A NEW ALGORITHM FOR FINDING WEAK COMPONENTS

by

Robert Endre Tarjan

Memorandum No. ERL-432

3 April 1974

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720
A NEW ALGORITHM FOR FINDING WEAK COMPONENTS

Robert Endre Tarjan
Department of Electrical Engineering
and Computer Sciences
University of California
Berkeley, California

ABSTRACT

Pacault has devised an algorithm for finding the weak components of a directed graph G having V vertices and E edges in \(O(V+E)\) time.

This paper presents another \(O(V+E)\) algorithm which is more straightforward and which calculates weak components of certain subgraphs of G as well as of G.

Keywords: algorithm, connectivity, directed graph, strong components, topological sorting, weak components.

† This work was partially supported by the National Science Foundation, Grant Number NSF-GJ-35604X, and by a Miller Research Fellowship.
A NEW ALGORITHM FOR FINDING WEAK COMPONENTS

Robert Endre Tarjan

Let $v$ and $w$ be two vertices in a directed graph $G$. There is a non-path from $v$ to $w$ if there is no path from $v$ to $w$. If there is a non-path from $v_1$ to $v_{i+1}$ for $1 \leq i \leq n$, then $v_1, v_2, \ldots, v_n$ is a sequence of non-path steps from $v_1$ to $v_n$. Vertices $v$ and $w$ are in the same weak component of $G$ if they are in the same strong component (there is a path from $v$ to $w$ and from $w$ to $v$) or if there is a sequence of non-path steps from $v$ to $w$ and from $w$ to $v$. The relation "$v$ and $w$ are in the same weak component" is an equivalence relation on the vertices of $G$ [1].

Pacault [2] has devised an algorithm to find the weak components of $G$ in $O(V+E)$ time and space, if $G$ has $V$ vertices and $E$ edges. Here is another $O(V+E)$ algorithm which is directly related to Pacault's but is more straightforward. It gives the weak components of certain subgraphs of $G$ as well as of the entire graph.

The weak components of $G$ are unions of its strong components.

1. Find the strong components $C_i$ of $G$ and shrink each to a single vertex $i$, with an edge $(i,j)$ in the new graph if and only if there is an edge $(v,w)$ in $G$ with $v \in C_i$ and $w \in C_j$.

The weak components of the resulting acyclic graph give the weak components of $G$. Shrinking the strong components requires $O(V+E)$ time using depth-first search [3]. Henceforth, assume that $G$ is acyclic.

2. Number the vertices of $G$ from 1 to $V$ so that all arcs run from a lower numbered to a higher numbered vertex.
This operation is called **topologically sorting** the vertices of $G$ and may be carried out in $O(V+E)$ time using depth-first search [4] or other methods [5]. In fact, steps (1) and (2) may if desired be carried out using a single depth-first search. Henceforth assume that vertices are identified by number.

**Lemma 1**

The vertices of any weak component of $G$ are consecutive.

**Proof**

Let $i < j < k$ with $i$ and $k$ in the same weak component. There is a non-path from $k$ to $j$, a non-path from $j$ to $i$, and a sequence of non-path steps from $i$ to $k$. Thus, $j$ is in the component.

Q.E.D.

Now we use the obvious approach; namely, we examine vertices in order, one by one, computing weak components as we go. By Lemma 1, the vertex numbering gives an ordering of the weak components, and the set of largest vertices of the weak components uniquely characterizes them. For $1 \leq k \leq V$, let $B(k) = \{j|\ (j,k) \text{ is an edge}\}$ and let $G_k$ be the induced subgraph of $G$ with vertices $\{j|\ 1 \leq j \leq k\}$. Let $L_k = \{v_1 < v_2 < \ldots < v_n\}$ be the set of largest vertices of the weak components of $G_k$. For $1 \leq i \leq n$, let $W_k(v_i)$ be the weak component containing vertex $v_i$. A **sink** $j \in W_k(v_i)$ is a vertex such that all edges from $j$ lead to vertices larger than $v_i$.

Let $S(v_i)$ be the set of sinks of $W_k(v_i)$.

**Lemma 2**

Let $1 \leq k < V$, and let $v_\downarrow = \min\{v_j|\ v_j \geq \max(\{i \in B(k+1)\} \cup \{0\})\}$.

(A) If $S_k(v_\downarrow) \subseteq B(k+1)$ then $L_{k+1} = \{v_1, v_2, \ldots, v_\downarrow, k+1\}$; otherwise $L_{k+1} = \{v_1, v_2, \ldots, v_{\downarrow-1}, k+1\}$.
Proof

Vertex \( k+1 \) has no edges leading to smaller numbered vertices; thus, if \( i, j \leq k \) there is a path from \( i \) to \( j \) in \( G_k \) iff there is a path from \( i \) to \( j \) in \( G_{k+1} \). It follows that the weak components of \( G_{k+1} \) are unions of \( \{k+1\} \) and the weak components of \( G_k \). There is a non-path from \( v_{\ell} + 1 \) to \( k + 1 \), so \( \bigcup_{i \geq \ell} W_k(v_i) \subseteq W_{k+1}(k+1) \). If \( S_k(v_\ell) \not\subseteq \mathcal{B}(k+1) \) then there is a non-path from some vertex in \( S_k(v_\ell) \) to \( k + 1 \), and \( W_k(v_\ell) \subseteq W_{k+1}(k+1) \). There is a non-path from every vertex in \( W_k(v_j) \) to every vertex in \( W_k(v_\ell) \) for \( j > \ell \), so there must be a path from every vertex in \( W_k(v_\ell) \) to every vertex in \( W_k(v_j) \) for all \( j > \ell \). Thus, there is a path from every vertex in \( W_k(v_j) \) with \( j < \ell \) to \( k + 1 \), and if \( S_k(v_\ell) \subseteq \mathcal{B}(k+1) \), then there is a path from every vertex in \( W_k(v_\ell) \) to \( k + 1 \). It follows that \( L_{k+1} \) has the value given in (A). (B) follows immediately from the definition of sinks.

Q.E.D.

Step (3) below is a program in Algol-like notation which uses Lemma 2 to calculate weak components. The implementation is a little tricky. At the beginning of the \( k \)th iteration of while loop \( a \), \( L = L_k \) and \( S(v) = S_k(v) \) for each \( v \in L_k \). We assume that \( L \) is a stack with highest element on top and the \( S(v) \)'s are linked lists with elements in decreasing order. At the beginning of the \( k \)th iteration of while loop \( a \), if vertex \( i \) appears in \( S(k) \), \( p(i) \) is a pointer to that occurrence; otherwise, if \( i \leq k \), \( p(i) = 0 \). We need the array \( p \) to carry out the calculation in (B) of Lemma 2.

Assume that the elements of \( \mathcal{B}(k) \) are in order from highest to lowest;
if not, we can sort them in \( O(V+E) \) time using a radix sort \([6]\).

(3) \textbf{begin} L:= S(1):= \{1\};

set \( p(1) \) to point to \( 1 \) in \( S(1) \);

\textbf{a:} \textbf{for} \( k = 1 \) \textbf{until} \( V - 1 \) \textbf{do begin}

\( m:= \max(\{j \in B(k+1)\} \cup \{0\}) \);

let \( v \) be the first element of \( L \);

\textbf{b:} \textbf{while} \( v > m \) \textbf{do begin}

\( w:= v; \)

delete \( v \) from \( L \);

let \( v \) be the new first element of \( L \) (-1 if none exists);

\textbf{end;}

\textbf{c:} \textbf{if} \( S(w) \subseteq B(k+1) \) \textbf{then begin}

add \( w \) to the front of \( L \);

\textbf{if} \( w = k \) \textbf{then begin}

\( S(k+1):= \emptyset; \)

\textbf{for} \( j \in B(k+1) \) \textbf{do} \( p(j):= 0; \)

\textbf{go to} \textbf{ADD;}

\textbf{end end;}

\textbf{d:} \( S(k+1):= S(k); \)

\textbf{ADD:} add \( k + 1 \) to the front of \( S(k + 1); \)

set \( p(k+1) \) to point to \( k + 1 \) in \( S(k+1); \)

\textbf{for} \( j \in B(k+1) \) \textbf{do if} \( p(j) \neq 0 \) \textbf{then begin}

delete \( j \) from \( S(k+1) \) using pointer \( p(j); \)

\( p(j):= 0; \)

-4-
end;
add k + 1 to the front of L;
end end;

Clearly, step (3) is an implementation of the result in Lemma 2, and $L = L_V$ when the algorithm is completed. Consider the test in statement c. Since $S(v)$ and $B(k+1)$ are in order, highest to lowest, we can step through $S(v)$ and $B(k+1)$ from highest element to lowest, and test whether $S(v) \subseteq B(k+1)$ in time proportional to $|B(k+1)|$. Statement d only requires setting the list pointer of $S(k+1)$ to point at list $S(k)$ and thus this statement requires a fixed finite time. At most $2V - 1$ elements are ever added to set $L$, and each execution of loop b deletes an element from $L$, so loop b is executed at most $2V$ times. A simple count shows that step (3) requires $O(V+E)$ time total. Thus steps (1) - (3) give an $O(V+E)$ algorithm to find weak components. The storage required in step (3) is $2V + E$ words for $B$, $V$ words for $L$, $V$ words for $P$, $2V$ words for $S$, and the total is $6V + E + C$ for some constant $C$.

This algorithm processes the vertices in order and has the nice feature that we can read off the weak components of $G_k$ for any $k$ as the calculation proceeds, which is not true of Pacault's original algorithm. The algorithm can also be modified to process vertices from highest to lowest instead of from lowest to highest.
REFERENCES


