A survey of Dominator construction algorithms

Chinmay Chandak, Tapan Sahni

cs13b1011@iith.ac.in, cs13b1030@iith.ac.in
Indian Institute of Technology, Hyderabad

December 9, 2016

Abstract

This survey deals with the work done in the field of dominator tree construction. We will try to comment on various algorithms, their complexity and the recent improvements on Dominator construction.

1 Introduction

The Lengauer-Tarjan [1] algorithm (classically also known as the LT algorithm), first published in 1979, is the best known and the most widely known algorithm for fast dominance. It was the breakthrough algorithm for finding immediate dominators in a CFG after a series of efforts made to improve the time bound for the problem, some of the prominent ones being: Aho and Ullman, Purdom and Moores (this is actually a mistake in giving credit! This interesting point in the history of dominator construction algorithms has been discussed later on) quadratic time bound iterative algorithm (complexity is $\mathcal{O}(nm)$) which involved making a naive pass for each vertex to compute the set of nodes it dominated; another bit-manipulative algorithm proposed by Aho and Ullman, but this also had a quadratic time complexity.

The simplistic implementation of the LT algorithm has a time complexity of $\mathcal{O}(m \log n)$, while a more sophisticated implementation is able to reduce the complexity to $\mathcal{O}(m\alpha(m, n))$, where $\alpha(m, n)$ is the slow-growing functional inverse of the Ackermann Function. The LT algorithm involves a pre-order depth-first search to order the nodes, a semi-dominator computation step and then the immediate dominator computation stage.

2 Dominator Computation

We compute semi-dominator for every vertex w as follows

$$sdom(w) = \{v \mid there\ is\ a\ path\ v = v_0,\ v_1,\ \dots,\ v_k = w\ such\ that\ v_i > w\ for\ 1 \leq i \leq k-1\}$$

As mentioned in the paper, the computation of semi-dominators is a intermediate step for $idom(w)$ computation because of a couple of properties. If w $\neq$ r (r is the entry block of the flowgraph), then $sdom(w)$ is a proper ancestor of w in T (the MST of G), and $idom(w)$ is (not necessarily proper) an ancestor of $sdom(w)$. Also, if we replace the set of non-tree edges of G by the set of edges $\{(sdom(w), w) \mid w \in V and\ w \neq r\}$, then the dominators of vertices in G are unchanged. Thus, once we know the MST and the semi-dominators, we can compute the dominators.

During the computation of immediate dominator for every vertex, the algorithm maintains a Union-Find data structure for the set of processed vertices and edges. This helps us in calculating the immediate dominator (given by theorem above) in O(logn).The Union-Find data structure has 2 operations

$$LINK(u, v): \ Add\ the\ edge\ (u, v)\ to\ the\ forest.$$

$$EVAL(u): \ It\ returns\ the\ vertex\ u\ on\ the\ path\ r \xrightarrow{+} v\ with\ minimum\ semi-dominator.$$

Once the spanning tree has been constructed, the LT algorithm uses path compression to carry out the EVAL step. The path compression uses the Union-Find data structure: There is an assignment of an ancestor to each vertex and simultaneous updations to a label array based upon the semi-dominators.

3 Complexity

The running time of the algorithm is $\mathcal{O}(m+n)$ for the DFS plus the time for $n-1$ LINKs and $m+n-1$ EVALs. The running time of the aforementioned LINKs and EVALs is $\mathcal{O}(m \log n)$, and hence the overall complexity is also $\mathcal{O}(m \log n)$.

The time-bound can be reduced to $\mathcal{O}(m\alpha(m, n))$ if the LINK function is changed so that the path compression is only performed on balanced trees (in the simple version skewed trees form the worst-case analysis). Thus, the LINK operation now has the work of not only adding an edge in a forest, but make sure that after the edge is added, the height of the tree is balanced by adjusting the sizes of each of the subtrees (making the root of the smaller subtree equal to the new root), and propagating the changes till the bottom. Thus, it tries to maintain the height of the subtrees while adding an edge by making the root of the smaller subtree (i.e height) as our new root.This makes the total complexity of $n-1$ LINKs and $m+n-1$ EVALs, and hence the overall algorithms complexity to be $\mathcal{O}(m\alpha(m, n))$.

4 Recent Improvements

The advent of static single assignment form (SSA) has rekindled interest in dominance and related concepts. New algorithms for several problems in optimization and code generation have built on dominance. An extensive amount of work has been done after the publication of this paper to improve the speed of the computation of the dominators. First, lets revisit the series of works that have made progress in the same direction.

In 1970, Allen , in her work on control-flow analysis, formulated the dominance computation as a global data-flow problem. In a 1972 paper with Cocke [2], she showed an iterative algorithm to solve these equations in $\mathcal{O}(n^2)$. Hecht and Ullman [3], in 1975, the showed that a reverse postorder iterative scheme solves these equations in a single pass over the CFG for reducible graphs. Both Hechts book and Aho and Ullmans Dragon book present the iterative algorithm for dominance without restricting it to reducible graphs. Unfortunately, Aho and Ullman mistakenly credit the algorithm to Purdom and Moore, rather than to Allen and Cocke.

Aho and Ullman approached the problem from another direction. Their algorithm takes the graph and successively removes nodes. Any nodes in the entire graph (rather than a single path) that cannot subsequently be reached are dominated by the removed node. This algorithm works in quadratic time, in the number of nodes. With Hopcroft, they improved this time bound to $\mathcal{O}(m \log m)$ for reducible graphs, where m is the number of edges in the graph, by using an efficient method of finding ancestors in trees.

In 1974, Tarjan [4] proposed an algorithm that uses depth-first search and union-find to achieve an asymptotic complexity of $\mathcal{O}(n \log n + m)$. Five years later, Lengauer and Tarjan [1] built on this work to produce an algorithm with almost linear complexity (slightly super-linear complexity due to the inverse Ackermann function). This is the classic LT algorithm which has been discussed above.

In 1983, Gabow and Tarjan [5] founded a linear-time Union-Find algorithm for disjoint sets. In this, the tree, say T resulting from the Union operations is known in advance; thus there is an additional constraint on the $Union(v, w)$ function being successful only if the edge (v,w) is present in T. The linear time is achieved by using memoization of Union-Find within small microtrees of size $\mathcal{O}(n \log n)$. In 1985, Harel [6] proposed an offline linear time algorithm for computing immediate dominators using this faster version of Union-Find. He basically tried to convert the linear Union-Find algorithm into the $EVAL$-$LINK$ algorithm. However, this introduced the construction of pseudo-dominators, which were used to then compute semi-dominators; this faced the problem of complicated case analysis and correctness testing issues when the order of $LINK$ operations did not follow a specific pattern.

In 1998, Buchsbaum et al. [7] presented a linear-time algorithm based on Lengauer-Tarjan. Their algorithm is essentially a divide-and-conquer algorithm that groups the bottom nodes of the depth-first search tree into microtrees. By solving the local problem for the microtrees, they can perform the subsequent union-find operations needed for Lengauer-Tarjan in linear time. Thus, their algorithm has better asymptotic complexity than Lengauer-Tarjan. However, they state that their algorithm runs 10 to 20 % slower than Lengauer-Tarjan on real flowgraphs.

In 1999, Alstrup et al. [8] presented a more correct version of Harels work which involved using Williams Q-Heaps to memoize $EVAL$ instructions. This also added an assumption that we know that the order of the $LINK$ instructions would be in reverse $DFS-order$. The authors posit that the actual complexity of the algorithm, using practical data structures, is $\mathcal{O}(m + n \log \log \log n)$. The paper does not provide any experimental data that shows the algorithms measured behavior versus Lengauer-Tarjan.

A major controversy arose in 2001, when Cooper stated that a simple iterative scheme for this problem can be engineered to outrun the more complicated techniques built on union-find, despite the higher asymptotic complexity of the iterative data-flow approach. From this perspective, Lengauer-Tarjan is an appropriate comparison; the other algorithms expend additional effort to speed up the union-find operations when, in fact, the asymptotic advantage from union-find does not show up until the problem size becomes unrealistically large. While Lengauer-Tarjan has faster asymptotic complexity, it requires unreasonably large CFGs on the order of 30,000 nodes before this asymptotic advantage catches up with a well-engineered iterative scheme. Cooper thus stated that since the iterative algorithm is simpler, easier to understand, easier to implement, and faster in practice, it should be the technique of choice for computing dominators.

According to Cooper [11], to compute dominance information with data-flow techniques, we can pose the problem as a set of data-flow equations and solve them with a reverse-postorder iterative algorithm. Bit vectors wasted time on the intersections of sparsely populated sets for large graphs, and also in copying sets from one node to another. The final engineered algorithm implemented a sub-algorithm: Two-finger algorithm which made the intersections of sparse sets fast by implementing a step similar to the merge stage of the merge-sort algorithm. The complexity of this engineered iterative framework was $\mathcal{O}(d(N + ED))$ where d is a constant which is observed to have a relatively small value, while D is the size of the largest DOM set.

In 2004, Georgiadis [9] made a thorough comparison of the LT and Cooper algorithms and proposed a new hybrid algorithm known as SEMI-NCA. It simplifies the link-eval stage by ensuring that vertices are processed in preorder during the computation of idoms from sdoms. The complexity of this algorithm is $\mathcal{O}(n^2 + m \log n)$. Georgiadis showed that the hybrid algorithm and the LT algorithm performed better than the iterative algorithm (hence opposing the conclusion by Cooper), thus reinforcing the controversy even further.

In 2005, Buchsbaum et al. [7] published another algorithm simpler than previous linear-time algorithms: rather than employ complicated data structures, they combine the use of microtrees (the partitioning of the DFS-tree is also based on certain constraints), topological sort and memoization with new observations on a restricted class of path compressions. Compared to the standard, slightly superlinear algorithm of Lengauer and Tarjan, which has much less overhead, the algorithm runs 10 - 20% slower on real flowgraphs of reasonable size and only a few percent slower on very large flowgraphs.
In 2008, Buchsbaum et al. [10] combined all the above works to work-out a linear-time solution to some of the most common problems in control flow theory: finding dominators, path evaluation, finding nearest ancestor, dominance frontier etc. A lot of work is currently in

progress which aims at making the construction of the dominator tree and solving its related problems with simpler, faster, and easier-to-understand implementations.

Very recently, Gabow (2010) and Fraczak et al [12] (2013). presented linear-time algorithms that are based on a different approach, and require only simple data structures and a data structure for static tree set union. Gabows algorithm uses the concept of minimal set posets, while the algorithm of Fraczak et al. uses vertex contractions. Ramalingam and Repsl [13] presented an incremental algorithm for finding dominators in an acyclic graph. Their algorithm uses a data structure that computes nearest common ancestors in a tree that grows by leaf additions. For this incremental nearest common ancestors problem, Gabow gave an $\mathcal{O}(m)$ -time RAM algorithm, and Alstrup and Thorup [8] gave an $\mathcal{O}(m \log \log n)$-time pointer machine algorithm. These results give implementations of the Ramalingam-Reps algorithm that run in $\mathcal{O}(m)$ time on a RAM and in $\mathcal{O}(m \log \log n)$ time on a pointer machine. Ramalingam showed how to reduce the problem of computing dominators in an arbitrary flow graph to computing dominators in an acyclic graph. His reduction uses static-tree disjoint set union, so it runs in $\mathcal{O}(m\alpha(m, n))$ time on a pointer machine, and in $\mathcal{O}(m)$ time on a RAM. Therefore, the combination of any linear-time algorithm for computing dominators in an acyclic graph with Ramalingams reduction gives a linear-time RAM algorithm for computing dominators in a general graph.

All the iterative algorithms published so far have proved that they are not competitive with the more sophisticated algorithms based on the LT algorithm. Thus, production compilers like GCC and LLVM use the more efficient version of the classic Lengauer-Tarjan algorithm which remains the state-of-the-art for fast dominance queries and dominator/post-dominator tree construction. This has an asymptotic complexity of $\mathcal{O}(m + n \log n)$. The amortized quasi-linear complexity can be obtained using path compression and balanced tree properties as described before, but it does not seem to be implemented in practice in some compilers.

References

[1] Lengauer, Thomas, and Robert Endre Tarjan, A fast algorithm for finding dominators in a flowgraph *ACM Transactions on Programming Languages and Systems (TOPLAS) 1.1 (1979): 121-141.*

[2] F. E. Allen and J. Cocke, Graph theoretic constructs for program control flow analysis, *IBM T.J. Watson Research Center, Tech. Rep. IBM Res. Rep. RC 3923, 1972.*

[3] M. S. Hecht and J. D. Ullman,Characterizations of reducible flow graphs, *Journal of the ACM, vol. 21, no. 3, pp. 367375, 1974.*

[4] R. E. Tarjan, Finding dominators in directed graphs, *SIAM J. Comput., 3 (1974), pp. 6289.*

[5] H. N. Gabow and R. E. Tarjan, A linear-time algorithm for a special case of disjoint set union, *J. Comput. System Sci., 30 (1985), pp. 209221.*

[6] D. Harel, A linear time algorithm for finding dominators in flow graphs and related problems, *in Proceedings of the 17th ACM Symposium on Theory of Computing, 1985, pp. 185194.*

[7] A. L. Buchsbaum, H. Kaplan, A. Rogers, and J. R. Westbrook, A new, simpler linear-time dominators algorithm, *ACM Trans. Programming Languages and Systems, 20 (1998), pp. 12651296; Corrigendum, 27 (2005), pp. 383387.*

[8] S. Alstrup, D. Harel, P. W. Lauridsen, and M. Thorup, Dominators in linear time, *SIAM J. Comput., 28 (1999), pp. 21172132.*

[9] L. Georgiadis, Linear-Time Algorithms for Dominators and Related Problems, *Ph.D. thesis, Department of Computer Science, Princeton University, Princeton, NJ, 2005.*

[10] A. L. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. E. Tarjan, and J. Westbrook,Linear-time algorithms for dominators and other path-evaluation problems, *SIAM J. Comput., 38(4):15331573, 2008. doi:10. 1137/070693217.*

[11] K. D. Cooper, T. J. Harvey, and K. Kennedy,A simple, fast dominance algorithm,*Rice Computer Science, Tech. Rep. TR-06-38870, 2006.*

[12] W. Fraczak, L. Georgiadis, A. Miller, and R. E. Tarjan,Finding dominators via disjoint set union, *Journal of Discrete Algorithms, vol. 23, no. 0, pp. 220, 2013.*

[13] G. Ramalingam and T. Reps, An incremental algorithm for maintaining the dominator tree of a reducible flowgraph, *in Proc. 21th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, 1994, pp. 287296.*

[14] P. W. Purdom and E. F. Moore, Immediate predominators in a directed graph, *Comm. ACM, 15 (1972), pp. 777778.*

[15] S. Alstrup and P. W. Lauridsen, A simple and optimal algorithm for finding immediate dominators in reducible graphs, *Dept. of Computer Science, U. Copenhagen, Tech. Rep. DIKU TOPPS D-261, 1996.*

[16] L. Georgiadis and R. E. Tarjan, Finding dominators revisited, *in Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2004, pp. 862871.*

[17] L. Georgiadis, R. E. Tarjan, and R. F. Werneck, Finding dominators in practice, *J. Graph Algorithms Appl., 10 (2006), pp. 6994.*

[18] L. Georgiadis, N. Parotsidis, Dominators in directed graphs: a survey of recent results, applications, and open problems, *Proc. 2nd Intl Symp. Comput. Inform. Math., pp.15 20 (2013)*