Finding Dominators Revisited

Extended Abstract

Loukas Georgiadis[1] Robert E. Tarjan[1,2]

Abstract

The problem of finding dominators in a flowgraph arises in many kinds of global code optimization and other settings. In 1979 Lengauer and Tarjan gave an almost-linear-time algorithm to find dominators. In 1985 Harel claimed a linear-time algorithm, but this algorithm was incomplete; Alstrup et al. [1999] gave a complete and "simpler" linear-time algorithm on a random-access machine. In 1998, Buchsbaum et al. claimed a "new, simpler" linear-time algorithm with implementations both on a random access machine and on a pointer machine. In this paper, we begin by noting that the key lemma of Buchsbaum et al. does not in fact apply to their algorithm, and their algorithm does not run in linear time. Then we provide a complete, correct, simpler linear-time dominators algorithm. One key result is a linear-time reduction of the dominators problem to a nearest common ancestors problem, implementable on either a random-access machine or a pointer machine.

1 Introduction

We consider a flowgraph $G = (V, A, r)$, which is a directed graph with $|V| = n$ vertices and $|A| = m$ arcs such that every vertex is reachable from a distinguished start vertex $r \in V$. A vertex w *dominates* a vertex v if every path from r to v includes w. Our problem is to find for each vertex v in $V - \{r\}$ the set $dom(v)$ which consists of the vertices that dominate v. The *immediate dominator* of v, denoted by $idom(v)$, is the unique vertex w that dominates v and is dominated by all the vertices in $dom(v) - \{v\}$. The *immediate dominators tree* is a directed tree I rooted at r which is formed by the edges $\{(idom(v), v) \mid v \in V - \{r\}\}$. A vertex w dominates v if and only if w is an ancestor of v in I [2]. Thus it suffices to compute $idom(v)$ for all vertices.

The dominators problem appears in program optimization and code generation [2, 4]. Lengauer and Tarjan [14] gave an $O(m\alpha(m, n))$-time algorithm, where $\alpha(m, n)$ is an extremely slow-growing functional inverse of the Ackermann function. A complicated linear-time algorithm was announced by Harel [12], but the description is incomplete and contains some false arguments (see [3]). Based on Harel's work, Alstrup et al. [3] gave a complete linear-time algorithm for the RAM model, but using complicated data structures, in particular Fredman and Willard's Q-heaps [8]. Buchsbaum et al. [6] attempted to give the first simple linear-time dominators algorithm by applying a new special-case analysis of the path compression technique used in disjoint set union and other problems [17]. We show, however, that their analysis does not apply to their dominators algorithm which means that despite being more complicated, the Buchsbaum et al. algorithm is not asymptotically faster than the Lengauer-Tarjan algorithm. We give an overview of the Buchsbaum et al. algorithm in Section 3 together with a discussion of the reason it is not linear. Our new algorithm is based on the Buchsbaum et al. algorithm and can be implemented either on a random-access or on a pointer machine.

2 Preliminaries

2.1 Partitioning the DFS tree Let D be a depth-first search (DFS) tree [16] of the control flow graph $G = (V, A, r)$. We assume that G is represented by adjacency lists. The DFS tree D is rooted at r and $parent(v)$ is the parent of vertex v in D. For simplicity we refer to the vertices by their DFS numbers, so $v < u$ means that the DFS number of v is less than the DFS number of u. As in [7] we partition D into trivial and nontrivial microtrees. Each nontrivial microtree is a maximal subtree of D that has at most g vertices and contains at least one leaf of D, where g is a parameter that we will fix appropriately. Each trivial microtree is a singleton internal vertex of D. For any vertex v we denote by $micro(v)$ the microtree that contains v. Also we denote by $\mathcal{M}$ the set of nontrivial microtrees.

The trivial microtrees constitute a tree with relatively few leaves, the *core* C of the depth-first search tree. Let τ be any microtree of D and let $root(\tau)$ be its root. If $root(\tau) \neq r$ then $v = parent(root(\tau))$ is a

[1]Department of Computer Science, Princeton University, Princeton NJ, 08540. E-mail: {lgeorgia,ret}@cs.princeton.edu. Research at Princeton University partially supported by the National Science Foundation, under the Aladin Project, Grant No CCR-0122581, Carnegie Mellon.

[2]Hewlett-Packard, Palo Alto CA.

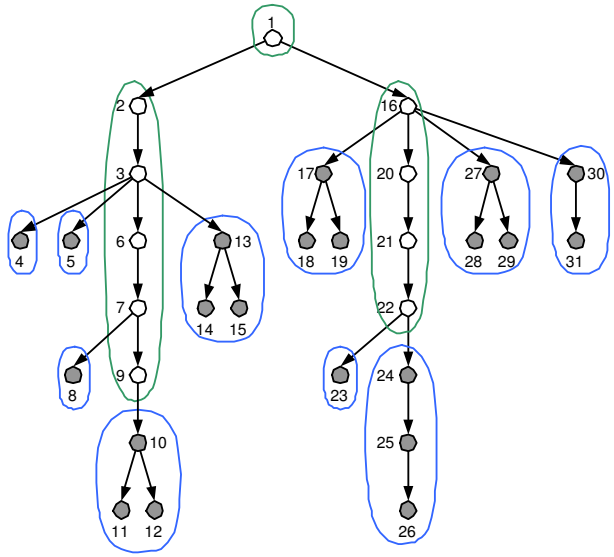

Figure 1: Partition of the DFS tree. There are 10 nontrivial microtrees and 3 lines, which are encircled. The nontrivial microtrees have size at most 3 and are shown with dark vertices.

vertex in C. Also any vertex of the core has at least g proper descendants in D. We note that v is a leaf of the tree C if and only if all of its children in D are roots of nontrivial microtrees. Therefore C has at most n/g leaves. We can group the vertices of C into a set $\mathcal{L}$ of paths with the following properties: if $v_1, v_2, \ldots, v_k$ is a path in $\mathcal{L}$ then v_{i+1} is the unique child of v_i in C for $1 \leq i \leq k-1$, and v_k has either zero or more than one children in C. We call such a path a *line*. We denote by $line(v)$ the line that contains vertex v. Let L be the number of lines. It is easy to verify that L is bounded by twice the number of leaves in C i.e., $L \leq 2n/g$. Figure 1 shows an example of this decomposition.

Let $l = (v_1 = s, v_2, \ldots, v_{k-1}, v_k = t)$ be a line in $\mathcal{L}$. The subtree T_t rooted at t is a union of microtrees in $\mathcal{M}$ and a part of the core. For example in Figure 2.1, T_1 is D and $T_9 = \{9, 10, 11, 12\}$. For each vertex i, $1 \leq i \leq k-1$, we define a right subtree T_i and a left subtree T'_i of T_{v_i}. T_i contains v_i and all descendants of v_i that are greater than v_{i+1} and are not descendants of v_{i+1}; T'_i contains v_i and all descendants of v_i that are less than v_{i+1}. The trees T_i, T'_i and $T_{v_{i+1}}$ partition the proper descendants of v_i. All the children of v_i, except for v_{i+1}, are roots of nontrivial microtrees. We also define $T(l) = \bigcup_{i=1}^{k-1} \left(T_i - \{v_i\}\right)$ to be the set of vertices in the right subtrees of l and $T'(l) = \bigcup_{i=1}^{k-1} \left(T'_i - \{v_i\}\right)$ to be the set of vertices in the left subtrees of l. Now consider the core tree C; if we contract every line in $\mathcal{L}$ into a single vertex we get a tree C' with L vertices. Let l and q be two vertices in C' that correspond to the lines $(v_1, \ldots, v_i)$ and $(u_1, \ldots, u_j)$. Then l is the parent of q in C' if and only if $parent(u_1) = v_i$. In this case we say $l = parent_{C'}(q)$.

2.2 Semidominators and External Dominators

In this section we review some of the definitions and results that are given in [14] and [7]. A path $P = (u = v_0, v_1, \ldots, v_{k-1}, v_k = v)$ is a *semidominator path* (SDOM path) if $v_i > v$ for $1 \leq i \leq k-1$. The *semidominator* of vertex v is $sdom(v) = \min(\{u \mid \text{there is an SDOM path from } u \text{ to } v\})$. From [14] we know that for any vertex $w \neq r$, $idom(w) \xrightarrow{*} sdom(w)$ and $sdom(w) \xrightarrow{+} w$[1]. Moreover, if $v \xrightarrow{+} w$ then either $v \xrightarrow{*} idom(w)$ or $idom(w) \xrightarrow{*} idom(v)$. The following result [14] allows efficient computation of semidominators and computation of immediate dominators from semidominators.

LEMMA 2.1. *Let $w \neq r$ and let u be a vertex for which $sdom(u)$ is minimum among vertices u satisfying $sdom(w) \xrightarrow{+} u \xrightarrow{*} w$. Then $sdom(u) \leq sdom(w)$ and $idom(u) = idom(w)$. Moreover, if $sdom(u) = sdom(w)$ then $idom(w) = sdom(w)$.*

Let $micro(v)$ be the microtree (either trivial or nontrivial) that contains v. A path $P = (u = v_0, v_1, \ldots, v_{k-1}, v_k = v)$ is an *external dominator path* (XDOM path) if P is an SDOM path and $v_i \notin micro(v)$ for $0 \leq i \leq k-1$. The *external dominator* of vertex v is

$$xdom(v) = \min(\{v\} \cup \{u \mid \text{there is an XDOM path from } u \text{ to } v\}).$$

A path $P = (u = v_0, v_1, \ldots, v_{k-1}, v_k = v)$ is a *pushed external dominator path* (PXDOM path) if $v_i \geq root(micro(v))$ for $1 \leq i \leq k-1$. The *pushed external dominator* of vertex v is

$$pxdom(v) = \min(\{u \mid \text{there is a PXDOM path from } u \text{ to } v\}).$$

An XDOM path is also an SDOM path and an SDOM path is also a PXDOM path, thus $pxdom(v) \leq sdom(v) \leq xdom(v)$. If $micro(u) = \{u\}$, which is true if u is in C, then $pxdom(u) = xdom(u) = sdom(u)$. The next result from [7] allows computation of immediate dominators from pushed external dominators.

[1] Throughout this paper the notation "$v \xrightarrow{*}_T u$" means that v is an ancestor of u in the tree T. "$v \xrightarrow{+}_T u$" means that v is a proper ancestor of u in T. We omit the subscript T when $T = D$.

LEMMA 2.2. *For any v, there exists a $w \in micro(v)$ such that*

(1) $w \xrightarrow{*} v$;
(2) $pxdom(v) = pxdom(w)$;
(3) $pxdom(w) = sdom(w)$;
(4) $idom(w) \notin micro(v)$.

Moreover, if $idom(v) \notin micro(v)$ then $idom(v) = idom(w)$.

3 The Buchsbaum et al. algorithm

3.1 Overview of the algorithm The Buchsbaum et al. algorithm preprocesses each nontrivial microtree and finds the immediate dominators for the vertices whose immediate dominators are inside the microtree. Then it computes $pxdom(v)$ for each vertex v in the graph and uses Lemma 2.2 to find $idom(v)$ for each v such that $idom(v) \notin micro(v)$. The $pxdom$'s are computed by processing the microtrees in reverse preorder. Let $\text{NCA}(D, u, v)$ be the nearest common ancestor of u and v in the DFS tree D. Also let $EN(v) = \{u \mid (u, v) \in A,\ u \notin micro(v)\}$ be the set of external neighbors of v. We associate each vertex $v \in V$ with a value $label(v)$ which is initially set equal to v. After processing $micro(v)$ we will have $label(v) = pxdom(v)$. This is accomplished by executing the following procedure for each microtree τ.

for $v \in \tau$ **do**
  $a = parent(root(\tau))$
  for $u \in EN(v)$ **do**
    if $micro(u)$ is trivial **then** $b = u$
    else $b = parent(root(micro(u)))$ **endif**
    if $(b > a)$ **then**
      $x = \min(\{label(z) \mid \text{NCA}(D, a, b) \xrightarrow{+} z \xrightarrow{*} b\})$
    else $x = \infty$ **endif**
    $label(v) = \min(\{label(u), label(v), x\})$
  done /* at this point $label(v) = xdom(v)$ */
done
$PushLabels(\tau)$

If $b < a$ then b is a proper ancestor of a and it is assumed that $label(b) = b$. When $b > a$ then all the vertices $z \neq v$ that satisfy $\text{NCA}(D, a, b) \xrightarrow{+} z \xrightarrow{*} b$ have been processed before τ and the corresponding $pxdom$'s are known, i.e., $label(z) = pxdom(z)$. Procedure $PushLabels(\tau)$ completes the computation of $pxdom(v)$ for all $v \in \tau$ after all the $xdom$'s are known. This is accomplished in linear time by finding the strongly connected components of the graph $G(\tau)$ induced by the vertices of τ and and processing them in topological order (see [7] for the details).

In order to compute the immediate dominators from the $pxdom$'s we need to know whether $idom(v) \in micro(v)$. We start by constructing from $G(\tau)$ an augmented graph $aug(\tau)$ as follows. We add a vertex t and for each arc (u, v) such that $v \in \tau$ and $u \notin \tau$ we add the arc (t, v). Let $iidom(v)$ be the *internal immediate dominator* of a vertex $v \in \tau$, which is defined as the immediate dominator of v in $aug(\tau)$. From [6, 7] we have:

LEMMA 3.1. *Let v be any vertex of a microtree τ. Then $idom(v)$ is in τ if and only if $iidom(v) \neq t$.*

We can compute the $iidom$'s using any simple (but superlinear) dominators algorithm. Since there may be too many microtrees in D we can't afford to run the simple algorithm for each microtree individually. The way to get around this problem is to use memoization to avoid repeating the same computations for identical graphs. In the worst case we will compute the $iidom$'s for every possible $aug(\tau)$ of size $O(g^2)$ and use table lookups to retrieve the $iidom$ values in constant time. If $g = O(\log^{1/3} n)$ this computation can be done in linear time. Although this process seems to require a RAM it can in fact be implemented on a pointer machine. This is accomplished in [6] by sorting the graph encodings using a variation of radix sort. After the $iidom$'s are available we need to compute the immediate dominators of the every vertex v that satisfies $iidom(v) \notin micro(v)$. By Lemma 2.2 all we need to do is find the vertex y on the path $pxdom(v) \xrightarrow{+} a$ such that $pxdom(y) = \min(\{pxdom(z) \mid pxdom(v) \xrightarrow{+} z \xrightarrow{*} a\})$. If $pxdom(y) = pxdom(v)$ then $idom(v) = pxdom(v)$. Otherwise y is a relative dominator of v, i.e., $idom(v) = idom(y)$. For vertices with relative dominators we complete the calculation of immediate dominators in a preorder pass.

3.2 The problem in the analysis The Buchsbaum et al. algorithm uses the same link-eval structure as the Lengauer-Tarjan algorithm (Section 5 of [17]) to implement the computation of x and y above (taking minima on tree paths), but these computations involve only nodes in the core. In fact the two algorithms are essentially the same when the size bound on the nontrivial microtrees is $g = 1$. In [6, 7] it is concluded that the operations of the link-eval data structure take linear time because the links are performed bottom-up. That is we link v to $parent(v)$ only after all the vertices in the subtree rooted at v have been linked. Let T_1 and T_2 be the trees in the link-eval forest that contain $parent(v)$ and v respectively. It is straightforward to verify that T_2 always contains at least one leaf in C and T_1 is either a singleton or also contains at least one leaf in C. The analysis assumes that all the non-singleton trees in the link-eval forest have roots which are leaves in C. In that case we only need

to link two leaves or a single vertex to a leaf, which implies $O(m\alpha(m, L) + n) = O(m + n)$ running time. The problem with this argument is that Tarjan's data structure [17] is not sufficiently flexible, in the sense that we are not free to choose the roots of the trees in the link-eval forest suitably so that the disjoint set union result of [6] would apply. For example consider the case where D is just a path $(v_1 = r, v_2, \ldots, v_n)$. Then $C = (v_1, v_2, \ldots, v_k)$, where $k = n - g = O(n)$. Each successive link is of v_{i+1} to v_i, and v_i becomes the root of the tree that contains v_j, $i \leq j \leq k$. Suppose we have $pxdom(v_k) = v_1$. Then a careful look at the implementation of the link-eval structure in [17, 14] reveals that v_k will only participate in the first link and all the other links will involve internal vertices of C, while the analysis in [6, 7] assumes that every vertex will eventually be linked to v_k. What we really get is the Lengauer-Tarjan algorithm applied on a graph with $O(n)$ vertices and $O(m)$ arcs, so the running time is not linear. We note that the special-case analysis in [6, 7] for disjoint set union does apply to their offline nearest common ancestors algorithm. In fact, the pointer machine version of our algorithm requires this result in order to achieve linear running time.

4 Our linear-time algorithm

4.1 Overview of our approach The conclusion from the previous section is that we need a different way to evaluate minima on paths inside a line. In particular we show that we can use nearest common ancestor queries on a fixed tree instead of using the standard data structure of [17] for evaluating minima inside a single line. We note that our definition of lines is similar to the I-paths of [3], but we perform different operations on the lines. To get a linear time bound the nearest common ancestor queries have to be offline. We can assure this if we deviate from the usual reverse preorder processing of the vertices when it is necessary to do so. Our results require neither complicated data structures nor a linear-time disjoint set union algorithm, but they do require a linear-time solution to the offline nearest common ancestors problem as well as the memoization for small subproblems proposed by Buchsbaum et al. [7, 6]. For clarity we present an algorithm that consists of two phases. As in [14] and [7] the two phases can be integrated into one, which gives a more efficient solution although the asymptotic running time is the same. In Phase A we compute pushed external dominators for each vertex and in Phase B we find the immediate dominators. In Section 5 we show how to remove an intermediate preprocessing step to get a more efficient algorithm.

We will use a mark bit to distinguish among different states that the vertices go through during the course of the algorithm. Initially none of the vertices in D is marked. In Phase A we start visiting the vertices in reverse preorder and compute the *pxdom*'s of the vertices that haven't been processed yet. We use the mark bit to keep track of the vertices in the core whose *pxdom*'s have already been computed. For the vertices in $\mathcal{M}$ we use the mark bit for a different purpose that we describe later. When we find a vertex v in the core that is not marked we process all the vertices in $line(v)$ and mark them. When we visit an already marked vertex we proceed to the next vertex in reverse preorder. This means that if $l = (v_1 = s, v_2, \ldots, v_{k-1}, v_k = t)$ is a line in $\mathcal{L}$, t will be the first vertex of l that we visit. After visiting t we will process all the vertices in l bottom-up. Note that when we start processing l all the vertices in $T(l)$ and T_t and none of the vertices in $T'(l)$ have been processed. To process a nontrivial microtree we can use the procedure of Section 3.1, but we use a new data structure to compute x in the inner loop.

The second phase is similar to the corresponding part of Buchsbaum et al. algorithm [6, 7]. We start by finding the internal immediate dominators in the microtrees of $\mathcal{M}$. Then we process all the vertices in reverse preorder and locate the immediate dominators for the vertices in C and for the vertices $v \in \mathcal{M}$ that have immediate dominators outside $micro(v)$. Again we use our new data structure to evaluate minima on tree paths.

4.2 Range Minimum Queries Our first algorithm will use as a subroutine a solution to the offline Range Minimum Query (RMQ) problem. In an instance of this problem we are given an array $A = \begin{bmatrix} a_1 & a_2 & \ldots & a_n \end{bmatrix}$ and a set Q_A of queries. A query is a pair of array indices (i, j), such that $1 \leq i \leq j \leq n$. For each pair $(i, j) \in Q_A$ we want to find the index of a smallest element in the subarray $\begin{bmatrix} a_i & a_{i+1} & \ldots & a_j \end{bmatrix}$; that is $\mathrm{RMQ}(A, i, j) = \arg\min_{k \in [i \ldots j]} a_k$ is the answer to query (i, j). It is known that the RMQ problem can be reduced in linear time to the offline Nearest Common Ancestor (NCA) problem and vice versa [10]. An instance of the NCA problem consists of a tree T and a set Q_T of queries. A query in Q_T is a pair (u, v) where u and v are vertices of T. If we refer to the vertices of T by their preorder numbers then,

$$\mathrm{NCA}(T, u, v) = \max(\{w \mid w \in T \text{ such that } w \xrightarrow{*} u \text{ and } w \xrightarrow{*} v\}).$$

Such queries can be answered offline on a pointer machine in $O(|T| + |Q_T|)$-time [6]. Also there are solutions to the offline NCA problem on a random-

access machine that answer each query in constant time using $O(|T|)$ time for preprocessing [13, 15, 5]. We note that the linear-time pointer machine algorithm requires Q_T to be known a priori, while the RAM algorithms don't have this restriction.

For our algorithm we want to be able to answer range minimum queries for the array $\big[sdom(v_1) \quad \dots \quad sdom(v_k)\big]$, where $l = (v_1, v_2, \dots, v_k)$ is a line in $\mathcal{L}$. This problem can be solved offline if the semidominators of the line have been computed before we need to answer any query. The order of the *pxdom* calculations ensures that this is the case. In later sections we use the notation

$$\mathrm{RMQ}(v_i, v_j) = \arg \min_{v_{j'} \in v_i \xrightarrow{*} v_j} sdom(v_{j'}).$$

In Section 5 we show how to use nearest common ancestor calculations on a fixed tree directly; that is, we remove the intermediate step that reduces RMQ to NCA.

4.3 A data structure for $Link()$ and $Eval()$ in C In this section we implement a data structure that can perform links and evaluations of path minima on a forest F induced by the vertices of C. We start by implementing a data structure that performs the same operations on a forest F' induced by the vertices of C'. Initially each node in C' is a singleton tree in F'. For any vertex $l \in C'$ let $root_{F'}(l)$ be the root of the tree the contains l in F'. The operations are:

$Link(l, q)$: Link the tree rooted at l in F' to the tree rooted at q in F'. In our algorithm we will always have $q = parent_{C'}(l)$.

$Eval(l)$: If $l = root_{F'}(l)$ return l. Otherwise, return a vertex of minimum value among the vertices q that satisfy $root_{F'}(l) \xrightarrow{+}_{C'} q \xrightarrow{*}_{C'} l$.

We implement these operations using the data structure of Section 5 in [17], but our algorithm will not call $Link()$ and $Eval()$. Instead it will call two procedures $VLink()$ (Virtual $Link$) and $VEval$ (Virtual $Eval$) which are built on top of $Link()$ and $Eval()$ and operate on C. A line $l = (v_1, v_2, \dots, v_k)$ is represented in F' by its top vertex v_1. We associate with l a label which, after all the vertices in l are linked, will be equal to a vertex with minimum semidominator among the vertices of l. To that end we use a field $lineLabel$ that is only defined for the top vertices of the lines and is initialized to ∞. After linking v_j to v_{j-1} we will have

$$lineLabel(v_1) = \arg \min_{u \in l \text{ and } u \geq v_j} sdom(u).$$

$VLink(v)$ performs a *virtual* link of v and $u = parent(v)$. When both v and u belong to the same line $l = (v_1, \dots, v_k)$ then we only have to set $lineLabel(v_1) = u$, if $sdom(u) < sdom(lineLabel(v_1))$. Our algorithm will process all the vertices in l before any of them is linked to its parent, therefore $VLink()$ maintains the correct result for $lineLabel$. In the case where $v = v_1$ we also need to link v_1 with u_1, where u_1 is the top vertex of $line(u)$. This is accomplished by executing the *real* link $Link(v_1, u_1)$. The implementation of $VEval()$ assumes that for each vertex $v \in C$ we store a value

$$up(v) = \arg \min_{\substack{u \in line(v) \\ \text{and } u \leq v}} sdom(u).$$

These values can be calculated in a straightforward manner by a top-down pass over the line, after all the $sdom$'s in the line are known. We further assume that the lines whose semidominators have been computed are preprocessed so that all the necessary RMQ's are answered in linear time. The role of $VEval()$ is to return a vertex with minimum $sdom$ among the vertices z that satisfy $root_F(b) \xrightarrow{+} z \xrightarrow{*} b$, where $root_F(b)$ is the root of the virtual tree built by $VLink()$ that contains b. If both $root_F(b)$ and b are in the same line then the value that we need is $\mathrm{RMQ}(c, b)$, where c is such that $parent_D(c) = root_F(b)$. Otherwise the required value is the vertex with minimum $sdom$ among $up(b)$, $lineLabel(Eval(parent_{C'}(b')))$ and $lineLabel(c')$, where b' is the top vertex of $line(b)$ and c' is the top vertex of $line(c)$. Since C' has $L = O(n/g)$ vertices and we call $VEval()$ for less than m arcs, the total time that will be spent on $Link()$ and $Eval()$ is $O(m\alpha(m, L) + L) = O(m)$, for $g = O(\log^{1/3} n)$. This implies that n calls to $VLink()$ and $O(n+m)$ calls to $VEval()$ take $O(n+m)$ time.

4.4 Pushed external dominators of the nontrivial microtrees We process a nontrivial microtree τ as in Section 3.1, but we use the data structure of the previous section to maintain the link-eval forest in C. A vertex $v \in C$ can be linked to $parent(v)$ only after $line(v)$ has been processed. Specifically v is linked when it becomes the next vertex in reverse preorder and is not processed again because it was marked after processing $line(v)$. Let c be the last vertex of the core that was linked to its parent. Also suppose $v \in \tau$ is the next vertex to be processed. Figure 2 shows the various situations that may happen when we examine the arc (u, v) for $u \in EN(v)$. Here b is the nearest ancestor of u in C and similarly a is the nearest ancestor of v in C. Also t is the bottom vertex of $l = line(a)$. Since τ is

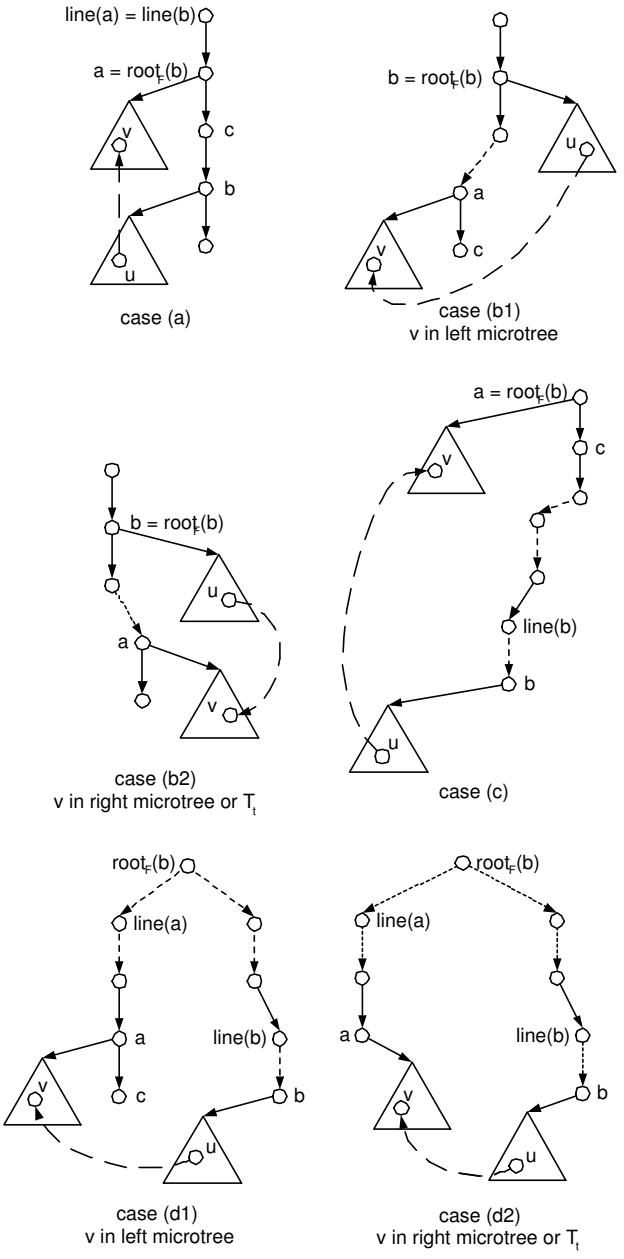

Figure 2: Four possible locations of $u \in EN(v)$. Any of the four cases may apply when v is in $T'(l)$, but only (b) or (d) may apply when v is in $T(l)$ or in $T_t \cap \mathcal{M}$. In all cases we may have $u = b$. In case (a), $a \xrightarrow{+} b$ and $a, b \in l$. In case (b), $b \xrightarrow{*} a$. In case (c), $a \xrightarrow{+} b$ and $b \in T_t$. Finally, in case (d) we have neither $a \xrightarrow{+} b$ nor $b \xrightarrow{*} a$. Cases (a), (c) and (d) are handled in $VEval()$ and case (b) is handled separately.

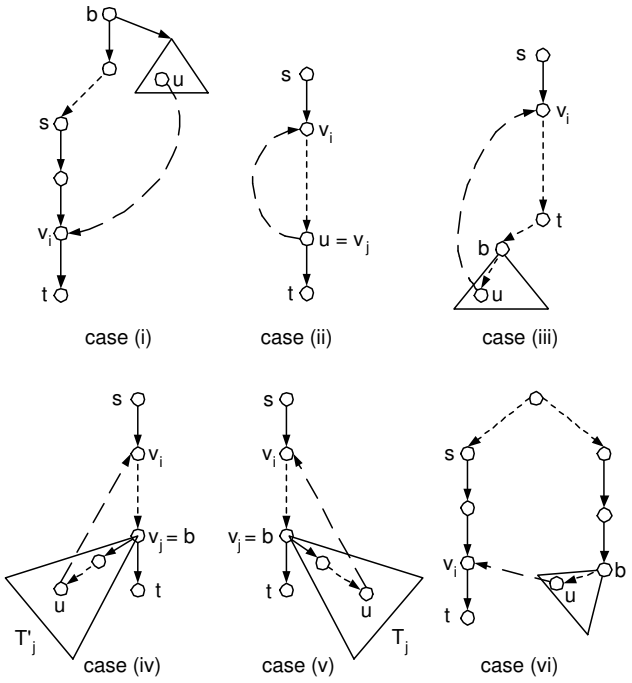

Figure 3: Computing the semidominators of the line $l = (v_1 = s, v_2, \ldots, v_k = t)$

nontrivial, $a = parent(root(\tau))$. Then v is in $T'(l)$ if and only if $parent(c) = a$. This implies $a = root_F(b)$ when $line(a) = line(b)$ (case (a) of Figure 2) or $b \in T_t$ (case (c) of Figure 2). Also when $a < b$ but not $a \xrightarrow{+} b$ then $v \notin T'(l)$ if and only if $parent(c) = \text{NCA}(D, a, b)$ (case (d2) in Figure 2). Case (b) is handled separately without a call to $VEval()$ because it doesn't require evaluations of path minima and moreover b may have already been processed so we must be careful not to set $sdom(v) = sdom(b)$ when $b = u$.

4.5 Semidominators in the core In this section we give an algorithm $ProcessLine(t)$ that computes the semidominators of a line $l = (s = v_1, v_2, \ldots, v_{k-1}, v_k = t)$ in $\mathcal{L}$. The algorithm assumes that we have $label(u) = pxdom(u)$ for all vertices $u > t$, and $label(u) = u$ for any $u \leq t$. In particular we don't know the $pxdom$'s in $T'(l)$. The computation of the semidominators relies on finding strongly connected components on l as we walk upwards from t to s. At vertex v_i we examine the incoming arcs (u, v_i). The actions that we need to execute depend on the location of u. Again b is the nearest ancestor of u in C. We consider the following cases (see Figure 3):

(i). $b \xrightarrow{+} v_i$. We update $label(v_i) = \min(\{label(v_i), label(u)\})$.

(ii). $u = v_j$ for $j > i$. We contract the vertices of the path $v_i \xrightarrow{+} v_j$ into a single vertex v_i and set $label(v_i)$ to be the minimum label in this path.

(iii). $u \in T_t$. We find $x = \min(\{label(w) \mid t \xrightarrow{+} w \xrightarrow{*} b)\})$. Then we contract the path $v_i \xrightarrow{+} t$ as if there were an arc (t, v_i) and update $label(v_i)$ as done in (ii). Finally we update $label(v_i) = \min(\{x, label(v_i), label(u)\})$.

(iv). $u \in T'_j$. It must be that $j \geq i$. We handle this case by contracting the cycle $(v_j \xrightarrow{+} u, v_i \xrightarrow{+} v_j)$ into a single vertex v_i. The $v_i \xrightarrow{+} v_j$ part is done as in (ii). For the $v_j \xrightarrow{+} u$ part we start from u and walk backwards until we reach v_j or a marked vertex. For each vertex $w \neq v_j$ that we visit in this walk, we mark w and for each arc (z, w) we add (z, v_i) at the end of the list of incoming arcs of v_i. (The added arcs will eventually be examined while processing v_i.)

(v). $u \in T_j$. We update $label(v_i) = \min(\{label(u), label(v_i)\})$. If $v_j > v_i$ we contract the path $v_i \xrightarrow{+} v_j$ as in (ii).

(vi). Neither $v_i \xrightarrow{+} b$ nor $b \xrightarrow{+} v_i$. Then $v_i < b$. We find $x = \min(\{label(w) \mid \text{NCA}(D, u, v_i) \xrightarrow{+} w \xrightarrow{*} b\})$. Then we update $label(v_i) = \min(\{x, label(v_i), label(u)\})$.

The process that we use to contract a path $v_i \xrightarrow{+} v_j$ is similar to Gabow's linear-time algorithm for strong components [9]. A stack S is sufficient to keep track of the contracted paths in l. Each element of the stack corresponds to a strongly connected component scc of successive vertices in l and can be represented by the highest vertex v in the component i.e., $v = \min(\{u \mid u \in scc\})$. Then it is also true that $sdom(v) = \min(\{sdom(u) \mid u \in scc\})$. Suppose that when we reach vertex v the current status of the stack is $S = (u_d, u_{d-1}, \ldots, u_1)$, where $v = parent(u_d)$ and $u_i < u_{i-1}$ for $2 \leq i \leq d$. Then we keep removing the top element of the stack until we reach a proper descendant of u or empty the stack. As we pop each element $w < u$ in S we update $label(v) = \min(\{label(v), label(w)\})$. The other subroutine that we need carries out the backwards walk from a vertex u in a left nontrivial microtree τ to $root(\tau)$. To achieve the effect of contracting the path $P = (root(\tau) \xrightarrow{*} u)$ we set the mark bit of w for all $w \in P$. Since we don't actually remove the vertices of P we stop the backwards walk once we have reached a vertex z such that it is either marked or in the core. Note that if z has been marked then every vertex in the path $root(\tau) \xrightarrow{*} z$ has also been marked. We can prove by induction that $ProcessLine()$ correctly computes semidominators in lines.

THEOREM 4.1. *Let $l = (v_1 = s, v_2, \ldots, v_{k-1}, v_k = t)$ in $\mathcal{L}$. Assume that just before the execution of $ProcessLine(t)$ we have $label(u) = pxdom(u)$ for any vertex u such that $u > t$. Then after the execution of $ProcessLine(t)$ we have $label(v_i) = sdom(v_i)$ for $1 \leq i \leq k$.*

Proofs and implementation details are omitted in this extended abstract. The x values that we use in this procedure correspond to path minima and are computed by $VEval()$. After a vertex u in $T'(l)$ is marked, every subsequent visit to u will cost us $O(1)$ time. Moreover, the number of stack operations is $O(k)$. Therefore, it is not hard to verify that the execution of $ProcessLine()$ takes linear time with respect to the size of $l \cup T'(l)$ and the number of predecessors of the vertices in $l \cup T'(l)$, plus the time spent on $VEval()$. A similar statement holds for the procedure of Section 4.4 which processes nontrivial microtrees. Since the total time time spent on $VLink()$ and $VEval()$ is linear the whole algorithm runs in linear time.

5 An NCA-based algorithm

Our first algorithm uses RMQ to evaluate minima on paths inside a line. We have already mentioned that in order to solve the offline RMQ for an array A we can use a linear-time reduction to the offline NCA problem for a tree T_A that is constructed based on the values of A. This seems to be a redundant step, which we wish to avoid. In this section we modify our algorithm so that it directly uses NCA queries on a tree $I(l)$ that is constructed based on the immediate dominators tree of a line l. This method is more desirable also because in contrast to T_A the immediate dominators tree has to be constructed by the dominators algorithm anyway.

5.1 Line dominators tree The next lemma suggests a method to construct the immediate dominators tree I once we have computed the semidominators. (We omit the proof in this extended abstract.) We note that this process resembles the Aho, Hopcroft and Ullman algorithm for finding dominators in reducible graphs [1].

LEMMA 5.1. *For any vertex $w \neq r$, $idom(w)$ is the nearest common ancestor of $sdom(w)$ and $parent(w)$ in the immediate dominators tree I, i.e., $idom(w) = \text{NCA}(I, parent(w), sdom(w))$.*

We can locate NCA$(I, parent(w), sdom(w))$ easily by using the fact that when $idom(w) \neq sdom(w)$, $parent(w)$ is not dominated by any vertex v that satisfies $idom(w) \xrightarrow{+} v \xrightarrow{*} sdom(w)$. If we start from $parent(w)$ and go up on the I-tree path from $parent(w)$ to r until we meet the first vertex x such that $x \leq sdom(w)$, then $x = idom(w)$. In the simple case where D is just a path $(1, 2, \ldots, n)$ we have $idom(w) =$ NCA$(I, w-1, sdom(w))$. So after the calculation of the semidominators we can construct the dominators tree I incrementally in linear time. For a general graph, however, this procedure can take $O(n^2)$ time. Now we apply this method to a line $l = (v_1, v_2, \ldots, v_k)$ in $\mathcal{L}$ and construct a tree $I(l)$, which we call *line dominators tree* of l. We denote by $parent_{I(l)}(v)$ the parent of v in $I(l)$. The construction goes as follows. Initially $I(l)$ consists of the vertices v_1 and $root = sdom(v_1)$ where $sdom(v_1) = parent_{I(l)}(v_1)$ and is the current root of the tree. When we process v_i we first check whether $sdom(v_i) < root$. If this is true we set $parent_{I(l)}(root) = sdom(v_i)$ and make $sdom(v_i)$ the new root of $I(l)$. Otherwise we traverse the path in $I(l)$ from v_{i-1} to $root$ until we get to a vertex x such that $x \leq sdom(w)$. Then we set $parent_{I(l)}(root) = x$. An example is shown in Figure 4. Note that unless $v_1 = r$, $I(l)$ contains vertices outside l and in particular $root \xrightarrow{+} v_1$. $I(l)$ can be constructed in the same top-down pass of l in which the *up* values are computed and requires $O(k)$ time. The next Lemma describes the properties of $I(l)$.

LEMMA 5.2. *Let $I(l)$ be the line dominators tree of $l = (v_1, \ldots, v_k)$. For any $v_i \in l$, if $parent_{I(l)}(v_i) \in l$ then $parent_{I(l)}(v_i) = idom(v_i)$. Otherwise, let z be a vertex that satisfies $parent_{I(l)}(v_i) \xrightarrow{+} z \xrightarrow{+} v_1$ and $sdom(z) = \min(\{sdom(w) \mid parent_{I(l)}(v_i) \xrightarrow{+} w \xrightarrow{+} v_1\})$. If $sdom(z) \geq parent_{I(l)}(v_i)$ then $idom(v_i) = parent_{I(l)}(v_i)$; otherwise $idom(v_i) = idom(z)$.*

We also note that the previous lemma implies that all the vertices that point to the same vertex $w \notin l$ have the same dominators. So it suffices to locate the immediate dominator of $u = child_{I(l)}(w)$ and set $idom(v) = idom(u)$ for all $v \in l$ such that $parent_{I(l)}(v) = w$.

5.2 Processing the nontrivial microtrees We process a vertex in a nontrivial microtree as in Section 4.4, but we evaluate paths using procedure $MVEval()$ which is a modified version of $VEval()$. The difference between these two version is that in case (a) of Figure 2, $MVEval()$ uses the result of NCA$(I(l), a, b)$ instead of RMQ(c, a). Also in cases (c) and (d), $MVEval()$ returns the minimum label itself of any vertex z that satisfies $root_F(b) \xrightarrow{+} z \xrightarrow{*} b$ instead of a vertex with the

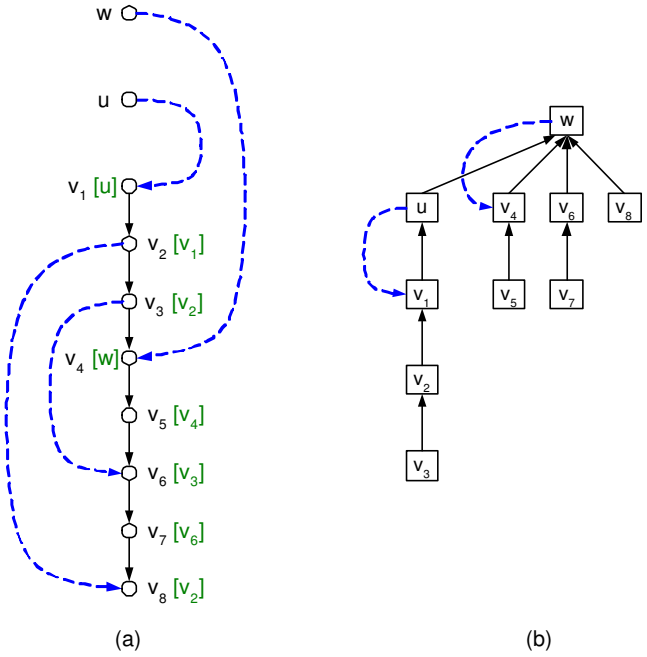

Figure 4: (a) $l = (v_1, \ldots, v_8)$ is a line of the core. w and u are semidominators of v_4 and v_1 respectively that lie outside l. The dashed arcs connect $sdom(v_i) \neq parent(v_i)$ to v and the values inside the brackets correspond to semidominators. (b) The line dominators tree $I(l)$. A solid arc (v_i, z) indicates that $parent_{I(l)}(v_i) = z$. A dashed arc (z, v_i) indicates that $child_{I(l)}(z) = v_i$, which means that during the process of constructing $I(l)$, z was once the root of the tree and v_i was the first child of z in $I(l)$.

minimum label in the same path, as done in $VEval()$. The consequence of replacing the range minimum query by a nearest common ancestors calculation is that may get $label(v) \neq pxdom(v)$, for $v \in T'(l)$. Nevertheless, the labels we compute using our modified procedure give sufficient information to find the external dominators of vertices outside $l \cup T(l) \cup T'(l)$ and the immediate dominators of vertices in $T'(l)$. This fact is stated in the next theorem.

THEOREM 5.1. *Let τ be a microtree in $\mathcal{M}$ and $a = parent(root(\tau))$. After processing all the vertices $v \in \tau$ we have*

$$\min(\{label(v), label(up(a))\}) = \min(\{pxdom(v), label(up(a))\}).$$

Moreover, if τ is not a left microtree then $label(v) = pxdom(v)$. Otherwise, if $idom(v) \notin \tau$ then at least one of the following statements is true:

(a) $label(v) = pxdom(v)$;

(b) $label(v) = idom(v)$;

(c) $label(v) = w \notin \tau$, where w is such that $idom(v) = idom(z)$ and z satisfies $w \xrightarrow{+} z \xrightarrow{+} v_1$ and $sdom(z) = \min(\{sdom(u) \mid w \xrightarrow{+} u \xrightarrow{+} v_1\})$.

Theorem 5.1 implies that the algorithm of Section 4.5 will still be correct, so for any $v \in C$ we get $label(v) = sdom(v)$. Since our goal is to eliminate the need for range minimum queries we need a way to compute $idom(v)$ when $pxdom(v) \in line(a)$. The next result, which is derived from Lemma 2.2, Lemma 5.1, and Lemma 5.2 gives such an alternative solution.

LEMMA 5.3. *Let $l = (v_1, v_2, \ldots, v_k)$ be a line at the core of the DFS tree and $v \neq v_i$ be any vertex in either T_i or T'_i. If $idom(v)$ is not in $micro(v)$ then $idom(v) = \text{NCA}(I, v_i, pxdom(v))$. Moreover, let $y = \text{NCA}(I(l), v_i, pxdom(v))$. Then one of the following statements is true:*

(a) $y = idom(v) \in l$;

(b) $y \notin l$ and either $idom(v) = y$ or $idom(v) = idom(z)$ where z is such that $y \xrightarrow{+} z \xrightarrow{+} v_1$ and $sdom(z) = \min(\{sdom(u) \mid y \xrightarrow{+} u \xrightarrow{+} v_1\})$.

5.3 Immediate dominators In order to complete the description of our modified algorithm we now give the details of computing the immediate dominators. Suppose that u is the parent of the last vertex that was linked. We process the vertices $v \in C$ such that $parent_{I(line(v))}(v) = u$, and the vertices $v \in \tau$ such that $\tau \in \mathcal{M}$, $idom(v) \notin \tau$ and $\text{NCA}(I(line(a)), a, label(v)) = u$, where $a = parent(root(\tau))$. First we consider the case $v \in C$. If $u \in line(v)$ then $idom(v) = u$ by Lemma 5.2. Otherwise we compute a vertex z with minimum label among the vertices that satisfy $u \xrightarrow{+} z \xrightarrow{+} v_1$. This is done using the appropriate call to $VEval()$. We note that since $u \leq \min(\{sdom(z) \mid v_1 \xrightarrow{*} z \xrightarrow{*} v\})$ we get the same result if we evaluate the minimum label of the path $u \xrightarrow{+} v$. In this case we know that $line(v) \neq line(u)$ therefore $VEval()$ will not make a range minimum query. By applying Lemma 5.2 we conclude that the result is either the immediate dominator of v or a relative dominator of v. If v belongs to a nontrivial microtree τ then we assume we have first computed $u = \text{NCA}(I(line(a)), a, label(v))$. Again this calculation is done on a fixed tree so we can use a linear-time offline NCA algorithm. Now we consider the cases listed in Theorem 5.1. If $label(v) = pxdom(v)$ then we apply Lemma 5.3 and process v the same way we processed the vertices of the core. In the other cases we have $\text{NCA}(I(line(a)), a, label(v)) = label(v)$ and we can apply Theorem 5.1. Therefore in any case we have either $idom(v) = u$ or we need an additional call to $VEval()$ in order to compute the immediate or a relative dominator of v.

References

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman. On finding lowest common ancestors in trees. SIAM J. Comput. 5, 1, 1976, Pages 115-133, 2000.

[2] A. V. Aho, R. Sethi and J. D. Ullman. Compilers: Principles, techniques and tools. Addison-Wesley, Reading, Mass., 1986.

[3] S. Alstrup, D. Harel, P. W. Lauridsen and M. Thorup: Dominators in Linear Time. SIAM J. Comput. 28(6), Pages 2117-2132, 1999.

[4] A. Appel. Modern compiler implementation in ML. Cambridge University Press, Cambridge, Cambridge UK, 1998.

[5] M. A. Bender and M. Farach-Colton. The LCA problem revisited. LATIN, Pages 88-94, 2000.

[6] A. L. Buchsbaum, H. Kaplan, A. Rogers and J. R. Westbrook. Linear-Time Pointer-Machine Algorithms for Least Common Ancestors, MST Verification, and Dominators. STOC 1998, Pages 279-288.

[7] A. L. Buchsbaum, H. Kaplan, A. Rogers and J. R. Westbrook. A New, Simpler Linear-Time Dominators Algorithm. ACM Transactions on Programming Languages and Systems 20, 6, November 1998, Pages 1265-1296.

[8] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. Journal of Computer and System Sciences, 48(3), Pages 533–551, 1994.

[9] H. N. Gabow. Path-based depth-first search for strong and biconnected components. Information Processing Letters 74, 2000, Pages 107-114.

[10] H. N. Gabow, J. L. Bentley and R. E. Tarjan. Scaling and related techniques for computational geometry. STOC 1984, Pages 135-143.

[11] H. N. Gabow and R. E. Tarjan. A Linear-Time Algorithm for a Special Case of Disjoint Set Union. JCSS 30(2), Pages 209-221, 1985.

[12] D. Harel. A linear time algorithm for finding dominators in flow graphs and related problems. In Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, Pages 185–194, May 1985.

[13] D. Harel and R. E. Tarjan. Fast Algorithms for Finding Nearest Common Ancestors. SIAM J. Comput. 13(2), Pages 338-355, 1984.

[14] T. Lengauer and R. E. Tarjan, A fast algorithm for finding dominators in a flowgraph. ACM Trans. Program. Lang. Syst. 1, 1, 1979, Pages 121-141.

[15] B. Schieber and U. Vishkin. On Finding Lowest Common Ancestors: Simplification and Parallelization. SIAM J. Comput. 17(6), Pages 1253-1262, 1988.

[16] R. E. Tarjan, Depth-first search and linear graph algorithms. SIAM J. Comput., 1,(2), Pages 146-160, 1972.

[17] R. E. Tarjan, Applications of path compression on balanced trees. J. ACM 26, 4, 1979, Pages 690-715.