Science of Computer Programming 12 (1989) 151-155
North-Holland

AN ALGORITHM FOR TRANSITIVE REDUCTION OF AN ACYCLIC GRAPH

David GRIES
Computer Science Department, Cornell University, Ithaca, NY 14853-7501, U.S.A.

Alain J. MARTIN
California Institute of Technology, Pasadena, CA 91125, U.S.A.

Jan L. A. van de SNEPSCHEUT and Jan Tijmen UDDING
Department of Computing Science, Groningen University, Groningen, Netherlands

Communicated by M. Rem
Received June 1987
Revised March 1988

1. Introduction

We derive a new algorithm for computing the transitive reduction of a directed acyclic graph. The algorithm was developed by attempting to invert Warshall's algorithm for computing the transitive closure of a graph [4]. Thus, the concept of program inversion (cf. [2, 3]) led to the algorithm, although it could not be strictly used.

The transitive closure of a finite, directed graph is obtained by adding a directed edge wherever the original graph contains a path from the source to the destination node of the new edge. The graph thus obtained is called *closed.* For any graph, its transitive closure is uniquely determined. Further, the transitive closure of an acyclic graph is itself acyclic.

The transitive reduction of a finite, directed graph is obtained by removing edges whose absence do not affect the transitive closure. A graph thus obtained is called *reduced.* The intersection of acyclic graphs with the same transitive closure has that same transitive closure. Hence, the transitive reduction of an acyclic graph is uniquely determined—as the intersection of all acyclic graphs with the same transitive closure. On the other hand, the transitive reduction of a cyclic directed graph is not unique (see [1] for an analysis).

Because of the properties just mentioned, we restrict our attention to directed acyclic graphs.

In the sequel, we first describe Warshall's algorithm. We then analyze the algorithm informally and suggest an invariant for an algorithm for reducing an initially closed graph, which is in some sense the inverse of Warshall's algorithm. Finally, we obtain an algorithm for the transitive reduction of an acyclic graph.

We use the following notation. For expressions P and e and identifier x, P_e^x denotes P with all free occurrences of x textually replaced by e.

The graph contains N nodes numbered 0 onwards. For nodes i and j and integer k, $0 \leq k \leq N$, predicate $i \rightarrow^{\geq k} j$ denotes the presence in the initial graph of a path from i to j via one or more intermediate nodes, all of which have a number at least k; predicate $i \rightarrow^{<k} j$ denotes the presence in the initial graph of a path from i to j via one or more intermediate nodes, all of which have a number less than k. In the sequel, we omit the ranges of i, j, and k.

The graph that is being modified is recorded in variable b as an adjacency matrix: b_{ij} is equivalent to the presence of edge (i, j) in the graph. The initial value of variable b is given by constant a.

We list some properties of predicate $i \rightarrow^{<k} j$ and its "dual" $i \rightarrow^{\geq k} j$.

(0) $\forall(i,j :: b_{ij} \equiv a_{ij} \vee i \rightarrow^{<0} j) \equiv (b = a)$.

(1) $\forall(i,j :: b_{ij} \equiv a_{ij} \wedge \neg i \rightarrow^{\geq N} j) \equiv (b = a)$.

(2) $\forall(i,j :: b_{ij} \equiv a_{ij} \vee i \rightarrow^{<N} j) \equiv (b = \text{closure of } a)$.

(3) $\forall(i,j :: b_{ij} \equiv a_{ij} \wedge \neg i \rightarrow^{\geq 0} j) \equiv (b = \text{reduction of } a)$.

(4) $i \rightarrow^{<k+1} j \equiv i \rightarrow^{<k} j \vee ((i \rightarrow^{<k} k \vee a_{ik}) \wedge (k \rightarrow^{<k} j \vee a_{kj}))$.

(5) $i \rightarrow^{\geq k} j \equiv i \rightarrow^{\geq k+1} j \vee ((i \rightarrow^{\geq k+1} k \vee a_{ik}) \wedge (k \rightarrow^{\geq k+1} j \vee a_{kj}))$.

2. Warshall's algorithm for computing transitive closure

Warshall's algorithm is

```
TC:  $k := 0$; $\{C\}$
     do $k \neq N \rightarrow$ forall$(i, j$: $b_{ik} \wedge b_{kj}$: $b_{ij} := true)$; $\{C_{k+1}^k\}$
                 $k := k+1$ $\{C\}$
     od.
```

Its invariant C is

$$C: \quad b_{ij} \equiv a_{ij} \vee i \rightarrow^{<k} j.$$

We have omitted the universal quantification over i and j, as well as the ranges of i., j, and k. Notice that the **forall** statement can also be written as

$$\textbf{forall}(i, j :: b_{ij} := b_{ij} \vee (b_{ik} \wedge b_{kj})).$$

We prefer the latter form in our proofs and the original form in the program text. C_0^k is vacuously true on account of (0) and the initial condition $b = a$. $C \wedge k = N$ implies, on account of (2), that b is the transitive closure of a.

We must show the invariance of C over the loop body. We investigate the right-hand side of C^k_{k+1}.

$$\begin{aligned}
& a_{ij} \vee i \rightarrow^{<k+1} j \\
= \quad & \{(4)\} \\
& a_{ij} \vee i \rightarrow^{<k} j \vee ((i \rightarrow^{<k} k \vee a_{ik}) \wedge (k \rightarrow^{<k} j \vee a_{kj})) \\
= \quad & \{C\} \\
& b_{ij} \vee (b_{ik} \wedge b_{kj}).
\end{aligned}$$

It follows that simultaneous execution of the up-to-N^2 assignments to b_{ij} maintain C. The assignments may, however, also be executed in any order since the values b_{ik} and b_{kj} are not changed: for example, b_{ik} is assigned the value *true* if $b_{ik} \wedge b_{kk}$ holds, which implies that b_{ik} is already *true*.

3. From reduced to closed graph and vice versa

We now consider the operational aspects of Warshall's algorithm. Iteration k of the loop of Warshall's algorithm adds edges between nodes that are connected by a path all of whose intermediate nodes have a number less than k. A corresponding reduction algorithm will undo the actions of the closure algorithm: edges will be deleted only between nodes that are connected by a path whose intermediate nodes are at least k. If we succeed in finding the exact "undoing", we have two algorithms with the same invariant. However, this is too strong to hope for, because $i \rightarrow^{<k} j$ and $i \rightarrow^{\geq k} j$ may hold simultaneously. In the closure algorithm, an edge (i, j) is added as soon as $i \rightarrow^{<k} j$ is found to be true, even if $i \rightarrow^{\geq k} j$ is also true. Similarly, in the proposed reduction algorithm, an edge (i, j) is deleted as soon as $i \rightarrow^{\geq k} j$ is found to be true, even if $i \rightarrow^{<k} j$ is also true. Thus, we obtain two distinct invariants, C for the closure and R for the reduction algorithm.

$$C: \quad b_{ij} \equiv a_{ij} \vee i \rightarrow^{<k} j.$$

$$R: \quad b_{ij} \equiv a_{ij} \wedge \neg i \rightarrow^{\geq k} j.$$

Inspired by the discussion leading to R, we propose the following algorithm for the transitive reduction of an initially closed graph.

```
TR:  $k := N$; $\{R\}$
     do $k \neq 0 \rightarrow k := k - 1$; $\{R^k_{k+1}\}$
                   forall($i, j$: $b_{ik} \wedge b_{kj}$: $b_{ij} := false$) $\{R\}$
     od.
```

We need to check the invariance of R in algorithm TR. The initialization establishes R since R^k_N holds on account of (1). $R \wedge k = 0$ implies, on account of (3), that b is the transitive reduction of a. We check the invariance of R over the loop body of

TR. We do so by manipulating the right-hand side of R while using the fact that a is closed.

$$\begin{aligned}
&a_{ij} \wedge \neg i \rightarrow^{\geqslant k} j\\
=\quad &\{(5)\}\\
&a_{ij} \wedge \neg(i \rightarrow^{\geqslant k+1} j \vee ((i \rightarrow^{\geqslant k+1} k \vee a_{ik}) \wedge (k \rightarrow^{\geqslant k+1} j \vee a_{kj})))\\
=\quad &\{\text{since } a \text{ is closed } i \rightarrow^{\geqslant k+1} k \text{ implies } a_{ik}\}\\
&a_{ij} \wedge \neg(i \rightarrow^{\geqslant k+1} j \vee (a_{ik} \wedge a_{kj}))\\
=\quad &\{\text{calculus}\}\\
&a_{ij} \wedge \neg i \rightarrow^{\geqslant k+1} j \wedge \neg(a_{ik} \wedge a_{kj})\\
=\quad &\{\text{rewrite } a_{ik} \wedge a_{kj} \text{ using } x = (x \wedge y) \vee (x \wedge \neg y);\\
&\ \text{on account of } R^{k}_{k+1} \text{ we aim at term of the form } a_{ij} \wedge \neg i \rightarrow^{\geqslant k+1} j\}\\
&a_{ij} \wedge \neg i \rightarrow^{\geqslant k+1} j\\
&\wedge \neg((a_{ik} \wedge \neg i \rightarrow^{\geqslant k+1} k \wedge a_{kj} \wedge \neg k \rightarrow^{\geqslant k+1} j)\\
&\qquad \vee (a_{ik} \wedge a_{kj} \wedge (i \rightarrow^{\geqslant k+1} k \vee k \rightarrow^{\geqslant k+1} j)))\\
=\quad &\{\text{since } a \text{ is closed } i \rightarrow^{\geqslant k+1} j \text{ is implied by}\\
&\ a_{ik} \wedge a_{kj} \wedge (i \rightarrow^{\geqslant k+1} k \vee k \rightarrow^{\geqslant k+1} j)\}\\
&a_{ij} \wedge \neg i \rightarrow^{\geqslant k+1} j \wedge \neg(a_{ik} \wedge \neg i \rightarrow^{\geqslant k+1} k \wedge a_{kj} \wedge \neg k \rightarrow^{\geqslant k+1} j)\\
=\quad &\{R^{k}_{k+1}\}\\
&b_{ij} \wedge \neg(b_{ik} \wedge b_{kj}).
\end{aligned}$$

It follows that simultaneous execution of the up-to-N^2 assignments to b_{ij} maintains R. Again, the assignments may also be executed in any order: whether an assignment is executed depends only on the values b_{ik} and b_{kj}, and these are not changed since the acyclic nature of the graph enforces $b_{kk} = \mathit{false}$.

4. Transitive reduction of any acyclic graph

Warshall's algorithm, *TC*, computes the transitive closure of an arbitrary graph; it has C as an invariant. The dual of algorithm *TC* is algorithm *TR* which reduces an initially closed, acyclic graph; it has invariant R. The optimist might hope that a weaker invariant of *TR* exists that can be used to prove that *TR* reduces an arbitrary acyclic graph. The example below shows that this is not the case, for execution of *TR* with this graph does not eliminate the redundant edge $(0, 3)$.

Fortunately there is a way out: algorithm *TC*; *TR* computes the transitive close of an arbitrary acyclic graph and then reduces the closed graph—which has the same transitive reduction as the original graph. The time complexity of the in-situ

$$\begin{tikzcd}
0 \arrow[r] \arrow[rrr, bend left] & 1 \arrow[r] & 2 \arrow[r] & 3
\end{tikzcd}$$

algorithm *TC*; *TR* is $O(N^3)$. In [1] it is shown that transitive closure and transitive reduction have equal time complexity.

References

[1] A.V. Aho, M.R. Garey and J.D. Ullman, The transitive reduction of a directed graph, *SIAM J. Comput.* **1**(2) (1972) 131-137.
[2] E.W. Dijkstra, *Selected Writings on Computing: A Personal Perspective* (Springer, New York, 1982) 351-354.
[3] D. Gries, *The Science of Programming* (Springer, New York, 1981) 265-274.
[4] S. Warshall, A theorem on Boolean matrices, *J. ACM* **9** (1962) 11-12.