Immediate Dominators in Linear Time

An Elegant and Non-Amortized Algorithm
EXTENDED ABSTRACT

Marco T. Morazán

Seton Hall Uniersity

morazanm@shu.edu

Abstract

Categories and Subject Descriptors D.3.3 [*Programming Languages*]

Keywords Immediate Dominators, Algorithms

Computing dominators is a fundamental problem in the implementation of programming languages. Dominators are used in compiler optimizations involving loop-invariant computations and code motion [2]. Dominators also play a role in program transformation techniques involving static single assignment form [8], lambda lifting [11], and lambda dropping [10]. In addition to applications in programming languages, dominators are used in software testing to achieve good testing coverage [1], in VLSI testing to find faults [5], and in computational biology to study species extinction [3, 4].

Modern algorithms to compute dominators, have taken two major approaches: an equation-based approach and a spanning-tree-based approach. The equation-based approach, also known as the data-flow approach, aims to solve a system of recursive set-equations–one equation for each node in the call-graph. The spanning tree approach aims to exploit properties of the depth-first spanning tree to determine the dominator relationship. Both approaches have strived to develop "fast" and "practical" algorithms with varying degrees of success. Equation-based approaches have not developed an $O(N)$ algorithm, but have yielded algorithms that are elegant and that in practice are expected to run fast [7]. Approaches using spanning trees have developed an asymptotically optimal $O(N)$ algorithm [6]. This asymptotically optimal algorithm, however, is conceptually complex, difficult to explain, and difficult to implement.

Spanning-tree-based algorithms aim to build the dominator tree of a call-graph, G, by exploiting properties of, ST_G, its depth-firt spanning tree. Several spanning tree algorithms have been proposed in the search for an $O(n)$ algorithm [6, 9, 12, 13]. Of these algorithms, the best known is the almost linear algorithm developed by Langauer and Tarjan (LT) [13] which has served as the basis for other spanning-tree-based algorithms. The most recent refinement has been done by Buchsbaum et al. (B) obtaining an $O(n)$ algorithm [6]. These algorithms work using three conceptual steps:

1. Compute semidominators.
2. From semidominators compute relative dominators.
3. From relative dominators compute immediate dominators.

The LT and B algorithms compute each of these steps differently, but both are based on visiting and processing nodes.

This article presents a new spanning-tree-based linear-time algorithm that eliminates the need to compute semidominators and relative dominators. Its novel approach is based on processing edges, not nodes, during a traversal of the nodes in reversed order from the spanning tree traversal order. A forest-like data structure is maintained that dynamically tracks dominator information across edges when the tail of an edge is visited, not when the head is visited. The immediate dominator of a node is not computed until it is its turn to be connected within the forest and its immediate dominator is decided solely based on the heads of the edges for which it is the tail. At each step, the forest consists solely of trees of height 0–nodes not yet connected–and trees of height 1–the root of such trees is the largest node so far from which its children are reachable. This property guarantees that each edge can be processed in constant time. Therefore, the algorithm is $O(v + e)$, or simply $O(n)$, where v is the number of nodes and e is the number of edges. Furthermore, the algorithm is remarkably simple, elegant, and easily implemented.

References

[1] H. Agrawal. Efficient Coverage Testing Using Global Dominator Graphs. In *Proceedings of the SIGPLAN/SIGSOFT Workshop on Program Analysis For Software Tools and Engineering*, pages 11–20, 1999.

[2] A. V. Aho and J. D. Ullman. *Principles of Compiler Design*. Addison-Wesley Publishing Company, 1979.

[3] S. Allesina and A. Bodini. Who Dominates Whom in the Ecosystem? Energy Flow Bottlenecks and Cascading Extinctions. *Journal of Theoretical Biology*, 230(3):351–358, 2004.

[4] S. Allesina, A. Bodinia, and C. Bondavalli. Secondary Extinctions in Ecological Networks: Bottlenecks Unveiled. *Ecological Modelling*, 194(1-3):150–161, 2006.

[5] M. E. Amyeen, W. K. Fuchs, I. Pomeranz, and V. Boppana. Fault Equivalence Identification Using Redundancy Information and Static and Dynamic Extraction. In *Proceedings of the 19th IEEE VLSI Test Symposium*, pages 124–130, 2001.

[6] A. L. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. E. Tarjan, and J. R. Westbrook. Linear-Time Algorithms for Dominators and Other Path-Evaluation Problems. *Siam J. Compt.*, 38(4):1533–1573, 2008.

[7] K. D. Cooper, T. J. Harvey, and K. Kennedy. A Simple, Fast Dominance Algorithm. *Software Practice and Experience*, 4, 2001.

[8] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Efficiently Computing Static Single Assignment Form and the Control Dependence Graph. *ACM Trans. Program. Lang. Syst.*, 13:451–490, October 1991. ISSN 0164-0925.

[9] D. Harel. A Linear Time Algorithm for Finding Dominators in Flow Graphs and Related Problems. In *Proc. of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 185–194, 1985.

[10] O. Danvy and U. P. Schultz. Lambda-Dropping: Transforming Recursive Equations into Programs with Block Structure. *Theoretical Computer Science*, 248(1–2):243–287, 2000.

[11] M. T. Morazán and U. P. Schultz. Optimal Lambda Lifting in Quadratic Time. In O. Chitil, editor, *Implementation and Application of Functional Languages*, volume 5083 of *Lecture Notes in Computer Science*, pages 37–56. Springer Verlag, 2008.

[12] Stephen Alstrup and Dov Harel and Peter W. Lauridsen and Mikkel Thorup. Dominators in Linear Time. *SIAM Journal on Computing*, 28 (6):2117–2132, 1999.

[13] Thomas Lengauer and Robert Endre Tarjan. A Fast Algorithm for Finding Dominators in a Flowgraph. *ACM Transactions on Programming Languages and Systems* , 1(1):121 – 141, 1979.