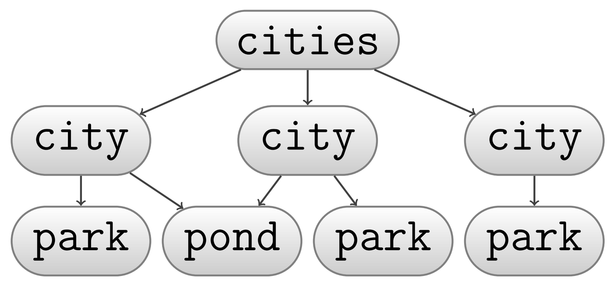
Efficient External-Memory Bisimulation on DAGs

Jelle Hellings[1], George H. L. Fletcher[2], and Herman Haverkort[2]
[1] *Hasselt University and transnational University of Limburg*
[2] *Eindhoven University of Technology*

Introduction

Graphs are fundamental structures which arise in numerous areas: social networks, biological and biomedical ontologies, scientific workflows, business process models, etc.



A graph describing points of interests in some cities.

Graph data sets are typically huge, compression techniques are necessary in order to support efficient analysis of the resulting graphs. One popular method to perform compression uses the concept of bisimulation partitioning.

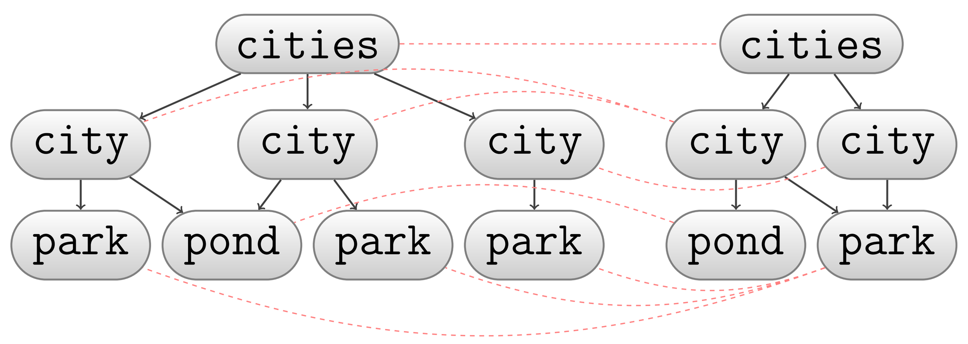
So far performing bisimulation partitioning has however been a very time-consuming step, in particular when the original graph is so large that it does not fit into the computer's main memory but must be stored on disk. Existing algorithms then spend most of their time transferring data back and forth between main memory and disk, which is extremely costly.

Bisimulation

Bisimulation is an equivalence relation relating nodes that 'have the same behaviour'. Formally DAG 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;
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 $n_1' \approx n_2'$.

Often one can compress a graph significantly by *bisimulation partitioning*, replacing each group of bisimilar nodes in the graph by a single node. The resulting graph retains much of the original graph's structure.



A directed acyclic graph on the left and the graph compressed using bisimulation partitioning on the right. The dotted red lines relate bisimilar nodes.

External memory bisimulation

We can give each bisimilarity cluster a unique identifier called the *bisimilarity identifier*. The *bisimilarity family* of a node is the set of bisimilarity identifiers of the partitions to which its children belong. The *bisimilarity decision value* of a node is the combination of its label, rank and bisimilarity family.

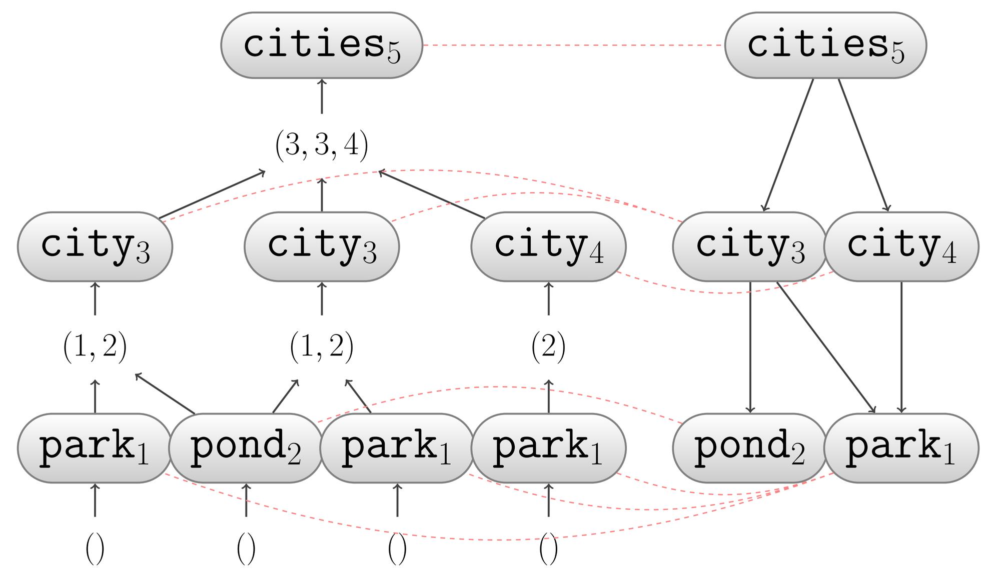
Theorem
Nodes in the same bisimilarity cluster have the same bisimilarity decision value.

This observation leads to the following algorithm; whereby N is the set of nodes, E is the set of edges and l is a node-labeling function.

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

Sort nodes on rank
for rank $r = 0$ to maximum rank **do**
  Determine bisimilarity decision value of nodes with rank r
  Sort on bisimilarity decision value
  Assign unique identifier to each group of bisimilarity decision values
  Send assigned bisimilarity identifiers to parent nodes
end for

We use well known external-memory programming tools for some of the parts of the algorithm. For assigning ranks and sending bisimilarity identifiers to parent nodes we can use *time-forward processing* and for sorting on bisimilarity decision values we can use *string-sorting*.



Execution of the algorithm: showing what values are send to each node and the bisimilarity identifiers assigned to each node.

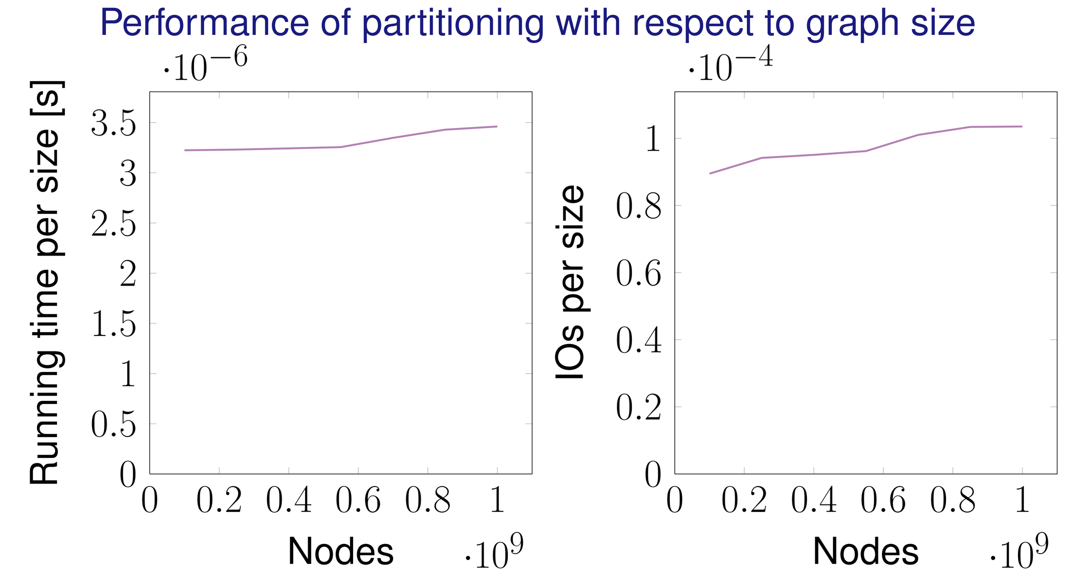
Theorem
We can compute the bisimilarity equivalence classes of a directed acyclic graph in $O(\textsc{Sort}(|N| + |E|))$.

Our methods can be specialized to construct several XML indices efficiently. These specializations include construct the 1-index in $O(\textsc{Sort}(|N|))$ and constructing the $A(k)$-index in $O(\textsc{Sort}(k|N|))$.

Our open-source implementation is available at `http://jhellings.nl`.

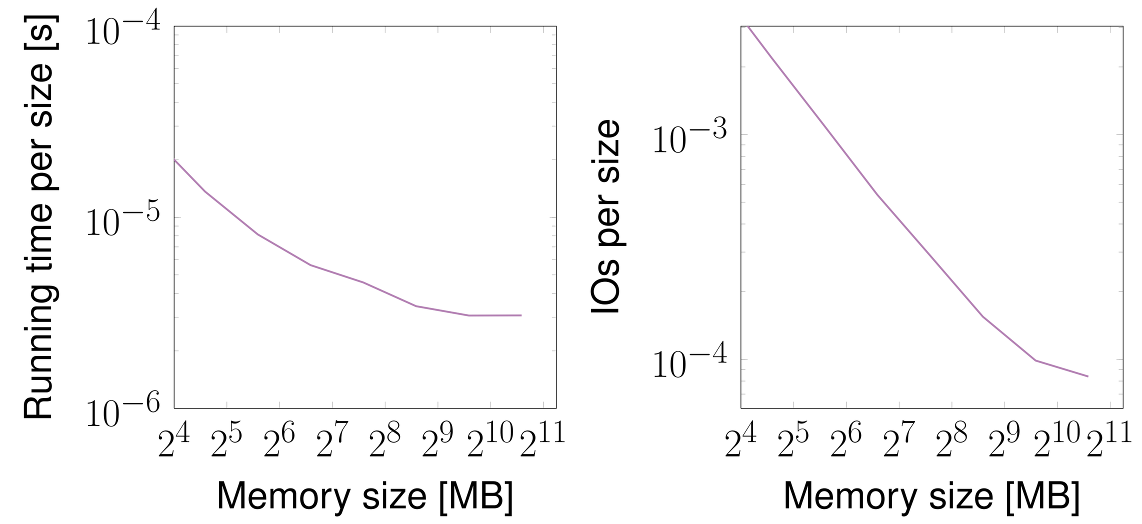
Empirical results

Performance of partitioning with respect to graph size



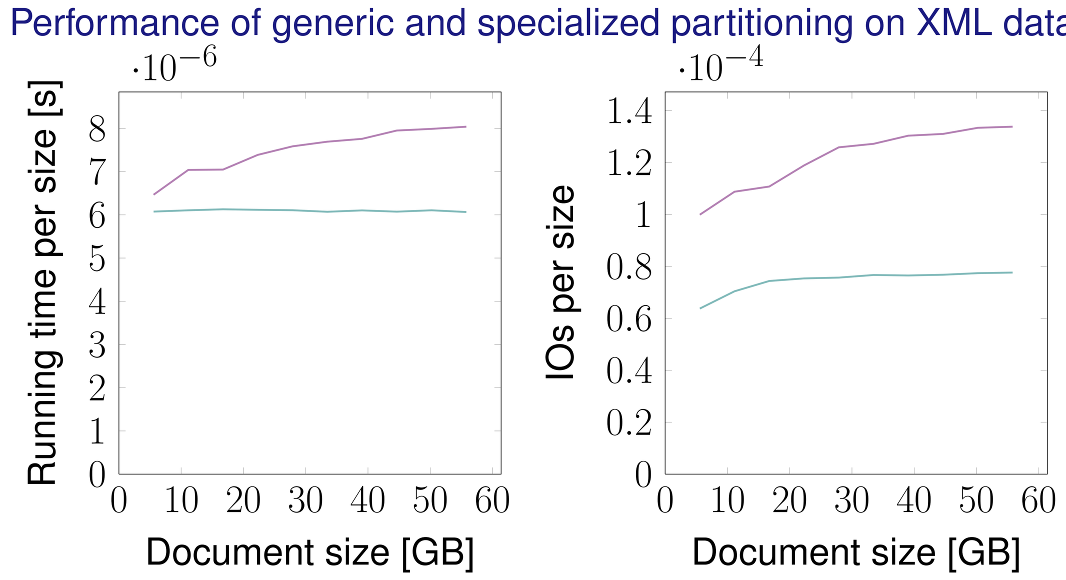
Running time and IOs performed per node and edge for bisimulation partitioning of directed acyclic graphs of large sizes.

Performance of partitioning with respect to available internal memory



Running time and IOs performed per node and edge for bisimulation partitioning of a directed acyclic graph in a restricted memory environment.

Performance of generic and specialized partitioning on XML data



Comparison between the performance of the general algorithm and a specialized (faster) 1-index construction algorithm on XML documents.

Future work

The conceptual and practical results developed pave the way for a variety of further investigations; including *generalizing bisimulation partitioning*, *partition maintenance*, and *practical output formatting*.