Solving Path Problems on Directed Graphs

Robert Endre Tarjan
Computer Science Department
Stanford University
Stanford, California 94305

October 1975

Abstract

This paper considers path problems on directed graphs which are solvable by a method similar to Gaussian elimination. The paper gives an axiom system for such problems which is a weakening of Salomaa's axioms for a regular algebra. The paper presents a general solution method which requires $O(n^3)$ time for dense graphs with n vertices and considerably less time for sparse graphs.

The paper also presents a decomposition method which solves a path problem by breaking it into subproblems, solving each sub-problem by elimination, and combining the solutions. This method is a generalization of the "reducibility" notion of data flow analysis, and is a kind of single-element "tearing". Efficiently implemented, the method requires $O(m\ \alpha(m,n))$ time plus time to solve the subproblems, for problem graphs with n vertices and m edges. Here $\alpha(m,n)$ is a very slowly growing function which is a functional inverse of Ackermann's function.

The paper considers variants of the axiom system for which the solution methods still work, and presents several applications, including solving simultaneous linear equations and analyzing control flow in computer programs.

This research was supported by National Science Foundation grant DCR72-03752. Reproduction in whole or in part is permitted for any purpose of the United States Government.

Keywords : Ackermann's function, algorithm, dominators, flow graph reducibility, Gaussian elimination, global flow anelysis, graph, shortest path, simultaneous linear equations, transitive closure.

1. Introduction.

Consider a system of linear equations $Ax = c$, where A is an n by n, real-valued, non-singular matrix, x is an n by one vector of variables, and c is an n by one vector of constants. Mathematicians have developed many methods for solving such systems [15,44], including Gaussian elimination and its variants and a host of iterative methods.

Some of these linear algebra techniques apply in other settings. Klenne [25] and others [6,36,37] have described the use of elimination methods to compute regular expressions for finite automata. Floyd [13] and others [9,20] have used similar methods to find certain kinds of optimum paths in graphs. Not all of these researchers have realized the connection of their ideas with Gaussian elimination.

Independently, many canputer scientists have developed methods for collecting information about the flow of control in a computer program [4,5,10,14,16,17,21,22,23,24,29,31,43,45]. Some of these methods resemble iterative methods for solving systems of linear equations; others resemble Gaussian elimination. Flow graphs of computer programs often have a special property, called reducibility. For such programs, especially-efficient information collection algorithms exist [16,23,37,43].

In this paper we develop an elimination method for solving such path problems. We use an axiomatic setting which covers most of the problem domains described above. The axiom system is a weakening of Salomaa's axioms for a regular algebra [36,37], with right distributivity replaced by a monotonicity axiom suggested by Graham and Wegman [16] and by Wegbriet [45]. We discuss variants of the axiom system for which the method is also valid.

For convenience in presenting some of the results, we use a graph-theoretic framework in place of a matrix-theoretic one. For dense graphs with n vertices, the elimination method requires $O(n^3)$ time; for sparse graphs, the running time depends in a complicated way upon the sparsity.

We also describe two methods for solving a path problem by breaking it into several path problems on smaller graphs. The first, well-known by numerical analysts, uses the strongly connected components of the problem graph. For a graph with n vertices and m edges, the method requires $O(n+m)$ time plus the time to solve the subproblems. The second method, which generalizes the reducibility notion of global flow analysis, and which is a type of single-element "tearing", uses the dominators of the problem graph. This method requires $O(m\ \alpha(m,n))$ time plus the time to solve the subproblems, where $\alpha(m,n)$ is a very slowly growing function related to a functional inverse of Ackermann's function. For reducible flow graphs, the total running time is $O(m\ \alpha(m,n))$, better than the $O(m \log n)$ running time of the best previous algorithms [14,16,23,43].

The paper contains nine sections. Section 2 gives the necessary definitions from graph theory. Section 3 gives the axiom system for path problems and presents the elimination method. Section 4 discusses the effect of reversing the edges of the problem graph. Section 5 gives the strong components decomposition method. Section 6 presents the dominators decomposition method. Section 7 discusses changes to the axiom system. Section 8 gives examples of path problems. Section 9 contains further remarks and conclusions.

2. Directed Graphs.

A directed graph $G = (V,E)$ is a finite set V of $n = |V|$ elements called vertices and a finite set E of $m = |E|$ elements called edges. Associated with each edge e is a vertex $h(e)$ called the head of e and a vertex $t(e)$ called the tail of e. Edge e leaves $t(e)$ and enters $h(e)$. This definition allows loops (edges e with $h(e) = t(e)$) and multiple edges (edges e_1, e_2 with $h(e_1) = h(e_2)$, $t(e_1) = t(e_2)$).

A path p of length k from v to w is a sequence of edges $p = e_1, e_2, \ldots, e_k$ such that $h(e_i) = t(e_{i+1})$ for $1 \leq i \leq k-1$, $t(e_1) = v$, and $h(e_k) = w$. We extend h and t by defining $h(p) = h(e_k)$, $t(p) = t(e_1)$. The path p contains edges $e_1, e_2, \ldots, e_k$ and vertices $t(e_1), h(e_1), h(e_2), \ldots, h(e_k)$, and avoids all other edges and vertices. By convention there is an empty path (containing no edges) from every vertex to itself. A cycle is a path p, containing at least one edge, such that $h(p) = t(p)$.

A graph $G' = (V',E')$ is a subgraph of a graph $G = (V,E)$ if $V' \subseteq V$ and $E' \subset E$. If $E' = E(V') = \{e \in E \mid h(e), t(e) \in V'\}$, then G' is the subgraph of G induced by the set of vertices V'. Similarly, if $V' = V(E') = \{v \in V \mid \exists e \in E$ with $h(e) = v$ or $t(e) = v\}$, then G' is the subgraph of G induced by the set of edges E'.

If there is a path from a vertex v to a vertex w in a graph G, then w is reachable from v in G. A graph is strongly connected if any vertex in it is reachable from any other vertex. The maximal strongly connected subgraphs of a graph G are vertex-disjoint and are called its strongly connected components.

A (directed, rooted) tree T is a graph with a distinguished vertex r such that there is a unique path from r to any vertex in T . If a vertex v is on the path from r to a vertex w , then v is an ancestor of w and w is a descendant of v . We denote this relation by $v \overset{*}{\rightarrow} w$. We denote the fact that (v,w) is a tree edge by $v \rightarrow w$, and the fact that $v \overset{*}{\rightarrow} w$ and $v \neq w$ by $v \overset{+}{\rightarrow} w$.

3. Path Problems.

Let $R = (S, \oplus, \odot, *, \underline{0}, \underline{1})$ be an algebra consisting of a domain S, two binary operations $\oplus$ and $\odot$, a unary operation $*$, and two constants $\underline{0}, \underline{1} \in S$, satisfying the following axioms.

A1: $(x \oplus y) \oplus z = x \oplus (y \oplus z)$ -- A5: $x \odot (y \odot z) = (x \odot y) \odot z$

A2: $x \oplus y = y \oplus x$ A6: $\underline{0} \odot x = x \odot \underline{0} = \underline{0}$

A3: $x \oplus x = x$ A7: $x \odot \underline{1} = \underline{1} \odot x = x$

A4: $\underline{0} \oplus x = x \oplus \underline{0} = x$ A8: $x \odot (y \oplus z) = x \odot y \oplus x \odot z$.

As a consequence of A1, A2, A3 we can define a partial order $\leq$ on S by: $x \leq y$ ifandonlyif $x \oplus y = y$.

A9: $x \leq y$ implies $x \odot z \leq y \odot z$.

For $x \in S$ and i a non-negative integer, let $x^i = \underline{1}$ if $i = 0$, $x^i = x \odot x^{i-1}$ if $i > 0$.

A10: $x^i \leq x^*$ for all non-negative integers i .

A11: $(z \odot x) \oplus y \leq z$ implies $y \odot x^* \leq z$.

These axioms are a weakening of Salomaa's axioms for a regular algebra [36,37], with right distributivity replaced by a monotonicity axiom (A9) suggested by Graham and Wegman[16] and by Wegbreit [45]. Note that if S contains no zero element $\underline{0}$ but satisfies A1 - A3, A5, A7 -A11 we can always create a zero element $\underline{0}$, defining $\underline{0} \oplus x = x \oplus \underline{0} = x$, $\underline{0} \cdot x = x \cdot \underline{0} = \underline{0}$, $\underline{0}^* = \underline{1}$. It is easy to verify that A1-A11 hold for $S \cup \{\underline{0}\}$.

Lemma 1. If $x \leq y$ and $w \leq z$, then $x \oplus w \leq y \oplus z$ and $x \odot w \leq y \odot z$.

Proof. By A1, A2, and the definition of $\leq$, $x \oplus w \oplus y \oplus z = x \oplus y \oplus w \oplus z = y \oplus z$. Thus $x \oplus w \leq y \oplus z$. By A9, $x \odot w \leq y \odot w$. By A8 and the definition of $\leq$, $y \odot w \oplus y \odot z = y \odot (w + z) = y \odot z$. By the transitivity of $\leq$, $x \odot w \leq y \odot w \leq y \odot z$. 3

Let $G = (V,E)$ be a graph. Let $a: E \to S$, $c: V \to S$. Then (G,a,c) is a path problem. For any path $p = e_1, e_2, \ldots, e_k$ in G we extend a to p by defining $a(p) = a(e_1) \odot a(e_2) \odot \ldots \odot a(e_k)$. If p is a path of no edges, we let $a(p) = 1$. A solution to the path problem (G,a,c) is a mapping $x: V \to S$ such that

C1: $c(t(p)) \odot a(p) < x(h(p))$ for all paths p;

C2: $x(v) \leq z(v)$ for all mappings $z: V \to S$ satisfying the set of inequalities

$$Q(E) = \left\{ \sum_{\substack{e \in E \\ h(e) = v}} z(t(e)) \odot a(e) \oplus c(v) \leq z(v) \mid v \in V(E) \right\} .$$

Lemma 2. Let $z: V \to S$ satisfy the set of inequalities $Q(E)$. Then $c(t(p)) \odot a(p) \leq z(h(p))$ for all paths p.

Proof. Let $p = e_1, e_2, \ldots, e_k$ be any path in G. We prove $c(t(p)) \odot a(p) < z(h(p))$ by induction on k. If $k = 0$, the result is immediate. Suppose the result is true for $k \geq 0$. Then

$$\begin{aligned} c(t(p)) \odot a(p) &= c(t(p)) \odot [a(e_1) \odot \ldots \odot a(e_k)] \odot a(e_{k+1}) \\ &\leq x(t(e_{k+1})) \odot a(e_{k+1}) \quad \text{by the induction hypothesis and A9} \\ &\leq x(h(e_{k+1})) = x(h(p)) \quad \text{by } Q(E) . \end{aligned}$$

□

If f is any function, let $f|_X$ denote the restriction of f to the domain X.

Lemma 3. Let $z: V \to S$ satisfy the set of inequalities $Q(E)$. Let $E' \subseteq E$. Then $z|_{V(E')}$ satisfies $Q(E')$.

Proof. Immediate. □

We shall present a two-step method for solving path problems. The first step is analogous to finding the LU decomposition of a numeric matrix by Gaussian elimination [15]. The second step is analogous to numeric backsolving [15] and is also related to propagation methods for data flow analysis [17,21,23,24]. To present the algorithm we need a few more definitions.

Let $G = (V,E)$ be a graph and let (G,a,c) be a path problem. Let $v,w \in V$ and let P be a set of paths from v to w in G. A value $y \in S$ is a tag for the triple (v,w,P) if

T1: $a(p) \geq y$ for all paths $p \in P$.

T2: $z(w) \geq z(v) \odot y$ for all mappings $z: V \to S$ satisfying $Q(E)$.

Suppose the vertices of a graph $G = (V,E)$ are numbered from one to n and identified by number. A sequence of triples $(v(1),w(1),P(1)), \ldots,(v(k),w(k),P(k))$ with $v(i),w(i) \in V$, $P(i)$ a set of paths from $v(i)$ to $w(i)$ in G is a propagation sequence for G if

P1: $v(i) = w(i) = i$ for $1 \leq i \leq n$ and $v(i) \neq w(i)$ for $n+1 \leq i \leq k$.

P2: Each path p in G can be represented as

$p = p(i_1),p(i_2),\ldots,p(i_{2\ell+1})$, where $i_2 < i_4 < \ldots < i_{2\ell}$,

$1 \leq i_{2j+1} \leq n$ for $0 \leq j \leq \ell$, $n+1 \leq i_{2j} \leq k$ for $1 \leq j \leq \ell$,

and $p(i_j) \in P(i_j)$ for $1 \leq j \leq 2^{l-t-1}$.

Given a propagation sequence $(v(i),w(i),P(i))$ for G and a tag $y(i)$ for each triple in the sequence, the following algorithm computes a solution to the path problem (G,a,c) .

```
SOLVE: begin
   init:  for i := 1 until n do x(i) := c(i) ⊙ y(i);
   main:  for i := n+1 until k do
              x(w(i)) := x(w(i)) ⊕ x(v(i)) ⊙ y(i) ⊙ y(w(i));
      end SOLVE;
```

Theorem 1. The mapping $x: V \to S$ computed by SOLVE is a solution to (G,a,c) .

proof. Let $p = p(i_1),p(i_2),\ldots,p(i_{2\ell+1})$ be a path in G , represented as in P2. We prove by induction on ℓ that C1 holds for p after iteration $i_{2\ell}$ of main . Suppose $\ell = 0$. Then

$$c(t(p)) \odot a(p) = c(t(p(i_1))) \odot a(p(i_1))$$
$$\leq c(i_1) \odot y(i_1) \qquad \text{by T1 and Lemma 1,}$$

and C1 holds after execution of init .

Suppose $\ell \geq 0$. By the induction hypothesis, $c(t(p)) \odot a(p(i_1)) \odot \ldots \odot a(p(i_{2\ell+1})) \leq x(t(p(i_{2\ell+2})))$ after iteration $i_{2\ell}$ of main . Thus

$$c(t(p)) \odot a(p) < x(t(p(i_{2\ell+2}))) \odot a(p(i_{2\ell+2})) \odot a(p(i_{2\ell+3}))$$
$$\leq x(v(i_{2\ell+2})) \odot y(i_{2\ell+2}) \odot y(w(i_{2\ell+2})) \text{ before iteration } i_{2\ell+2} \text{ of main}$$
$$\leq x(w(i_{2\ell+2})) \text{ after iteration } i_{2\ell+2} \text{ of main .}$$

Thus C1 holds for any path p .

To complete the proof, we show by induction on i that x satisfies C2 after iteration i of main . Let z satisfy $Q(E)$.

Then, for $1 \leq i \leq n$, $z(i) \geq c(i)$, and $z(i) \geq z(i) \odot y(i)$ by T2. Hence $z(i) \geq c(i) \odot y(i)$, and x satisfies $z(i) \geq x(i)$ for $1 \leq i < n$ after init. Suppose x satisfies $z(v) > x(v)$ before iteration $i > n+1$ of main. The only value of $x(v)$ which changes during iteration i of main is $x(w(i))$. By the induction hypothesis and T2, $x(v(i)) \odot y(i) \leq z(v(i)) \odot y(i) < z(w(i))$ before iteration i. Also $z(w(i)) \odot y(w(i)) \leq z(w(i))$ by T2. Hence $x(v(i)) \odot y(i) \odot y(w(i)) < z(w(i))$ before iteration i, and $x(w(i)) \leq z(w(i))$ after iteration i. By induction, C2 holds for the final value of x. □

We note several important facts about SOLVE. First, its running time is $O(k)$, where k is the length of the propagation sequence (if $\oplus$, $\odot$, and * require constant time). Also, tags for a propagation sequence depend only on a and not on c. Thus we can solve a set of path problems $(G,a,c_1), (G,a,c_2), \ldots, (G,a,c_\ell)$ by finding a set of tags for a single propagation sequence and then using SOLVE once for each c_i. SOLVE is a generalization of the backsolving step used to solve simultaneous linear equations, and is also related to propagation methods of global flow analysis.

In order to apply SOLVE, we must first compute a propagation sequence and appropriate tags. The following lemmas lead to a way to compute a propagation sequence.

Lemma 4. Let $e \in E$. [illegible] Then a(e) is a tag for $(t(e),h(e),\{e\})$.

Proof. Immediate. □

Lemma 5. Let x be a tag for some triple (v,v,P). Then x^* is a tag for (v,v,P^*), where P^* is the set of all paths formed by concatenating zero or more paths in P. (P^* includes the empty path from v to v.)

Proof. Let p be any path from v to v in $G(E')$. If p is empty, $a(p) = 1 \leq x^*$ by A10. If p is non-empty, p can be represented as $p = p_1, p_2, \ldots, p_k$, where each p_i is a path in P. By T1, $a(p_i) < x$ for all i. Thus $a(p) \leq x^k \leq x^*$ by A10. Hence T1 holds. Let $z: V \to S$ satisfy $Q(E)$. Then $z(v) \geq z(v) \odot x \oplus z(v)$ by T2, and $z(v) \geq z(v) \odot x^*$ by A11. Hence T2 holds. □

Lemma 6. Let y_1, y_2 be tags for (v,w,P_1) and (v,w,P_2), respectively. Then $y_1 \oplus y_2$ is a tag for $(v,w, P_1 \cup P_2)$.

Proof. Let p be any path in $P_1 \cup P_2$. Then p is a path in either P_1 or P_2, say P_1. Hence $a(p) \leq y_1 \leq y_1 \oplus y_2$, and T1 holds for p. Let $z: V \to S$ satisfy $Q(E)$. Then $z(w) \geq z(v) \odot y_1 \oplus z(v) \odot y_2 = z(v)(y_1 \oplus y_2)$ by A8. Hence T2 holds. □

Lemma 7. Let y_1, y_2 be tags for (u,v,P_1) and (v,w,P_2), respectively. Then $y_1 \odot y_2$ is a tag for $(u,w,P_1 \cdot P_2)$, where $P_1 \cdot P_2$ is the set of all paths formed by concatenating a path from P_1 with a path from P_2.

Proof. Let p be any path in $P_1 \cdot P_2$. Then p can be represented as $p = p_1, p_2$ with $p_1 \in P_1$, $p_2 \in P_2$. Hence $a(p) = a(p_1) \odot a(p_2) \leq y_1 \odot y_2$, and T1 holds. Let z satisfy $Q(E)$. Then $z(w) \geq z(v) \odot y_2$ and $z(v) \geq z(u) \odot y_1$. Hence

$z(w) \geq z(u) \odot y_1 \odot y_2$ by A9, and T2 holds. □

The following algorithm, a version of Gaussian elimination, computes tags for certain triples which form a propagation sequence. The algorithm assumes that the vertices of the problem graph G are numbered from one to n and identified by number.

```
ELIMINATE: begin
        for v := 1 until n do for w := 1 until n do y(v,w) := 0;
        for e∈E do y(t(e),h(e)) := y(t(e),h(e)) ⊕a(e);
  loop: for v := 1 until n do begin
            a: y(v, v) := y(v,v)*;
            b: for (u,v) , (v,w) with (u,w > v) and (y(u,v),y(v,w) ≠ 0)
                   do
                       y(u,w) := y(u,w) ⊕y(u,v) ⊙y(v,v) ⊙y(v,w);
end end ELIMINATE;
```

For $u,v,w \in V$, let $P_v(u,w) = \{p = e_1,e_2,\ldots,e_\ell \mid t(p) = u,$ $h(P) = w,$ and $h(e_i) \leq v,\ h(e_i) \notin \{u,w\}$ for $1 \leq i \leq \ell-1\}$. Let $P(u,w) = P_{\min\{u,w\}}(u,w)$. Notice that $P(u,w) = P_{\min\{u,w\}-1}(u, w)$, and $P(v,v)^* = \{p = e_1,e_2,\ldots,e_\ell \mid t(p) = v,\ h(p) = v,$ and $h(e_i) < v$ for $1 \leq i \leq \ell-1\}$.

Theorem 2. For each final value of $y(v,w)$ computed by ELIMINATE, $y(v,w)$ is a tag for $(v,w,P(v,w))$. If $v = w$, $y(v,w)$ is a tag for $(v,w,P(v,w)^*)$.

Proof. We prove by induction on v that after iteration v of loop each value of $y(u,w)$ so far computed is a tag for $(u,w, P_{\min\{u,v,w\}}(u,w))$, and. $y(u,w)$ is a tag for $(u,w,P(u,w)^*)$ if $u = w \leq v$. The hypothesis is true after the first two for loops of

ELIMINATE by Lemma 4 and Lemma 6. Suppose the hypothesis is true after iteration v-1 of loop . Consider iteration v . Execution of step a causes $y(v,v)$ to become a tag for $(v,v,P(v,v)^*)$ by Lemma 5. Consider any set of paths $P_v(u,w)$ with $u,w > v$. This set of paths can be represented as

$P_v(u,w) = P_{v-1}(u,w) \cup P_{v-1}(u,v) \cdot P(v,v)^* \cdot P(v,w)$. Step b computes a tag for each such $P_v(u,w)$ using Lemmas 6 and 7. By induction, the hypothesis holds in general. The theorem follows. □

Theorem 3. The following is a propagation sequence for G .

(1) The elements of $\{(v,v,P(v,v)) \mid v \in V\}$ in any order, followed by

(2) the elements of $\{(v,w, P(v,w) \mid v,w \in V, v < w\}$ in increasing order on v (or on w), followed by

(3) the elements of $\{(v,w,P(v,w)) \mid v,w \in V, v > w\}$ in decreasing order on v (or on w).

Proof. Let p be any path in G . Let $v_1 = t(p)$. For $i > 1$, let v_{i+1} be the first vertex $u > v_i$ following v_i on p . Let v_j be the last such v_i definable (v_j is the largest vertex on p). Similarly let $w_1 = h(p)$. For $i > 1$, let w_{i+1} be the last vertex $u > w_i$ preceding w_i on p . Let w_ℓ be the last such w_i definable. Then $v_j = w_\ell$. We can represent p as

$p = p_1, p_2, \cdots, p_{2j}, p_{2j+1}, \cdots, p_{2j+2\ell-3}$, where $p_{2i} \in P(v_i, v_{i+1})$ for $1 \leq i \leq j-1$, $p_{2j+2i-2} \in P(w_{\ell-i+1}, w_{\ell-i})$ for $1 \leq i \leq \ell-1$, $p_{2i-1} \in P(v_i, v_i)$ for $1 \leq i \leq j$, and $p_{2j+2i-1} \in P(w_{\ell-i}, w_{\ell-i})$ for $1 \leq i \leq \ell-1$. The theorem follows. □

The complete algorithm for solving a path problem consists of three steps:

S1: Apply ELIMINATE to compute tags.

S2: Form the propagation sequence given by Theorem 3, omitting triples $(v,w,P(v,w))$ with tag $\underline{0}$.

S3: Apply SOLVE.

Steps S2 and S3 require $O(k)$ time and space, where k is the number of non-zero tags computed by ELIMINATE. The running time of ELIMINATE depends in a complicated way upon the number of non-zero tags. By rearranging the computations and using appropriate data structures, we can implement ELIMINATE to run in

$$O\left(k + \sum_{v=1}^{n} |\{(u,v) \mid f(u,v) \neq \underline{0},\ u > v\}| \cdot |\{(v,w) \mid f(v,w) \neq \underline{0},\ w > v\}|\right)$$

time and $O(k)$ storage space [7,40]. (By only storing values of $f(v,w)$ which eventually become non-zero, we can avoid spending time zeroing $f(v,w)$ for all v and w .)

For dense graphs the storage bound is $O(n^2)$ and the time bound is $O(n^3)$. For sparse graphs, the resource requirements depend upon the vertex numbering chosen. Numerical analysts have devoted much effort to finding good numbering schemes, both for arbitrary sparse graphs and for graphs with special structure. See [7,32,33,34,35,40].

4. Graph Reversal.

If E is a set of edges, let E^R, the reversal of E, be the set of edges formed by switching the head and tail of each edge in E. If G = (V,E) is a graph, $G^R = (V,E^R)$ is the reversal of G. (Reversing the edges of a graph corresponds to transposing the corresponding adjacency matrix.) Suppose we have a method to solve path problems on G. We would like to transform this method so that it solves path problems on G^R.

Theorem 4. Let $(v,(1),w(1),P(1)), \ldots,(v(k),w(k),P(k))$ be a propagation sequence for G. Then

$(v(1),w(1),P(1)^R),\ldots,(v(n),w(n),P(n)^R)$,

$(w(k),v(k),P(k)^R),\ldots,(w(n+1),v(n+1),P(n+1)^R)$ is a propagation sequence for G^R, where $P^R = \{e_\ell^R, e_{\ell-1}^R,\ldots, e_1^R \mid e_1,e_2,\ldots, e_\ell \in P\}$.

Proof. Immediate. □

Thus any propagation sequence for a graph G can be easily converted into a propagation sequence for its reversal. Furthermore, any computation of tags based on Lemmas 4 -7 can be converted into a computation of tags for the reversal graph by exchanging the arguments of each ⊙ operation corresponding to an application of Lemma 7. Hence our solution method for the path problem (G,a,c) also gives a solution method for the path problem (G^R,a,c).

5. Decomposition by Strong Components.

The purpose of' ELIMINATE is to gather information about the cycles of G . If G has no cycles, SOLVE can be used directly, assuming that the vertex numbering satisfies $t(e) < h(e)$ for each edge e . A numbering which satisfies this property is called a topological ordering [26]. We can find such a numbering in $O(n+m)$ time [26,38]. Thus, for acyclic graphs, there is a simple $O(n+m)$ solution algorithm.

We can generalize this idea. Let G be an arbitrary graph and let $G_1 = (V_1,E_1)$, $G_2 = (V_2,E_2)$, . . . , $G_k = (V_k,E_k)$ be the strongly connected components of G . Using depth-first search, we can compute the components G_i and topologically order them; that is, arrange them so that $e \in E$ with $t(e) \in V_i$ and $h(e) \in V_j$ implies $i \leq j$. This computation requires $O(n+m)$ time [38].

For $1 \leq i \leq k$, let $((v(i,j),w(i,j),P(i,j))$, $1 \leq j \leq \ell_i$, be a propagation sequence for G_i . For $1 \leq i \leq k$, let $n_i = |V_i|$, The following algorithm computes a propagation sequence for G .

```
STRONGSEQ: begin
              SEQ := ∅;
              for i := 1 until k do for j := 1 until n_1 do
                 add (v(i,j),w(i,j),P(i,j)) to SEQ;
              for i := 1 until k do begin
                 for j := n_i+1 until k_i do
                    add (v(i,j),w(i,j),P(i,j)) to SEQ;
                 for e∈E with (t(e) ∈ V_i and (h(e) ∈ V_j ⇒ j > i) do
                    add (t(e),h(e),{e}) to SEQ;
           end end STRONGSEQ;
```

Theorem 5. The sequence computed by STRONGSEQ is a propagation sequence for G .

Proof. Immediate. □ ,

We can compute a propagation sequence with tags for each component G_i by using ELIMINATE. It follows from Lemma 3 that the computed tags are also tags with respect to the graph G . Thus the time to solve a path problem on G is $O(n+m)$ plus the time to apply ELIMINATE to each strong component of G .

Henceforth, we shall assume that the problem graph G is strongly connected; if not, we compute a propagation sequence with tags for each strongly connected component and form a propagation sequence for G using STRONGSEQ. This algorithm corresponds to solving a system of linear equations by decomposing the matrix of coefficients into "irreducible" blocks [15,44]. A "reducible" matrix should not be confused with a "reducible" graph as defined in the next section.

6. Decomposition by Dominators.

The decomposition method presented in Section 5 is quite efficient. However, in most practical problems the problem graph G is strongly connected and the Section 5 method accomplishes nothing. In this section we present a more powerful decomposition method, based upon the dominators of the problem graph, which is efficient and which applies to a large collection of problem graphs which occur in practice.

Let $G = (V,E)$ be a strongly connected directed graph. Let r be a fixed, distinguished vertex of G. If $v,w \in V$ and every path p from r to w contains v, we say v dominates w in G.

Lemma 8. There is a tree T, called the dominator tree of G, such that $v \overset{*}{\rightarrow} w$ in T if and only if v dominates w. Vertex r is the root of T and T contains every vertex in G.

Proof. See [4]. □

For any vertex $w \neq r$, the immediate dominator of w in G is the vertex v such that $v \rightarrow w$ in the dominator tree T. We denote this relationship by $v = \underline{idom}(w)$. By convention $\underline{idom}(r) = 0$. We can compute $\underline{idom}(w)$ for all vertices w in $O(m\ \alpha(m,n))$ time by using depth-first search and a sophisticated data manipulation algorithm [42].

Lemma 9. If $e \in E$, then $\underline{idom}(h(e))$ $h(e)$ in T.

Proof. Every path from r to $h(e)$ contains $\underline{idom}(h(e))$. By adding edge e to any path from r to $t(e)$, we get a path from

r to h(e) . Thus any path from r to t(e) contains idom(h(e)) , and idom(h(e)) dominates t(e) . □

For any edge $e \in E$, let v(e) be idom(h(e)) if t(e) = idom(h(e)) , and let v(e) be the vertex u such that idom(h(e)) $\rightarrow u \overset{*}{\rightarrow}$ t(e) if t(e) $\neq$ idom(h(e)) . Let e* be an edge with $h(e^*) = h(e)$, t(e*) = v(e) . For $v \in V$, let $G(v) = (V(v), E^*(v))$, where $V(v) = \{w \mid \text{idom}(w) = v\}$, $E^*(v) = \{e^* \mid e \in E$ such that idom(h(e)) = v $\neq$ t(e)$\}$. We call the strongly connected components of the graphs G(v) the dominator strong components of G . The dominator strong components partition the vertices of G (excluding r).

The idea of our algorithm is to compute a propagation sequence with tags for G by using a method like ELIMINATE only within the dominator strong components of G . For parts of the propagation sequence connecting dominator strong components, we use the O(m $\alpha(m,n)$) method described in [42] for computing functions defined on a tree (in this case, defined on the dominator tree T). If the strong dominator components *are* small, the resulting algorithm is very efficient; if each strong dominator component contains a single vertex, the entire solution process requires O(m $\alpha(m,n)$) time and space. Luckily, this special case occurs frequently in same of the application areas.

The first part of the algorithm analyzes the graph G . First, we compute the dominator tree T of G using the O(m $\alpha(m,n)$) algorithm of [42]. Next, we compute v(e) for each edge e using the O(m $\alpha(m,n)$) least common ancestors algorithm of [1], also described in [42]. Next, we find the strongly connected components

of each graph $G(v)$ using the $O(m)$ algorithm of [8].

Finally, we number the vertices of G from one to n so that

(1) if $e \in E$ has $v(e)$, $h(e)$ in different dominator strong components of G, then $v(e) > h(e)$.

(2) $v \overset{+}{\rightarrow} w$ *in* T implies $v > w$.

For any edge $e \in E$ with $v(e)$, $h(e)$ in different dominator strong components, either $v(e) = \underline{idom}(h(e))$ or $\underline{idom}(v(e)) = \underline{idom}(h(e))$. If $v(e) = \underline{idom}(h(e))$, then $v(e) > h(e)$ by both condition (1) and condition (2). If $\underline{idom}(v(e)) = \underline{idom}(h(e))$, then $v(e) > h(e)$ by condition (1) and condition (2) does not apply. It follows that there is a numbering satisfying both (1) and (2). We can find such anumbering in $O(m)$ time by using a topological sorting algorithm. The entire graph analysis thus requires $O(m\ \alpha(m,n))$ time (and $O(m)$ space).

The second part of the algorithm computes tags for various triples associated with the graph. An outline appears below.

```
DELIM:  begin
        for v := 1 until n do for w := 1 until n do y(v,w) := 0;
        for e∈E do y(t(e),h(e)) := y(t(e),h(e)) ⊕a(e);
        for v := 1 until n do begin
TREE:       for e∈E such that idom(h(e)) = v do
                compute a tag y(v(e),h(e)) for (v(e),h(e),P1(e));
CYCLE :     for w∈V such that idom(w) = v do begin
                compute a tag y(w,w) for (w, w, P(w, w) ) ;
                Compute a tag y(v,w) for (v,w, P2(v,w)) ;
            end CYCLE;
        end;
        y(n,n) := y(n,n)*;
    end DELIM;
```

In this program $P_1(e) = P_{u(e)}(v(e),h(e))$, where $u(e) = \min\{u \mid \underline{idom}(u) = \underline{idom}(h(e))\}-1$; and $P_2(v,w) = \{p = e_1,e_2, \ldots,e_k \mid t(p) = v,\ h(p) = w,\ h(e_j) < v$ for $1 \leq j \leq k$, and $t(e_j) = w \Rightarrow \exists j' > j$ with $t(e_{j'}) > w\}$.

Step TREE in DELIM uses in its computations the tags computed by previous iterations of CYCLE and TREE. The tags computed by TREE correspond to the edges e* with $\underline{idom}(h(e)) = v$. TREE uses a functional procedure EVAL(v(e),t(e)) such that EVAL(v(e),t(e)) returns the value $\underline{1}$ if $v(e) = t(e)$ and returns the value $y(v_1,v_2) \odot y(v_2,v_2) \odot y(v_2,v_3) \odot \ldots \odot y(v_{\ell-1},v_{\ell-1}) \odot y(v_{\ell-1},v_\ell)$ if $v(e) \neq t(e)$, where $v(e) = v_1,v_2,\ldots,v_\ell = t(e)$ is the sequence of vertices on the path from v(e) to t(e) in T . Here is a more detailed implementation of TREE.

```
TREE :  for e∈E such that idom(h(e)) = v do
            if v(e) ≠ t(e) then
                y(v(e),h(e)) := y(v(e),h(e)) ⊕EVAL(v(e),t(e)) ⊙y(t(e),t(e))
                                                              ⊙y(t(e),h(e));
```

EVAL(v(e),t(e)) computes a tag for $(v(e),t(e),P_2(v(e),t(e))$ by using assignments of the form $y(v_i,v_k) = y(v_i,v_j) \odot y(v_j,v_j) \odot y(v_j,v_k)$, where $v(e) \overset{*}{\rightarrow} v_i \overset{+}{\rightarrow} v_j \overset{+}{\rightarrow} v_k \overset{*}{\rightarrow} t(e)$ in T, $y(v_i,v_j)$ is a previously computed tag for $(v_i,v_j,P_2(v_i,v_j))$, $y(v_j,v_j)$ is a previously computed tag for $(v_j,v_j,P(v_j,v_j))$, and $y(v_j,v_k)$ is a previously computed tag $(v_j,v_k,P_2(v_j,v_k))$. By Lemma 7, each $y(v_i,v_k)$ computed in this way is a tag for $(v_i,v_k,P_2(v_i,v_k))$. After a sufficient number of such assignments, EVAL has computed a tag $y(v_1,v_\ell)$ for $(v_1,v_\ell,E_2(v_1,v_\ell))$. Then, also by Lemma 7, $y(v_1,v_\ell) \odot y(v_\ell,v_\ell) \odot y(v_\ell,h(e))$ is a tag for $(v(e),h(e),P_1(v(e),h(e)) \cap \{p \mid p \text{ contains } e\})$. By Lemma 6, each value $y(v(e),h(e))$ computed by TREE is a tag for $(v(e),h(e),P_1(v(e),h(e)))$.

The total number of EVAL operations carried out by DELIM is m. These operations require $O(m\ \alpha(m,n))$ time if EVAL is implemented as described in [42]. The secrets of this implementation are to save the computed intermediate values $y(v_i,v_k)$ for use in later calls on EVAL, and to order the computations in a clever fashion. Procedure DELIM saves the intermediate values $y(v_i,v_k)$ not only for use in Later calls on EVAL, but also for use as tags in the propagation sequence to be constructed.

Step CYCLE applies versions of STRONGSEQ, ELIMINATE, and SOLVE to the tags computed by TREE. Here is an implementation of CYCLE.

```
CYCLE : begin
   CE: for w∈V such that idom(w) = v, in increasing order of w, do begin
          y(w,w) := y(w,w)*;
          for (u,w),(w,x) with (u,x > w) and (y(u,w),y(w,x) ≠ 0)
             and u,x in same dominator strong component as w do
                y(u,x) := y(u,x) ⊕y(u,w) ⊙y(w,w) ⊙y(w,x);
       end;
   CS: for G_i a dominator strong component of G(v), in topologically
       increasing order, do begin
          for w a vertex of G_i, in increasing order of w, do
             for x a vertex of G_i with (x > w) and (y(w,x) ≠ 0) do
                y(v,w) := y(v,w) ⊕y(v,w) ⊙y(w,w) ⊙y(w,x);
          for w a vertex of G_i, in decreasing order of w, do
             for x a vertex with (idom(x) = v) and (x < w) and (y(w,x) ≠0) do
                y(v,x) := y(v,x) ⊕y(v,w) ⊙y(w,w) ⊙y(w,x);
       end;
end;
```

In this implementation of CYCLE, CE applies the idea of ELIMINATE to each strong component of G(v) , Each value y(u,x) computed by CE is a tag for (u,x,P(u,x)) , assuming that the previous iteration of TREE has correctly computed a tag y(v(e),h(e)) for each e∈E such that idom(h(e)) = v . This follows from a proof like that of Theorem 2.

Step CS of CYCLE uses the ideas in Theorem 3, STRONGSEQ, and SOLVE to compute, for each vertex w such that idom(w) = v , a tag y(v,w) for $(v,w,P_2(v,w))$. This follows easily from a proof using Lemma 6, Lemma 7, and ideas in the proofs of Theorem 1, Theorem 3, and Theorem 5.

Summarizing the above observations, we have the following theorem.

Theorem 6. The procedure DELIM, with TREE and CYCLE implemented as described, computes tags for the following triples.

TR1: $(v,v,P(v,v))$ for $v \in V$.

TR2: $(v,w,P_2(v,w))$ for $v \to w$ in T .

TR3: $(v(e),h(e),P_1(e))$ for $e \in E$.

TR4: $(v,w,P(v,w))$ for each pair of vertices v, w such that v, w are in the same dominator strong component and there is a path $p = e_1^*, e_2^*, \ldots, e_\ell^*$ in the component with $t(e_1^*) = v$, $h(e_\ell^*) = w$, $h(e_i^*) < \min\{v,w\}$ for $1 \le i \le \ell-1$.

TR5: $(v,w,P_2(v,w))$ for a subset SB of the pairs of vertices v, w such that $v \overset{+}{\to} w$ in T, where SB satisfies

(i) $v(e) \neq t(e) \Rightarrow (v(e),t(e)) \in SB$.

(ii) $(v,w) \in SB$ and $\neg(v \to w$ in $T) \Rightarrow \exists x$ such that $v \overset{+}{\to} x \overset{+}{\to} w$ in T and $(v,x), (x,w) \in SB$. We assume that an appropriate x for each $(v,w) \in SB$ is saved by procedure EVAL.

The total amount of computation time required by DELIM is proportionalto $m\,\alpha(m, n)$ plus the time required to apply ELIMINATE to each strong dominator component of G . The amount of storage space required by DELIM is proportionalto $m\,\alpha(m,n)$ (for triples of types TR1, TR2, TR3, TR5) plus k (for triples of type TR4), where k is the total number of non-zero tags resulting from applying ELIMINATE to each strong dominator component of G .

The third part of the algorithm arranges triples of types TR1 - TR5 into a propagation sequence. First, we construct a set of list:: $L(v)$,

one for each vertex v . Each list contains a set of ordered pairs of vertices (v,w) such that $v \overset{+}{\rightarrow} w$ in T . We construct the lists using the following algorithm.

```
LISTS: begin
       for v := 1 until n do L(v) = ∅;
       for each triple (u,w,P2(u,w)) of type TR5 do
          if ¬(u → w in T) then begin
             let u →+ v →+ w in T be such that (u,v),(v,w) ∈SB;
             add (v,w) to L(u);
   end end LISTS;
```

Next, we remove duplicates from each list L(u) and order the pairs (v,w) on each L(u) in decreasing order on w . A radix sort [28] accomplishes this in $O(m\ \alpha(m,n))$ time and space, since the total length of the lists is $O(m\ \alpha(m,n))$. Finally, we apply the following algorithm to compute a propagation sequence.

```
PROP:  begin
          Ps := ∅;
P1:       for v := 1 until n do add (v,v,P(v,v)) to PS;
loop:     for v := 1 until n do
             for G_i a dominator strong component of G(v),
                in topologically increasing order, do begin
P2:             for w a vertex of G_i do
                   for e∈E such that h(e) = w do
                      if t(e) ≠ v(e) then
                         add (h(e),t(e),{e}) to PS;
P3:             for w a vertex of G_i, in increasing order of w, do
                   for x a vertex of G_i with (x<w) and (y(w,x)≠0) do
                      add (w,x,P(w,x)) to PS;
P4:             for w a vertex of G_i in decreasing order of w, do
                   for x a vertex with (idom(x) = v) and
                      (x > w) and (y(w,x) ≠ 0) do
                         add (w,x,P(w,x)) to PS;
P5:             for w a vertex of G_i do for (u,x) ∈ L(w)
                      add (u,x,P_2(u,x)) to PS;
             end;
P6:       for v := n-1 step -1 until1 do begin
             let u → v in T;
             add (u,v,P_2(u,v)) to PS;
          end end PROP;
```

Theorem 7. The sequence PS computed by PROP is a propagation sequence for G .

Proof. Let p be any path in G . Let $v_1 = t(p)$. For $i \geq 1$, let v_{i+1} be the first vertex u following v_i on p such that $u > v_1$ and $\neg(\underline{idom}(v_i) \overset{+}{\rightarrow} \underline{idom}(u)$ in T) . Let v_ℓ be the last such v_i definable in this way. For $1 \leq i \leq \ell$, let w_i be the last vertex u between v_i and v_{i+1} on p such that

$\underline{idom}(v_i) = \underline{idom}(u)$ in T (if there is no such u, let $w_i = v_i$). Then we can represent p as $p = p_1, p_2, \ldots, p_{2\ell-3}, p_{2\ell-2}$, where $t(p_{2i-1}) = v_i$, $t(p_{2i}) = w_i$ for $1 \leq i \leq P-1$, $h(p_{2\ell-2}) = h(p)$, p_{2i-1} contains only proper descendants of $\underline{idom}(v_i)$, and p_{2i} contains only proper descendants of w_i (with the exception of $t(p_{2i})$ and $h(p_{2i})$). Note that any path p_{2i-1} can be empty, as can $p_{2\ell-2}$.

Since every vertex u on $p_{2\ell-2}$ except $t(p_{2\ell-2})$ satisfies $t(p_{2\ell-2}) \xrightarrow{+} u$ in T, we can write $p_{2\ell-2}$ as $p_{2\ell-2} = p_{2\ell-2,1}, p_{2\ell-2,2}, \ldots, p_{2\ell-2,2k}$, where $t(p_{2\ell-2,2j-1}) \rightarrow h(p_{2\ell-2,2j-1})$ in T, $t(p_{2\ell-2,2j}) = h(p_{2\ell-2,2j})$, $p_{2\ell-2,2j-1}$ is a path in $P_2(t(p_{2\ell-2,2j-1}), h(p_{2\ell-2,2j-1}))$, and $p_{2\ell-2,2j}$ is a path in $P(t(p_{2\ell-2,2j}), h(p_{2\ell-2,2j}))$. The triples $(v, w, P_2(v,w))$ for $v \rightarrow w$ in T are added to the end of PS, in decreasing order of w, by step P6 of PROP.

For each $1 \leq i \leq \ell-2$, every vertex u on p_{2i-1} satisfies $\underline{idom}(v_i) \xrightarrow{+} u$. Applying the ideas in Theorem 3 and Theorem 5, we can represent p_{2i-1} as a sequence of paths selected, in order, from the path sets $P(w,x)$ added to PS during steps P3 and P4 in iteration $\underline{idom}(v_i)$ of loop , alternating with paths selected from the path sets $P(v,v)$ added to PS during step P1.

What remains to be shown is that, for each $1 \leq i \leq L-2$, p_{2i} can be represented as a sequence of paths selected, in order, from the path sets $P_2(v,w)$ added to PS during step P5 in iterations $\underline{idom}(w_i)$ to $\underline{idom}(v_{i+1})-1$ of loop , alternating with paths selected

from appropriate path sets $F(v, v)$, and ending with a path set $\{e\}$ added to PS during iteration $\underline{idom}(v_{i+1})$ of step P3. Thus, consider any path p_{2i}. Let $p_{2i,1} = e$ be the path consisting of the last edge e on p_{2i}. Then $p_{2i} = x_1, p_{2i,2}, p_{2i,1}$, where x_1 is a path in $P_2(t(x_1),h(x_1))$, $p_{2i,2}$ is a path in $P(t(p_{2i,1}),t(p_{2i,1}))$, and DELIM has computed a tag for $(v(e),t(e),P_2(v(e),t(e)))$.

Let $j = 1$, $z_j = v(e)$. We repeat the following step until reaching a value of j for which $t(x_j) = h(x_j)$. We have $z_j \xrightarrow{+} t(x_j) \xrightarrow{*} h(x_j)$ and $(z_j,h(x_j),P_2(z_j,t(x_j))) \in SB$. If $t(x_j) \neq h(x_j)$, there is some z such that $z_j \xrightarrow{+} z \xrightarrow{+} h(x_j)$ and $(z_j,z,P_2(z_j,z)) \cdot (z,h(x_j),P_2(z_j,z)) \in SB$. If $t(x_j) \xrightarrow{*} z$, let $z_{j+1} = z_j$, $x_j = x_{j+1}, p_{2i,2j+2}, p_{2i,2j+1}$, where x_{j+1} is a path in $P_2(t(x_j),z)$, $p_{2i,2j+2}$ is a path in $P(z,z)$, and $p_{2i,2j+1}$ is a path in $P_2(z,h(x_j))$. If $z \xrightarrow{+} t(x_j)$, let $z_{j+1} = z$, $x_{j+1} = x_j, p_{2i,2j+2}, p_{2i,2j+1}$, where $p_{2i,2j+2}$ and $p_{2i,2j+1}$ are empty paths. Since the distance between z_j and $h(x_j)$ in T strictly decreases with increasing j, eventually we reach a value of j, say k, for which $t(x_k) = h(x_k)$. Then x_k and $p_{2i,2k}$ are empty paths, and we have decomposed p_{2i} as $p_{2i} = p_{2i,2k-1}, p_{2i,2k-2}, \ldots, p_{2i,2}, p_{2i,1}$, where $p_{2i,2j+1}$ is a path in $P_2(t(p_{2i,2j+1}),h(p_{2i,2j+1}))$, $p_{2i,2j}$ is a path in $P(t(p_{2i,2j}),h(p_{2i,2j}))$, and $(t(p_{2i,2j+1}),h(p_{2i,2j+1}))$ is on $L(z_j)$ if $p_{2i,2j+1}$ is not empty, for $1 < j < k-1$. Since either $z_j \xrightarrow{+} z_{j+1}$

or $t(x_{j+1}) \xrightarrow{h} t(x_j)$ in T , the triples corresponding to the non-empty paths $p_{2i,2j+1}$ are added to PS in the order

$(t(p_{2i,2k-1}), h(p_{2i,2k-1}), P_2(t(p_{2i,2k-1}), h(p_{2i,2k-1}))), \ldots,$

$(t(p_{2i,j}), h(p_{2i,j}), P_2(t(p_{2i,j}), h(p_{2i,j})))$ during step S5 in iterations $\underline{idom}(w_i)$ to $\underline{idom}(v_{i+2})-1$ of loop . Triple $(t(e),h(e),\{e\})$ is added to PS during step S2 in iteration $\underline{idom}(v_{i+1})$

Combining the decompositions of the paths p_{2i-1} , p_{2i} $(1 \leq i < t-2)$, $p_{2\ell-3}$, and $p_{2\ell-2}$ gives a decomposition of p which satisfies the condition for a Propagation sequence. □

Below is a summary of the decomposition algorithm for solving path problems.

Step 1: Analyze the graph G to find its dominator strong components and number its vertices.

Time: $O(m\ \alpha(m,n))$

Space: $O(m)$.

Step 2: Apply DELIM to compute tags.

Time: $O(m\ \alpha(m,n)$ + elimination time within dominator strong components)

Space: $O(m\ \alpha(m,n) + k)$, where k is fill-in within dominator strong components.

Step 3: Apply LISTS and PROP to compute a Propagation sequence.

Time: $O(m\ \alpha(m,n) + k)$

Space: $O(m\ \alpha(m,n) + k)$.

Step 4 : Apply SOLVE.

Time: $O(m\ \alpha(m,n) + k)$

Space: $O(m\ \alpha(m,n) + k)$.

We see that the total running time of the algorithm is proportional to $m\alpha(m,n)$ plus the elimination time within the dominator strong components, and the storage requirements are proportional to $m\ \alpha(m, n)$ plus the fill-in within the dominator strong components. In summary, this algorithm allows us to trade a slightly non-linear overhead cost for large savings in elimination time, if the graph G has more than a few dominator strong components. Using Theorem 4, we can also apply the algorithm profitably to graphs whose reversal has more than a few dominator strong components.

The power of this algorithm lies in the fact that in several important application areas, most of the graphs of interest readily decompose into many dominator strong components. A graph such that each of its dominator strong components has a single vertex we call a reducible graph (relative to the fixed vertex r). This definition is not the standard one, but it is equivalent to many other characterizations; see [18,19,41]. On reducible graphs, the decomposition algorithm carries out no elimination; the total time and space requirements are $O(m\ \alpha(m,n))$. (In this case the algorithm can also be simplified somewhat.)

Ullman [43], Kennedy [23], and Graham and Wegman [16] have proposed O(m log n) time algorithms for solving global flow analysis problems on directed graphs. Our algorithm constitutes a generalization of the Graham-Wegman algorithm to arbitrary graphs, and to solving

arbitrary path problems. By using the improved data manipulation algorithm of [42], we have reduced the time bound to $O(m\ \alpha(m,n))$ for reducible graphs. The extension to arbitrary graphs using dominator strong components seems to be a natural idea, apparently overlooked by previous researchers.

7. Variants of the Axiom System.

This section considers several ways in which the axioms can be modified without affecting the validity of the algorithms presented.

Boundedness.

In some applications (especially in global flow analysis [14,16,21]), the * operation is not present. Instead, an axiom of the form

AB: $\quad x^{k+1} \leq \sum_{i=0}^{k} x^i$

is assumed. In this case we can define $x^* = (1+x)^k$. It is then easy to prove A10 and A11. To compute x^*, we apply the formula

$$x^* = (1 \oplus x)^{2^{\log_2 k}} = (\ldots((1 \oplus x)^2)^2 \ldots)^2 ,$$

which uses $\log_2 k$ $\oplus$ and $\odot$ operations to compute x^*.

Distributivity.

In applications to regular expressions [6,25,36,37], We can strengthen axioms A9, A10, A11 to

A9D: $\quad (x \oplus y) \odot z = (x \odot z) \oplus (y \odot z)$

A10D: $\quad (y \odot x^* \odot x) \oplus y = y \odot x^*$

A11D: $\quad z \odot x \oplus y = z \quad$ implies $\quad y \odot x^* \leq z$.

In this case the solution to a path problem (G,a,c) is the minimum solution to the set of equations

$$QE(E) = \left\{ \sum_{\substack{e \in E \\ h(e) = v}} z(t(e)) \odot a(e) \oplus c(v) = z(v) \mid v \in V \right\} .$$

Inverses.

In applications to numeric problems, axiom A3 does not hold. Instead of A3, A9, A10, A11 we assume

A3I: For all x there is an element $\ominus x \in S$ such that

$$x \oplus (\ominus x) = \ominus x \oplus x = 0 \ .$$

A9I: $(x \oplus y) \odot z = x \odot z \oplus y \odot z$.

A10I: For all $x \neq 0$ there is an element $x^{-1} \in S$ such that

$$x \odot x^{-1} = x^{-1} \odot x = 1 \ .$$

These are the axioms of a division ring. We define $x^* = (1 \oplus \ominus x)^{-1}$ for $x \neq 1$. Then $z = y \odot x^*$ is the unique solution to the equation $(z \odot x) \oplus y = z$. A solution to a numeric path problem is a vector z satisfying QE(E) .

The definitions and proofs in Sections 3 - 6 are not valid for numeric path problems, because deletion of axiom A3 means there is no partial order defined on the set S . However, the solution algorithms presented in Sections 3 - 6 are still valid. For a development of the ideas necessary for new proofs, see [15,44]. See [40] for further discussion of a numeric version of the decomposition algorithm in Section 6.

An added difficulty in the numeric case is that 1^* is undefined. This means that not all path problems have solutions. Furthermore the elimination methods in Sections 3 - 6 may not find solutions even for path problems which have them. Numerical analysts have developed various pivoting schemes to overcome this problem [15]. It is interesting to note that the existence of additive inverses allows the use of independent permutations of rows and columns in the matrix of coefficients

to rearrange the computations [15]. In the non-numeric applications covered by the Section 3 axiom system, only simultaneous permutations of rows and columns are valid. In addition, the existence of multiplicative inverses allows simplification of the tree manipulation method underlying the algorithm of Section 6 (see [42]).

8. Applications.

This section presents several of the more common types of path problems. Many others undoubtedly exist.

Applications on Acyclic Graphs.

Suppose we wish to find the transitive closure of a graph $G = (V,E)$. We can assume that G is acyclic (if not, we first find its strongly connected components and reduce each to a single vertex). Let $S = \{Y \mid Y \subseteq V\}$, $c(v) = \{v\}$ for $v \in V$, $a(e) = \emptyset$ for $e \in E$, $Y \oplus Z = Y \cup Z$, $Y \odot Z = Y \cup Z$. If $x(v)$ is a solution to (G,a,c), then $x(v)$ is the set of vertices from which v is reachable in G. A solution $x(v)$ can be computed in $O(n+m)$ set union operations using the method suggested in Section 4. For an exposition of this well-known algorithm, see [12].

We can use the same idea to compute dominators in an acyclic graph. Let $G = (V,E)$ be acyclic and let r ve a fixed vertex. Let $S = \{Y \mid Y \subseteq V\}$, $c(v) = \{v\}$ for $v \in V$, $a(e) = \{h(e)\}$ for $e \in E$, $Y \oplus Z = Y \cap Z$, $Y \odot Z = Y \cup Z$. If $x(v)$ is a solution to (G,a,c), then $x(v)$ Is the set of dominators of v. The Section 4 method computes the sets $x(v)$ in $O(n+m)$ set operations. This algorithm is due to Hecht and Ullman [17]. Note that the dominators for an arbitrary graph can be computed in $O(m\ \alpha(m,n) + \sum_{v \in V} |x(v)|)$ time without using set operations [42].

As a last application of this kind, consider critical path analysis. Let $G = (V,E)$ be an acyclic directed graph with a source vertex s, a sinkvertex t, and length a(e) on each edge. We desire the length and location of a longest path from s to t. Let

$c(s) = 0$, $C(v) = -\infty$ for $v \in V-\{s\}$, $y \oplus z = \max\{y,z\}$, $y \odot z = y+z$. If $x(v)$ is a solution to (G,a,c) , then $x(t)$ is the length of a longest path from s to t , and such a path can be constructed by examining $x(v)$ for appropriate vertices v . Computing $x(v)$ requires $O(n+m)$ time. See [8].

Simple Applications on Graphs with Cycles.

Let G be a graph, let Σ be a finite set, and let Σ^* denote the set of finite strings over Σ . Let A denote the empty string. Let S denote the set of subsets of Σ^* . For $e \in E$, let $a(e) = [w(e))$, where each $w(e)$ is some word in Σ^* . Let $\oplus$ denote set -union, let $\odot$ denote set concatenation $(Y \odot Z = \{yz \mid y \in Y \text{ and } z \in Z\})$, and let * denote transitive closure $(Y^* = \bigcup_{i=0}^{w} Y^i$, where $Y^0 = \{\Lambda\}$ and $Y^i = Y^{i-1} \odot Y)$. Let r be a fixed vertex in G and let $c(r) = \{\Lambda\}$, $c(v) = \emptyset$ if $v \in V-\{r\}$. If $x(v)$ is a solution to (G,a,c) , then $x(v) = \{a(e_1) \odot a(e_2) \odot \cdots \odot a(e_k) \mid p = e_1, e_2, \ldots, e_k$ is a path from r to v in G$\}$. Computing the regular set recognized b-y a finite automaton is thus a path problem. See [6,25,36,37].

Let $G = (V,E)$ be a graph and let $a(e)$ for $e \in E$ denote the length of the edge e . Let r be a fixed vertex of G . We desire the length of the shortest path from r to every other vertex. Alternately, we desire the length of the shortest paths between all pairs of vertices. We allow negative edge lengths. Let $c(r) = 0$, $c(v) = \infty$ for $v \in V-\{r\}$, $y \oplus z = \min\{y,z\}$, $y \odot z = y+z$, $y^* = \begin{cases} 0 & \text{if } y \geq 0 \\ -\infty & \text{if } y < 0 \end{cases}$. Then a solution $x(v)$ to (G,a,c) gives

the length of a shortest path from r to v . By computing a propagation sequence and applying SOLVE n times, we can find shortest paths for all vertex pairs. The time required by this method for either the single source or the all pairs problem is $O(n^3)$ for a dense graph and less for a sparse graph. See [9,13,20] for shortest path algorithms which use elimination methods.

Dijkstra [11] has given an $O(n^2)$ algorithm for the sin@ source problem with non-negative edge lengths. This algorithm runs in $O(\min\{n^2, m \log n\})$ time if the proper data structures are used [20].

Global Flow Analysis.

The following application is an abstraction of a problem which arises often when doing global flow analysis of computer programs. Let L be a set with an operation $\oplus$ and a zero element 0 satisfying A1, A2, A3, A4. Let S be a set of functions $f: L \to L$ satisfying the following axioms.

F1: S is closed under composition and $\oplus$ (where $f \oplus g$ is the function h defined by $h(x) = f(x) \oplus g(x)$).

F2: S contains an identity function 1 such that $1(x) = x$.

F3: Each function in S is monotonic; that is, if $f \in S$, $x,y \in L$, and $x < y$, then $f(x) \leq f(y)$.

If $f,g \in S$, let $f \odot g$ be the function h such that $h(x) = g(f(x))$. If $f \in S$, $x \in L$, let $f \odot x$ denote $f(x)$ .With this definition, A1 - A9 hold on S .

F4: For all $f \in S$, there is an $f^* \in S$ satisfying A10 and A11.

The idea of these definitions is the following. Let $G = (V,E)$ be a directed graph with a fixed vertex r . Let $a(e) \in S$ for all $e \in E$. Let $c(r) \in L$. Let $c(v) = 0$ for $v \in V-\{r\}$. The graph G represents a computer program; each vertex of G represents a basic block of code (a block with only a single entry and a single exit point). The set L represents a set of properties which can hold in various blocks of the program. The vertex r is the start of the program. For each edge e , $a(e)(z)$ is the property which holds at t(e) if the property z holds at h(e) and the program takes the branch corresponding to edge e .

Assume that the property $c(r) \in L$ holds at the start of the program. We desire, for each $v \in V$, a property $x(v) \in L$ such that

P1: $x(v)$ holds at block v , independent of the execution sequence which causes control to reach block v .

In theory, we would like the "best" such set of properties x(v) ("best" means "smallest relative to $\leq$ "). In general there may not be such a "best" set, and even if there is, the set may not be effectively computable [22]. We will settle for a set of properties x(v) satisfying C1 and C2. We can construct such a set of properties by using the algorithms in Sections 3 -6. First we compute tags *for a* propagation sequence by using function addition, composition, and transitive closure. Next we apply SOLVE, which finds a solution by using function application and addition of elements in L .

Many authors have studied algorithms for this data flow problem and discussed concrete examples of it (see [3,4,5,10,14,16,17,21,22, 23,24,29,31,43,45]). For most applications, $\oplus$, $\odot$, and * can be computed efficiently (i.e. in constant time; see [14]).

For arbitrary graphs the worst-case running times ofallkncwn algorithms are O(nm) or worse; the running time of the elimination algorithm of Section 3 is $O(n^3)$ (faster if G is sparse). For some restricted classes of graphs, such as reducible graphs, there are faster algorithms. Cocke and Allen [5,10] introduced reducible graphs, but their global flow algorithms were O(nm) . Ullman [43] devised an O(m log n) algorithm for eliminating common subexpressions in computer programs with reducible flow graphs; Fong, Kam, and Ullman [14] later extended this algorithm to an abstract setting. Kennedy [23] devised an algorithm for all graphs which uses node listings. For reducible graphs, the algorithm is O(m log n) by a result of Aho and Ullman[3]; this bound is tight [30]. Graham and Wegman [16] devised another O(m log n) algorithm, which Reif [31] simplified and extended. The Graham-Wegman algorithm served as the starting point for the faster and more general algorithm of Section 6.

The node listing method of Kennedy is a propagation method; it uses only function application and addition of elements in L . References [17,21,24] describe less efficient propagation algorithms. In order for these propagation methods to work, the boundedness axiom AB described in Section 7 must hold. Otherwise, x^* cannot be computed from x . The methods of Ullman, Graham and Wegman, and our methods do not require the boundedness condition.

We make the following conjectures. Consider a global flow problem on a graph G such that the underlying algebra satisfies the boundedness condition AB for k = 1 and the right distributivity axioms A9D - A11D (see Section 7). Suppose G is reducible, with O(n) edges. Then (1) any propagation method (i.e., a method which uses only function

application and addition of elements in L) requires at least $C n \log n$ operations to solve the global flow problem (in the worst case), where C is some positive constant. Furthermore (2) any method which uses function application, function composition, and addition of either elements of L or functions requires at least $C n\alpha(n,n)$ operations to solve the global flow problem (in the worst case).

The ideas in [30] and [39,42] may lead to proofs of (1) and (2).

Numeric Applications.

As discussed in Section 7, the algorithms of Sections 3 - 6 can be used to solve systems of linear equations with numeric coefficients. For any system whose underlying graph is reducible or almost-reducible, the algorithm of Section 6 will be very efficient. Two related examples of cases in which this may happen are when computing steady-state probabilities for a Markov chain (especially if the chain represents an operating system or other computer program) and when using Kirkoff's laws to compute the number of times each step in a computer program is executed [27].

9. Remarks and Conclusions.

This paper has given an axiomatic framework for path problems on directed graphs, described a method similar to Gaussian elimination for solving them, and presented two decomposition schemes for speeding up the elimination method. The first decomposition scheme uses the strongly connected components of the problem graph G ; the scheme is well-known to numerical analysts. The second method, more powerful than the first, uses the dominators of G .

The second method reduces the time to solve path problems on reducible graphs from $O(m \log n)$ to $O(m\,\alpha(m,n))$, where G has n vertices and m edges. The method improves and generalizes an algorithm of Graham and Wegman for solving global flow problems on reducible graphs. We conjecture that the method is optimum to within a constant factor for solving path problems on reducible graphs. The method is likely to be not only theoretically efficient but practically efficient as well.

By combining the dominators decomposition method and the corresponding method for the reversal of a graph, we get an even more powerful decomposition method. It may be possible to define the "strongly biconnected components" of a directed graph and to extend the dominators decomposition idea to these components. Doling this remains an open problem.

References

[1] A. Aho, J. Hopcroft, and J. Ullman, "On computing least common ancestors in trees," Proc. Fifth Annual ACM Symposium on Theory of Computing (1973), 253-265.

[2] A. Aho, J. Hopcroft, and J. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass. (1974), 195-201.

[3] A. V. Aho and J. D. Ullman, "Node listings for reducible flow graphs," Proc. Seventh Annual ACM Symposium on Theory of Computing (1975), 177-185.

[4] A. V. Aho and J. D. Ullman, The Theory of Parsing, Translation, and Compiling, Vol. II: Compiling, Prentice-Hall, Englewood Cliffs, N. J. (1972).

[5] F. E. Allen, "Control flow analysis," SIGPLAN Notices, Vol. 5 (1970), 1-19.

[6] R. C. Backhouse and B. A. Carré, "Regular algebra applied to path-finding problems/ J. Inst. Maths. Applics., Vol. 15 (1975), 161-186,.

[7] J. R. Bunch and D. J. Rose, "Partitioning, tearing, and modification of sparse linear systems," J. Math. Analysis and Applications, Vol. 48 (1974), 574-593.

[8] R. G. Busacker and T. L. Saaty, Finite Graphs and Networks, McGraw-Hill, New York (1965), 128-135.

[9] B. A. Carré, "An algebra for network routing problems," J. Inst. Maths. Applics., Vol. 7 (1971), 273-294.

[10] J. Cocke, "Global common subexpression elimination," SIGPLAN Notices, Vol. 5 (1970), 20-24.

[11] E. W. Dijkstra, "A note on two problems in connection with graphs," Numerische Math., Vol. 1 (1959), 269-271.

[12] J. Eve, "On computing the transitive closure of a relation," STAN-CS-75-508, Computer Science Dept., Stanford University (1975).

C13] R. Floyd, "Algorithm 97: shortest path," Comm. ACM 5 (1962), 345.

[14] A. Fong, J. Kam, and J. Ullman, "Application of lattice algebra to loop optimization," Conf. Record of the Second ACM Symposium on Principles of Prog. Lang. (1975), 1-g.

[15] G. E. Forsythe and C. B. Moler, Computer Solution of Linear Algebraic Equations, Prentice-Hall, Englewood Cliffs, N. J. (1967).

[16] S. Graham and M. Wegman, "A fast and usually linear algorithm for global flow analysis," Conf. Record of the Second ACM Symposium on Principles of Prog. Lang. (1975), 22-34.

[17] M. S. Hecht and J. D. Ullman, "Analysis of a simple algorithm for global flow problems," Conf. Record of the ACM Symposium on Principles of Prog. Lang. (1973), 207-217.

[18] M. S. Hecht and J .D. Ullman, "Flow graph reducibility," Vol. 1 (1972), 188-202.

[19] M. S. Hecht and J. D. Ullman, "Characterizations of reducible flow graphs," J. ACM., Vol. 21 (1974),367-375.

[20] D. B. Johnson, "Algorithms for shortest paths," Ph.D. Thesis, Cornell University, Ithaca, N.Y. (1973).

[21] J. Kam and J. D. Ullman, "Global optimization problems and iterative algorithms," TR-146, Dept. of Electrical Engineering, Princeton University (1974).

[22] J. Kam and J. D. Ullman, "Monotone data flow analysis frameworks," unpublished report, Princeton University (1975).

[23] K. W. Kennedy, "Node listings applied to data flow analysis," Conf. Record of the Second ACM Symposium on Principles of Prog. Lang. (1973), 10-21.

[24] G. A. Kildall, "A unified approach to global program optimization," Conf. Record of the ACM Symposium on Principles of Prog. Lang. (1973), 194-206.

[25] S. C. Kleene, "Representation of events in nerve nets and finite automata," Automata Studies, Shannon and McCarthy, eds., Princeton University Press, Princeton, N.J. (1956), 3-40.

[26] D. Knuth, The Art of Computer Programming, Vol. 1: Fundamental Algorithms, Addison-Wesley, Reading, Mass. (1968), 258-265.

[27] ibid, 364-370.

[28] D. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley, Reading, Mass. (1973), 170-178.

1291 E. S. Lorry and C. W. Medlock, "Object code optimization," Comm. ACM., Vol. 12 (1969), 13-22.

[30] G. Markowsky and R. Tarjan, "Lower bounds on the lengths of node sequences in directed graphs," IBM Research Report No. RC 5477, Yorktown Heights, New York (1975).

[31] J. Reif, private communication, Harvard University (1975).

[32] D. Rose, "Triangulated graphs and the elimination process," J. Math. Analysis and Applications, Vol. 32 (1970), 597-609.

[33] D. Rose, "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations," Graph Theory and Computing, R. Read, ed., Academic Press, N.Y. (1973), 183-217.

[34] D. Rose, R. Tarjan, and G. Lueker, "Algorithmic aspects of vertex elimination on graphs," SIAM J. Comput., to appear.

[35] D. Rose and R. Tarjan, "Algorithmic aspects of vertex elimination," Proc. Seventh Annual ACM Symposium on Theory of Computing (1975), 245-254.

[36] A. Salomaa, "Two complete axiom systems for the algebra of regular events," J. ACM., Vol. 13 (1966), 158-169.

[37] A. Salomaa, Theory of Automata, Pergamon Press, Oxford, England (1969), 120-127.

[38] R. Tarjan, "Depth-first search and linear graph algorithms," SIAM J. Comput., vol. 1 (1972), 146-160.

[39] R. Tarjan, "Efficiency of a good but not linear set union algorithm," J. ACM., vol. 22 (1975), 215-225.

[40] R. Tarjan, "Graph theory and Gaussian elimination," Sparse Matrix Computations, J. Bunch and D. Rose, eds., Academic Press, New York, to appear.

[41] R. Tarjan, "Testing flow graph reducibility," J. Comp. Sys. Sciences, Vol. 9 (1974), 355-365.

[42] R. Tarjan, "Applications of path compression on balanced trees," STAN-CS-75-512, Computer Science Dept., Stanford University (1975).

[43] J. D. Ullman, "A fast algorithm for the elimination of common subexpressions," Acta Informatica, Vol. 2 (1973), 191-213.

[44] R. S. Varga, Matrix Iterative Analysis, Prentice-Hall, Englewood Cliffs, N.J. (1962).

[45] B. Wegbreit, "Property extraction in well-founded property sets," Computer Science Division, Bolt Beranek and Newman, Inc., Cambridge, Mass. (1973).