MIT/LCS/TM-76

A NOTE ON THE AVERAGE TIME TO COMPUTE TRANSITIVE CLOSURES

P. A. BLONIARZ
M. J. FISCHER
A. R. MEYER

SEPTEMBER 1976

MIT/LCS/TM-76

A NOTE ON THE AVERAGE TIME TO COMPUTE TRANSITIVE CLOSURES

P.A. Bloniarz
M.J. Fischer
A.R. Meyer

September 1976

Massachusetts Institute of Technology
Laboratory for Computer Science
(formerly Project MAC)
Cambridge, Massachusetts
02139

P.A. Bloniarz*
M.J. Fischer**
A.R. Meyer*

Revised July 1976

A NOTE ON THE AVERAGE TIME TO COMPUTE TRANSITIVE CLOSURES

An algorithm which finds shortest paths between all pairs of nodes in an n node weighted, directed graph using an average of $O(n^2 \cdot (\log n)^2)$ basic steps has been described by Spira [10]. A special case of the shortest path problem is the transitive closure problem for Boolean matrices.

In this note we point out a simple restriction of Spira's algorithm which allows the computation of the transitive closure of a Boolean matrix in average time $O(n^2 \cdot \log n)$. (This time bound for the average case was obtained independently by D. Angluin [2] using a different algorithm.) In the course of verifying the restricted algorithm, we isolate a lacuna in Spira's original procedure - namely Spira's algorithm does not specify from which node to search when several nodes are equidistant from a source. We describe a counter-example based on this lacuna showing that Spira's algorithm may run in $\Omega(n^3)$† average steps on certain ensembles of graphs when "tie-breaking" in the case of equidistant nodes is decided by a plausible but improper convention. With a proper tie-breaking procedure, Spira's algorithm indeed can be shown to run in $O(n^2 \cdot (\log n)^2)$ steps on the average for a somewhat larger class of probability measures on graphs than he originally claimed, although we do not prove this latter fact in the present note.

2. DEFINITIONS AND PRELIMINARIES

We assume the reader is familiar with the standard definitions of Boolean matrices and directed graphs as contained in [1]. In particular, a directed graph G is a pair (V,E), where

---

*Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 545 Main Street, Cambridge, Massachusetts 02139, U.S.A.

**Department of Computer Science, University of Washington, Seattle, Washington 98195, U.S.A.

† A function f is $\Omega(g(n))$ iff there is a constant $c > 0$ such that $f(n) \geq c \cdot g(n)$ for all sufficiently large n.

This report was prepared with the support of the National Science Foundation under research grant no. GJ-43634X, contract no. DCR74-12997-A01.

$V = \{1,2,\ldots,n\}$ is a set of vertices, and $E \subseteq V \times V$ is a set of edges. The adjacency set of a node $i \in V$ is the set $A(i) = \{j \in V \mid (i,j) \in E\}$. A node j is said to be reachable from i if there is a path from i to j. One representation of a graph (V,E) is the $n \times n$ Boolean incidence matrix M_G, defined by $M_G(i,j) = 1$ if and only if edge (i,j) is in E; otherwise, $M_G(i,j) = 0$. The transitive closure of a Boolean matrix M is defined by $M^* = I \vee M \vee M^2 \vee M^3 \vee \ldots$, where I is the identity matrix. Note that $M^*(i,j) = 1$ if and only if $i = j$ or there is a path from node i to node j in the graph represented by M.

A weighted directed graph is a directed graph whose edges have weights which are non-negative real numbers or positive infinity (∞). By possibly adding arcs with weight ∞, we may assume all n^2 edges are present in the graph. The cost (or length) of a path is the sum of the weights of its edges, and the minimum cost matrix C is the $n \times n$ matrix such that $C(i,j)$ is the minimum of the costs of all paths in the graph from i to j. We may identify an unweighted directed graph $G = (V,E)$ with the weighted graph with nodes V such that all edges in E are assigned weight 0 and all edges in $(V \times V) - E$ are assigned weight ∞. Thus if C is the minimum cost matrix of the weighted version of G, then $C(i,j) = 0$ if and only if $M_G^*(i,j) = 1$. Hence any algorithm which computes the shortest distance matrix of a weighted graph can be used directly to compute the transitive closure of a Boolean matrix.

The model of computing machine used is that of a random access machine in which address determinations have unit cost [1].

3. A TRANSITIVE CLOSURE ALGORITHM

We first informally describe our transitive closure algorithm and verify its correctness. The algorithm constructs the transitive closure of a matrix M one row at a time; i.e., in graph terminology, it finds the set of nodes reachable from a given node. Observe that if there is a path from a particular node (say node i) to node j, then trivially there is also a path from node i to all vertices in the adjacency set A(j). Using this observation, we may produce the set of nodes reachable from node i by initializing a set variable L to the set A(i); then we repetitively add to L the adjacency sets A(j) of all nodes j which are in L. We halt when either L contains all nodes, or the adjacency sets of all nodes in L have been placed in L. It is easy to show by induction on the number of arcs in a path between i and any node reachable from i that when the algorithm is halted, L will equal the set of all nodes reachable from node i.

We thus obtain the following algorithm $\underset{\sim}{R}(i)$ which computes the set of nodes reachable from node i, given the adjacency sets $A(j)$ $(j = 1,2,\ldots,n)$.

For efficiency, the set of reachable nodes is represented as a bit array $L(j)$, where, at the conclusion of the algorithm, $L(j) = 1$ if and only if node j is reachable from node i. We say node j is labelled at any stage in the algorithm if $L(j) = 1$ at that stage. A push-down stack is used to contain those labelled nodes whose adjacency sets have not yet been explicitly included in L. There are assumed to be two basic stack operations: push (i), which places node i on the top of the stack; and pop, which removes and returns the top node on the stack, if there is one, or EMPTY, if the stack is empty. We assume the sets $A(j)$ for $j = 1,2,\ldots,n$ are represented as lists. The variable CARD represents the number of labelled nodes, and is used to test whether all nodes have been labelled.

Algorithm $\underset{\sim}{R}(i)$, which computes the ith row of the matrix M_G^*:

```
(1) L(i) ← 1
    L(j) ← 0 for j ≠ i        {Initially, only node i is labelled.}
    CARD ← 1
    j ← i
(2) While j ≠ EMPTY and CARD ≠ n do:
      (2a) For each node k ε A(j), if L(k) = 0,
              then do:                          {If node k on the
                        L(k) ← 1                 adjacency list A(j)
                        CARD ← CARD + 1          is not yet
                        push(k)                  labelled, then
              end                                label k.}
      (2b) j ← pop
    end
(3) M*(i,j) ← L(j) for j = 1,2,...,n.
```

Algorithm $\underset{\sim}{TC}$, which computes the transitive closure of the matrix M:

(1) Extract the adjacency lists $A(j)$ $(j = 1,2,\ldots,n)$ of the graph corresponding to M.
(2) For $i = 1,2,\ldots,n$, do $\underset{\sim}{R}(i)$.

4. ANALYSIS OF THE ALGORITHM

The time required by algorithm $\underset{\sim}{TC}$ of course depends on the matrix on which it is computing. A time of $\Omega(n^3)$ operations is taken on the matrix corresponding to an n node graph consisting of an n-1 node clique and a single isolated vertex. The average time the algorithm takes over wide classes of graphs is $O(n^2 \cdot \log n)$, as we now show.

In the following, M is an $n \times n$ Boolean matrix, and $T(M)[T_i(M)]$ is the time taken by algorithm $\underset{\sim}{TC}$[algorithm $\underset{\sim}{R}(i)$]

on matrix M. If $\mathfrak{M}$ is a probability distribution on matrices, then $\overline{T}_{\mathfrak{M}}[\overline{T}_i(\mathfrak{M})]$ is the average value of $T(M)[T_i(M)]$ as M ranges over $\mathfrak{M}$. We will omit mention of $\mathfrak{M}$ when there is no ambiguity.

Two $n \times n$ incidence matrices M, M' are called <u>out-degree matching</u> iff $\sum_{j=1}^{n} M(i,j) = \sum_{j=1}^{n} M'(i,j)$ for each $i = 1,\ldots,n$. That is, if A and A' are the adjacency lists of the graphs corresponding to M and M' respectively, then $|A(i)| = |A'(i)|$ for $i = 1,\ldots,n$*.

<u>Theorem</u>. Suppose $\mathfrak{M}$ is any probability distribution of $n \times n$ Boolean matrices such that for any out-degree matching matrices M and M',

Probability (M) = Probability (M') in $\mathfrak{M}$.

Then $\overline{T}_{\mathfrak{M}} = O(n^2 + \min(n\cdot\overline{E}, n^2\cdot\log n))$, where $\overline{E}$ is the average number of ones in the matrices of $\mathfrak{M}$.

<u>Proof</u>. We first make a few observations. For any matrix M,

$$T(M) = O(n^2) + \sum_{i=1}^{n} T_i(M),$$

where the first term accounts for the extraction of the adjacency lists and all bookkeeping tasks. Hence,

$$\overline{T} = O(n^2) + \sum_{i=1}^{n} \overline{T}_i,$$

and it thus suffices to show $\overline{T}_i \le k\cdot\min(n\cdot\log n, \overline{E})$ for some constant k, and all i, $1 \le i \le n$.

For the execution of algorithm $\underset{\sim}{R}(i)$ on matrix M, let $N_i(M)$ be the number of times the test "if L(k) = 0" in step (2a) of the algorithm is performed; let $\overline{N}_i(\mathfrak{M})$ be the corresponding average value of $N_i(M)$ as M ranges over $\mathfrak{M}$. Since $T_i(M)$ is clearly proportional to $N_i(M)$, it is sufficient to show that $\overline{N}_i \le k'\cdot\min(n\cdot\log n, \overline{E})$ for some constant k' independent of i and n.

Note that for any matrix M, $\sum_{j=1}^{n} |A(j)| = E =$ the number of one's in M. The test "if L(k) = 0" in step (2a) of $\underset{\sim}{R}(i)$ is made at most once for each entry on any adjacency list, so we have $N_i(M) \le$ the number of 1's in M. Thus $\overline{N}_i \le \overline{E}$.

*$|A|$ denotes the cardinality of a set A.

To show $\overline{N}_i \le k' \cdot n \log n$, define for each $\alpha : \{1,2,\ldots,n\} \to \{0,1,2,\ldots,n\}$, the set $\mathfrak{M}_\alpha = \{M \in \mathfrak{M} \mid \sum_{j=1}^{n} M(\ell,j) = \alpha(\ell)$ for $\ell = 1,\ldots,n\}$. For any fixed α any two matrices in $\mathfrak{M}_\alpha$ are out-degree matching and hence equiprobable in $\mathfrak{M}$ by hypothesis. Thus, it suffices to prove that there is a fixed k' such that $\overline{N}_i(\mathfrak{M}_\alpha) \le k' \cdot n \cdot \log n$ for the probability distribution in which all matrices in $\mathfrak{M}_\alpha$ are equiprobable and all other matrices have probability zero. (Abusing notation slightly, we also refer to this distribution as $\mathfrak{M}_\alpha$.)

Consider now another mechanism for building up the elements of M_α dynamically. For each $j = 1,2,\ldots,n$, choose, independently and at random with equal probabilities, a set $A_0(j)$ of $\alpha(j)$ distinct elements of V. If M_0 is the matrix corresponding to the adjacency lists $A_0(j)$, then $\overline{N}_i$ is equal to the expected value of $N_i(M_0)$ (since each element of $\mathfrak{M}_\alpha$ has an equal probability of being M_0).

We now modify algorithm $\underset{\sim}{R}(i)$ to be a probabilistic procedure $\underset{\sim}{P}(i)$ which constructs M_0 and simultaneously runs $\underset{\sim}{R}(i)$ on it. L and A are set variables corresponding to the vector L and the lists A(j) in algorithm $\underset{\sim}{R}(i)$.

```
  Procedure P(i):
(1) L ← {i}
    j ← i
(2) While j ≠ EMPTY and |L| ≠ n do:
        A ← ∅
        While |A| < α(j) do:                    ⎫
              Choose, at random and with       ⎪
              equal probabilities, a node      ⎬ Build up A_0(j)
              k ∈ V.                           ⎪
              If k ∉ A, then A ← A ∪ {k}       ⎭
        end
   (2a) For each node k ∈ A, if k ∉ L then
              L ← L ∪ {k} and push(k).
   (2b) j ← pop
    end
```

By the above remarks, $\overline{N}_i$ is the expected number of times the test "if $k \notin L$" is performed at step (2a) of procedure $\underset{\sim}{P}(i)$.

Notice that in the execution of $\underset{\sim}{P}(i)$, once a node becomes a member of $L \cup A$ it remains so at subsequent steps of the procedure. Moreover, if $|L \cup A| = n$ just before an execution of step (2a), then at most $|A| \leq n$ further executions of the test "if $k \notin L$" will be executed at step (2a) before $\underset{\sim}{P}(i)$ terminates.

Let $\overline{N}_{i'}$ be the expected number of times nodes are chosen at random in step (2) throughout the execution of $\underset{\sim}{P}(i)$ <u>until</u> $\underline{|L \cup A| = n}$. Since at any point in the execution of $\underset{\sim}{P}(i)$ the number of times the test "if $k \notin L$" has been performed is trivially at most equal to the number of times nodes have been chosen at random at step (2), we conclude that $\overline{N}_i \leq \overline{N}_{i'} + n$.

But $\overline{N}_{i'}$ is nothing but the expected number of times nodes from the set $\{1,\ldots,n\}$ must be chosen randomly (with repetitions) and put in the set $L \cup A$ until $L \cup A = \{1,\ldots,n\}$. The expected number of random selections with repetition from the set $\{1,\ldots,n\}$ until all elements have been selected is well known to be asymptotic to $n\log_e n$ ([4], p. 225), so $\overline{N}_{i'} = O(n\log n)$, which completes the argument. □

Spira originally stated his result for graphs in which each edge weight was an independent non-negative real-valued random variable with the same distribution for each edge. We note that additional probability measures on unweighted graphs to which the hypotheses of the above theorem apply include:

(1) Boolean matrices in which entry $M(i,j)$ is an independent random variable which has probability p_i of being equal to 1 (the probability may vary for different rows of the matrix and may depend on n),

and

(2) For every ℓ, $1 \leq \ell \leq n^2$, the set $\mathfrak{M}_\ell$ of all Boolean matrices which contain exactly ℓ ones, with equal probability.

5. A LACUNA IN SPIRA'S ALGORITHM

Our analysis of the algorithms $\underset{\sim}{R}(i)$ and $\underset{\sim}{TC}$ for computing transitive closures was prompted by Spira's analysis of his more general algorithm for computing minimum cost matrices. Indeed our algorithms may be obtained by restricting Spira's algorithm to the special case when all edge weights are zero or ∞, except that in this case Spira's algorithm leaves unspecified the order in which adjacency lists of labelled nodes are to be searched. (Spira suggests a search order based on minimum weights, but these will all be zero in the zero-∞ case.)

It is natural to assume that any specification of the missing tie-breaking rule for choice among equal weight paths would lead to an efficient algorithm. This is not the case. We next describe transitive closure algorithm BADTC which is a special case of Spira's algorithm in which a tie-breaking rule is determined by using a first-in first-out queue (instead of the last-in first-out pushdown stack of algorithm TC). However, the expected running time of BADTC is $\Omega(n^3)$.

We remark that the tie-breaking lacuna may be repaired easily in Spira's general algorithm for weighted graphs along the lines of the pushdown mechanism we used in TC. Spira's claim of an average running time of $O(n^2\log^2 n)$ may then be verified over a somewhat larger class of probability distributions on weighted graphs than he originally claimed. We postpone further discussion of Spira's general algorithm to a later paper.

The algorithm BADTC is obtained by performing BADR(i) for i = 1,...,n where BADR(i) computes the set of nodes reachable from node i. In BADR(i), a bit array L indicating "labelled" nodes reachable from node i is used as in algorithm R(i). Instead of the pushdown store of nodes whose adjacency lists must be searched as in R(i), the algorithm BADR(i) maintains a first-in, first-out queue of labelled nodes whose adjacency lists have not been completely searched. The queue operations are tail(j) which inserts node j at the tail end of the queue, and head which removes and takes the value of the first element of the queue, or returns the value EMPTY if the queue is empty. Finally, we assume the adjacency sets A(j) are stored as lists and the operation next(A(j)) removes and takes the value of the first element of the list, or returns the value EMPTY if the list is empty.

Algorithm BADR(i), which computes the ith row of the matrix M_G^*:

```
(1) L(i) ← 1                                  }
    L(j) ← 0 for j ≠ i                        }{Initially, only node i
    CARD ← 1                                  }  is labelled.}
    j ← i
                                               {An element from A(j) will be
(2) While j ≠ EMPTY and CARD ≠ n do:            labelled next.}
        (2a) k ← next(A(j))
        (2b) If k ≠ EMPTY, then do:
                 tail (j)                      }{Return j to the queue if
                                               }  A(j) was still not empty.}
                 If L(k) = 0, then do:         {If the next-reachable-
                     L(k) ← 1                   node k is not already
                     CARD ← CARD + 1            labelled, then label
                     tail(k)                    it and add it to the
                 end                            queue.}
             end
        (2c) j ← head
    end
(3) M*(i,j) ← L(j) for j = 1,...,n.
```

Theorem: Let $\mathfrak{M}$ be the uniform probability distribution on $n \times n$ Boolean matrices. For any $n \times n$ matrix M, assume that the adjacency lists A(j) for $j = 1,\ldots,n$ are sorted by increasing node index. Then the expected running time on $\mathfrak{M}$ of algorithm BADR(i) is at least $\epsilon \cdot n^2$ for $i = 1,\ldots,n$, and the expected running time of algorithm BADTC is at least $\epsilon \cdot n^3$, for some constant $\epsilon > 0$.

Proof: A graph G corresponding to a matrix M in $\mathfrak{M}$ has on the average at least n/3 nodes on each adjacency list A(i) for $i = 1,\ldots,n$ and is strongly connected, as is easily verified (cf. the remark in section 6 below). In particular, application of algorithm BADR(i) to such a graph G will cause node n to be labelled eventually for $i = 1,\ldots,n$ since node n will be reachable from node i.

Consider an application of algorithm BADR(i) to G for $i \neq n$. Let j_0 be the value of the variable j at that execution of step (2a) at which the next-reachable-node-variable k is first set to n. Since $|A(j_0)| \geq n/3$ and since n is the last element of $A(j_0)$ (which by hypothesis is sorted in order of increasing node index), j_0 must have been labelled, placed on the queue, and subsequently reappeared at the head of the queue at step (2c) at least n/3-1 times.

Following each appearance of j_0 at the head of the queue at step (2c), a new element of $A(j_0)$ is selected at step (2a) and added to the queue at step (2b) unless that element had previously been placed on the queue. The queue discipline implies that the ℓ^{th} element of $A(j_0)$ will appear at the head of the queue at step (2c) in between each successive appearance of j_0 at the head of the queue, starting from the ℓ^{th} appearance of j_0 and continuing until either n is labelled (after at least $n/3 - \ell - 1$ further appearances of j_0), or until the adjacency list of this ℓ^{th} element is exhausted. Thus, the ℓ^{th} element of $A(j_0)$ appears at the head of the queue at step (2c) at least $\min(n/3 - \ell - 1, n/3) = n/3 - \ell - 1$ times. Since there are at least $n/3-1$ elements of $A(j_0)$ before n, at least $\sum_{\ell=1}^{n/3-1} (n/3 - \ell - 1) = \Omega(n^2)$ executions of step (2c) must occur. $\square$

6. REMARKS

The analysis of the algorithm which was presented in section 4 has implications for the properties of "random" graphs. For example, if $\mathfrak{M}$ is the uniform probability distribution on n x n Boolean matrices [graphs], we know that $N_i(\mathfrak{M}) = n\log_e n + o(n\log n)$. Thus, for an "average" graph G, algorithm $\underset{\sim}{R}(i)$ halts after about $n\log_e n$ edges from labelled nodes have been examined in step (2a). This occurs either because (1) node i is discovered to be connected to all nother nodes, or (2) all adjacency sets from all labelled nodes have been exhausted. However, case (2) does not occur on the average because more than $\Omega(n\log n)$ must have been examined in this case: G has on the average at least n/3 nodes on each adjacency set A(j) for j = 1,2,...,n. Hence there are at least n/3 labelled nodes immediately adjacent to node i, and each has an adjacency set of at least n/3 nodes, giving a total of at least $n^2/9$ edges for labelled nodes to be examined in verifying case (2). Thus on the average the algorithm halts because of case (1). In fact, closer analysis shows that the probability that node i is connected to all other nodes is $1 - o(\frac{1}{n})$, and thus the probability that a random graph in $\mathfrak{M}$ is strongly connected is $1 - o(1)$. Results similar to these, along with a more precise description of the behavior of "random" undirected graphs, have been described previously by P. Erdos and others [3], and the interested reader is referred to this work.

Finally, we note that M. Fredman [6] has shown that Spira's algorithm may be modified to find the shortest distance matrix in $O(n^2 \cdot \log_2 n)$ average number of edge weight <u>comparisons</u>, but at a considerable increase in total running time.

REFERENCES

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

[2] Angluin, D. Private communication.

[3] Erdos, P. & J. Spencer (1974) Probabilistic Methods in Combinatorics. New York: Academic Press.

[4] Feller, W. (1968) An Introduction to Probability Theory and its Applications. New York: Wiley, Vol. 1, 3rd edition.

[5] Fischer, M.J. & A.R. Meyer (1971) "Boolean Matrix Multiplication and Transitive Closure." Twelfth Annual Symp. on Switching and Automata Theory, 129-31.

[6] Fredman, M.L. (1975) "On the Decision Tree Complexity of the Shortest Path Problems." Sixteenth Annual Symp. on Foundations of Computer Science, 98-9.

[7] Furman, M.E. (1970) "Applications of a Method of Fast Multiplication of Matrices in the Problem of Finding the Transitive Closure of a Graph." Dokl. Akad. Nauk SSSR 194, 3. (Soviet Math. Dokl. 11, 5 (1970), 1252.)

[8] Munro, I. (1971) "Efficient Determination of the Transitive Closure of a Directed Graph." Information Processing Letters, 1, 2.

[9] O'Neil, P., & E. O'Neil (1973) A Fast Expected Time Algorithm for Boolean Matrix Multiplication and Transitive Closure, in Courant Computer Science Symp. 9: Combinatorial Algorithms (ed. R. Rustin), 59-68. New York: Algorithmics Press.

[10] Spira, P.M. (1973) "A New Algorithm for Finding Shortest Paths in a Graph of Positive Arcs in Average Time $O(n^2 \log^2 n)$." SIAM J. Computing, 2, 1, 28-32.

[11] Strassen, V. (1969) "Gaussian Elimination is not Optimal." Numer. Math. 13, 354-6.