Computing Frequency Dominators and Related Problems

Loukas Georgiadis*

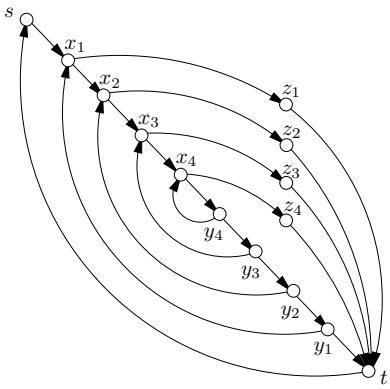
Hewlett-Packard Laboratories, Palo Alto, CA, USA

Abstract. We consider the problem of finding *frequency dominators* in a directed graph with a single source vertex and a single terminal vertex. A vertex x is a *frequency dominator* of a vertex y if and only if in each source to terminal path, the number of occurrences of x is at least equal to the number of occurrences of y. This problem was introduced in a paper by Lee et al. [11] in the context of dynamic program optimization, where an efficient algorithm to compute the frequency dominance relation in reducible graphs was given. In this paper we show that frequency dominators can be efficiently computed in general directed graphs. Specifically, we present an algorithm that computes a sparse ($O(n)$-space), implicit representation of the frequency dominance relation in $O(m + n)$ time, where n is the number of vertices and m is the number of arcs of the input graph. Given this representation we can report all the frequency dominators of a vertex in time proportional to the output size, and answer queries of whether a vertex x is a frequency dominator of a vertex y in constant time. Therefore, we get the same asymptotic complexity as for the regular dominators problem. We also show that, given our representation of frequency dominance, we can partition the vertex set into *regions* in $O(n)$ time, such that all vertices in the same region have equal number of appearances in any source to terminal path. The computation of regions has applications in program optimization and parallelization.

1 Introduction

Let $G = (V, A, s, t)$ be a flowgraph with a distinguished source vertex s and terminal vertex t, which is a directed graph such that every vertex is reachable from s and reaches t. The concept of *frequency dominators* was introduced in [11] with an application in dynamic program optimization. Formally, vertex x is a frequency dominator of vertex y if and only if in each s-t path, the occurrences of x are at least as many as the occurrences of y. We will denote by $fdom(y)$ the set of frequency dominators of y. Frequency dominators are related to the well-studied problem of finding *dominators*, which has applications in several areas [9]. A vertex x *dominates* a vertex y if and only if every s-y path contains x. Similarly, a vertex x *postdominates* a vertex y if and only if every y-t path contains x. This is equivalent to stating that y dominates x in the reverse flowgraph $G^r =$

* Current address: Informatics and Telecommunications Engineering Department, University of Western Macedonia, Kozani, Greece. E-mail: lgeorg@uowm.gr



vertex y	$fdom(y)$
s	$\{s, x_1, t\}$
x_1	$\{x_1\}$
x_2	$\{x_2\}$
x_3	$\{x_3\}$
x_4	$\{x_4\}$
y_1	$\{x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4\}$
y_2	$\{x_2, x_3, x_4, y_2, y_3, y_4\}$
y_3	$\{x_3, x_4, y_3, y_4\}$
y_4	$\{x_4, y_4\}$
z_1	$\{s, x_1, z_1, t\}$
z_2	$\{s, x_1, x_2, z_2, t\}$
z_3	$\{s, x_1, x_2, x_3, z_3, t\}$
z_4	$\{s, x_1, x_2, x_3, x_4, z_4, t\}$
t	$\{s, x_1, t\}$

Fig. 1. The frequency dominators of a family of reducible graphs with $n = 3k + 2$ vertices and $m = 5k + 2$ arcs. (In this figure $k = 4$.) For each z_i, $fdom(z_i)$ contains a subset of the x_j's. However, no pair of x_j's is related under *fdom*. Hence, any transitive reduction of the *fdom* relation without Steiner nodes requires $\Theta(k^2)$ arcs.

(V, A^r, s^r, t^r), where $A^r = \{(x, y) \mid (y, x) \in A\}$, $s^r = t$, and $t^r = s$. We will denote by $dom(y)$ the set of dominators of y, and by $dom^r(y)$ the set of postdominators of y.

Both the dominance and the frequency dominance relations are reflexive and transitive, but only the former is antisymmetric. The transitive reduction of *dom* is the *dominator tree*, which we denote by D. This tree is rooted at s and satisfies the following property: For any two vertices x and y, x dominates y if and only if x is an ancestor of y in D [1]. For any vertex $v \neq s$, the *immediate dominator* of v, denoted by $d(v)$, is the parent of v in D. It is the unique vertex that dominates v and is dominated by all the vertices in $dom(v) \setminus \{v\}$. On the other hand, a transitive reduction of *fdom* can have complex structure; refer to Figure 1 for an example.

Lee et al. [11] presented an efficient algorithm for computing frequency dominators in *reducible* graphs. A flowgraph G is reducible when the repeated application of the following operations

(i) delete a loop (v, v);
(ii) if (v, w) is the only arc entering $w \neq s$ delete w and replace each arc (w, x) with (v, x),

yields a single node. Equivalently, G is reducible if every loop has a single entry vertex from s. Reducible graphs are useful in program optimization as they represent the control flow in structured programs, i.e., in programs that use a restricted set of constructs, such as IF-THEN and WHILE-DO. However, the use of GOTO constructs, or the transformations performed by some optimizing compilers can often produce irreducible graphs. Although there are techniques

that can transform an irreducible graph to an equivalent reducible graph, it has been shown that there exist irreducible graphs such that any equivalent reducible graph must be exponentially larger [5]. The algorithm of Lee et al. can compute $fdom(y)$, for any vertex y, in $O(|fdom(y)|)$ time, after a preprocessing phase. This preprocessing phase consists of computing the dominators, postdominators, and the loop structure of G. Each of these computations can be performed in $O(m+n)$ time [4, 8], or in $O(m\alpha(m,n))$[1] time with simpler and more practical algorithms [12, 14].

Another concept related to frequency dominators, that has also been used in optimizing and parallelizing compilers, is that of *regions* [3]. Two vertices x and y are in the same region if and only if they appear equal number of times in any s-t path. Hence, x and y are in the same region if and only if $x \in fdom(y)$ and $y \in fdom(x)$. Regions define an equivalence relation on the flowgraph vertices, therefore we say that x and y are *equivalent* if and only if they belong to the same region. Johnson et al. [10] showed that the problem of finding regions can be solved by computing the *cycle equivalence* relation: Two vertices are cycle equivalent if and only if they belong to the same set of cycles. Cycle equivalence for edges can be defined similarly. In [10], Johnson et al. show that vertex cycle equivalence can be easily reduced to edge cycle equivalence in an *undirected* graph. Then, they develop an $O(m+n)$-time algorithm, based on depth-first search, for computing edge cycle equivalence.

1.1 Overview and Results

In Section 2 we present an efficient algorithm for computing frequency dominators in general flowgraphs. This algorithm is a generalization of the algorithm of Lee et al. for reducible graphs, and achieves the same, asymptotically optimal, time and space bounds. Then, in Section 3, we show that the equivalence relation can be derived from our representation of the frequency dominance relation in $O(n)$ time. Our main results are summarized by the next theorem.

Theorem 1. *Let $G = (V, A, s, t)$ be any flowgraph with $|V| = n$ vertices and $|A| = m$ arcs. We can compute an $O(n)$-space representation of the frequency dominance relation of G in $O(m+n)$ time, such that:*

- *For any vertex y we can report all the frequency dominators of y in $O(|fdom(y)|)$ time,*
- *for any two vertices x and y, we can test in constant time whether $x \in fdom(y)$, and*
- *we can compute the regions of G in $O(n)$ time.*

Some of the methods we use to obtain these results may be useful in other graph problems related to program optimization.

Finally, in Section 4, we describe a general framework that can express the problems we considered in Sections 2 and 3 as a type of reachability problems

[1] $\alpha(m,n)$ is a functional inverse of Ackermann's function and is very slow-growing [15].

in graphs representing transitive binary relations, and for which an efficient reporting algorithm is required. We believe that this formulation gives rise to questions that deserve further investigation.

1.2 Preliminaries and notation

We assume that G is represented by adjacency lists. With this representation it is easy to construct the adjacency list representation of the reverse flowgraph G^r in linear-time. Given a depth-first search (DFS) tree T of G rooted at s, the notation "$v \xrightarrow{*} u$" means that v is an ancestor of u in T and "$v \xrightarrow{+} u$" means that v is a proper ancestor of u in T. An arc $a = (u, v)$ in G is a *DFS-tree arc* if $a \in T$, a *forward arc* if $u \xrightarrow{+} v$ and $a \notin T$, a *back arc* if $v \xrightarrow{+} u$, and a *cross arc* otherwise (when u and v are unrelated in T). For any rooted tree T, we let $p_T(v)$ denote the parent of v in T, and we let $T(v)$ denote the subtree of T rooted at v. Finally, for any function or set f defined on G, we denote by f^r the corresponding function or set operating on G^r.

2 Efficient computation of the *fdom* relation

First we state some useful properties of frequency dominators. As noted in [10, 11], it is convenient to assume that G contains the arc (t, s); adding this arc does not affect the *fdom* relation and allows us to avoid several boundary cases. Henceforth we assume that $(t, s) \in A$, which implies that G is strongly connected. In particular, note that there is a cycle containing all vertices in $dom(y) \cup dom^r(y)$, for any vertex y. We also have the following characterization of the *fdom* relation.

Lemma 1 ([11]). *For any vertices x and y, $x \in fdom(y)$ if and only if x belongs to all cycles containing y.*

Obviously, reversing the orientation of all arcs in a directed graph does not change the set of cycles that contain a vertex. Thus, from Lemma 1 we get:

Lemma 2. *For any vertex y, $fdom(y) = fdom^r(y)$.*

Our algorithm will be able to identify easily a subset of $fdom(y)$ using certain structures that are derived from G. The remaining vertices in $fdom(y)$ will be found by performing the symmetric computations on G^r.

Let T be a depth-first search tree of G rooted at s. We identify the vertices by their preorder number with respect to the DFS: $v < w$ means that v was visited before w. For any vertex $v \neq s$, the *head* of v is defined as

$$h(v) = \max\{u : u \neq v \text{ and there is a path in } G \text{ from } v \text{ to } u \text{ containing only vertices in } T(u)\},$$

and we let $h(s) = null$. Informally, $h(v)$ is the deepest ancestor u of v in T such that G contains a v-u path visiting only vertices in $T(u)$. Note that the h function is well-defined since G is strongly connected. The heads define a tree H, called

the *interval tree* of G, where $h(v) = p_H(v)$. It follows that the vertices in each subtree $H(v)$ of H induce a strongly connected subgraph of G, which contains only vertices in $T(v)$. Tarjan [16] gave a practical algorithm for computing the interval tree in $O(m\alpha(m,n))$ time using nearest common ancestor computations. A more complicated linear-time construction was given by Buchsbaum et al. [4].

Now we also define

$$\hat{h}(v) = \begin{cases} v, & \text{if there is a back arc entering } v, \\ h(v), & \text{otherwise.} \end{cases}$$

We say that a vertex v is *special* if $\hat{h}(v) = v$. The following lemma provides necessary conditions for frequency dominance.

Lemma 3. *If $x \in fdom(y)$ then the following statements hold:*

(a) $x \in dom(y) \cup dom^r(y)$,
(b) $x \in H(\hat{h}(y))$, and
(c) $x \in H^r(\hat{h}^r(y))$.

Proof. The proof follows directly from the definitions. For (a), consider a vertex x that neither dominates nor postdominates y. Then, there exist two paths P and Q, from s to y and from y to t respectively, that both avoid x. The catenation of P and Q followed by (t, s) is a cycle that passes through y but not x, hence $x \notin fdom(y)$ by Lemma 1. In order to show (b), suppose $x \in fdom(y)$ but $x \notin H(\hat{h}(y))$. Since $H(\hat{h}(y))$ induces a strongly connected subgraph, there is a cycle C passing through y that contains only vertices in $H(\hat{h}(y))$. Then, $x \notin C$ which contradicts Lemma 1. Finally, (c) is implied by (b) and Lemma 2. □

Similarly to the algorithm given in [11], our algorithm is also based on computing dominators and postdominators. Here, however, we make explicit use of the reverse graph and of properties of depth-first search. In particular, we will need the following fact.

Lemma 4 ([13]). *Any path from x to y with $x < y$ contains a common ancestor of x and y.*

Now we provide sufficient conditions for frequency dominance.

Lemma 5. *If w is in $H(\hat{h}(v))$ and dominates v then w is a frequency dominator of v.*

Proof. Let $w \in dom(v)$ and $w \in H(\hat{h}(v))$. Notice that w satisfies $\hat{h}(v) \xrightarrow{*} w \xrightarrow{*} v$. Therefore, the lemma is trivially true when $\hat{h}(v) = v$. Now suppose $\hat{h}(v) = h(v)$. For contradiction, assume that w is not in $fdom(v)$. Then, Lemma 1 implies that there is a cycle C containing v but not w. Let z be the minimum vertex on C. Note that by Lemma 4, z is an ancestor of v. Hence, v belongs to $H(z)$ and by the definition of $h(v)$ we have that $z \xrightarrow{*} h(v)$. Also, $w \neq z$ (because $w \notin C$),

thus $z \xrightarrow{+} w$. Now observe that the DFS-tree path from s to z followed by the part of C from z to v is an s-v path avoiding w. This contradicts the fact that $w \in dom(v)$. □

Now, from Lemma 5 and Lemma 2 we get:

Corollary 1. *Let w be vertex such that*

(a) $w \in H(\hat{h}(v))$ and $w \in dom(v)$, or
(b) $w \in H^r(\hat{h}^r(v))$ and $w \in dom^r(v)$.

Then w is a frequency dominator of v.

It is easy to show, using Lemma 3, that the above conditions suffice to compute frequency dominators. (See Lemma 6 below.) Note that in general $dom(v) \cap dom^r(v) \supseteq \{v\}$, so several vertices may satisfy both (a) and (b) of Corollary 1. Also, we point out that a vertex w that satisfies $w \in dom^r(v)$ and $w \in H(\hat{h}(v))$ may not be a frequency dominator of v. (Consider $w = e$ and $v = c$ in Figure 2 for an example.)

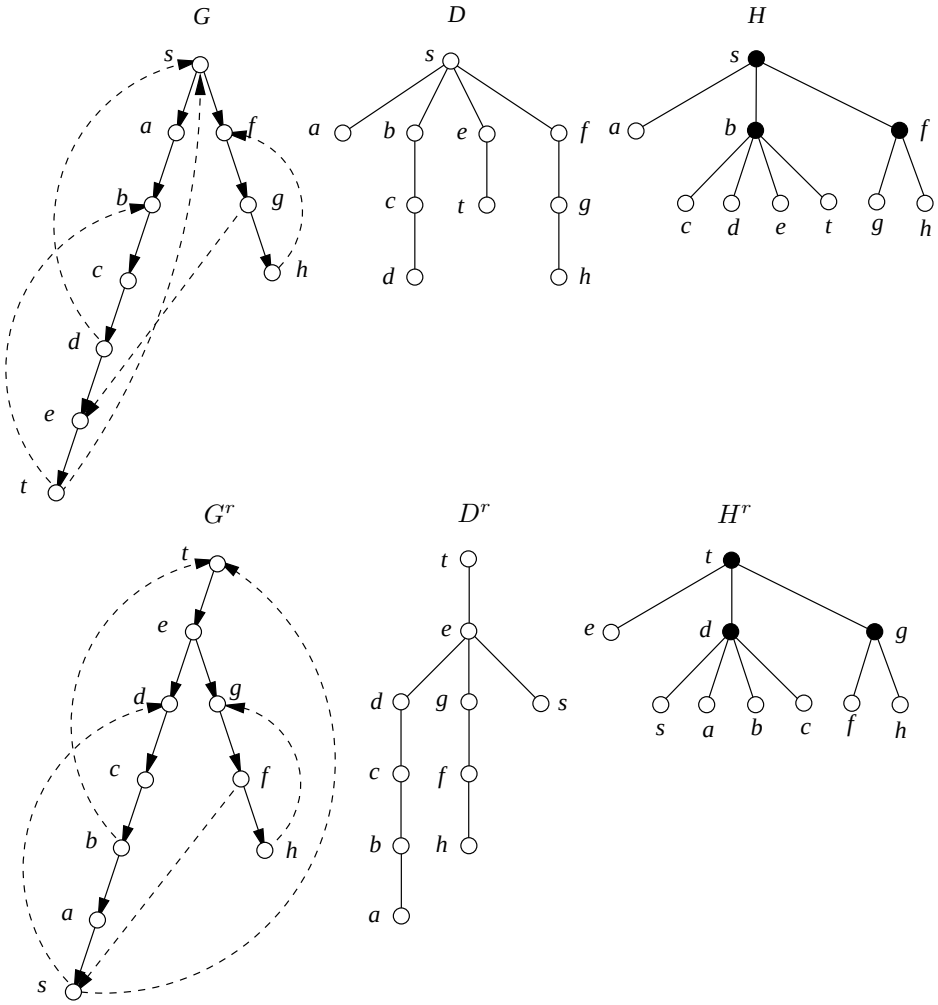
Preprocessing. In the preprocessing phase we compute the interval tree H, the dominator tree D, and the $\hat{h}$ function in G. We also compute the corresponding structures in G^r. For the query algorithm we need to test in constant time if two vertices are related in one of the trees that we have computed. To that end, we assign to the vertices in each tree $T \in \{H, H^r, D, D^r\}$ a preorder and a postorder number[2], denoted by pre_T and $post_T$ respectively. Then x is an ancestor of y in T if and only if $pre_T(x) \leq pre_T(y)$ and $post_T(x) \geq post_T(y)$. Calculating these numbers takes $O(n)$ time.

Queries. To find $fdom(y)$ we report y and all the proper ancestors x of y in D such that x is a descendant of $\hat{h}(y)$ in H. To that end, we visit the dominators of y starting from $d(y)$ and moving towards s. At each visited vertex x we test whether $x \in H(\hat{h}(y))$, which can be done in constant time using the pre_H and $post_H$ numberings. We show below that we can stop the reporting process as soon as we find a dominator of y not in $H(\hat{h}(y))$. (See Lemma 6.) Similarly, we report all the proper ancestors x of y in D^r such that x is a descendant of $\hat{h}^r(y)$ in H^r. Therefore, we can answer the reporting query in $O(|fdom(y)|)$ time.

To test whether $x \in fdom(y)$ in constant time, it suffices to test if condition (a) or condition (b) of Corollary 1 holds. Again this is easily accomplished with the use of the preorder and postorder numbers for D and D^r.

Figure 2 illustrates the computations performed by our algorithm. Next, we show that the algorithm finds all the frequency dominators of a query vertex.

[2] A preorder traversal of a tree visits a vertex before visiting its descendants; a postorder traversal visits a vertex after visiting all its descendants. The corresponding numberings are produced by assigning number i to the ith visited vertex.



vertex y	$dom(y) \cap H(\hat{h}(y))$	$dom^r(y) \cap H^r(\hat{h}^r(y))$	$fdom(y)$	region of y
s	$\{s\}$	$\{s\}$	$\{s\}$	$\{s\}$
a	$\{s, a\}$	$\{a, b, c, d\}$	$\{s, a, b, c, d\}$	$\{a\}$
b	$\{b\}$	$\{b, c, d\}$	$\{b, c, d\}$	$\{b, c, d\}$
c	$\{b, c\}$	$\{c, d\}$	$\{b, c, d\}$	$\{b, c, d\}$
d	$\{b, c, d\}$	$\{d\}$	$\{b, c, d\}$	$\{b, c, d\}$
e	$\{e\}$	$\{e, t\}$	$\{e, t\}$	$\{e, t\}$
f	$\{f\}$	$\{f, g\}$	$\{f, g\}$	$\{f, g\}$
g	$\{f, g\}$	$\{g\}$	$\{f, g\}$	$\{f, g\}$
h	$\{f, g, h\}$	$\{f, g, h\}$	$\{f, g, h\}$	$\{h\}$
t	$\{e, t\}$	$\{t\}$	$\{e, t\}$	$\{e, t\}$

Fig. 2. Computation of the *fdom* relation using dominator trees and interval trees. The DFS-tree arcs of G and G^r are solid and remaining arcs are dashed. Special vertices are filled in the interval trees.

Lemma 6. *The query algorithm is correct.*

Proof. From Corollary 1 we have that all the vertices that are reported during a query for $fdom(y)$ are indeed frequency dominators of y. Now we argue that any frequency dominator x of y that is not found using condition (a) of Corollary 1 is found using condition (b). In this case, Lemma 3(a) implies that x is a postdominator of y. This fact combined with Lemma 3(c) means that condition (b) of Corollary 1 applies.

We also need to argue that we can stop the search procedure for the vertices in $fdom(y)$ as soon as we find the deepest dominator of y that is not in $H(\hat{h}(y))$. To that end, note that for any vertex z such that $\hat{h}(y) \xrightarrow{*} z \xrightarrow{*} y$, $z \in H(\hat{h}(y))$. Therefore, if $x \in dom(y)$ and $x \notin H(\hat{h}(y))$ then $x \xrightarrow{+} \hat{h}(y)$, which proves our claim. □

Strict frequency dominance. Let us call vertex x a *strict frequency dominator* of vertex y if and only if in each s-t path containing y, the number of appearances of x is strictly greater than the number of appearances of y. E.g., in Figure 2, f and g are strict frequency dominators of h. We can show that such an x must satisfy both conditions (a) and (b) of Corollary 1. Hence, with our *fdom* structure we can test if x is a strict frequency dominator of y in constant time, and report all the strict frequency dominators of y in $O(|fdom(y)|)$ time. Note, however, that this bound is not proportional to the size of the output.

Inverse frequency dominance. Let $fdom^{-1}(x)$ denote the set of vertices y such that $x \in fdom(y)$. The problem of reporting $fdom^{-1}(x)$ can be reduced to the following task. We are given an arbitrary rooted tree T with n vertices, where each vertex x is assigned an integer label $\ell(x)$ in $[1, n]$, and we wish to support the following type of queries: For a vertex x and a label j, report all vertices $y \in T(x)$ with $\ell(y) \geq j$. Let k be the number of such vertices. We can achieve an $O(k \log n)$ query time (using $O(n)$ space), thus getting an $O(|fdom^{-1}(x)| \log n)$ bound for reporting $fdom^{-1}(x)$.

3 Computing regions

Recall that two vertices x and y are equivalent if and only if they appear equal number of times in any s-t path. Hence, x and y are equivalent if and only if $x \in fdom(y)$ and $y \in fdom(x)$. As in Section 2, we assume that G contains the arc (t, s). Then, by Lemma 1 it follows that x and y are equivalent if and only if they appear in the same set of cycles.

The structure of Section 2 clearly supports constant time queries that ask if two vertices are equivalent. Therefore, it is also straightforward to find the equivalence class of a given vertex y in $O(|fdom(y)|)$ time. Still, with this method we can only guarantee an $O(n^2)$ bound for computing the complete equivalence relation for all vertices (i.e., the regions of G). In this section we show how to

achieve this computation in overall $O(n)$ time, with a somewhat more involved manipulation of the structure of Section 2.

We start with a technical lemma that relates the structure of the dominator tree to that of the interval tree.

Lemma 7. *Let v be any vertex other than s. Then, all vertices w in $H(v)$ such that $d(w) \xrightarrow{+} v$ have the same immediate dominator $x = d(v)$.*

Proof. Let x be the minimum vertex that is an immediate dominator of any vertex in $H(v)$. Since $v \neq s$, $d(v)$ is a proper ancestor of v and so $x \xrightarrow{+} v$. Let w be a vertex such that $d(w) = x$. Then, w is not dominated by any vertex z that satisfies $x \xrightarrow{+} z \xrightarrow{+} w$, thus for each such z there exists a path P from x to w that avoids z. Also, since the vertices in $H(v)$ induce a strongly connected subgraph, then for any $y \in H(v)$ there is a path Q from w to y that contains only vertices in $H(v)$. Therefore, the catenation of P and Q is a path from x to y avoiding z. This implies that either $d(y) = x$ or $d(y) \in H(v)$. Setting $y = v$ gives $d(v) = x$. □

From the above lemma we immediately get:

Corollary 2. *The subgraph of D induced by the vertices in $H(v)$ and $d(v)$ is a tree rooted at $d(v)$.*

We begin with an initial partition of the vertices, which after a refinement process (described in Section 3.1) will produce the actual equivalence classes. Our first step is to label each vertex w in D with $\hat{h}(w)$. Then, each vertex has a unique label and two vertices are equivalent only if they have the same label. Let $L(v)$ denote the set of vertices labeled with v. Note that $L(v) = \emptyset$ if v is not special; otherwise $L(v)$ consists of v and the children of v in H that are not special. Our goal is to compute the equivalent vertices for each $L(v)$ separately, using information from the dominator tree D. To that end, let w be any vertex in $L(v)$. We define $\hat{d}(w)$ to be the nearest proper ancestor z of w in D with label v. If such a z does not exist for w, then we set $\hat{d}(w) = d(v)$. A simple and efficient way to compute the $\hat{d}$ function is by using *path compression*: Starting from the parent of w in D, we follow parent pointers until we reach a vertex z which is either the first vertex in $L(v)$ or $d(v)$. Then, we set $\hat{d}(w) = z$ and make z the parent of all the visited vertices. In order to guarantee that this process returns the correct $\hat{d}(w)$ for all w, we need to process the sets $L(v)$ in an appropriate order. We argue that it suffices to process the special vertices by decreasing depth in H.

Lemma 8. *Suppose that for each special vertex v, we process $L(v)$ after we have processed $L(u)$ for all special vertices $u \in H(v)$. Then, the path compression algorithm computes the $\hat{d}$ function correctly.*

Proof. Let v be the currently processed special vertex. We denote by $\hat{D}$ the dominator tree after the changes due to the path compressions performed so far, just before processing v. Let u be a special vertex in $H(v)$. Then Corollary 2 implies

that when the path compression algorithm was applied to $L(u)$, only vertices in $H(u) \cup \{d(u)\}$ were visited. Moreover, Lemma 7 and the fact that $H(u) \subset H(v)$ imply $d(v) \xrightarrow{*} d(u)$. Also note that the parent of $d(u)$ did not change. Now, since v is an ancestor of u in H, there is no vertex in $H(u)$ with label v. Hence, for any $y \in L(v)$ and $x \in L(v) \cup \{d(v)\}$, x is an ancestor of y in $\hat{D}$ if and only if it is an ancestor of y in D. The claim follows. □

The worst-case running time of this algorithm is $O(n \log n)$ [17], but in practice it is expected to be linear [9]. We can get an actual $O(n)$ bound for computing the $\hat{d}$ function via a simple reduction to the *marked ancestors problem* [2]. In the marked ancestors problem we are given a rooted tree T, the vertices of which can be either *marked* or *unmarked*. We wish to support the following operations: mark or unmark a given vertex, and find the nearest marked ancestor of a query vertex. In our case $T = D$ and initially all vertices are marked. Again we process the special vertices by decreasing depth in H. When we process a special vertex v we perform a nearest marked ancestor query for all $w \in L(v)$; we set $\hat{d}(w)$ to be the answer to such a query for w. Then we unmark all vertices in $L(v)$. It follows from similar arguments as in the proof of Lemma 8 that this process computes the correct $\hat{d}$ function. Also, since we never mark any vertices, this process corresponds to a special case of the disjoint set union (DSU) problem, for which the result of Gabow and Tarjan [7] gives constant amortized time complexity per operation. Since we perform two DSU operations per vertex, the $O(n)$ time bound follows.

Now it remains to show how to compute the equivalence classes for each $L(v)$ using the $\hat{d}$ function.

3.1 Equivalence classes in $L(v)$

Let D_v be the graph with vertex set $L(v) \cup \{d(v)\}$ and edge set $\{(\hat{d}(w), w) \mid w \in L(v)\}$. From Corollary 2 we have that D_v is a tree rooted at $d(v)$. Let D'_v be the forest that results after removing $d(v)$ and its adjacent edges from D_v; the trees in this forest are rooted at the children of $d(v)$ in D_v. Let w be any vertex in D'_v. Note that by condition (a) of Corollary 1 we have that all ancestors of w in D'_v are in $fdom(w)$. The next lemma allows us to compute the equivalence classes in each tree of D'_v separately.

Lemma 9. *Let x be a vertex in D, and let x_1 and x_2 be two distinct children of x in D. Then no pair of vertices $y_1 \in D(x_1)$ and $y_2 \in D(x_2)$ are equivalent. Also x can be equivalent with at most one of y_1 and y_2.*

Proof. Let y and z be two equivalent vertices. Then $y \in fdom(z)$ and $z \in fdom(y)$. Since the *dom* relation is antisymmetric, Lemma 3(a) implies (with no loss of generality) that $y \in dom(z)$ and $z \in dom^r(y)$. Hence, the vertices y_1 and y_2, defined in the statement of the lemma, cannot be equivalent. So, they also cannot be both equivalent with x. □

Now we can consider a single tree T in D'_v. Lemma 9 also implies that only vertices with ancestor-descendant relation in T can be equivalent. Our plan is to process T bottom-up, and at each vertex w of T test if $w \in \mathit{fdom}(p_T(w))$. If the outcome of the test is true then we add $p_T(w)$ to the same equivalence class as w, since we already know that $p_T(w) \in \mathit{fdom}(w)$. If, on the other hand, the outcome of the test is false, then the next lemma infers that no other vertex can be in the same equivalence class with w.

Lemma 10. *Let T be a tree of D'_v, and let w be a non-root vertex in T. If w is not equivalent with $p_T(w)$ then w is not equivalent with any of its proper ancestors in T.*

Proof. For contradiction, assume w is equivalent with a proper ancestor z of $u = p_T(w)$ in T. Notice that $v \xrightarrow{*} z \xrightarrow{+} u$. Then, we either have a cycle C_{-u} through w and z but not u, or a cycle C_u through u but not w and z. The first case contradicts the fact that $u \in \mathit{fdom}(w)$. In the second case, let x be the minimum vertex on C_u. From Lemma 4 and the fact that $u \in L(v)$, we have $x \xrightarrow{*} v \xrightarrow{*} z$. But then, the DFS-tree path from x to u, followed by the part of C_u from u to x, forms a cycle through z that does not contain w. This contradicts the fact that w and z are equivalent. □

4 A General Framework

We can show that the problems we have considered in this paper can be formulated in more general terms as follows. We are given a collection $\mathcal{G}$ of k directed graphs $G_i = (V_i, A_i)$, $1 \leq i \leq k$, where each graph G_i represents a transitive binary relation R_i over a set of elements $U \subseteq V_i$. That is, for any $a, b \in U$, we have aR_ib if and only if b is reachable from a in G_i. Let R be the relation defined by: aRb if and only if aR_ib for all $i \in \{1, \ldots, k\}$. I.e., b is reachable from a in all graphs in $\mathcal{G}$. (Note that R is also transitive.) For instance, consider the relation that is defined by the pairs (a, b) of vertices of a directed graph, such that a is both a dominator and a postdominator of b. In this case we have $\mathcal{G} = \{D, D^r\}$ (where the edges of the dominator trees are directed from parent to child).

Our goal is to find an efficient representation of R with a data structure that, for any given $b \in U$, can report fast all elements a satisfying aRb. Another related problem is to bound the combinatorial complexity of R, i.e., the size of a directed graph $G = (V, A)$, with $U \subseteq V$, such that for any $a, b \in U$, aRb if and only if b is reachable from a in G. Some interesting results can be obtained for several special cases. For instance, in the simple case where $k = 2$ and G_1 and G_2 are (directed) paths, it can be shown that the size of G is $O(n \log n)$, and moreover that there exist paths that force any G to have $\Omega(n \log n)$ size [18]. On the other hand, we can represent R with a Cartesian tree in $O(n)$ space, and answer reporting queries in time proportional to the number of reported elements [6].

Acknowledgements. We would like to thank Bob Tarjan, Renato Werneck, and Li Zhang for some useful discussions.

References

1. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA, 1986.
2. S. Alstrup, T. Husfeldt, and T. Rauhe. Marked ancestor problems. In *Proc. 39th IEEE Symp. on Foundations of Computer Science*, page 534.
3. T. Ball. What's in a region?: or computing control dependence regions in near-linear time for reducible control flow. *ACM Lett. Program. Lang. Syst.*, 2(1-4):1–16, 1993.
4. 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 Journal on Computing*. To appear. Preliminary version available online at `http://arxiv.org/abs/cs.DS/0207061`.
5. L. Carter, J. Ferrante, and C. Thomborson. Folklore confirmed: reducible flow graphs are exponentially larger. In *Proc. 30th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 106–114, 2003.
6. H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th ACM Symp. on Theory of Computing*, pages 135–143, 1984.
7. H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–21, 1985.
8. L. Georgiadis and R. E. Tarjan. Finding dominators revisited. In *Proc. 15th ACM-SIAM Symp. on Discrete Algorithms*, pages 862–871, 2004.
9. L. Georgiadis, R. E. Tarjan, and R. F. Werneck. Finding dominators in practice. *Journal of Graph Algorithms and Applications (JGAA)*, 10(1):69–94, 2006.
10. R. Johnson, D. Pearson, and K. Pingali. The program structure tree: computing control regions in linear time. *SIGPLAN Not.*, 29(6):171–185, 1994.
11. B. Lee, K. Resnick, M. D. Bond, and K. S. McKinley. Correcting the dynamic call graph using control-flow constraints. In *Proc. Compiler Construction, 16th International Conference, CC 2007*, pages 80–95, 2007. Full paper appears as technical report TR-06-55, The University of Texas at Austin, Department of Computer Sciences, 2006.
12. T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems*, 1(1):121–41, 1979.
13. R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–59, 1972.
14. R. E. Tarjan. Testing flow graph reducibility. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, pages 96–107, 1973.
15. R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.
16. R. E. Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6(2):171–85, 1976.
17. R. E. Tarjan and J. van Leeuwen. Worst-case analysis of set union algorithms. *Journal of the ACM*, 31(2):245–81, 1984.
18. L. Zhang. Personal communication, 2008.