Efficient external-memory bisimulation on DAGs*

Jelle Hellings †
Databases and Theoretical Computer Science
Universiteit Hasselt

November 3, 2011

Directed acyclic graphs are widely used to model data. Examples range from XML documents to workflow logs. The notion of bisimilarity plays a central role in handling with these large data sets; grouping bisimilar nodes is often used as a first step to reduce the size of the data sets. Examples of the use of (variants of) bisimulation can be found in the 1-index[1] and the A(k)-index[2]. Both are index types used to index XML documents for efficient query evaluation.

Grouping bisimilar equivalent nodes is called bisimulation partitioning. A well-known bismulation partitioning algorithm is by Paige and Tarjan[3]. This algorithm has a running time of $\mathcal{O}(E \log N)$ with N the number of nodes and E the number of edges. Refinements[4] provide running time improvements for special cases such as DAGs.

The size of real world data sets often exceeds available main memory. Hence, there is a practical need for an efficient approach to computing bisimulation in external memory. The algorithm by Paige and Tarjan and its refinements are however very hard to adapt on graphs that do not fit in main memory; this due to their tendency to jump to parts of the graph without any particular order.

Motivated by these observations, we have developed the first efficient external-memory algorithm to compute the bisimulation partitioning of DAGs. Our algorithm has a worst-case IO-complexity of $\mathcal{O}(\textsc{Sort}(N + E))$, where $\textsc{Sort}(n)$ is the number of accesses to external memory needed to sort an input of size n.

We have also provided an implementation of this algorithm in C++ using the STXXL library[5]. This implementation has an expected IO-complexity of $\mathcal{O}(\textsc{Sort}(N + E))$ and experiments show that the implementation can easily handle directed acyclic graphs containing billions of nodes and edges.

A major application of our bisimulation partitioning algorithm is to construct bisimulation-based indices on XML documents. Thereby we can utilize the simple tree-structure of XML documents to our advantage. This results in an 1-index construction algorithm and implementation with worst case IO-complexity of $\mathcal{O}(\textsc{Sort}(N))$. For constructing the A(k)-index we provide an algorithm with worst case IO-complexity of $\mathcal{O}(\textsc{Scan}(k) + \textsc{Sort}(kN))$; whereby $\textsc{Scan}(n)$ is the number of accesses to external memory needed to read or write an input of size n.

Measurements on the implementation of the 1-index construction algorithm verify efficient performance on XML documents with a size up to 55.8GB and having billions of nodes and edges. These measurements also shows that the 1-index construction algorithm outperforms the general bisimulation partition algorithm on these XML trees.

The proposed bisimulation partitioning algorithms are simple enough for practical implementation and use, and open the door for further study of external-memory bisimulation algorithms. To this end, the full open-source implementation has been made freely available.

*See `http://jhellings.nl/projects/exbisim/` for more information on this topic.

†Supervised by George Fletcher and Herman Haverkort while completing my Master Thesis at the Eindhoven University of Technology.

[1] T. Milo and D. Suciu. Index structures for path expressions. In *Proc. Int. Conf. on Database Theory (ICDT)*, pages 277–295, Jerusalem, 1999.

[2] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting local similarity for indexing paths in graph-structured data. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, pages 129–140, 2002.

[3] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[4] A. Dovier, C. Piazza, and A. Policriti. An efficient algorithm for computing bisimulation equivalence. *Theor. Comput. Sci.*, 311(1-3):221–256, 2004.

[5] See `http://stxxl.sourceforge.net/`.