Corrigendum: A New, Simpler Linear-Time Dominators Algorithm

ADAM L. BUCHSBAUM
AT&T Labs
and
HAIM KAPLAN
Tel Aviv University
and
ANNE ROGERS
University of Chicago
and
JEFFERY R. WESTBROOK
San Diego, CA

Corrigendum to *ACM Trans. on Programming Languages and Systems*, **20**(6):1265–1296, 1998.

Categories and Subject Descriptors: D.3.4 [**Programming Languages**]: Processors—*compilers*; E.1 [**Data Structures**]: Graphs and Networks; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*

General Terms: Algorithms, Languages, Performance, Theory

Additional Key Words and Phrases: Compilers, dominators, flowgraphs, microtrees, path compression

---

1. INTRODUCTION

The complexity analysis of the dominators algorithm we described in our original paper [Buchsbaum et al. 1998] required a special *link-eval* data structure (defined below), which we claimed performed m operations on an n-node, ℓ-leaf tree in $O((m+n)\alpha(m+\ell,\ell))$ time. This claim was based on the application of a new bottom-up disjoint set union result to the original link/eval data structure presented

---

Authors' addresses: A. L. Buchsbaum, AT&T Labs, Shannon Laboratory, 180 Park Ave., Florham Park, NJ 07932, USA, e-mail: alb@research.att.com; H. Kaplan, School of Computer Science, Tel Aviv University, Tel Aviv, Israel, e-mail: haimk@math.tau.ac.il; A. Rogers, Dept. of Computer Science, University of Chicago, 1100 E 58th St., Chicago, IL 60637, USA, e-mail: amr@cs.uchicago.edu; J. R. Westbrook, 4031 South Hempstead Cr., San Diego, CA 92116, USA, e-mail: jwestbrook@acm.org.

by Tarjan [1979]. That application is faulty: Tarjan's implementation of link/eval where the operation is min does not perform set-union operations as we claimed; a counterexample can be constructed using a simple path as the link-eval tree.

To repair the running time of our algorithm, we present a modified link-eval data structure with the required time bound. It effects a linear-time reduction to the case of a path and then applies an existing linear-time solution for paths [Alstrup et al. 1999]. The path solution in turn relies on a data structure for disjoint set union when the union tree is known in advance [Gabow and Tarjan 1985].

2. LINK/EVAL WITH BOTTOM-UP LINKING

Let T be a rooted tree with real-valued nodes. We want to maintain a forest F on the nodes of T. Initially each node of T is a singleton tree in F. Let $p_T(u)$ denote the parent of u in T. The *link-eval data structure* operations are:

link(u). Add edge $\{u, p_T(u)\}$ to F, making u a child of $p_T(u)$ in F.

eval(u). Let r be the root of the tree in F that contains u. Return any minimum-valued node on the path from r to u.

update(u, a). Replace the value of u with the minimum of its current value and a. Valid only if u is a tree root in F.

Tarjan [1979] shows how any sequence of m link-eval operations can be performed in $O((m+n)\alpha(m+n, n))$ time, where n is the number of nodes in T. In our dominators algorithm [Buchsbaum et al. 1998] we use a slightly different variation of *eval*(u), which we address in Section 3; for ease of exposition, we describe a data structure to handle the original definitions given above.

A sequence of *link*, *eval*, and *update* operations has the *bottom-up linking property* if no *link*(u) is performed before *link*(x) is performed for every proper descendent x of u in T. Let ℓ be the number of leaves of T. We describe a data structure that performs m operations with the bottom-up linking property in $O((m+n)\alpha(m+\ell, \ell))$ time. The idea is to isolate maximal unary paths in T; apply to them a special, linear-time link-eval data structure that operates on paths; and use Tarjan's [1979] data structure on the remainder of T, which will have $O(\ell)$ nodes.

2.1 Details

A *unary node* in a rooted tree is a node with precisely one child; a *unary path* is a top-down path of unary nodes terminating at a non-unary node. See Figure 1. Form T' by contracting maximal unary paths in T as follows. For each such path $(u_1, ..., u_i)$ in T such that $u_1, ..., u_{i-1}$ are unary, u_1 is the root of T or $p_T(u_1)$ is not unary, and u_i is not unary (leaves are not unary), there is a node u' in T'. Define $s(u_j) = u'$ for $1 \leq j \leq i$. Define $S(u') = \{u_1, ..., u_i\}$. For $v = u_j$ on such a path, define $\mathit{index}(v) = j$. In T', then, $p_{T'}(u') = s(p_T(u_1))$ (with an appropriate boundary definition to handle the root). This defines T'. Initially, $\mathit{value}_{T'}(u') = \mathit{value}_T(u_i)$ where $i = \max\{\mathit{index}(v) : v \in S(u')\}$, i.e., the value of the deepest node on the path contracted into u'. Note that every node u in T is in precisely one unary path; in the trivial case the path is the singleton node itself, and $\mathit{index}(u) = 1$.

Let *path-link*, *path-eval*, and *path-update* be the corresponding operations on link-eval structures that are restricted to act only on paths. Maintain such a structure

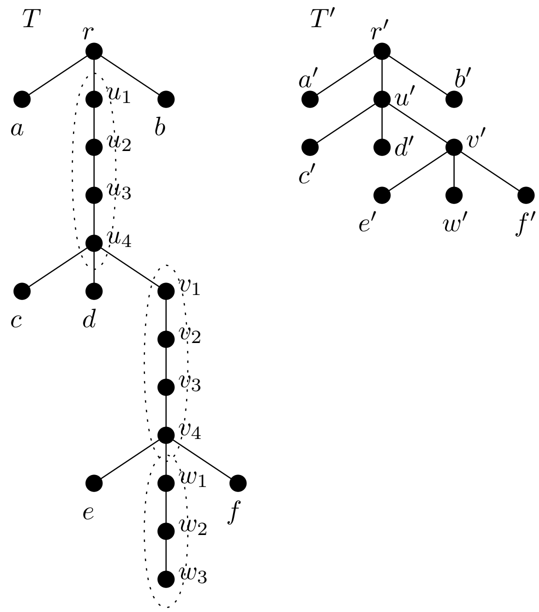

Fig. 1. Left: A link-eval tree T. Maximal unary paths are circled (dotted circles). Uncircled nodes are singleton unary paths. Right: The contracted tree T', with one node per unary path in T.

for each maximal unary path in T. Maintain a Tarjan link-eval structure on T'; call the operations on this data structure *link*$'$, *eval*$'$, and *update*$'$.

Implement *link*, *eval*, and *update* on T as follows.

link(u). Let $j = \mathit{index}(u)$ and $u' = s(u)$. If $j > 1$ (u is not the top node on its path) execute *path-link*(u) followed by *update*$'(u', \mathit{value}_T(p_T(u)))$. If $j = 1$ (u is the top node), execute *link*$'(u')$. This ensures that *eval*$'(v')$, where v' is a descendent of u' in T', returns the proper value; it assumes the bottom-up linking property.

eval(u). Let $a =$ *path-eval*(u); let $u' = s(u)$. If *link*$'(u')$ was previously performed, return whichever of a and *eval*$'(p_{T'}(u'))$ has minimum *value*. Otherwise (*link*$'(u')$ has not been performed), return a.

update(u, a). Let $u' = s(u)$. Perform *path-update*(u, a) followed by *update*$'(u', a)$.

For example, consider Figure 1 and say that all descendents of u_2 have been *link*ed. Thus, u_2 is a tree root in F, the forest corresponding to T, and u' is a tree root in F', the forest corresponding to T'. Consider *eval*(w_2): *path-eval*(w_2) returns a min-valued node among w_1 and w_2, and *eval*$'(v')$ ($v' = p_{T'}(w')$) returns a min-valued node in T' among u' and v'. By the prior operations, $\mathit{value}_{T'}(v')$ is the minimum value associated to the v_i's, and $\mathit{value}_{T'}(u')$ is the minimum value associated to u_2, u_3, and u_4. The result of *eval*(w_2) is thus correct.

In general, let F' be the forest maintained by *link*$'$/*eval*$'$/*update*$'$ that corresponds to F. At any point in the sequence of operations, the following can be seen to be true by induction.

(1) If u is a tree root in F, then $s(u)$ is a tree root in F'.

(2) If u' is a tree root in F', then some member of $S(u')$ is a tree root in F.

(3) For any u' in F', if u' is a root, let $r(u')$ denote the deepest node in $S(u')$ that is a root in F; otherwise, let $r(u')$ denote the highest node in $S(u')$. Then

$$\mathit{value}_{T'}(u') = \min\{\mathit{value}_T(x) : x \in S(u'), x \text{ a descendent in } T \text{ of } r(u')\}.$$

Based on (1)–(3) and assuming bottom-up linking, we claim that the data structure is correct. That is, if the *link*, *eval*, and *update* operations were performed on a vanilla Tarjan data structure, the results of the *eval* operations would be the same as if they were performed as above. To see this, consider an *eval*(u), and let r be the root of the tree in F that contains u in the comparable vanilla Tarjan data structure. By (1) and (2), $r' = s(r)$ is the root of the tree in F' that contains $u' = s(u)$. If $r' = u'$, then u and r lie on the same unary path and $\mathit{link}'(u')$ has not yet been performed. Thus, *path-eval*(u) returns the correct node. Otherwise $r' \neq u'$, in which case u and r lie on different unary paths and $\mathit{link}'(u')$ was previously performed. In this case, *path-eval*(u) returns some min-valued node on the path from u to the root (call it v) of its unary path in T; by (3), $\mathit{eval}'(p_{T'}(u'))$ returns some min-valued node on the path from $p_T(v)$ to r. Thus the min-valued node in $\{\mathit{path\text{-}eval}(u), \mathit{eval}'(p_{T'}(u'))\}$ is the correct result.

If m operations are performed on T, then $O(m)$ operations are performed on T' and the path-link-eval data structures. Alstrup et al. [1999] show how to do the path-link-eval operations in linear time on a RAM, using Gabow and Tarjan's [1985] result for disjoint set union when the union tree is known in advance. The total time is thus $O((m + n)\alpha(m + \ell, \ell))$, where ℓ is the number of leaves in T, because there are $O(\ell)$ nodes in T'.

3. A VARIATION OF *EVAL*

Our dominators algorithm [Buchsbaum et al. 1998] as well as that of Lengauer and Tarjan [1979] use a different definition of *eval*(u):

eval(u). Let r be the root of the tree in F that contains u. If $u = r$, return r. Otherwise, let v be the child of r on the path from r to u, and return any minimum-valued node on the path from v to u.

It is easy to reduce this variation to the original Tarjan definitions in the general case. Assume a data structure that implements the operations as defined in Section 2; call those operations link_0, eval_0, and update_0. Let *link*, *eval*, and *update* refer to operations on a modified structure that abstracts *eval*(u) as defined in this section. Along with an underlying data structure that performs link_0, eval_0, and update_0, we maintain a linked list *PENDING*(u), initially empty, and a boolean bit *LINKED*(u), initially `false`, with each node $u \in T$. To implement *link*(u), we perform the following steps.

(1) If $\mathit{LINKED}(p_T(u))$ = `true`, then execute $\mathit{link}_0(u)$; otherwise append u to $\mathit{PENDING}(p_T(u))$.

(2) Set $\mathit{LINKED}(u) \leftarrow$ `true`.

(3) For all v in *PENDING*(u), remove v from *PENDING*(u), and execute $\mathit{link}_0(v)$.

We can then implement $eval(u)$ and $update(u,a)$ simply by executing $eval_0(u)$ and $update_0(u,a)$, respectively. We see that for all v: (a) after any operation $PENDING(v)$ is empty whenever $LINKED(v) = \texttt{true}$; and (b) $link_0(v)$ is not performed before $LINKED(p_T(v)) = \texttt{true}$. These observations imply the following invariant, which in turn implies the correctness of the reduction.

INVARIANT 3.1. *After any operation, consider any node v. Let r be the root of the tree containing v in the link-eval forest; let r_0 be the root of the tree containing v in the $link_0$-$eval_0$ forest. (Both forests are induced on T by the respective data structures.)*

(1) *If $v = r$, then $r = r_0$.*
(2) *If $v \neq r$, then r_0 is the child of r on the path from r to v.*

Alternatively, one could implement the data structure in Section 2 to support directly the above variation of $eval(u)$. In this case, the implementations of $link(u)$ and $eval(u)$ handle the "off-by-one" issue of *eval*'s terminating below the root, and the invariants become more complicated to describe.

4. CONCLUSION

While our original dominators algorithm can be implemented in linear time with the revised link-eval data structure, the link-eval structure requires a RAM. Implementing link-eval with bottom-up linking on a pointer machine remains open, although it can be done if *update* is not required. Georgiadis and Tarjan [2004] have devised a new dominators algorithm that runs in linear time on a pointer machine.

ACKNOWLEDGMENTS

We thank Loukas Georgiadis and Bob Tarjan for pointing out the flaw in our original paper.

REFERENCES

ALSTRUP, S., HAREL, D., LAURIDSEN, P. W., AND THORUP, M. 1999. Dominators in linear time. *SIAM J. Comput. 28,* 6, 2117–2132.

BUCHSBAUM, A. L., KAPLAN, H., ROGERS, A., AND WESTBROOK, J. R. 1998. A new, simpler linear-time dominators algorithm. *ACM Trans. Prog. Lang. Syst. 20,* 6, 1265–1296.

GABOW, H. N. AND TARJAN, R. E. 1985. A linear-time algorithm for a special case of disjoint set union. *J. Comput. Syst. Sci. 30,* 2, 209–221.

GEORGIADIS, L. AND TARJAN, R. E. 2004. Finding dominators revisited. In *Proc. 15th ACM-SIAM Symp. on Discrete Algorithms.* SIAM, Philadelphia, PA. To appear.

LENGAUER, T. AND TARJAN, R. E. 1979. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Prog. Lang. Syst. 1,* 1, 121–141.

TARJAN, R. E. 1979. Applications of path compression on balanced trees. *J. ACM 26,* 4, 690–715.