A TRANSITIVE CLOSURE ALGORITHM

PAUL PURDOM JR.

Abstract.

An algorithm is given for computing the transitive closure of a directed graph in a time no greater than \( a_1 N_1 n + a_2 n^3 \) for large \( n \) where \( a_1 \) and \( a_2 \) are constants depending on the computer used to execute the algorithm, \( n \) is the number of nodes in the graph and \( N_1 \) is the number of arcs (not counting those arcs which are part of a cycle and not counting those arcs which can be removed without changing the transitive closure). For graphs where each arc is selected at random with probability \( p \), the average time to compute the transitive closure is no greater than \( \min\{a_1 p n^3 + a_2 n^2, \frac{1}{2} a_1 n^3 p^{-2} + a_2 n^2\} \) for large \( n \). The algorithm will compute the transitive closure of an undirected graph in a time no greater than \( a_2 n^2 \) for large \( n \). The method uses about \( n^2 + n \) bits and \( 5n \) words of storage (where each word can hold \( n + 2 \) values).

1. Introduction.

The transitive closure \( T \) of a directed graph \( G \) is a directed graph such that there is an arc in \( T \) going from node \( i \) to node \( j \) if and only if there is a path in \( G \) going from node \( i \) to node \( j \). The transitive closure of a node \( i \) is the set of nodes on paths starting from node \( i \). For example the transitive closure of node \( k \) in Figure 1 is the set of nodes \( \{g, l, j, k, h\} \).

It is often useful to specify a graph \( G \) with nodes \( 1, 2, \ldots, n \) by an \( n \times n \) incidence matrix \( M \) with elements \( m_{ij} \) defined by

\[
\begin{cases} 
\text{true} & \text{if } G \text{ has an arc from node } i \text{ to node } j, \\
\text{false} & \text{otherwise}.
\end{cases}
\]

It has long been known that the incidence matrix \( M \) of a graph can be used to compute the incidence matrix \( T \) of the transitive closure of the graph with the equation

\[
T = \sum_{1 \leq i \leq n} M^i
\]

where \( M \) and \( T \) are boolean matrices. It takes about \( n^4 \) operations to compute \( T \) this way. Warshall [1] has a method to compute the transitive closure which takes between \( n^3 \) and \( n^2 \) operations. His algorithm to con-
vert the incidence matrix $M$ of a graph into the incidence matrix of the transitive closure of $G$ is equivalent to the following:

W1. For $1 \leq k \leq n$ do the remaining steps.
W2. For each $i$ such that $1 \leq i \leq n$ and $M[i,k]$ is true do step W3.
W3. For $1 \leq j \leq n$ set $M[i,j] \leftarrow M[i,j] \lor M[k,j]$.

A method for computing transitive closure using lists is given by Thorelli [2]. His method, however, will in many cases take about $n^4$ operations if the transitive closure has about $n^2$ arcs (with minor changes his algorithm can be done in $n^3$ steps.)

There are many algorithms which require the computing of transitive closure. The reader is referred to Weber and Wirth [3] and Lynch [4] for some practical problems in the field of syntactic analysis where it is necessary to find the transitive closure of a graph with one or two hundred nodes.

The algorithm in this paper is designed for computing the transitive closure of a graph with a moderately large number of nodes (the graph should, however, fit in the computer storage; this requires about $n^2$ bits of memory for a graph with $n$ nodes). In section 2 it is shown that the maximum running time for the algorithm is proportional to $n^2$, but there are cases (such as sparse graphs where the number of arcs is no more than a constant times the number of nodes, random graphs where each possible arc is selected with fixed probability, and undirected graphs) when the running time increases only as $n^2$. In section 4 the method is compared with Warshall's algorithm and cases are given where the method in this paper will be faster for large graphs.

The concepts of path equivalence and partial ordering are particularly important to understanding the algorithm. Two distinct nodes $x$ and $y$ are path equivalent if there is both a path from $x$ to $y$ and a path from $y$ to $x$. Also each node is path equivalent to itself. For any pair of nodes $x$ and $y$, there is a path from any node path equivalent to $x$ to any node path equivalent to $y$ if and only if there is a path from $x$ to $y$. In the following the term equivalent always refers to path equivalence. A directed graph is a partial ordering if and only if the graph has no cycles. Thus if no pair of distinct nodes in the graph are equivalent, the graph is a partial ordering. If the graph is a partial ordering it is possible to find a consistent linear ordering of the nodes [5]. This means that the nodes $1, 2, \ldots, n$ can be renumbered as $\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_n$ in such a way that if there is an arc from $x$ to $y$ then $\hat{i}_x$ precedes $\hat{i}_y$.

The algorithm consists of four parts. The first part finds all the classes of nodes which are equivalent and replaces each class by a single node.