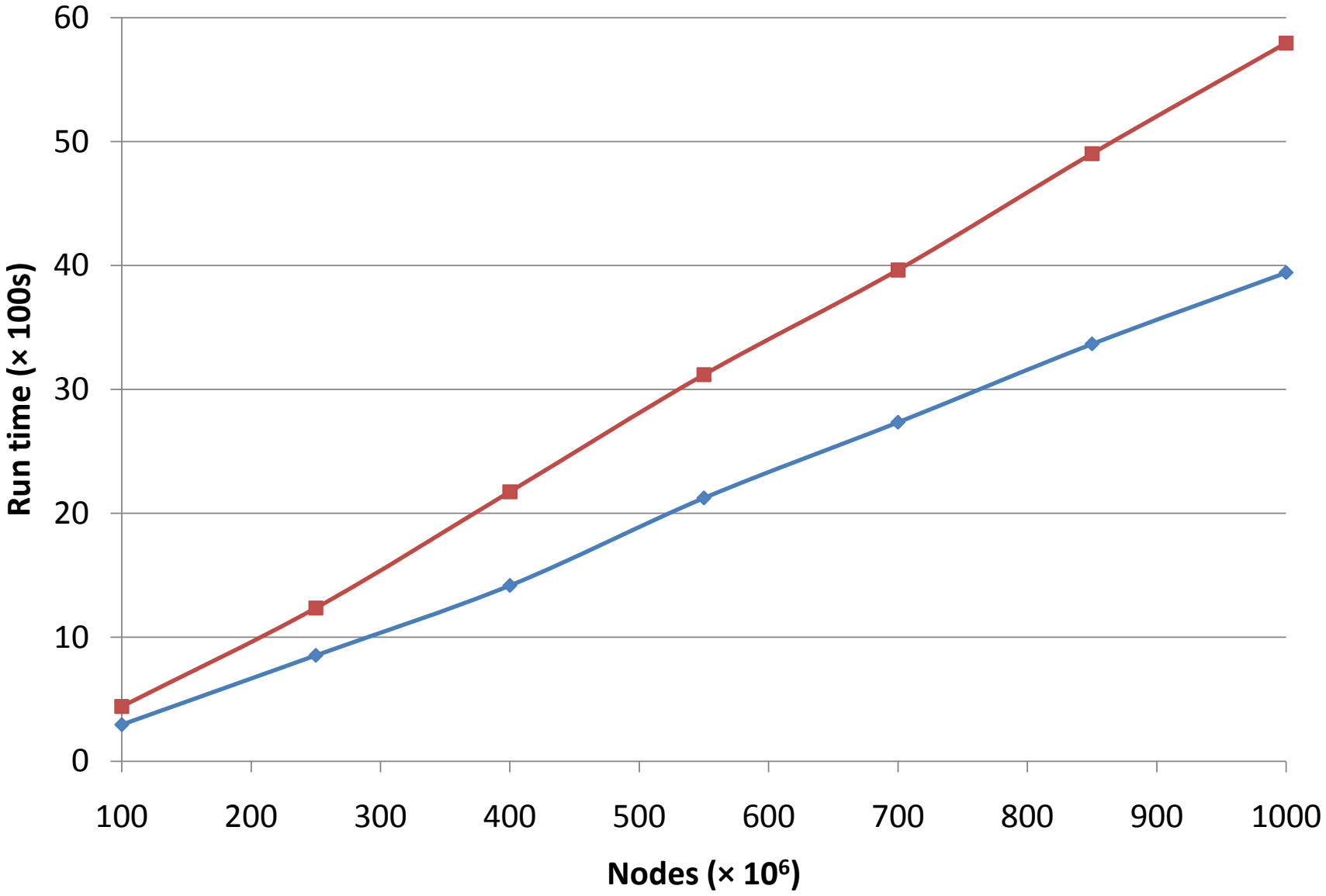
Running times for random graphs



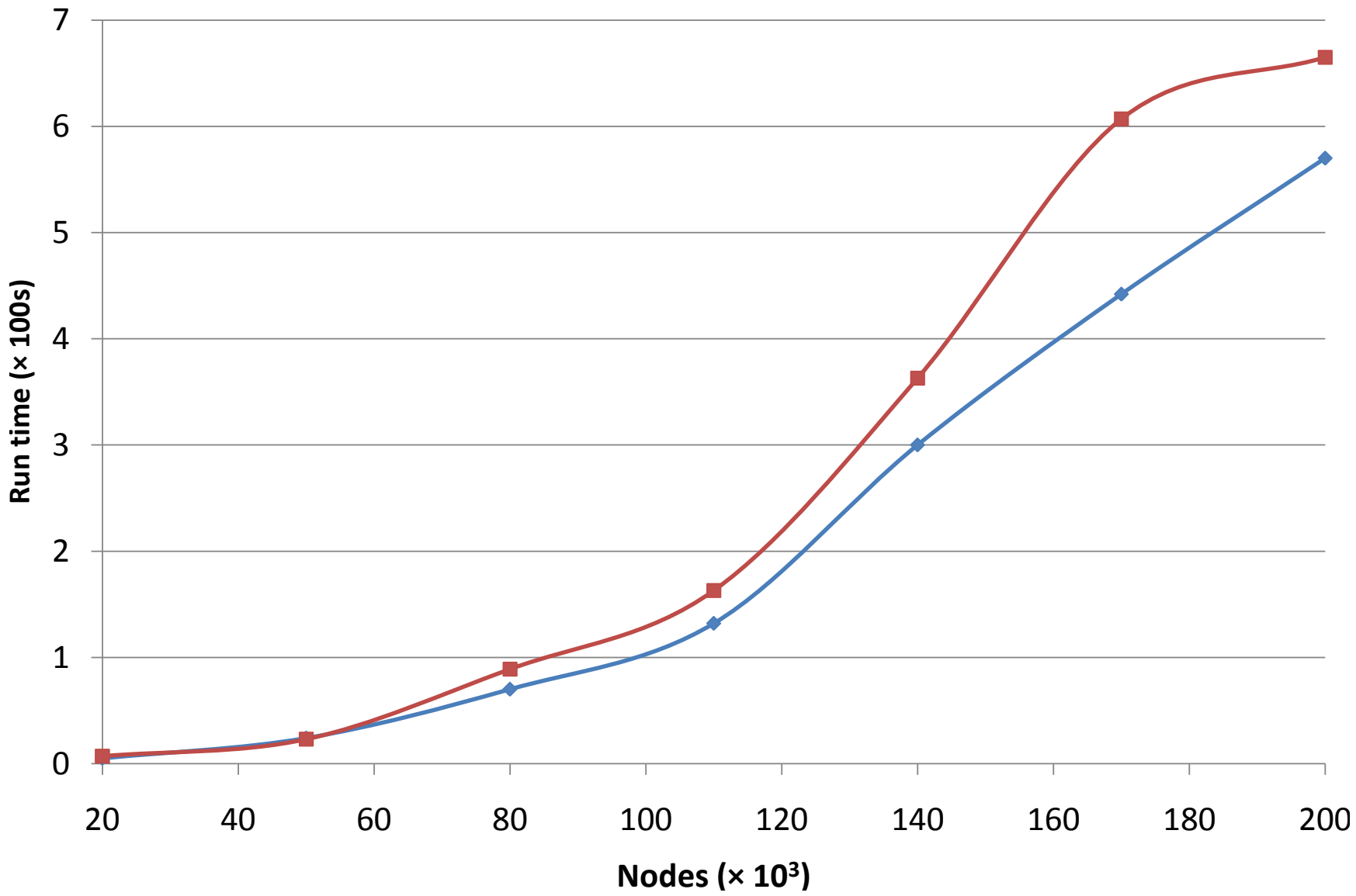
Running times for transitive closure graphs



Running times for random graphs



Running times for transitive closure graphs



Partitions for random graphs

	structural summaries		without structural summaries		
nodes	initial	local	initial	local	collisions
$100 \cdot 10^6$	99.86%	100.00%	0.02%	99.96%	3 (28709)
$250 \cdot 10^6$	99.83%	100.00%	0.01%	99.97%	4 (44245)
$400 \cdot 10^6$	99.83%	100.00%	0.00%	99.92%	6 (216302)
$550 \cdot 10^6$	99.82%	100.00%	0.00%	99.97%	5 (108743)
$700 \cdot 10^6$	99.81%	100.00%	0.00%	99.95%	5 (236386)
$850 \cdot 10^6$	99.81%	100.00%	0.00%	99.97%	6 (203958)
$1000 \cdot 10^6$	99.80%	100.00%	0.00%	99.96%	6 (278918)

Partitions for transitive closure graphs

	structural summaries		without structural summaries		
nodes	initial	local	initial	local	collisions
$20 \cdot 10^3$	99.62%	100.00%	10.04%	100.00%	0
$50 \cdot 10^3$	99.89%	100.00%	10.01%	100.00%	0
$80 \cdot 10^3$	99.91%	100.00%	10.01%	100.00%	0
$110 \cdot 10^3$	99.95%	100.00%	10.01%	100.00%	0
$140 \cdot 10^3$	99.39%	100.00%	10.00%	100.00%	0
$170 \cdot 10^3$	99.97%	100.00%	10.00%	100.00%	0
$200 \cdot 10^3$	99.96%	100.00%	10.00%	100.00%	0

Overview

Graph maintenance

Goal

Techniques for indexing graph datasets

Graph maintenance

Goal

Techniques for indexing graph datasets

What if data changes?

Can we update this graph index efficiently when:

- We add or remove a subgraph
- We add or remove edges

Summary

- Fast (expected) IO efficient bisimulation partitioning
  - Partition decision structure
  - Structural summaries
- Small scale experimental verification
  - Efficient on structured data
  - Efficient on unstructured data
  - Even ‘efficient’ for non-sparse graphs