Efficient External-Memory Bisimulation on DAGs

Jelle Hellings\textsuperscript{1}, George H. L. Fletcher\textsuperscript{2}, and Herman Haverkort\textsuperscript{2}
\textsuperscript{1} Hasselt University and transnational University of Limburg
\textsuperscript{2} Eindhoven University of Technology

May 25, 2012
Introduction

Definitions

Algorithm

Experimental verification

Conclusion
Graphs

- Used as formal model
  - Graph and XML databases
  - Model checking
- Graphs can become large for modern applications
  - Large XML databases
  - Social networks
  - Biological interaction networks
Bisimilarity

- Idea: ‘compress’ graph to speed up computations
- Intuitively: group nodes that ‘reflect the same behavior’
Internal memory algorithm

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

Even faster algorithm for DAGs by Dovier et al. [2004]
▷ Runtime complexity: $O(|N| + |E|)$
▷ Depends on node ranks
External memory

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
Node-labeled graphs

Definition
A (node labeled) graph is represented by a triple \( G = \langle N, E, l \rangle \):

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

Note
We work with directed acyclic graphs
Node bisimilarity

Definition
Nodes $n_1$ and $n_2$ are bisimilar to each other, denoted $n_1 \approx n_2$, if and only if:

1. the nodes have the same label: $l_1(n_1) = l_2(n_2)$;
2. for every node $n'_1 \in children(n_1)$ there is a node $n'_2 \in children(n_2)$ such that $n'_1 \approx n'_2$; and,
3. for every node $n'_2 \in children(n_2)$ there is a node $n'_1 \in children(n_1)$ such that $n'_1 \approx n'_2$. 
Node bisimilarity: example
Node rank

- The rank of a node is the maximum distance to a leaf node.
- Rank is denoted by $\text{rank}(n)$.
  - Rank of leaf nodes is 0.
  - Rank of any other node $n$ is $1 + \max_{m \in \text{children}(n)} \text{rank}(n)$.

Fact

$m \approx n$ implies $\text{rank}(m) = \text{rank}(n)$. 
Main concepts

- Bisimilarity class is identified by unique identifier
  *Bisimilarity identifier*

- Set of bisimilarity identifiers of children
  *Bisimilarity family*

- Label, rank and bisimilarity family of a node
  *Bisimilarity decision value*

**Theorem**
Nodes have equivalent bisimilarity decision value if and only if they are bisimilar equivalent
**Algorithmic outline**

**Require:** Directed acyclic graph $G = \langle N, E, I \rangle$.  
**Ensure:** Bisimulation partitions of $N$.  

1. Sort nodes on rank  
2. For rank $r = 0$ to maximum rank do  
3. Determine bisimilarity decision value of nodes with rank $r$  
4. Sort on bisimilarity decision value  
5. Assign unique bisimilarity identifier to each group  
6. Send assigned bisimilarity identifiers to parent nodes  
7. End for
Algorithm: example

![Diagram of cities and locations]
Practical details

- Use external memory string sorting for grouping
- Use time-forward processing

**Theorem**

We can compute the bisimilarity equivalence classes of a DAG in \( O(\text{SORT}(|N| + |E|)) \) IOs
Specializations for XML indexing

- We can construct the 1-index in $O(\text{SORT}(|N|))$ IOs
- We can construct the $A(k)$-index in $O(\text{SORT}(k|N|))$ IOs
Implementation

- C++ implementation for DAGs
- C++ implementation for 1-index on XML documents
- On top of the STXXL library
  - Provides priority queue and sorting
  - No string sorting; sorting on hash values instead
Optimization

During rank partitioning we can already partition better

- Include labels
- Partition on ‘structural summaries’

Result: during bisimulation partitioning each group of investigated nodes is smaller.
Bisimulation partitioning of DAGs

Running time per size [s]

IOs per size

Nodes \( \times 10^9 \)

No pre-grouping during ranking

Pre-group nodes during ranking based on hash
Performance with respect to internal memory

![Graph](image_url)

- Running time per size [s]
- IOs per size

Memory size [MB]

- $10^{-6}$
- $10^{-5}$
- $10^{-4}$
- $10^{-3}$

$2^4$ $2^5$ $2^6$ $2^7$ $2^8$ $2^9$ $2^{10}$ $2^{11}$
XML Specialization: the 1-Index

![Graph showing running time and IOs per size against document size.]

- External memory bisimulation partitioning
- External memory 1-index construction
Results

- IO efficient bisimulation partitioning on DAGs
- Specializations for practical XML indices
- Open source implementations in C++
- Empirical validation
Future work

- Generalizing bisimulation partitioning
- Partition maintenance
- Practical output formatting
http://jhellings.nl