Volume 7, number 6 INFORMATION PROCESSING LETTERS October 1978

# A LINEAR-TIME ALGORITHM FOR FINDING ALL FEEDBACK VERTICES

Michael R. GAREY
*Bell Laboratories, Murray Hill, NJ, U.S.A.*

Robert E. TARJAN *
*Stanford University, Stanford, CA, U.S.A.*

Received 28 December 1977; revised version received 21 June 1978

Depth-first search, directed graphs, feedback vertex

Let $G = (V, A)$ be a directed graph with $n = |V|$ vertices and $m = |A|$ arcs. A *feedback vertex* of $G$ is a vertex contained in every cycle of $G$. More generally, a *feedback set* of $G$ is a set $S \subseteq V$ such that every cycle of $G$ contains at least one vertex in $S$. The problem of determining a minimum cardinality feedback set arises in several contexts (e.g. [4], [5]) but, unfortunately, when phrased as a yes-no question it is NP-complete [2].

In certain applications it may be sufficient to discover whether a feedback set of some small size $k$ exists. In this paper we present an O($m$)-time algorithm which uses depth-first search to find all feedback vertices in an arbitrary strongly connected graph. (If the problem graph is not strongly connected, one can solve the feedback set problem by finding strongly connected components using the algorithm of [6] and then finding feedback sets separately in each component.) Our algorithm can be used to find feedback sets of size $k$ in time $O(n^{k-1}m)$, whereas the obvious algorithm, which tests each subset of $k$ vertices to determine whether removing those vertices results in an acyclic graph, requires time $O(n^k m)$.

In the development below we shall use various properties of depth-first search without proving them; a reader unfamiliar with these properties or the terminology should consult at least one of [1], [6], [7]. A depth-first search of a strongly connected graph $G$ partitions the arcs of $G$ into four classes:

(i) *Tree arcs.* An arc is a tree arc if it leads to a previously unreached vertex when it is traversed during the search. The tree arcs define a directed spanning tree of $G$ rooted at the start vertex of the search.

(ii) *Cycle arcs.* An arc is a cycle arc if it leads from a descendant to an ancestor in the spanning tree. Every cycle of $G$ uses at least one cycle arc.

(iii) *Forward arcs.* An arc is a forward arc if it is not a tree arc, but it leads from an ancestor to a descendant in the spanning tree.

(iv) *Cross arcs.* An arc is a cross arc if it is not a tree, cycle, or forward arc. Cross arcs join two vertices which are neither ancestors nor descendants of one another in the spanning tree.

Suppose the vertices of $G$ are numbered from 1 to $n$ as they are reached during the search. This numbering is a preorder numbering [3] of the spanning tree, and it has the property that any cross arc leads from a higher-numbered to a lower-numbered vertex. We shall assume that each vertex of $G$ is identified by its number.

Let $s$ be the smallest vertex with no exiting tree

* The research of this author was supported in part by National Science Foundation grant MCS 75-22870, by Office of Naval Research contract N00014-76-C-0688, and by Bell Telephone Laboratories.

arc. Then $(1, 2, 3, ..., s-1, s)$ is the tree path [1] from 1 to $s$. Since $G$ is strongly connected, there must be some arc $(s, t)$ leaving $s$ in $G$, and since that arc was not included in the tree, we must have $1 \leqslant t \leqslant s-1$. Thus $(s, t)$ is a cycle arc, and the only possible feedback vertices are those in the set $\{t, t+1, ..., s-1, s\}$.

We can refine this set of possibilities as follows. Let $y$ be the largest vertex with an entering cycle arc, i.e.,

$$y = \max\{v: \quad \text{there is a cycle arc } (u, v)\}.$$

Let $z$ be the largest vertex which is an ancestor of all vertices that have exiting cycle arcs, i.e.,

$$z = \max\{w: \quad \text{for every cycle arc } (u, v), \ w \text{ is an ancestor of } u\}.$$

Notice that $z$ must be an ancestor of $s$ and hence $z \leqslant s$. Since each cycle arc $(u, v)$ defines a cycle in $G$ consisting of $(u, v)$ and the tree path from $v$ to $u$, and since any feedback vertex must lie on all such cycles, every feedback vertex must be both a descendant of $y$ and an ancestor of $z$. It follows that the only possible feedback vertices are those belonging to the set $\{v: y \leqslant v \leqslant z\}$. If $y > z$ or if the subgraph of $G$ obtained by deleting all these vertices is not acyclic, then no feedback vertices exist.

In order to determine which of these candidates actually are feedback vertices, we compute two values for each vertex $v \in V$. These are defined as follows:

$$\mathbf{maxi}(v) = \max\{w \neq v: \quad w \leqslant z \text{ and there is a path from } v \text{ to } w \text{ which uses no cycle arcs and which passes through [2] only vertices } u \text{ satisfying } u > z\}.$$

$$\mathbf{loop}(v) = \begin{cases} \textbf{true} & \text{if there is a path from } v \text{ to some proper descendant } w \text{ of } z \text{ which uses no cycle arcs and which passes through only vertices } u \text{ satisfying } u > z; \\ \textbf{false} & \text{otherwise.} \end{cases}$$

[1] I.e., path in the spanning tree.

[2] A path $p$ is said to *pass through* a vertex $u$ if $u$ lies on $p$ and $u$ is not an endpoint of $p$.

By convention, we set $\max(\emptyset) = 0$.

The depth-first search maintains a stack of all vertices encountered so far which have outgoing arcs that have not yet been examined. Vertices are removed from this stack in reverse topological order [3] with respect to the acyclic graph obtained by deleting all cycle arcs. That is, if $(u, v)$ is a non-cycle arc, then $v$ is unstacked before $u$ during the search. Thus, if $z$ is known ahead of time, the following equations can be used to compute **maxi** and **loop** during the search, vertex-by-vertex in unstacking order:

$$\mathbf{maxi}(v) = \max(\{w: \quad w \leqslant z \text{ and } (v, w) \text{ is a non-cycle arc}\} \cup \{\mathbf{maxi}(w): w > z \text{ and } (v, w) \text{ is a non-cycle arc}\});$$

$$\mathbf{loop}(v) = \textbf{true} \quad \text{if and only if either } v \text{ is a proper descendant of } z \text{ or there is a non-cycle arc } (v, w) \text{ such that } w > z \text{ and } \mathbf{loop}(w) = \textbf{true}.$$

The theorem below characterizes the feedback vertices.

**Theorem 1.** *A vertex $v \in V$ is a feedback vertex if and only if*

(i) $y \leqslant v \leqslant z$;
(ii) *the set* $\{x: y \leqslant x \leqslant z\}$ *is a feedback set*;
(iii) $\mathbf{maxi}(u) \leqslant v$ for $1 \leqslant u < v$; *and*
(iv) $\mathbf{loop}(u) = \textbf{false}$ *for* $y \leqslant u < v$.

**Proof.** Suppose $v$ is a feedback vertex. The validity of properties (i) and (ii) was noted previously. Suppose (iii) fails, i.e., there is a vertex $u < v$ such that $\mathbf{maxi}(u) > v$. Since $G$ is strongly connected, vertex 1 has an entering arc which must be a cycle arc, say $(x, 1)$. Vertex $x$ must be a descendant of $z$. Consider the cycle consisting of the tree path from 1 to $u$, the path avoiding $v$ from $u$ to $\mathbf{maxi}(u)$, the tree path from $\mathbf{maxi}(u)$ to $z$ to $x$, and the cycle arc $(x, 1)$. This cycle avoids $v$, which is a contradiction to $v$ being a feedback vertex. Thus (iii) holds.

Similarly, suppose (iv) fails, i.e., there is a vertex $u$ such that $y \leqslant u < v$ and $\mathbf{loop}(u) = \textbf{true}$. Let $w$ be a proper descendant of $z$ such that there is a path from $u$ to $w$ which avoids $v$. Since $G$ is strongly connected, there is a path from $w$ to some ancestor of $z$. Consider the shortest such path. The last arc on it must be a cycle arc which must enter an ancestor of $y$, say $x$. Consider the cycle consisting of the tree path from $x$ to $u$, the path avoiding $v$ from $u$ to $w$, and the shortest path from $w$ to $x$. This cycle avoids $v$, which again is a contradiction. Thus (iv) holds.

Conversely, suppose (i) through (iv) hold for $v$, but $v$ is not a feedback vertex. Consider some cycle in $G$ which avoids $v$. By (ii), this cycle must contain some vertex in the range $y \leqslant x \leqslant z$. The cycle must also contain a cycle arc, say $(u, w)$. By the definitions of $y$ and $z$, $w \leqslant y$ and $u$ is a descendant of $z$. Let $x_1$ be the last vertex in the range $y \leqslant x_1 \leqslant z$ preceding $w$ on the cycle, i.e., the path in the cycle from $x_1$ to $w$ involves no other vertices in this range.

Suppose $x_1 < v$. Then $u \neq z$, since otherwise $u$ would have been picked as $x_1$. By the choice of $x_1$, there is a path from $x_1$ to $u$ which avoids all vertices in the range $y \leqslant x \leqslant z$. Thus **loop**$(x_1)$ = **true**, which contradicts (iv). Suppose on the other hand that $x_1 > v$. Let $x_2$ be the last vertex smaller than $v$ preceding $x_1$ on the cycle (such a vertex exists because $w \leqslant y$). Let $x_3$ be the first vertex in the range $v < x_3 \leqslant x_1$ following $x_2$ on the cycle. Then **maxi**$(x_2) \geqslant x_3$, which contradicts (iii). Thus both possibilities for $x_1$ are impossible, so (i) through (iv) imply that $v$ is a feedback vertex.

Theorem 1 suggests the following algorithm for finding all the feedback vertices in a strongly connected graph $G$.

*Algorithm*

*Step* 1. Carry out a depth-first search on $G$, numbering the vertices from 1 to $n$ in search order and computing $y$ and $z$.

*Step* 2. Test whether $y \leqslant z$ and whether $G$ becomes acyclic when all vertices $x$ satisfying $y \leqslant x \leqslant z$ are deleted. If either test fails, stop ($G$ has no feedback vertices).

*Step* 3. Repeat the depth-first search, computing **maxi**$(v)$ and **loop**$(v)$ for all vertices $v \in V$ using the equations given earlier.

*Step* 4. Initialize **maxitest** $\leftarrow \max\{$**maxi**$(u)$: $1 \leqslant u < y\}$, initialize the set of feedback vertices to the empty set, and initialize $v \leftarrow y$. Then perform the following steps:

```
Step 4a. If maxitest ⩽ $v$, add $v$ to the set of
         feedback vertices.
Step 4b. Set maxitest ← max{maxitest, maxi($v$)}.
Step 4c. If loop($v$) = false and $v < z$, set
         $v \leftarrow v + 1$ and return to Step 4a.
         Otherwise, stop.
```

It is not hard to verify that the total amount of time required by this algorithm is $O(m)$, where $m$ is the number of arcs in $G$.

## References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms (Addison-Wesley, Reading, MA, 1974).

[2] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher, eds., Complexity of Computer Computations (Plenum Press, New York, 1972) 85–104.

[3] D.E. Knuth, The Art of Computer Programming, Volume 1: Fundamental algorithms (Addison-Wesley, Reading, MA, 1968).

[4] Z. Manna, Mathematical Theory of Computation (McGraw-Hill, New York, 1974) 171.

[5] G.W. Smith and R.B. Walford, The identification of a minimal feedback vertex of a directed graph, IEEE Trans. Circuits and Systems CAS-22 (1975) 9–15.

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

[7] R.E. Tarjan, Finding dominators in directed graphs, SIAM J. Comput. 3 (1974) 62–89.