Abstract

Immediate Dominators in Linear Time
An Elegant and Non-Amortized Algorithm
EXTENDED ABSTRACT

Marco T. Morazán
Seton Hall University
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 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-first 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 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


[Copyright notice will appear here once 'preprint' option is removed.]


