# GRAPH THEORY AND GAUSSIAN ELIMINATION

by

Robert E. Tarjan

STAN-CS-75-526
NOVEMBER 1975

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

# Graph Theory and Gaussian Elimination

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

Abstract

This paper surveys graph-theoretic ideas which apply to the problem of solving a sparse system of linear equations by Gaussian elimination. Included are a discussion of bandwidth, profile, and general sparse elimination schemes, and of two graph-theoretic partitioning methods. Algorithms based on these ideas are presented.

Keywords: bandwidth, dominators, Gaussian elimination, profile, sparse linear systems, strongly connected components.

This research was supported in part by National Science Foundation grant DCR72-03752 A02 and by the Office of Naval Research contract NR 044-402. Reproduction in whole or in part is permitted for any purpose of the United States Government.

## 1. Introduction.

Consider the system $xM = c$, where $M$ is a nonsingular real-valued $n$ by $n$ matrix, $x$ is a one by $n$ vector of variables, and $c$ is a one by $n$ vector of constants. We wish to solve this system of equations for $x$. In many applications, $M$ is a large sparse matrix; that is, $M$ has many zero elements. If the system is solved using Gaussian elimination or some other direct method, many of the zeros in $M$ may become non-zero. To make the solution process efficient, we would like to avoid having to explicitly examine the zeros of $M$, and to keep the number of non-zero elements small.

We can model the zero -non-zero structure of $M$ by a directed graph and study the effect of a solution method on this graph. This graph-theoretic analysis has several important benefits, including the following.

(1) For some sparse matrices, a graph-theoretic representation is a good one, allowing efficient access of non-zero matrix elements.

(2) We can devise a good solution procedure for an entire class of matrices (those with the same zero -non-zero structure) at a time. If several matrices with the same zero -non-zero structure occur in an application, then spending extra time initially to devise a good solution procedure may result in later savings as the procedure is reused.

(3) The approach illuminates the applicability of solution methods for linear systems to other kinds of graph problems such as those arising in global flow anayslis and operations research.

This paper surveys several graph-theoretic aspects of the solution of linear systems. We consider several graph-theoretic methods for choosing a good ordering scheme for Gaussian elimination. These include bandwidth minimization, profile minimization, and general sparse techniques. We also discuss graph-theoretic block methods based on the strongly-connected components and dominators of the underlying graph. Finally, we discuss the problem of choosing a set of pivot positions for Gaussian elimination.

The paper contains seven sections. Section 2 introduces necessary graph-theoretic notation. Section 3 discusses representation of a system of linear equations as a graph, a decomposition method which uses strongly connected components, and a graphical version of Gaussian elimination. Section 4 discusses methods for choosing a pivot order. Section 5 discusses a decomposition method using dominators. Section 6 discusses selection of a set of pivot positions. Section 7 contains further remarks. Results in Sections 3, 4, and 6 are not new, though some are as yet unavailable in print. Section 5 contains new results.

## 2. Graph-Theoretic Notation.

A directed graph $G = (V,E)$ is a finite set $V$ of $n = |V|$ elements called vertices and a finite set $E \subset V \times V$ of $m = |E|$ vertex pairs called edges. An edge of the form $(i,i)$ is a loop. If $(i,j) \in E$ if and only if $(j,i) \in E$, we say $G$ is symmetric. (A symmetric directed graph corresponds to the undirected graph given by making $\{i,j\}$ an undirected edge if $(i,j) \in E$. We prefer to use symmetric directed graphs instead of undirected graphs since they correspond more closely to the computer representation of graphs.)

A graph $G' = (V',E')$ is a subgraph of $G$ if $V' \subseteq V$ and $E' \subseteq E$. If $V'' \subseteq V$ and $G(V'') = (V'',E(V''))$ where $E(V'') = \{(i,j) \in E \mid i,j \in V''\}$, then $G(V'')$ is the subgraph of $G$ induced by the vertex set $V''$. Similarly, if $E'' \subset E$ and $G(E'') = (V(E''),E'')$ where $V(E'') = \{i \in V \mid \exists (i,j) \in E'' \text{ or } \exists (j,i) \in E''\}$, then $G(E'')$ is the subgraph of $E$ induced by the edge set $E''$.

A sequence of edges $p = (v_1,v_2),\ldots,(v_k,v_{k+1})$ is a path from $v_1$ to $v_{k+1}$. By convention there is a path of no edges from every vertex to itself. If $v_1 = v_{k+1}$ the path is a cycle. Every cycle contains at least one edge. The path is said to contain vertices $v_1,v_2,\ldots,v_{k+1}$ and edges $(v_1,v_2),\ldots,(v_k,v_{k+1})$ and to avoid all other vertices and edges. If $v_1,v_2,\ldots,v_{k+1}$ are distinct, except possibly $v_1 = v_{k+1}$, $p$ is simple.

If there is a path from a vertex v to a vertex w , . v is reachable from w . If every vertex in a graph G is reachable from every other vertex, G is strongly connected. The maximal strongly connected subgraphs of a graph G are vertex-disjoint and are called its strongly connected components. If u , v , w are distinct vertices of a graph G such that every path from u to w contains v , then v is a dominator of w with respect to u . If G contains no three distinct vertices u , v , w such that v dominates w with respect to u , G is strongly biconnected. The maximal strongly biconnected subgraphs of G are edge disjoint (except for loops) and and are called the strongly biconnected components of G .

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. If v is onthepathfrom r to w, we write $v \overset{*}{\rightarrow} w$ and say v is an ancestor of w and w is a descendant of v . If (v,w) is a tree edge, we write $v \rightarrow w$ and say v is the parent of w and w is a child of v . If $v \overset{*}{\rightarrow} w$ and $v \neq w$ , 'we write $v \overset{+}{\rightarrow} w$ and say v is a proper ancestor of w and w is a proper descendant of T .

If G = (V,E) is any graph, the symmetric (or undirected) extension of G is the graph $G' = (V, \{(i,j) \mid (i,j) \in E \text{ or } (j,i) \in E\})$ . If T is a tree, its symmetric extension is called a symmetric (or undirected) tree. If G = (V,E) is any graph, its reversal is the graph $G^R = (V, \{(j,i) \mid (i,j) \in E\})$ .

For a graph G = (V,E) an ordering $\alpha$ of V is bijection $\alpha: \{1,2,\ldots,n\} \leftrightarrow V$ . $G_\alpha = (V,E,\alpha)$ is an ordered graph.

## 3. Gaussian Elimination on a Graph.

Let xM = c be a set of n linear equations, where $M = (m_{ij})$ is an n by n non-singular matrix. We can represent the zero -non-zero structure of the matrix M by an ordered graph $G_\alpha = (V,E,\alpha)$ , where $V = \{1,2,\ldots,n\}$ , $E = \{(i,j) \mid m_{ij} \neq 0 \text{ or } i = j\}$ , and a(i) = i for $1 \leq i \leq n$ . The unordered graph G = (V,E) corresponds to the set of matrices $PMP^T$ , where P is a permutation matrix.

We can represent the system $xM = c$ by assigning to vertex $i$ the value $c(i) = -c_i$ and the variable $x(i) = x_i$ and assigning to edge $(i,j)$ the value $m(i,j) = m_{ij}$ if $i \neq j$, $m(i,j) = m_{ij}+1$ if $i = j$. The system $xM = c$ becomes

$$Q = \left\{ \sum_{(i,j)\in E} x(i)m(i,j)+c(i) = x(j) \mid 1 \leq j \leq n \right\} .$$

Henceforth we consider the system of equations defined graph-theoretically in this way. (The variable $x(j)$ appears on the right side of the $j$-th equation for reasons to be discussed later.)

Corresponding to any subgraph $G' = (V',E')$ of $G$ is a system of equations

$$Q' = \left\{ \sum_{(i,j)\in E'} x(i)m(i,j) + c(i) = x(j) \mid j\in V' \right\} .$$

We shall discuss solving the system $Q$ by Gaussian elimination. First, it is useful to consider a way of decomposing $Q$ into subsystems $Q'$ such that the solution to the subsystems gives the solution to the whole system. Let $G_1 = (V_1,E_1), \ldots, G_k = (V_k,E_k)$ be the strongly connected components of the graph $G$. These components can be ordered so that if $(v,w)$ is an edge of $G$ with $v\in V_i$ and $w\in V_j$, then $i \geq j$. Such an ordering is a topological sorting. [26] of the components. Given the components in a topologically sorted order, the following method solves the system $Q$.

```
SOLVE: for i := k step -1 until 1 do begin
       solve the system Q_i;
       for (v,w) ∈ E such that v∈V_i , w∈V_j with j > i do
            c(w) := c(w) + x(v)·m(v,w);
end SOLVE;
```

This scheme is well-known and its validity is easy to check. The strongly connected components of $G$ correspond to the irreducible blocks of the matrix $M$ [43].

We can find the strongly connected components of a graph G and topologically sort them in $O(n+m)$ time using depth-first search [40]. The running time of SOLVE is thus $O(n+m)$ plus the time to solve the substystems $Q_i$ if the graph G is represented as a set of adjacency lists [40]. Reference [ 63 contains a more detailed complexity analysis.

One special case of SOLVE is important. If each strongly connected component of G consists of a single vertex (i.e., G is acyclic except for loops), each subsystem $Q_i$ is a single equation $x(i)a(i) + c(i) = x(i)$ . Solving such an equation requires one subtraction and one division: $x(i) = c(i)[1-a(i)]^{-1}$ . In this case SOLVE requires $O(n+m)$ time total. This special case is the final step, called back solving, of the Gaussian elimination method.

The first step of Gaussian elimination consists of the following algorithm.

```
ELIMINATE: for j := 1 until n-1 do
     for (j,k) ∈ E with k > j do begin
a:   c(k) := c(k) + c(j)·[1-m(j,j)]^-1·m(j,k);
          for (i,j) ∈ E with i > j do begin
               if (i,k) ∉ E then add (i,k)
                    with value m(i,k) = 0 to E;
               m(i,k) := m(i,k) + m(i,j)·[1-m(j,j)]^-1·m(j,k);
end end ELIMINATE;
```

It is well-known and easy to verify that when ELIMINATE terminates, the solution to the original equation set Q can be found by applying SOLVE to the graph $G' = (V,E')$ defined by $E' = \{(i,j) \in E \cup F \mid i \geq j\}$ , where F is the set of added edges $(i,k)$ , called fill-in edges, created by ELIMINATE. The values on edges when ELIMINATE terminates give an LU decomposition of M [14].

ELIMINATE requires $O(n + |E \cup F|)$ numeric storage and

$$O\left(n + \sum_{\substack{(i,j) \in E \cup F \\ i > j}} \left(1 + \sum_{\substack{(j,k) \in E \cup F \\ k > j}} 1\right)\right)$$ arithmetic operations.

SOLVE requires $\Theta\left(n + \sum_{\substack{(i,j) \in E \cup F \\ i \geq j}} 1\right)$ time once ELIMINATE is applied.

Solving the system Q for a new set of constants requires $O(n + |E \cup F|)$ time given the LU decomposition computed by ELIMINATE. For a more detailed complixity analysis, see [6].

Implementing ELIMINATE to achieve the bounds above for total storage and total operation count is not simple. Two methods of implementation suggest themselves.

(1) Representation of $m(i,j)$ using a hash table [25].
(2) Representation of $m(i,j)$ using adjacency lists for G [40].

It is straightforward to implement ELIMINATE using a hash table to store edge values. This representation will achieve the desired storage bound in the worst case and the desired operation bound on the average (but not in the worst case). Because the hash table must be stored, the storage requirements will exceed the storage necessary for adjacency lists, but the average running time is apt to be faster than using adjacency lists.

Careful implementation of ELIMINATE using adjacency lists allows us to achieve the desired storage and time bounds in the worst case. Gustavson [19] discusses many of the ideas important in such an implementation. We use a two-pass method. First, we compute the set F of fill-in edges. An algorithm described in [34] is adequate for this step. Next we use the following modification of ELIMINATE for the LU decomposition. We assume that, for each vertex k , a list $B(k)$ of vertices j such that $(j,k) \in E \cup F$ is available, and that these vertices j are in order by number, smallest to largest, in the list $B(k)$ . Associated with each entry $j \in B(k)$ is the value $m(j,k)$ . The procedure below carries out the computation column-by-column. This method of elimination is sometimes called the Crout method or the Doolittle method [14].

```
CELIMINATE: begin
     for i := 1 until n do array(i) := 0;
     for k := 2 until n do begin
          for j ∈ B(k) do-array(j) := m(j,k);
          for j ∈ B(k) with j < k do begin
               c(k) := c(k)+ c(j)·[1-m(j,j)]^-1·m(j,k);
     b:   for i ∈ B(j) with i > j do
               array(i) := array(i) + m(i,j)·[1-m(j,j)]^-1·m(j,k);
     end;
     for (j,k) ∈ E ∪ F do m(j,k) := array(j);
end end CELIMINATE;
```

Variable array is used here to-make the computation in Step b easy. It is easy to see that this procedure works correctly and achieves the desired storage bound and operation count. The correctness of CELIMINATE depends on the fact that the entries in each list B(k) are in order by number. This representation seems to require that the fill-in F be precomputed.

So far, little is known about the efficiency of using adjacency lists versus using a hash table. Most likely, the hash table method uses less time, and the adjacency list method uses less space. See [8,19,23] for details concerning implementation of Gaussian elimination using adjacency lists.

The time and storage requirements of ELIMINATE depend only on the structure of G and on the ordering $\alpha$ . By reordering the vertices of G , we may greatly improve the efficiency of ELIMINATE. The next section discusses the problem of choosing a good ordering. Because of the complexity of implementing ELIMINATE for sparse graphs, various researchers have studied special methods which handle certain types of sparse graphs. Two such methods, the bandwidth method, and the profile method, are discussed in the next section, in addition to the general sparse method.

Symmetry plays an important role in the solution process. If the matrix M is symmetric (i.e., $m_{ij} = m_{ji}$ ), it is possible to save a factor of two in storage and computing time by using the

symmetry [19]. If the matrix M is structurally symmetric (i.e., G is symmetric), it is much easier to compute the fill-in and other properties of the elimination order [33]. In some applications it may be useful to make G symmetric by adding an edge (j,i) for each edge (i,j) . This may simplify the implementation of ELIMINATE and decrease the time necessary to find a good elimination ordering. These savings must be balanced against the time and storage costs for handling the added edges.

If one of the pivot elements m(j,j) equals one the j-th iteration of the main loop in ELIMINATE cannot be carried out. Furthermore if any of the m(j,j) are close to one, the method is numerically unstable [14]. For certain types of matrices, however, ELIMINATE is guaranteed to work and to be numerically stable. These include the diagonally dominant matrices and the symmetric positive definite matrices [ 143. Henceforth we shall not worry about numeric stability but shall assume that ELIMINATE using any vertex ordering will produce an acceptable answer. In practice, however, it is important to verify stability.

## 4. Elimination Schemes.

One method used to avoid the complexity of implementing ELIMINATE for general sparse graphs is the bandwidth method. If $\alpha$ is an ordering of the vertices of G , we define the bandwidth b of G to be $\max_{(i,j) \in E} |\alpha(i) - \alpha(j)|$. The bandwidth method finds a band of width 2b+1 about the main diagonal outside of which all entries are zero, and performs Gaussian elimination within the band. The bandwidth version of Gaussian elimination appears below.

```
BELIMINATE: for j := 1 until n-1 do
     for k := j+1 until j+b do begin
          c(k) := c(k)+ c(j)·[1-m(j,j)]^-1·m(j,k);
          for i := j+1 until j+b do
               m(i,k) := m(i,k) + m(i,j)·[1-m(j,j)]^-1·m(j,k);
end BELIMINATE;
```

Bandwidth elimination requires O(bn) storage using array storage and $O(b^2n)$ time. The difficulty with the bandwidth method is finding an ordering which produces a small bandwidth. A graph for which there is an ordering such that all edges within the bandwidth are present is called a dense bandwidth graph. It is easy to test in O(n+m) time whether a graph G is a dense bandwidth graph. If it is, the ordering which makes G a dense bandwidth graph is easy to compute.

A graph with an ordering which produces bandwidth one is tridiagonal [14]. (Edges within the bandwidth may be missing, so a tridiagonal graph need not be a dense bandwidth graph.) It is easy to test in O(n+m) time whether a graph is tridiagonal. Garey and Johnson [16] have devised an O(n+m) time method to find a bandwidth two ordering if one exists. We know of no efficient method to test for bandwidth three.

Various heuristics exist for finding orderings with small bandwidth. A breadth-first search method proposed by Cuthill and McKee [11] works well on some examples.

Unfortunately, the problem of determining whether a given graph G has an ordering which produces a bandwidth of a given size b or less belongs to a class of problems called NP-complete. The NP-complete problems have the following properties.

(1) If any NP-complete problem has a polynomial-time algorithm, then all NP-complete problems have polynomial-time algorithms.

(2) If any NP-complete problem has a polynomial-time algorithm, then any problem solvable non-deterministically in polynomial time has a deterministic polynomial-time algorithm.

Such well-studied problems as the travelling salesman problem, the tautology problem of propositional calculus, and the maximum clique problem are NP-complete. It seems unlikely that any NP-complete algorithm has a polynomial-time algorithm. Papadimitriou [29] first proved the minimum bandwidth problem NP complete; Garey and Johnson [16] proved the problem NP complete even for trees! This negative result reduces the appeal of the bandwidth scheme except for problems for which a good choice of ordering is explicit or implicit in the problem description.

An extension of the bandwidth method is the profile method. If $\alpha$ is an ordering of the vertices of G , the profile b(j) of vertex j is $\max\{\alpha(j) - \alpha(i) \mid (i,j) \in E \text{ or } (j,i) \in E \text{ and } a(j) > \alpha(i)\}$ . The profile method assumes that all entries are within an envelope of varying width about the main diagonal. For implementation of the profile method, see [38]. Profile elimination requires $O\left(\sum_{j=1}^{n} b(j)\right)$ storage and $O\left(\sum_{j=1}^{n} b(j)^2\right)$ time. As with the bandwidth method, there is still the problem of finding an ordering with small profile.

A graph G for which there is an ordering such that all edges within the profile are present is called a dense profile graph. That is, G = (V,E) is a dense profile graph if and only if G is symmetric and there is an ordering $\alpha$ of the vertices such that if $(i,j) \in E$ with $\alpha(i) < a(j)$ , and k satisfies $a(i) \leq \alpha(k) \leq \alpha(j)$ , then $(k,j) \in E$ .

There is a nice characterization of dense profile graphs which has apparently not appeared in print before. We call a graph G = (V,E) an interval graph if there is a mapping I of the vertices of G into sets of consecutive integers such that $(i,j) \in E$ if and only if $I(i) \cap I(j) \neq \emptyset$ .

Theorem 1. G is a dense profile graph if and only if G is an interval graph.

Proof. Suppose $G = (V,E)$ is dense profile with appropriate ordering $\alpha$. For each vertex $v \in V$, let $I(v) = \{a(w) \mid (w,v) \in E$ and $\alpha(w) \leq \alpha(v)\}$. By the dense profile property, each set $I(v)$ is a set of consecutive integers. Suppose $(i,j) \in E$ with $a(i) < a(j)$. Then $a(i) \in I(i) \cap I(j)$. Suppose $a(k) \in I(i) \cap I(j)$. Then $(k,i),(k,j) \in E$, $a(k) \leq a(i)$, $a(k) < a(j)$. Without loss of generality, suppose $\alpha(i) \leq \alpha(j)$. Then by the dense profile property, $(i,j) \in E$. Thus the intervals $I(v)$ faithfully represent the edges of $G$.

Conversely, suppose $G$ is an interval graph with appropriate intervals $I(v)$. $G$ is symmetric since $I(i) \cap I(j) = I(j) \cap I(i)$. Let $a(j)$ be an ordering such that $a(j) \leq \alpha(i)$ implies the largest integer in $I@(j))$ is no greater than the largest integer in $I(\alpha(i))$. Let $(i,j) \in E$ with $\alpha(i) < a(j)$ and suppose $\alpha(i) \leq \alpha(k) \leq \alpha(j)$. Then $I(\alpha(i)) \cap I(\alpha(j)) \neq \emptyset$, so $I(\alpha(j))$ contains all integers between the largest integer in $I@(i))$ and the largest integer in $I(\alpha(j))$. This set includes the largest integer in $I(\alpha(k))$. Thus $I(\alpha(j)) \cap I(\alpha(k)) \neq \emptyset$, and $(k,j) \in E$. □

Lueker and Booth [28] have devised an $O(n+m)$-time test for the interval graph property. The test is constructive, so an appropriate ordering for a dense profile graph can be found in $O(n+m)$ time.

The breadth-first search method of Cuthill and McKee produces small profile on some examples. A reverse breadth-first search based on the Cuthill-McKee method does as well or better [27]. Little is known theoretically about the behavior of such heuristics. The problem of finding an ordering to minimize $\sum_{i=1}^{n} b(j)$ (or $\sum_{i=1}^{n} b(j)^2$) has not yet been proved NP-complete. For results on the NP-completeness of a similar problem, see [15]. See [10] for further discussion of bandwidth, profile, and related ordering schemes.

It is easy to generalize the definitions of bandwidth and profile to allow different envelopes on either side of the diagonal. See [10]. In view of the difficulty of finding good orderings for minimizing symmetric bandwidth and profile, we do not pursue this idea further.

Several facts reduce the appeal of the bandwidth and profile schemes except on problems for which a good choice of ordering is explicit or implicit in the problem description. First, it is not easy to find a good ordering. Second, and more important, the bandwidth and profile schemes may be overly pessimistic in that they may examine many matrix elements which are in fact zero. This will happen with sparse graphs having large bandwidth or profile. A practical example is the square k by k grid graph, which arises in finite difference solutions to partial differential equations [14]. Any bandwidth or profile method for this problem requires $O(k^3)$ storage and $O(k^4)$ time [22,30], whereas the nested dissection method [17,36] a special type of general sparse ordering, requires only $O(k^2 \log k)$ storage and $O(k^3)$ time.

We consider now the general sparse method. A graph is a perfect elimination graph if there is an ordering which produces no fill-in. We can test for the perfect elimination property in O(nm) time [34]. This property is computationally at least as hard as testing a directed graph for transitivity, so improving the time bound beyond O(nm) would be a significant result. Given any ordering, we can compute its fill-in in O(nm) time [34]. Such an algorithm is useful if we wish to precompute the fill-in before performing the numeric calculations. Computing the fill-in is at least as hard as computing the transitive closure of a directed graph [34].

The problem of finding an ordering which minimizes the size of the fill-in is NP-complete [34]. However, a related problem has a polynomial time algorithm. We call a set of fill-in edges F minimal if no ordering produces a fill-in $F' \subset F$. If $\alpha$ is an ordering which produces fill-in F, $\alpha$ is a minimal ordering. Minimal orderings are not necessarily close to minimum, but given any ordering we can improve it to a minimal one in $O(n^4)$ time [34].

These problems are easier for symmetric graphs. We can test a symmetric graph for the perfect elimination property in $O(n+m)$ time, compute the fill-in of any ordering in $O(n+m)$ time, and find a minimal ordering in $O(nm)$ time [35]. These algorithms, especially the one to compute fill-in, may have important practical uses.

In view of the NP-completeness results, we cannot hope to solve the general problem of efficiently implementing sparse Gaussian elimination. We can only try to solve the problem for special cases. Approaches include the following.

(1) Develop and study heuristics for producing orderings with small fill-in. Several heuristics have been proposed, including the minimum degree and minimum fill-in heuristics [31,32]. These methods seem to work well in practice, but nothing is known about their theoretical behavior.

(2) Develop good ordering schemes for special types of graphs. A successful example of this approach is the nested dissection method [17,36].

(3) Develop methods which avoid the necessity of computing all the fill-in. In some cases values on fill-in edges can be stored implicitly rather than explicitly, resulting in a savings of time and storage.

We consider in the next section a method which combines ideas (3) and (4).

Another possible approach would be to study the average behavior of elimination methods. This approach is not a good one, however, for two reasons. (1) Most graphs which occur in practical problems are highly non-random in their structure. (2) Erdös and Even [13] have shown that "most" symmetric graphs with order n log n edges have a fill-in of order $n^2$ (most graphs with less than order n log n edges are not connected). Thus a dense matrix method is as good (to within a, constant factor) as any sparse method, on random graphs which are not too sparse.

## 5. A Decomposition Method Using Dominators.

This section presents a decomposition method for solving systems of linear equations which is more powerful than the decomposition into strongly connected components discussed in Section 3. The idea of the method is as follows. Suppose G = (V,E) is a directed graph and there exists a triple u, v, w of distinct vertices such that v

dominates w with respect to u . We can partition V into $V = \{v\} \cup V_1 \cup V_2$ such that $V_1$ contains u and all vertices reachable by a path from u which avoids v . Let $G_1 = (\{v\} \cup V_1, E(\{v\} \cup V_1))$ , $G_2 = (\{v\} \cup V_2, E(\{v\} \cup V_2))$. Suppose we are given a set of equations defined on G . We solve the set by the following method.

Step 1: For each vertex $w \in V_2$ , solve for x(w) in terms of x(v) using the system of equations defined on $G_2$ . That is, represent x(w) as $x(w) = x(v) \cdot a(v,w) + b(v,w)$ for some real values $a(v,w)$ , $b(v,w)$ .

Step 2: Replace each edge (x,y) with $x \in V_2$ , $y \in \{v\} \cup V_1$ , by an edge (v,y) with value $m(v,y) = 0$ , if such an edge does not exist already. Set $m(v,y) := a(v,x) \cdot m(x,y) + m(v,y)$ . Set $c(y) := b(v,x) \cdot m(x,y) + c(y)$ .

Step 3: In the new graph G' , solve the system of equations defined on $G_1'$ .

Step 4: using the equations found in Step 1, solve for the values of the variables x(w) , $w \in V_2$ .

This method solves the system of equations defined on graph G by solving the two smaller systems defined on $G_2$ and $G_1'$ and combining the solutions. It is equivalent to carrying out Gaussian elimination on G in an order so that all the vertices in $V_2$ are ordered first, followed by vertex v , followed by all the vertices in $V_1$ . For each edge (x,y) with $x \in V_2$ , $Y \in \{v\} \cup V_1$ , this elimination order may create a large number of fill-in edges (x',y) with $x' \in V_2$. None of these fill-in edges are really necessary to the computation; only the corresponding fill-in edge (v,y) is necessary. By computing the value of this edge directly, we avoid computing many of the fill-in edges and thus save time and storage space.

We generalize this scheme as follows. Henceforth we assume G = (V,E) is **Strongly** connected. Let r be some fixed, distinguished vertex of G . If v dominates w with respect to r and no vertex

dominates w with respect to v , we say v is the immediate dominator of w (with respect to r ). We denote this relationship by $v = \underline{idom}(w)$ .

Theorem 2 [ 1 ]. Each vertex $w \neq r$ has a unique immediate dominator. The rooted tree $T = (V, \{(\underline{idom}(v),v) \mid v \neq r\})$ , called the dominator tree of G , has the property that, for every vertex w , its dominators with respect to r are exactly its ancestors in T .

Our solution method works as follows.

Step 1: Choose a fixed vertex r of G . Compute the corresponding dominator tree T .

Step 2: Working from the leaves of T to the root, solve for each variable x(v) in terms of $x(\underline{idom}(v))$ .

Step 3: Solve for x(r) and for all other variables x(v) by backsolving using the equations computed in Step 2.

Step 2 will compute, for each variable x(v) , a pair of numbers a(v) and b(v) such that $x(v) = x(\underline{idom}(v))\cdot a(v) + b(v)$ . As we work through the tree in Step 2, we must compose such affine functions. We will assume the existence of two primitive instructions for this purpose. Given two ordered pairs (a,b) and (c,d) , let $(a,b)\cdot(c,d) = (ac , bc+d)$ (this operation corresponds to forming the composition of the affine functions $ax+b$ and $cy+d$ .

The two operations will construct T and place ordered pairs of real numbers on its edges. Initially T has no edges constructed. The operation LINK($\underline{idom}(v)$ , v , (a,b)) adds the edge $(\underline{idom}(v),v)$ , with associated value $c(\underline{idom}(v),v) = (a,b)$ to T . The operation EVAL(v) returns the ordered triple (u,x,y) such that $(x, y) = c(e_1)\cdot c(e_2)\cdot\ .\ .\ .\ \cdot\ c(e_k)$ , where $e_1,e_2,\ldots,e_k$ is the longest path to vertex v in the part of T so far constructed by LINK instructions, and this path starts at vertex u . (If v has no entering edge yet constructed, EVAL(v) returns the triple (v,1,0) ; the pair (1,0) corresponds to the identity function.)

Now we give the details of the algorithm.

Step 1: Choose a fixed vertex r of G . Compute the corresponding dominator tree T of G . Number the vertices of T from 1 to n in postorder. For each vertex v , let s(v) be the set of children of v in T .

Step 2:
```
for u := 1 until n do begin
    initialize E(u) = ∅;
    for v∈ s(u) do
        for each edge (w,v) of G do begin
            (z,a,b) := EVAL(w);
            if (z,v) is not an edge of E(u) then
                add (z,v) with value m(z,v) = m(w,v)·a to E(w)
            else m(z,v) := m(w,v)·a + m(z,v);
            c(v) := m(w,v)·b + c(v);
        end;
    find the strongly connected components of the graph
        G(u) = ({v} ∪ s(u), E(u)) and topologically sort them;
    solve the system of equations
        Q(u) = { Σ_{(w,v)∈E(u)} m(w,v)·x(w) + c(v) = x(v) | v∈s(u) }
        to give an equation x(v) = a(v)·x(u) + b(v) for
        each v∈s(u) , by using Gaussian elimination and
        the strongly connected components decomposition
        as discussed in Section 3;
    for v∈s(u) do LINK(u , v, (a(v),b(v)));
```

Step 3:
```
for each edge (w,n) of G do begin
    (z,a,b) := EVAL(w);
    m(n,n) := m(w,n)·a + m(n,n);
    c(n) := m(w,n)·b + c(n);
    x(n) := c(n)·[1-m(n,n)]^-1;
end;
for i := n-1 step -1 until1 do
    x(i) := x(idom(i))·a(i) + c(i);
```

This method uses Gaussian elimination on the strongly connected components of the graphs G(u) and combines the solutions to give the solution to the entire problem. The time to combine solutions is almost-linear in the size of G ; thus if the method breaks the graph into several parts it is certainly faster than Gaussian elimination applied to the whole graph.

More precisely, the running time of Step 1 is $O(m\ \alpha(m,n))$ [41], where $\alpha(m,n)$ is a very slowly growing function related to a functional inverse of Ackermann's function. Step 1 requires O(m) storage. Step 2 requires $O(m\ \alpha(m,n))$ time and O(m) storage for the LINK and EVAL instructions [41]. Step 2 requires O(m) time and storage except for the Gaussian elimination steps and the LINK and EVAL instructions. Step 3 requires O(n) time and storage. Thus the entire algorithm requires $O(m\ \alpha(m,n))$ time and O(m) storage exclusive of the Gaussian elimination steps.

If each strongly connected component of every graph G(u) consists of a single vertex, then the algorithm runs in $O(m\ \alpha(m,n))$ time total. A graph G for which this happens is called a reducible graph [29] (not to be confused with a reducible matrix). Though reducible graphs do not seem to arise in numerical problems, they often arise in global optimization of computer code, to which the ideas in this paper also apply. Thus this decomposition method may have considerable practical value. Indeed, similar methods for reducible graphs have been extensively studied by computer scientists [ 2, 9, 18,21,24,42 ] .

If no root r can be found for which G breaks into several pieces using this decomposition scheme, the same idea can be applied to the reverse of G . The algorithm must be changed somewhat, but the idea is similar. In fact, a more general algorithm which divides G into strongly biconnected components and solves a set of equations on each component can be developed. The trouble with such an algorithm is that at present no efficient method exists for dividing a graph into strongly biconnected components. Research is in progress in this area.

## 6. Selection of a Set of Pivot Positions.

When considering orderings for Gaussian elimination in Section 4, we restricted our attention to simultaneous row and column permutations, represented by a renumbering of vertices in the graph representing the system of equations. Thus we always used the positions on the main diagonal as pivot positions. In numeric problems, there is no reason to restrict our attention to such reorderings, however. We can easily allow independent row and column permutations, and thus use an arbitrary transversal of the matrix as a set of pivot positions (a matrix transversal is a set of n matrix elements, no two in the same row or column).

There are two reasons for selecting a transversal other than the main diagonal.

(1) To improve the stability of Gaussian elimination.
(2) To improve the resource requirements of Gaussian elimination.

The well-known partial and complete pivoting methods [14] choose a transversal to improve stability. They choose a set of matrix elements of large absolute value as pivots. These methods depend on the actual numeric entries and not on the zero -non-zero structure of the matrix.

If we do not know the actual entries of the matrix, but only its zero - non-zero structure, then any transversal consisting of non-zero elements is as good as any other for purposes of stability. Such a transversal may be found in $O(n^{1/2} m)$ time by using a bipartite matching algorithm of Hopcroft and Karp [44]. Dulmage and Mendelsohn [12] extensively discuss this and related problems. Essentially no research has been done on the problem of picking a non-zero transversal which minimizes resource requirements. One theorem is known however.

Theorem 2. Let M be any matrix. Let Q be any permutation matrix such that MQ has a non-zero main diagonal. Let G(Q) be the directed graph corresponding to MQ . Then the vertex partition induced by the strongly connected components of G(Q) is independent of **Q** .

This theorem follows from results of Dulmage and Mendelsohn [12]. Howell has given a nice proof [45]. The theorem implies that the strong component decomposition method discussed in Section 3 produces the same number of components independent of the transversal chosen, though the components themselves may be different.

Ignoring questions of stability, there is no reason not to choose a transversal some of whose elements are initially zero and only become non-zero as the elimination proceeds. Such a choice may result in substantial computational savings. Bank and Rose [4] have provided a practical example of this idea. Though their method is numerically unstable, it can be modified to make it stable without degrading its efficiency too much [5].

In summary, the problem of choosing the best set of pivot positions, for stability or efficiency or both, is very poorly understood. The results of Bank and Rose indicate that allowing only transversals which are initially non-zero is too restrictive. It is likely that the problem is too hard for a general solution, and the most promising areas for research seem to be the development of heuristics and special-case algorithms.

## 7. Remarks.

Though we have assumed throughout this discussion that the matrix M consists of numbers, there is no reason to do so. The techniques of linear algebra, such as Gaussian elimination, apply to other algebraic structures having two operations + and ·. Thus the methods discussed in this paper can be used to compute path sets in labelled graphs [3,37] (a problem of automata theory), find shortest paths and other kinds of optimal paths in directed graphs [7], and to do global flow analysis of computer code [2,9,18,24,42]. The algorithms remain the the same; only the interpretation changes.

We must assume the existence, for any $a$, of an element $a^*$ such that, for all $b$, $a^* \cdot b$ is a solution to the equation $x = a \cdot x + b$. For numbers, $a^* = [1-a]^{-1}$ exists whenever $a \neq 1$, and Gaussian elimination requires non-unit pivots.

References


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

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

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

[4] R. E. Bank and D. J. Rose, "An $O(n^2)$ method for solving constant coefficient boundary value problems in two dimensions," SIAM J. Numer. Anal., to appear.

[5] R. E. Bank and D. J. Rose, "Marching algorithms for elliptic boundary value problems I: the constant coefficient case," SIAM J. Numer. Anal., submitted.

[6] J. R. Bunch and D. J. Rose, "Partitioning, tearing, and modification of sparse linear systems," J. Math. Anal. Appl., 48 (1974), [illegible].

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

[8] A. Chang, "Application of sparse matrix methods in electric power systems," Sparse Matrix Proceedings, R. A. Willoughby, ed., IBM Research, Yorktown Heights, N.Y. (1968), 113-122.

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

[10] E. Cuthill, 'Several strategies for reducing the bandwidth of matrices," Sparse Matrices and Their Applications, D. Rose and R. Willoughby, eds., Plenum Press, N. Y. (1972), 157-166.

[11] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," Proc. ACM National Conference (1969), 157-172.

[12] A. Dulmage and N. Mendelsohn, "Graphs and Matrices," **Graph** Theory and Theoretical Physics, F. Harary, ed., Academic Press, N. Y. (1967), 161-227.

D-31 S. Even, private communication (1974).

V-41 G. E. Forsythe and C. B. Moler, Computer Solution of Linear Algebraic Equations, Prentice-Hall, Englewood Cliffs, N.J. (1967).

[15] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-Complete problems,'* Proc. Sixth Annual ACM Symp. on Theory of Computing (1974), 47-63.

[16] M. R. Garey and D. S. Johnson., arivate communication (1975).

[17] J. A. George, "Nested dissection of a regular finite element mesh," SIAM J. Numer. Anal., 10 (1973), 345-363.

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

[19] F. G. Gustavson, "Some basic techniques for solving sparse systems of linear equations," Sparse Matrices and Their Applications D. Rose and R. Willoughby, eds., Plenum Press, N.Y. (1972), 41-52.

[20] F. G. Gustavson, W. Liniger, and R. Willoughby, "Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations," J. ACM, 17 (1970), 87-109.

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

[22] A. J. Hoffman, M. S. Martin, and D. J. Rose, "Complexity bounds for regular finite difference and finite element grids," SIAM Journal of Numerical Analysis, 9 (1961), 364-369.

[23] A. Jennings, "A compact storage scheme for the solution of symmetric linear simultaneous equations," Comput. J., 9 (1966), 281-285.

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

[25] D. Knuth, The Art of Computer Programming, vol. 3: Sorting and Searching, Addison-Wesley, Reading, Mass. (1973), 506-549.

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

[27] W. H. Liu and A. H. Sherman, "Comparative analysis of the Cuthill-McKee and reverse Cuthill-McKee ordering algorithms for sparse matrices," SIAM J. Numer. Anal., to appear.

[28] G. S. Lueker and K. S. Booth, "Linear algorithms to recognize interval graphs and test for the consecutive ones property," Proc. Seventh Annual ACM Symp. on Theory of Computing (1975), 255-265.

[29] C. H Papadimitriou, "The NP-completeness of the bandwidth minimization problem" Computing, to appear.

[30] D. J. Rose, private communication (1975).

[31] D. J. Rose, "Triangulated graphs and the elimination process," J. Math. Anal. Appl., 32 (1970), 597-609.

[32] D. J. 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.

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

[34] D. J. Rose and R. E. Tarjan, "Algorithmic aspects of vertex elimination on directed graphs," to appear.

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

[36] D. J. Rose and G. F. Whitten, "Automatic nested dissection," Proc. ACM Conference (1974), 82-88.

[37] A. Salomaa, Theory of Automata, Pergamon Press, Oxford, England (1969), 120-123.

[38] A. H. Sherman, "Subroutines for envelope solution of sparse linear equations," Research Report No. 35, Dept. of Computer Science, Yale University (1974).

[39] A. H. Sherman, Ph.D. thesis, Yale University (1975).

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

[41] R. E. Tarjan, "Applications of path compression on balanced trees," to appear.

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

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

[44] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matching in bipartite graphs," SIAM J. Comput. 2 (1973), 225-231.

[45] T. D. Howell, "Partitioning using PAQ," to appear in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York.