A simple dynamic algorithm for maintaining a dominator tree

Stephen Alstrup E-mail:stephen@diku.dk
Peter W. Lauridsen E-mail:waern@diku.dk
Department of Computer Science
University of Copenhagen

Abstract

We present a simple algorithm which maintains the dominator tree for an arbitrary flow graph during a sequence of i edge insertions interspersed with q queries as "does x dominate y?". The complexity of the algorithm is $O(q + m * \min\{i, n\})$, where m and n respectively are the number of edges and nodes in the flow graph after the $i'th$ edge insertion. This improves the former best results from $O(q * \log n + m * i * \log n)$ in [13] and $O(q * n + m * i)$ in [6]. Furthermore, we show that the complexity of our algorithm for a single edge insertion is bounded by those nodes which no longer will be dominated by the same set of nodes.

1 Introduction

Dominator trees are used in control flow analysis [1, 7, 8, 9]. Algorithms for finding dominator trees for control flow graphs are given in [10, 11, 12] and the algorithm in [10] is linear. Recently dynamic algorithms [4, 6, 13][1] have been presented for maintaining dominator trees, but the complexities of these algorithms are worse or just as good as recomputation from scratch. In this paper we present an $O(q + m * \min\{i, n\})$ algorithm which maintains the dominator tree for a flow graph during a sequence of i edge insertions interspersed with q queries, where m and n respectively are the number of edges and nodes in the flow graph after the $i'th$ edge insertion. Our algorithm is the first bounded algorithm for the problem. Let W be the set of nodes, for which the depth in the dominator tree changes after insertion of an edge and let F be the set of outgoing edges incident with the nodes in W, then the complexity for a single edge insertion is $O(|W| + |F|)$. At the end of this paper we show the strongness of the bounded algorithm; we give an example, where our algorithm uses constant time and the algorithm in [6] uses linear time. The algorithm is simple and can be implemented on a pointer machine [14] with the same complexity for a series of insertions, however the complexity for a single edge insertion will be $O(|W| + |F| + \log\log n)$. The rest of this paper is organized as follows. In section 2 we describe the notation used in the paper. In section 3 our main observations are given. The algorithm is presented in section 4. In section 5 we give a complexity analysis and in section 6 we give additional remarks.

2 Notation

A control flowgraph $CFG(V, E, s)$ is a directed graph with a start node s. If a path exists from x to y we write $x \rightsquigarrow y$. The dominance relation is only defined for the

[1] The algorithms in [4, 13] are restricted to reducible graphs.

nodes reachable from s. Node x dominates y *iff* all paths from s to y pass through x. If x dominates y and $x \neq y$ then x strictly dominates y. The dominance relation is reflexive and transitive, and can be represented by a tree, called the dominator tree. If y is the parent of x in the dominator tree, then y immediately dominates x, denoted as $idom(x) = y$. The nearest common ancestor (nca) for two nodes is the node in the dominator tree with largest depth among those nodes that dominates both nodes. Notice that a node y dominates a node x *iff* $y = nca(x, y)$. For any function f (e.g. $idom$), we use f_{old} and f_{new} respectively for the value before and after insertion of an edge.

3 Observations

One of the reasons why former algorithms have been unbounded is found in the following cited observation.

"The inherent difficulty in the dominator update problems lies in the 'non-locality' of domination $\cdots$ Adding or removing a single flow graph edge - an act which can add or remove large numbers of paths - can thus affect domination between nodes arbitrarily far from the altered edge".

This observation is given in [6] as a citation from [4]. In [13] we find an observation very alike. One of our observations to get a bounded algorithm is that after insertion of an edge (x, y), a path exists from y to any node which changes immediate dominator, on which all nodes change depth in the dominator tree. (see lemma 1).

In the rest of this section we state the observations which are used in the algorithm presented in the next section. In all the observations (x, y) is the inserted edge and z is $nca(x, y)$.

Proposition 1 *Let v be a node which changes immediate dominator. We have the following properties*

1. *The immediate dominator for v is z after the insertion.*
2. *The node $idom(v)$ strictly dominates y before the insertion.*
3. *The node z strictly dominates $idom(v)$ before the insertion.*

Proof. see [6, 13]. $\Box$

Lemma 1 *If a node v changes depth in the dominator tree then a path exist from y to v where all nodes on the path change depth.*

Proof. Since v changes depth a node v' exists which dominates v and changes immediate dominator. According to proposition 1 $idom(v') \neq z$ is on any path from z to y. As v' changes immediate dominator a path P_1 from y to v' must exist, which avoids $idom(v')$. Since v' dominates v another path P_2 exists from v' to v which also avoids $idom(v')$. We now claim that the path $P = P_1 \bigcup P_2$ only includes nodes which change depth. Assume otherwise and let w be the first node on P, which does not change depth. Since the nodes on the path from y to w change depth, $idom(w)$ can not be one of these nodes. If $idom(w)$ was on a path from z to y, not including z, w would change depth, as a path from z through x and y to w and another path form z through $idom(w)$ to w would exist in the new graph. Consequently $idom(w)$ is on a path from s to z, and therefore a path exists from s through $idom(w)$ to w which avoids $idom(v')$. Now since v' changes dominator $w \neq v'$. The node w can not be on P_1 since a path exists from s through w to v', which avoids $idom(v')$. Likewise if w was on P_2 a path would exist from s through w to v, which avoids v' contradicting that v' dominates v. $\Box$

Corollary 1 *If a node v changes immediate dominator a path exists from y to v which does not include $idom(v)$ and on which all nodes change depth.*

Proof. Set $v = v'$ in the proof for lemma 1.□

Lemma 2 *If a path $v \rightsquigarrow w$ exists then $idom(w)$ is on the path or $idom(w)$ strictly dominates v.*

Proof. Assume $idom(w)$ does not strictly dominate v. Then a path $P_1 = s \rightsquigarrow v$ must exist, which avoids $idom(w)$. If the path P_2 from v to w does not include $idom(w)$ the path $P_1 \bigcup P_2$ is a path from s to w which does not include $idom(w)$, contradicting the definition of dominance. □

In order to determine the set of nodes which change immediate dominator we generalize proposition 1.

Proposition 2 *Let the nodes in the graph be numbered by their depth in the dominator tree.*
A node v changes immediate dominator iff

1. *z strictly dominates $idom(v)$.*
2. *$idom(v) < \max\{x | P \in \mathcal{P} \wedge x = \min\{u | u \in P\}\}$, where $\mathcal{P}$ is the set of all paths from y to v and $\mathcal{P} \neq \emptyset$.*

Proof.
$\Rightarrow$:

1. Follows directly from proposition 1.
2. Because v changes immediate dominator a path P from y to v exists, where $idom(v) \notin P$. Let $w = \min\{u | u \in P\}$ and assume $idom(v) \geq w$. Since $idom(v) \notin P$, $idom(v) \neq w$. Therefore a path from s to w exist, which avoids $idom(v)$. This implies that $idom(v)$ should be on any path from w to v, contradicting that $idom(v) \notin P$.

$\Leftarrow$: According to 2 a path P_2 from y to v exists, where $idom(v) < \min\{u | u \in P_2\}$, which means that $idom(v) \notin P_2$. By lemma 2 $idom(v)$ strictly dominates y. This and the fact that z strictly dominates $idom(v)$, means that a path P_1, from s through x to y exists, which does not include $idom(v)$, in the new graph. The path $P_1 \bigcup P_2$ is therefore a path from s to v, which avoids $idom(v)$.□

Corollary 2 *In the second condition of proposition 2 it is sufficient that $idom(v) < \max\{x | P \in \mathcal{P} \wedge x = \min\{u | u \in P \wedge$ u dominates $y\}\}$*

Proof. Inserting the restricted set in the proof for proposition 2 has no effect on the correctness. □

Lemma 3 *If z strictly dominates $idom(v)$, idom(v) strictly dominates y and v does not change immediate dominator, any path from y to v includes $idom(v)$.*

Proof. Assume a path exists from y to v which does not include $idom(v)$. This implies that a path from z through x and y to v not including $idom(v)$ exists in the new graph. □

Lemma 4 *If v changes immediate dominator a path P from y to v exists for which $\min\{u | u \in P \wedge$ u dominates $y\} > idom(v)$ and all nodes change depth.*

Proof. From corollary 1 a path P exists from y to v on which all nodes change depth, where $idom(v) \notin P$. Let $w = \min\{u | u \in P\}$ and assume that $w \leq idom(v)$. This implies that there is a path from s to w, which does not include $idom(v)$. Therefore $idom(v)$ is on all paths from w to v contradicting that $idom(v) \notin P$. □

4 Algorithm

In this section we describe how to update the dominator tree after insertion of an edge (x, y). To avoid trivial cases we assume that both x and y are reachable from s. In the first subsection we describe how to find nodes which change depth in the dominator tree and in the next subsection how to update the dominator tree.

4.1 Detection

The basic idea behind the algorithm is to search from the node y through nodes which change depth, in order to find nodes which change immediate dominator. In each iteration a search node is chosen and a search set is made, initially only containing the search node. The search is carried out from nodes in the search set until it is empty. Each time a node v is visited we distinguish between two cases:

1. If v changes immediate dominator by corollary 2, v is labeled with 'mark' and 'newidom'. If v does not dominate y it is included in the search set.
2. If v does not change immediate dominator then by lemma 2 and 3 $idom(v)$ has already been visited. Therefore we can check whether v changes depth by checking whether $idom(v)$ is marked. If v changes depth it is labeled with 'mark' and included in the search set.

During the search, nodes are only visited if they are not already marked. Initially we choose y as the search node. When the search set is empty, we choose the next search node by finding the first ancestor of y in the dominator tree, which is labeled with 'newidom'[2], if such a node exists. During this process we continuously update a variable 'mindepth', which contains the minimum depth of nodes which dominate y and change immediate dominator. The process stops when 'mindepth' is the depth of the search node after the search from the search node has been made. The algorithm is given in detail below.

```
1.  $z := nca(x, y)$;
2.  If $idom(y) \neq z$ then begin
3.    $mindepth := depth(y)$;
4.    $newsearchnode := y$;
5.    $newidom(y)$; $mark(y)$;
6.    repeat
7.      $searchnode := newsearchnode$;
8.      $M := \{searchnode\}$;
9.      repeat
10.       $M := M \backslash \{w\}$;
11.       $N := \{v | (w, v) \in E \wedge$ not $mark(v)$ $\}$
12.       For every $v \in N$ do begin
13.         if $z$ strictly dominates $idom(v) \wedge depth(idom(v)) > depth(searchnode)$
            then begin
```

[2] This search will only include nodes which change depth, since a node which dominates these nodes changes immediate dominator.

```
14.           $mark(v)$; $newidom(v)$;
15.           if $v$ dominates $y$ then $mindepth := \min\{mindepth, depth(v)\}$;
16.           else $M := M \bigcup \{v\}$
17.         end
18.         else if $mark(idom(v))$ then begin
19.           $mark(v)$; $M := M \bigcup \{v\}$
20.         end
21.       end;
22.     until $M = \emptyset$;
23.     if $depth(newsearchnode) \neq mindepth$ then repeat
24.       $newsearchnode := idom(newsearchnode)$
25.     until $newidom(newsearchnode)$;
26.   until $depth(searchnode) = mindepth$
27. end;
```

Algorithm

Lemma 5 *The algorithm correctly detects the nodes which change immediate dominator.*

Proof (sketch). The construction of the algorithm implies that all nodes which change depth are visited. Therefore by lemma 4 all nodes which change immediate dominator are visited. To establish the correctness we need to argue that the path P found from y to a node v, which changes immediate dominator, is a path which satisfy the conditions of lemma 4. Since all nodes on P change depth we only need to argue that $\min\{u | u \in P \wedge \text{ u dominates } y\} > idom(v)$. If this was not the case the construction of the algorithm yields that v is not reachable from y through any descendants of $idom(v)$ in the dominator tree. This implies that $idom(v)$ is on all paths from y to v, contradicting that v changes immediate dominator. □

4.2 Updating

After detecting the affected nodes we proceed as follows. The marked nodes are removed and added to the dominator tree in an order which guarantees that a node is a leaf when it is removed or added. To do this in a time to achieve the promised complexity for the algorithm, we have to maintain the dominator tree in such a way that adding/removing leaves from the dominator tree and queries about *nca* can be done in constant time. In [5] an algorithm for adding leaves and answering *nca*-queries in constant time is given. This algorithm can easily be extended to handle deletion of leaves in constant time [3].

Lemma 6 *An $O(L+Q)$ algorithm exists for inserting and deleting L leaves from a tree and interspersed answering Q nca-queries.*

Proof. Omitted.□

5 Complexity

Theorem 1 *The algorithm presented has the complexity $O(|W|+|F|)$ for a single edge insertion, where W is the set of nodes which change depth in the dominator tree and $F = \{(x,y)|(x,y) \in E \wedge x \in W\}$. Interspersed with edge insertions nca-queries can be answered in constant time.*

Proof. Because the algorithm only follows paths through nodes which changes depth, at most $O(|W|+|F|)$ nodes are examined. Each examination and change in the dominator tree can be done in constant time by lemma 6. $\square$

Theorem 2 *The algorithm presented performs i edge insertions interspersed with q nca-queries in $O(q + m * \min\{i, n\})$-time, where m and n respectively are the number of edges and nodes in the graph after the i'th insertion.*

Proof. At most n nodes change depth at most n times. For each change of depth all outgoing edges incident with a node are examined, giving the upper bound $O(\sum_{i=1}^{n} \sum_{v \in V} outdegree(v)) = O(n * m)$. $\square$

Theorem 3 *A pointer machine algorithm exists which has the complexity $O(|W|+|F|+\log\log n)$ for a single edge insertion, where W is the set of nodes which change depth in the dominator tree and $F = \{(x,y)|(x,y) \in E \wedge x \in W\}$. Interspersed with edge insertions queries as "does x dominate y" can be answered in constant time.*

Proof. In stead of using the results from [5] we do as follows. The pointer machine algorithm from [15] is used for maintaining a tree to answer domination-queries. The algorithm has the same complexity as the RAM-algorithm. The algorithm from [2] is used for maintaining a tree to answer *nca*-queries. The algorithm has the same complexity as the RAM-algorithm, except for *nca*-queries, which have the complexity $O(\log\log n)$. For each edge insertion only one *nca*-query is performed, which establishes the complexity $O(|W|+|F|+\log\log n)$. $\square$

Theorem 4 *An $O(q + m * \min\{i, n\})$ pointer machine algorithm exists which performs i edge insertions interspersed with q queries as "does x dominate y?", where m and n respectively are the number of edges and nodes in the graph after the i'th insertion.*

Proof. Omitted. $\square$

6 Notes

We now give an example of the strongness of our bounded algorithm. Consider the graph consisting of a simple path from the root, s, to a node, y. Inserting the edge (s, y) means that y and only y changes immediate dominator. Our algorithm will only examine the nodes which change depth, in this case y, whereas the algorithms in [6, 13] would examine all the nodes in the graph.
Previous work on the problem has included deletion of edges. But the results so far are worse than recomputation and unbounded. We are currently working on the problem.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. On finding lowest common ancestor in trees. In *Annual ACM Symposium on the theory of computing (STOC)*, volume 5, pages 115–132, 1973.

[2] S. Alstrup. Optimal algorithms for finding nearest common ancestor in dynamic trees. Technical Report 95-30, Department of computer science, University of Copenhagen, 1995.

[3] S. Alstrup and P.W. Lauridsen. A simple dynamic algorithm for maintaining a dominator tree. Technical Report 96-3, Department of Computer Science, University of Copenhagen, 1996.

[4] M. Carroll and B.G. Ryder. Incremental data flow update via attribute and dominator updates. In *ACM SIGPLAN-SIGACT Symposium on the Principles of Programming Languages*, pages 274–284, 1988.

[5] H.N. Gabow. Data structure for weighted matching and nearest common ancestors with linking. In *Annual ACM-SIAM Symposium on discrete algorithms (SODA)*, volume 1, pages 434–443, 1990.

[6] G.R. Gao, Y. Lee, and V.C. Sreedhar. Incremental computation of dominator trees. In *Proceedings of the ACM SIGPLAN Workshop on Intermediate Representations*, pages 1–12, 1995. SIGPLAN Notices, 30(3), March 1995.

[7] G.R. Gao and V.C. Sreedhar. A linear time algorithm for placing ϕ-nodes. In *ACM SIGPLAN-SIGACT Symposium on the Principles of Programming Languages*, January 1995.

[8] R. Gupta. Generalized dominators and post-dominator. In *Ann. ACM Symp. on Principles of Programming Languages*, volume 19, pages 246–257, 1992.

[9] R. Gupta. Generalized dominators. *Information processing letters*, 53:193–200, 1995.

[10] D. Harel. A linear time algorithm for finding dominator in flow graphs and related problems. In *Proc. 17th Annual ACM symposium on theory of computing*, pages 185–194, 1985.

[11] T. Lengauer and R.E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Programming Languages Systems*, 1:121–141, 1979.

[12] P.W. Purdom and E.F. Moore. Immediate predominators in a directed graph. *Comm. ACM*, 15(8):777–778, 1972.

[13] G. Ramalingam and T. Reps. An incremental algorithm for maintaining the dominator tree of a reducible flowgraph. In *Symposium on the Principles of Programming Languages*, pages 287–298, 1994.

[14] R.E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of computer and system sciences*, 18(2):110–127, 1979.

[15] A.K. Tsakalidis. Maintaining order in a generalized linked list. *Acta informatica*, 21:101–112, 1984.