ALGORITHMIC ASPECTS OF VERTEX ELIMINATION ON GRAPHS

by

Donald J. Rose and R. Endre Tarjan

Memorandum No. ERL-M483

November 1974

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ALGORITHMIC ASPECTS OF VERTEX ELIMINATION ON GRAPHS[†]

Donald J. Rose

Applied Mathematics, Aiken Computation Laboratory
Harvard University
Cambridge, Massachusetts

and

R. Endre Tarjan

Computer Science Division
Department of Electrical Engineering and Computer Sciences
and Electronics Research Laboratory
University of California, Berkeley

Abstract

We consider a graph-theoretic elimination process which is related to performing Gaussian elimination on sparse symmetric positive definite systems of linear equations. We give a new linear-time algorithm to calculate the fill-in produced by any elimination ordering, and we give two new related algorithms for finding orderings with small fill-in. One algorithm, based on breadth-first search, finds a perfect elimination ordering, if any exists, in $O(n+e)$ time, if the problem graph has n vertices and e edges. An extension of this algorithm finds a minimal (but not necessarily minimum) ordering in $O(ne)$ time. We conjecture that the problem of finding a minimum ordering is NP-complete.

Keywords: graph, breadth-first search, lexicographic search, Gaussian elimination, sparse linear equations, perfect elimination, triangulated graph, chordal graph

---

[†]Research partially sponsored by the Office of Naval Research under contract N00014-67-A-0298-0034 at Harvard University, and by the National Science Foundation, Grant GJ-35604X, and a Miller Research Fellowship at University of California, Berkeley. Part of this work was completed while the second author was visiting the Weizmann Institute of Science, Rehovot, Israel.

1. Introduction and Notation

A graph is a pair $G = (V,E)$ where V is a finite set of $n = |V|$ elements called vertices and

$$E \subseteq \{\{v,w\} \mid v, w \in V, v \neq w\}$$

is a set of $e = |E|$ unordered vertex pairs called edges. Given $v \in V$, the set

$$adj(v) = \{w \in V \mid \{v,w\} \in E\}$$

is the set of vertices adjacent to v. The notation $v \text{---} w$ means $w \in adj(v)$; $v \not\text{---} w$ means $w \notin adj(v)$. If $A \subseteq V$, the induced subgraph $G(A)$ of G is the subgraph $G(A) = (A,E(A))$ where $E(A) = \{\{x,y\} \in E \mid x, y \in A\}$.

For distinct vertices $v, w \in V$, a v,w chain of length k is a sequence of distinct vertices $\mu = [v = v_1, v_2, \ldots, v_{k+1} = w]$ such that $v_{i+1} \text{---} v_i$ for $i = 1,2,\ldots,k$. Similarly, a cycle of length k is a sequence of distinct vertices $\mu = [v_1, v_2, \ldots, v_k]$ such that $v_{i+1} \text{---} v_i$ for $i = 1,2,\ldots,k-1$ and $v_k \text{---} v_1$. We will always assume that G is connected; that is, for each pair of distinct vertices $v, w \in V$, there is a v,w chain.

For a graph $G = (V,E)$ with $|V| = n$, an ordering of V is a bijection $\alpha: \{1,2,\ldots,n\} \leftrightarrow V$. Sometimes we denote an ordering by $V = \{x_i\}_{i=1}^{n}$. $G_\alpha = (V,E,\alpha)$ is an ordered graph. In G_α, the set of vertices monotonely adjacent to a vertex v is defined by

$$madj(v) = adj(v) \cap \{w \in V \mid \alpha^{-1}(v) < \alpha^{-1}(w)\} \quad .$$

The notation $v \rightarrow w$ means $w \in madj(v)$.

For a vertex v, the deficiency $D(v)$ is the set of edges defined by

$$D(v) = \{\{x,y\} \mid v\text{—}x,\ v\text{—}y,\ x \not\text{—} y,\ x \neq y\} \quad .$$

The graph G_v obtained from G by

(1) adding edges so that all vertices in $adj(v)$ are pairwise adjacent, and

(2) deleting v and its incident edges

is the v-elimination graph of G. That is,

$$G_v = (V-\{v\},\ E(V-\{v\}) \cup D(v)) \quad .$$

For an ordered graph $G_\alpha = (V,E,\alpha)$, the elimination process

$$P(G_\alpha) = [G = G_0, G_1, G_2, \ldots, G_{n-1}]$$

is the sequence of elimination graphs defined recursively by $G_0 = G$, $G_i = (G_{i-1})_{x_i}$ for $i = 1,2,\ldots,n-1$. If $G_i = (V_i,E_i)$ for $i = 0,1,\ldots,n-1$, the fill-in $F(G_\alpha)$ is defined by

$$F(G_\alpha) = \bigcup_{i=1}^{n-1} \tau_i$$

where $\tau_i = D(x_i)$ in G_{i-1}, and the elimination graph G_α^* is defined by

$$G_\alpha^* = (V,\ E \cup F(G_\alpha)) \quad .$$

The notion of vertex elimination arises in the solution of sparse symmetric positive definite systems of linear equations by Gaussian elimination. Given any symmetric $n \times n$ matrix $M = (m_{ij})$ which

represents the coefficients of a set of linear equations, we can construct an ordered graph $G_\alpha = (V,E,\alpha)$, where vertex x_i corresponds to row i (and variable i), and $\{x_i,x_j\} \in E$ if and only if $m_{ij} \neq 0$ and $i \neq j$. The unordered graph $G = (V,E)$ corresponds to the equivalence class of matrices PMP^T, where P is any permutation matrix.

Suppose we solve the system with coefficients M using Gaussian elimination, eliminating variables in the order $1,2,\ldots,n$. Assuming no lucky cancellation of non-zero elements, the edges τ_i correspond exactly to the new non-zero elements created when row i is eliminated. For further discussion of this correspondence, see [19,21]. In order to make the elimination process efficient, we would like to create no more non-zero elements than necessary; that is, we would like to find an elimination ordering which minimizes the fill-in.

Given a graph $G = (V,E)$, an ordering α of V is a perfect elimination ordering of G if $F(G_\alpha) = \phi$. Thus α is a perfect elimination ordering if $v \rightarrow w$ and $v \rightarrow x$ in G_α imply $w \text{---} x$ or $w = x$. A graph which has a perfect elimination ordering is a perfect elimination graph. An ordering α is a minimal elimination ordering of G if no other ordering β satisfies $F(G_\beta) \subset F(G_\alpha)$ where the containment is proper. An ordering α is a minimum elimination ordering of G if no other ordering β satisfies $|F(G_\beta)| < |F(G_\alpha)|$.

Any elimination graph G_α^* is a perfect elimination graph, since α is a perfect ordering of this graph. Any perfect ordering of a graph is minimum, and any minimum ordering of a graph is minimal. If a graph is a perfect elimination graph, any minimal ordering is perfect.

The problem we would like to solve is that of finding a minimum elimination ordering for any graph G. However, this seems to be a very

difficult task in general; we conjecture that the problem of finding a minimum elimination order is NP-complete.† Thus we restrict our attention to finding a minimal ordering for any graph and finding a perfect ordering for any perfect elimination graph.

Perfect elimination graphs arise in contexts other than Gaussian elimination. Rose [19,21] has given several characterizations of perfect elimination graphs, including the one below.

A graph G is called <u>triangulated</u> if for every cycle $\mu = [v_1, v_2, \ldots, v_k]$ of length $k > 3$, there is an edge of G joining two nonconsecutive vertices of μ. Such an edge is called a <u>chord</u> of the cycle. For an arbitrary graph $G = (V,E)$ a set of edges F is a <u>triangulation</u> if $G' = (V, E \cup F)$ is triangulated. F is a <u>minimal triangulation</u> if $G_o = (V, E \cup F_o)$ is not triangulated for any $F_o \subset F$.

<u>Theorem A</u> (Rose [19,21], Dirac [5]): A graph G is a perfect elimination graph if and only if it is triangulated.

Triangulated or perfect elimination graphs have also been called <u>chordal</u> [8], <u>monotone transitive</u> [21], and <u>rigid circuit graphs</u> [5]. Gavril [7] has presented efficient algorithms for finding all cliques, maximum cliques, minimum colorings, maximum independent sets, and minimum clique coverings in triangulated graphs (for arbitrary graphs, these problems are NP-complete). These algorithms depend upon exploiting the necessary perfect elimination ordering. Assuming that such an ordering is given, it is easy to implement Gavril's algorithms to run in $O(n+e)$ time. ($O(n+e)$ is optimum to within a constant factor; the time bounds he gives are not tight.) Several

†The NP-complete problems, roughly speaking, are the hardest problems solvable using <u>non-deterministic</u> polynomial-time algorithms. Either <u>all</u> the NP-complete problems have deterministic polynomial-time algorithms or none of them do. Many people have tried and failed to find polynomial-time algorithms for problems in this class, but no one has proved that such algorithms do not exist. The tautology problem of propositional calculus, the travelling salesman problem, the maximum clique problem, and many other well-known problems are NP-complete. See [4,14] for further information.

important classes of graphs, such as trees, k-trees [20], and interval graphs [6,10], are triangulated, and a recognition algorithm for triangulated graphs can be used to recognize interval graphs efficiently [2,6,10].

The recognition problem for perfect elimination graphs bears a superficial resemblance to the problem of testing a directed graph for transitivity. It is easy to construct an $O(ne)$ algorithm to find a perfect ordering of a graph, if one exists. Gavril [8] has developed a way to find a perfect ordering of a graph G in $O(t(n,e)+n+e)$ time, where $t(n,e)$ is the time required to square the adjacency matrix of G. If Strassen's method of matrix multiplication [24] is used, this algorithm has an $O(n^{2.81})$ time bound. Below we present an $O(n+e)$ algorithm for finding perfect orderings. The algorithm uses a lexicographic search (or ordering scheme) which is a special type of breadth-first search. Surprisingly, a similar ordering scheme is useful in solving certain scheduling problems [3,13,23].

Ohtsuki, Cheung, and Fujisawa [17] have extended Rose's results in order to develop algorithms for finding minimal orderings. Given a graph $G = (V,E)$, let V^* be the set of vertices such that $v \in V^*$ if and only if, for each $\{x,y\} \in D(v)$, there is a chain from x to y which contains no vertices in $\{v\} \cup adj(v) - \{x,y\}$.

Theorem B [17]: An ordering α on G is a minimal elimination ordering if and only if $\alpha(i) \in V^*_{i-1}$, where $P(G_\alpha) = [G_0,G_1,G_2,\ldots,G_{n-1}]$ and $G_i = (V_i,E_i)$ for $i = 0,1,\ldots,n-1$.

This theorem leads to an algorithm for finding minimal orderings. Ohtsuki [18] has refined this method to get an $O(ne)$ algorithm for finding minimal orderings. The lexicographic ordering we consider here gives a significantly different $O(ne)$ algorithm for finding minimal

orderings, and our analysis of the properties of a lexicographic search leads to a characterization of minimal triangulations in terms of the cycle structure of the minimal triangulated graph.

We shall first study the more general problem of finding minimal orderings and then streamline our algorithm to solve the easier problem of finding perfect orderings. In §2 we derive some properties of minimal and perfect orderings. In §3 we motivate the idea of a lexicographic search by considering the relationship between breadth-first searches and perfect orderings. In §4 and §5 we consider in detail the analysis and implementation of lexicographic orderings, and in §6 we present some additional remarks. Although our results deal mainly with the application of lexicographic search to produce minimal and perfect elimination orderings, we feel the notion of a lexicographic search is algorithmically interesting and may have wider application.

2. Properties of Elimination Orderings and Fill-In

If we are to develop algorithms to find good ordering schemes, we must know some properties of elimination orderings and of the fill-in they produce.

Lemma 1: Let α be a perfect elimination ordering of a triangulated graph G. Let $x \in V$. Then α is also a perfect ordering of $G' = (V, E \cup D(x))$.

Proof: We must show that for any $\{w,y\}, \{z,y\} \in E \cup D(x)$ with $\alpha^{-1}(y) < \min(\alpha^{-1}(w),\alpha^{-1}(z))$ and $w \neq z$, we have $\{w,z\} \in E \cup D(x)$. There are three cases. If $\{w,y\}, \{z,y\} \in E$, then $\{w,z\} \in E$ since α is perfect. If

$\{w,y\}$, $\{z,y\} \in D(x)$, then $w, z \in adj(x)$ and $\{w,z\} \in E \cup D(x)$. The last case is $\{w,y\} \in E$, $\{z,y\} \in D(x)$ (or equivalently $\{w,y\} \in D(x)$, $\{z,y\} \in E$). This means $y, z \in adj(x)$ and $\{y,z\} \notin E$. If $w = x$, $z \in adj(x)$ means $\{w,z\} \in E$. Otherwise (i.e. if $w \neq x$), $\alpha^{-1}(x) > \alpha^{-1}(y)$ since α is perfect, and $\{x,w\} \in E$ since $x, w \in adj(y)$ and α is perfect. But $w, z \in adj(x)$ imply $\{w,z\} \in E \cup D(x)$. □

Corollary 1: If $G = (V,E)$ is triangulated and x is any vertex, the elimination graph $G_x = (V-\{x\}, E(V-\{x\}) \cup D(x))$ is triangulated. (This Corollary is also proved in [21].)

Corollary 2: If $G = (V,E)$ is triangulated and x is any vertex with $D(x) = \phi$, there is a perfect elimination ordering α with $\alpha(1) = x$.

Lemma 2: Let $G = (V,E)$ be a triangulated graph. Suppose $G' = (V, E \cup F)$ with $F \neq \phi$, $E \cap F = \phi$ is also triangulated. Then there exists some $f \in F$ such that $G' - f = (V, E \cup F-\{f\})$ is triangulated.

Proof: We prove the theorem by induction on $n = |V|$. If $n \leq 3$, the result is obvious since any graph with three or fewer vertices is triangulated. Suppose the result is true for $n \leq n_o$ and let $n = n_o + 1$. Let $R = \{x \mid D(x) = \phi\}$ where $D(x)$ is the deficiency in G. Let $S = \{x \mid D'(x) = \phi\}$ where $D'(x)$ is the deficiency in G'. We know $R \neq \phi$ and $S \neq \phi$. There are two cases.

(i) For some $x \in S$ there exists an edge $f = (u,x) \in F$. By Corollary 2 there is a perfect elimination order β for G' with $\beta(1) = x$. Then β is also a perfect elimination order for $G' - f$.

(ii) Case (i) does not hold. We prove that there exists some $x \in S$ with $F \not\subseteq D(x)$. Pick any $z \in S$. If $F \not\subseteq D(z)$, let $x = z$. Otherwise,

since $D(z) \subseteq E \cup F$, $E \cup F = E \cup D(z)$. In this case let x be any vertex such that $x \in R$. By Corollary 2, there is a perfect ordering α of G such that $\alpha(1) = x$, and by Lemma 1, α is a perfect ordering of G'. Thus $x \in S$. Since $D(x) = \phi$, $F \not\subseteq D(x)$.

Now $G_x = (V-\{x\}, E(V-\{x\}) \cup D(x))$ and $G'_x = (V-\{x\}, E(V-\{x\}) \cup F \cup D(x))$ are triangulated by Corollary 1. By the induction hypothesis there exists $f \in F - D(x)$ such that $G'_x - f$ is triangulated. But then $G' - f$ is triangulated since $f \notin D(x)$. □

Theorem 1: Let $G = (V,E)$ be a graph and let $G' = (V, E \cup F)$ be triangulated with $E \cap F = \phi$. F is a minimal triangulation if and only if for each $f \in F$, $G' - f = (V, E \cup F - \{f\})$ is not triangulated.

Proof: One direction is immediate from the definition of minimal triangulation; Lemma 2 gives the other direction. □

Theorem 2: Let $G = (V,E)$ be a graph and $G' = (V, E \cup F)$ be triangulated. Then F is a minimal triangulation if and only if each $f \in F$ is a unique chord of a 4-cycle in G'.

Proof: If F is minimal and $f \in F$, $G' - f$ is not triangulated and hence contains an unchorded cycle, say μ of length $\ell \geq 4$. But if μ has length $\ell > 4$, a proper subset of μ is an unchorded cycle in G' of length $\ell \geq 4$. Thus $G' - f$ must contain an unchorded cycle of length 4, and f is the unique chord of this cycle in G'.

The converse is immediate from Theorem 1 since $G' - f$ has an unchorded 4-cycle for any $f \in F$. □

Theorems 1 and 2 provide two useful characterizations of minimal

triangulations (and of minimal orderings, since α is a minimal ordering if and only if $F(G_\alpha)$ is a minimal triangulation). In §4 we prove that the lexicographic ordering scheme defined there produces an ordering whose fill-in satisfies the unique chord property of Theorem 2. Thus any lexicographic ordering is minimal.

Lemma 3: Let $G_\alpha = (V,E,\alpha)$ be an ordered graph. Then $\{v,w\}$ is an edge of $G_\alpha^* = (V, E \cup F(G_\alpha))$ if and only if there exists a chain $\mu = [v = v_1, v_2, \ldots, v_{k+1} = w]$ in G_α such that

$$\alpha^{-1}(v_i) < \min(\alpha^{-1}(v), \alpha^{-1}(w)) \ , \quad 2 \leq i \leq k \quad .^{\dagger} \qquad (P)$$

Proof: We show by induction on $\ell = \min(\alpha^{-1}(v), \alpha^{-1}(w))$ that, given any edge $\{v,w\}$ in G_α^*, there exists a v,w chain with the required property (P). Suppose $\ell = 1$. Then v—w in G_α, hence in G_α^*, and (P) holds vacuously. Suppose the result holds when $\ell \leq \ell_o$ and consider the case $\ell = \ell_o + 1$. If v—w in G_α then again (P) holds vacuously. Otherwise $\{v,w\} \in F(G_\alpha)$ and we have by the definition of $F(G_\alpha)$ an $x \in V$ with $\alpha^{-1}(x) < \min(\alpha^{-1}(v), \alpha^{-1}(w))$ and x—v, x—w in G_α^*. The induction hypothesis implies the existence of x, v and x, w chains in G_α satisfying (P) and combining these chains gives the required v, w chain.

The converse is established by induction on k, the length of μ. If $k = 1$, v—w in G_α^* trivially. Suppose the converse holds for $k \leq k_o$ and consider the case $k = k_o + 1$. Let $\mu = [v = v_1, v_2, \ldots, v_{k+1} = w]$ and choose $x = v_i$ where $\alpha^{-1}(v_i) = \max\{\alpha^{-1}(v_j) \mid 2 \leq j \leq k\}$. The induction hypothesis implies that v—x and x—w in G_α^*; hence v—w in G_α^*. □

†For convenience, although at the risk of possible confusion, we adopt the following convention: In the vacuous case where v—w and hence $\{v_2, \ldots, v_k\} = \phi$, the condition is regarded as satisfied. Similar conventions are followed below.

This lemma provides a characterization of the fill-in produced by any elimination ordering. It is useful to have an efficient algorithm for calculating the fill-in. We derive another characterization for this purpose.

Lemma 4: Let $G_\alpha = (V,E,\alpha)$ be an ordered graph. Then $E \cup F(G_\alpha)$ is the smallest set E^* of edges such that $E \subseteq E^*$ and

$$v \to w \text{ in } E^* \text{ implies } m(v) = w \text{ or } m(v) \to w \text{ in } E^* \qquad (Q)$$

where $m(v)$ is the vertex u with minimum $\alpha^{-1}(u)$ such that $v \to u$ in E^*.

Proof: $E^* = E \cup F(G_\alpha)$ clearly satisfies $E \subseteq E^*$ and (Q). We must prove that any set E^* satisfying $E \subseteq E^*$ and (Q) contains $E \cup F(G_\alpha)$. Suppose $(v,w) \in E \cup F(G_\alpha)$. We prove by induction on $i = \min(\alpha^{-1}(v),\alpha^{-1}(w))$ that $(v,w) \in E^*$. Suppose this result holds for $i \leq i_o$ and consider the case $i = i_o + 1$. If $(v,w) \in E$, then $(v,w) \in E^*$. If $(v,w) \in F(G_\alpha)$, there is some u with $u \to v$, $u \to w$ in G_α^*. By the induction hypothesis $u \to v$, $u \to w$ in E^*. Pick the u such that $u \to v$, $u \to w$ in E^* and $\alpha^{-1}(u)$ is maximum. Then $v = m(u)$ (otherwise $m(u) \to v$, $m(u) \to w$ in E^* by (Q) and $\alpha^{-1}(u)$ is not maximum). But then $m(u) \to w$ in E^* by (Q); i.e. $(v,w) \in E^*$. □

The following algorithm uses Lemma 4 to compute the edges in G_α^* for any ordered graph $G_\alpha = (V,E,\alpha)$. If $A(v) = \{w \mid v \to w \text{ in } G_\alpha\}$ for all vertices v when the algorithm starts, then $A(v) = \{w \mid v \to w \text{ in } G_\alpha^*\}$ for all v when the algorithm finishes.

```
Algorithm FILL: begin
   loop: for i:=1 until n-1 do begin
            v:=α(i)
            m(v):=α(min{α^-1(w)| w ε A(v)});
         add: for w ε A(v) do if w ≠ m(v) then
                 add w to A(m(v));
end end FILL;
```

It is immediate from Lemma 4 that FILL correctly computes the edges of G^*_α. Efficient implementation of FILL is discussed in §5.

3. Motivation: Breadth-First Search and Perfect Orderings

Given a graph G, a breadth-first search of G starting at a vertex s is a systematic examination of the edges of G using the following algorithm:

```
Algorithm BFS: begin
     initialize queue to contain s;
     level(s):=0;
     while queue is non-empty do begin
        remove first vertex v on queue;
        mark v explored;
explore: for w ε adj(v) do if w is unexplored then begin
           add w to end of queue;
           level(w):=level(v)+1;
           mark {v,w} as a tree edge;
     end end end BFS;
```

At each step, this algorithm picks an edge incident to the oldest reached vertex and finds out where the edge leads. The edge may lead either to a vertex already reached or to a new vertex, which is now considered reached.

During execution of this algorithm, statement explore processes each edge exactly twice; once for $w \in adj(v)$ and once for $v \in adj(w)$. The effect of BFS is (1) to generate a spanning tree of G, given by the edges $\{v,w\}$ such that w is unreached when statement explore is executed with $w \in adj(v)$; and (2) to partition the vertices of G into levels: If v is a vertex, $\underline{level}(v) = i$ if the shortest chain from s to v has length i. Figure 1 illustrates BFS applied to a graph.

Each edge joins two vertices on the same level or on adjacent levels. If α is a perfect elimination order for a graph G, and a vertex v is joined to a vertex w with $\underline{level}(w) = \underline{level}(v) + 1$ and to a vertex u with $\underline{level}(u) = \underline{level}(v) - 1$, then $\alpha^{-1}(v) > \min(\alpha^{-1}(u),\alpha^{-1}(w))$ since $\{u,w\}$ is not an edge of G. This strongly suggests that any perfect elimination graph has a perfect ordering which is consistent with the partial ordering by levels. This conjecture is true. The numbering in Figure 1 shows a perfect ordering with this property.

Thus the levels given by BFS convey some information about perfect orderings. But we must break ties within the levels. We can use a breadth-first search within each level to accomplish this, if the new searches are guided by the inter-level edges. This idea leads to a highly complicated way of generating perfect orderings which uses BFS applied recursively. Fortunately, there is a simpler way to look at this method. We present it in the next section. In its full generality,

the resultant algorithm gives minimal orderings, not just perfect orderings, and it is highly efficient.

4. Lexicographic Search: Minimal and Perfect Orderings

To find minimal orderings and perfect orderings, we use a lexicographic ordering scheme which is a special type of breadth-first search. The vertices of the graph are numbered from n to 1. During the search, each vertex v has an associated label consisting of a set of numbers selected from $\{1,2,\ldots,n\}$, ordered in decreasing order. Given two labels $L_1 = [p_1,p_2,\ldots,p_k]$ and $L_2 = [q_1,q_2,\ldots,q_\ell]$, we define $L_1 < L_2$ if, for some j, $p_i = q_i$ for $i = 1,2,\ldots,j-1$ and $p_j < q_j$, or if $p_i = q_i$ for $i = 1,2,\ldots,k$ and $k < \ell$. $L_1 = L_2$ if $k = \ell$ and $p_i = q_i$, $1 \leq i \leq k$.

Minimal Orderings

Consider the following ordering scheme:

```
Algorithm LEX M: begin
     assign the label ∅ to all vertices;
     for i := n step -1 until 1 do begin
select: pick an unnumbered vertex v with largest label;
        comment assign v the number i;
        α(i) := v;
update: for each unnumbered vertex w such that there is a chain
           [v = v_1,v_2,...,v_{k+1} = w] with v_j unnumbered and
           label(v_j) < label(w) for j = 2,3,...,k do add i to
              label(w);
     end end LEX M;
```

This algorithm constructs an ordering α for an initially unordered graph $G = (V,E)$ and constructs a label $L(v)$ given by the final value of label(v) for each $v \in V$. We call any ordering which can be generated by LEX M a lexicographic ordering. Figure 2 shows the application of this algorithm to an example graph. The complicated condition in statement update for updating labels is necessary because there may be fill-in edges; we are trying to find minimal orderings, not just perfect ones. Label updating can be simplified if the object is to find perfect orderings, as we shall see.

To establish the fact that algorithm LEX M produces a minimal ordering, we will prove that the fill-in produced by a lexicographic ordering has the unique chord property of Theorem 2. We need two lemmas which characterize the labels $L(v)$. For any vertex w with label $L(w)$, let

$$L_i(w) = L(w) \cap \{n,n-1,\ldots,i+1\} \quad ;$$

that is, $L_i(w)$ is the value of label(w) in LEX M just before the number i is assigned to a vertex.

Lemma 5: Let $G = (V,E)$ be a graph with lexicographic ordering α and labels $L(v)$, $v \in V$.

(1) If $L_i(v) < L_i(w)$ then $L_j(v) < L_j(w)$ for all $1 \leq j \leq i$.

(2) $L_i(w) \leq L_j(w)$ for all $j \leq i$.

(3) If $\alpha^{-1}(w) = j < \alpha^{-1}(v) = i$ then either $L_i(w) < L_i(v)$ and $L(w) < L(v)$ or $L_i(w) = L_i(v)$ and $L(v) \leq L(w)$.

(4) If $L(w) < L(v)$ with $\alpha^{-1}(w) = j$ and $\alpha^{-1}(v) = i$, then either $j < i$ with $L_i(w) < L_i(v)$ or $i < j$ with $L_j(w) = L_j(v)$.

(5) $j \in L(w)$ if and only if $\alpha^{-1}(w) < j$ and there exists a $v = \alpha(j)$, w chain $[v = v_1, v_2, \ldots, v_{k+1} = w]$ such that $L_j(v_i) < L_j(w)$ and $\alpha^{-1}(v_i) < j$, $2 \leq i \leq k$.

(6) $\{v \in V \mid n \in L(v)\} = adj(\alpha(n))$.

The proofs of (1)-(6) are straightforward. Properties (1) and (2) follow from the definitions of the labels and the order relation; (3) and (4) summarize statement select of LEX M. Property (5) follows from (1), (2) and statement update of LEX M; (5) means that the updated labels produced by one execution of statement update depend only on the old labels and not on the order of updating. Property (6) is an immediate corollary of (5).

Lemma 6: If α is a lexicographic ordering of $G = (V,E)$, then in G^*_α

$$L(w) = \{\alpha^{-1}(v) \mid w \rightarrow v\}$$

for $w \in V$.

Proof: Suppose $i = \alpha^{-1}(v) \in L(w)$. Then, by (5) of Lemma 5, $\alpha^{-1}(w) < i$ and there exists a chain $[v = v_1, v_2, \ldots, v_{k+1} = w]$ with $L_i(v_j) < L_i(w)$ for $2 \leq j \leq k$. Thus, for each such j, $L_m(v_j) < L_m(w)$ for all $m \leq i$ and hence $\alpha^{-1}(v_j) < \alpha^{-1}(w)$. Then $w \rightarrow v$ follows from Lemma 3.

The proof of the converse is somewhat less direct. Suppose that $w \rightarrow v$ and let $i = \alpha^{-1}(v)$. By Lemma 3 there exists a chain $[v = v_1, v_2, \ldots, v_{k+1} = w]$ with $\alpha^{-1}(v_j) < \alpha^{-1}(w)$, $2 \leq j \leq k$. Since $\alpha^{-1}(v_j) < \alpha^{-1}(w) < i$, we have $L_i(v_j) \leq L_i(w)$, $2 \leq j \leq k$. Now suppose

j_0 is the least such j with $L_i(v_{j_0}) = L_i(w)$. Since $\alpha^{-1}(v_{j_0}) < \alpha^{-1}(w) = m$ (say) we have, in addition (Lemma 5, (1),(3)), $L_p(v_{j_0}) \leq L_p(w)$, $m \leq p \leq i$. The chain $[v = v_1, v_2, \ldots, v_{j_0}]$ is such that $L_i(v_p) < L_i(j_0)$, $2 \leq p \leq j_0$; hence $i \in L_{i-1}(v_{j_0})$. But $L_{i-1}(v_{j_0}) \leq L_{i-1}(w)$, $L_i(v_{j_0}) = L_i(w)$, and $i \in L_{i-1}(v_{j_0})$ imply $i \in L_{i-1}(w)$ by the lexicographic ordering of labels. □

Theorem 3: Let $G = (V,E)$ be a graph with lexicographic ordering α and labels $L(v)$, $v \in V$. Then any edge $\{v,w\} \in F(G_\alpha)$ is the unique chord of some 4-cycle $\mu = [p,v,q,w]$ in G^*_α.

Proof: Without loss of generality we may assume $\alpha^{-1}(w) < \alpha^{-1}(v)$. By Lemma 5, part (5) there exists a v, w chain $[v = v_1, v_2, \ldots, v_{k+1} = w]$ in G_α such that $L_j(v_i) < L_j(w)$ and $\alpha^{-1}(v_i) < j$ for $2 \leq i \leq k$, where $j = \alpha^{-1}(v)$. Let $\ell = \max\{\alpha^{-1}(v_i) \mid 2 \leq i \leq k\}$ and let $p = \alpha(\ell)$. Then $p \to v$ and $p \to w$ in G^*_α by Lemma 3.

Now $p \to w$ in G^*_α implies $\mathrm{madj}(p) - \{w\} \subseteq \mathrm{adj}(w)$ in G^*_α. Thus $L_j(p) \subseteq L_j(w)$ by Lemma 6.

Since $L_j(p) < L_j(w)$, there is some $q \in \mathrm{madj}(w) - \mathrm{madj}(p)$ with $\alpha^{-1}(q) > j$. Then $p \not\to q$ in G^*_α, $w \to q$ in G^*_α, and $v - q$ in G^*_α since $w \to v$ and α is a perfect elimination order for G^*_α. Hence $\mu = [p,v,q,w]$ satisfies the theorem. □

Theorems 2 and 3 now immediately imply

Theorem 4: Let $G = (V,E)$ be a graph with lexicographic ordering α. Then α is a minimal ordering.

Proof: $F(G_\alpha)$ is a minimal triangulation by Theorems 2 and 3. □

Perfect Orderings

Since any minimal ordering of a perfect elimination graph is perfect, we can test a graph G to see if it is perfect by generating a minimal ordering α using LEX M and testing whether $F(G_\alpha) = \phi$ using FILL. There is a better way, however. If G is perfect and α is a lexicographic ordering, then $F(G_\alpha) = \phi$; and by Lemma 6, $L(w) = \{\alpha^{-1}(v) \mid w \rightarrow v\}$ in G. Suppose we modify LEX M by simplifying statement update.

```
Algorithm LEX P: begin
     assign the label φ to all vertices;
     for i:=n step -1 until 1 do begin
select: pick an unnumbered vertex v with largest label;
        comment assign v the number i,
        α(i):=v;
update2: for each unnumbered vertex w ε adj(v) do add i to label(w);
     end end LEX P;
```

Algorithm LEX P will generate an ordering α which, by the observation above, must be perfect if G has any perfect orderings, although if G has no perfect orderings α may not be minimal (see Figure 3). Figure 4 shows the application of LEX P to the graph in Figure 1. The relationship between LEX M, LEX P, and BFS is as follows: Any ordering generated by LEX M can be generated by LEX P, and any ordering generated by LEX P can be generated by performing a breadth-first search, numbering the vertices from n to 1 as they are reached. Thus the LEX M orderings are a subset of the LEX P orderings, which are in turn a subset of the breadth-first search orderings.

5. Implementation and Complexity

In this section we give linear-time implementations of FILL and LEX P, and an $O(ne)$ implementation of LEX M.

Computation of Fill-in

To get algorithm FILL to run fast, we must make sure that the adjacency lists A(v) do not contain too many redundant elements. During the i^{th} iteration of statement loop in FILL, we read through the elements in $A(\alpha(i))$ and eliminate duplicates. We use a Boolean array test(j), setting test(j) to true if and only if $\alpha(j) \in A(\alpha(i))$. The implementation appears below in Algol-like notation.

```
Algorithm FILL: begin
     for j:=1 until n do test(j):=false;
loop: for i:=1 until n-1 do begin
        k:=n;
        v:=α(i);
        comment  eliminate duplicates in A(v) and compute m(v);
   dup: for w ∈ A(v) do
           if test(α^-1(w)) then delete w from A(v)
           else begin
              test(α^-1(w)):=true;
              k:=min(k,α^-1(w));
           end;
        m(v):=k;
        comment  add required fill-in edges and reset test;
   add: for w ∈ A(v) do begin
           test(α^-1(w)):=false;
           if w ≠ m(v) then add w to A(m(v));
  end end end FILL;
```

Suppose this algorithm is applied to an ordered graph $G_\alpha = (V,E,\alpha)$ whose elimination graph G_α^* has e' edges. Each time statement add is executed, A(v) is free of redundancies since dup eliminates such

redundancies. The number of entries made to adjacency lists by one execution of add is thus bounded by $|A(v)|$, and the total number of additions to adjacency lists made by add over all iterations of loop is bounded by e'. It follows that the total time spent in dup over all iterations of loop is $O(n+e')$, and the running time of FILL is $O(n+e')$.

Perfect Orderings

The programming of LEX P is an interesting exercise in list processing, since the search requires that vertices be kept in a particular order depending on their labels. To make the implementation efficient, we do not actually calculate the labels of the vertices. For each label value, we keep a set of all vertices which have that label. We keep the sets in a queue ordered lexicographically by label (highest to lowest). When a new vertex v is numbered, we create a new set S' for each old set S containing a vertex w such that v—w. We delete from S all such vertices w and add them to the new set S', which is then inserted in the queue of sets just in front of S. It is easy to see that this method maintains the proper lexicographic ordering without actually calculating the labels.

An implementation of this method is given below in an Algol-like notation. To maintain the queue and the sets, we use cells, each containing four items: flag, head, next, and back. Certain cells are used as set headers. These cells are doubly linked using head and back in a queue which is ordered lexicographically by set label. A single cell is used as a header for this queue (it does not head a set of vertices).

Other cells are used to contain the vertices in the sets. Cells representing a set are doubly linked using next and back. If c is the

header cell of a set, next(c) points to the list of elements in the set. If c is an element cell, head(c) contains the name of the vertex in the cell and flag(c) points to the header of the set containing the cell. Figure 5 gives an example of this data structure.

The program uses certain other variables. If v is a vertex, cell(v) points to the cell containing v. The list fixlist contains pointers to the headers of the new sets created after a vertex v is numbered. Each such header h has flag(h) = 1 until after all the new sets are constructed. The algorithm then empties fixlist, resetting all the flags of the corresponding headers to zero.

```
Algorithm LEX P: begin
    comment (implicity) assign label φ to all vertices;
    head(1):=2;
    back(2):=1;
    head(2):=back(1):=next(1):=flag(1):=flag(2):=0;
    c:=3;
    comment  c is the number of the first empty cell;
    for  v ε V do begin
       head(c):=v;
       cell(v):=next(c-1):=c;
       flag(c):=2;
       back(c):=c-1;
       c:=c+1;
       α^-1(v):=0;
    end;
    next(c-1):=0;
    for i:=n step -1 until 1 do begin
       comment  skip empty sets;
       while next(head(1)):=0 do begin head(1):=head(head(1));
          back(head(1)):=1 end;
       comment  pick next vertex to number;
```

```
select: p:=next(head(1));
        comment  delete cell of vertex from set;
        next(head(1)):=next(p);
        if next(head(1)) ≠ 0 then back(next(head(1)):=head(1);
        v:=head(p);
        comment  assign v the number i;
        α(i):=v;
        α^-1(v):=i;
        fixlist:=φ;
update2: for w ε adj(v) do if α^-1(w) = 0 then begin
           comment  delete cell of w from set;
           next(back(cell(w)):=next(cell(w));
           if next(cell(w)) ≠ 0 then back(next(cell(w)):=back(cell(w));
           h:=back(flag(cell(w));
           comment  if h is an old set then create a new set;
           if flag(h) = 0 then begin
              head(c):=head(h);
              head(h):=c;
              back(head(c)):=c;
              back(c):=h;
              flag(c):=1;
              next(c):=0;
              add c to fixlist;
              h:=c;
              c:=c+1;
           end;
           comment  add cell of w to new set;
           next(cell(w)):=next(h);
           if next(h) ≠ 0 then back(next(h)):=cell(w);
           flag(cell(w)):=back(cell(w)):=h;
           next(h):=cell(w);
        end;
        for h ε fixlist do flag(h):=0;
     end end LEX P;
```

It is routine to verify that algorithm LEX P, as implemented above, operates correctly and requires $O(n+e)$ time and space to order a graph.

We can use LEX P to generate an ordering and then use FILL to calculate its fill-in. If the fill-in is empty, the graph is triangulated, and the ordering is perfect. If the fill-in is non-empty, the graph is not triangulated. Thus we have an $O(n+e)$ algorithm to test whether a graph is a perfect elimination graph and to generate a perfect elimination ordering if there is one.

Minimal Orderings

Algorithm LEX M apparently requires more execution time than LEX P, because statement update in LEX M requires more graph searching than statement update2 in LEX P. Here is an implementation of LEX M which runs in $O(ne)$ time.

To keep track of the labels, we use a less complicated scheme than used in the implementation of LEX P. Each unnumbered vertex w has an associated label number $\ell(w)$, such that $\ell(y) = \ell(z)$ if and only if y and z have the same label, and $\ell(y) < \ell(z)$ if and only if the label of y is less than the label of z. These label numbers are integers between 1 and k, where k is the number of distinct labels. When a new vertex v is numbered, each vertex w connected to v by a chain of the type defined in statement update is assigned a new label number $\ell'(w) = \ell(w) + \frac{1}{2}$. Label numbers of other vertices are not changed. The resultant label numbers are then sorted (using a radix sort) and new label numbers assigned so that all label numbers are integers between 1 and the new value of k.

To find chains of the type defined in statement update, we conduct

a search starting from the newly numbered vertex v. First the search passes only through vertices of highest label. Then the search is extended through vertices of second highest label, and so on. In this way all appropriate chains can be found efficiently. The program appears below in Algol-like notation.

```
Algorithm LEX M: begin
     for v ε V do begin ℓ(v):=1; α^-1(v):=0 end;
     k:=1;
loop: for i:=n step -1 until 1 do begin
 select: pick an unnumbered vertex v with ℓ(v) = k;
         comment assign v the number i;
         α(i):=v;
         α^-1(v):=i;
         for j:=1 until k do reach(j):=φ;
         mark all unnumbered vertices unreached;
         for w ε adj(v) and α^-1(w) = 0 do begin
            add w to reach(ℓ(w));
            mark w reached;
            ℓ(w):=ℓ(w)+1/2;
            mark {v,w} as an edge of G*_α;
         end;
search: for j:=k step -1 until 1 do
            while reach(k) ≠ φ do begin
               delete a vertex w from reach(k);
               for z ε adj(w) and z unreached do begin
                  mark z reached;
                  if ℓ(z) < k then begin
                     add z to reach(ℓ(z));
                     ℓ(z):=ℓ(z)+1/2;
                     mark {v,z} as an edge of G*_α;
                  end else add z to reach(k);
            end end;
   sort: sort unnumbered vertices by ℓ(w) value;
         reassign ℓ(w) values to be integers from 1 to k,
            redefining k appropriately;
      end end LEX M;
```

It is an easy exercise to show that this program correctly implements algorithm LEX M to compute a minimal ordering. The time required per execution of statement search is $O(e)$ since each vertex can only be marked "reached" once and thus each edge can only be examined once. Statement sort requires $O(n)$ time when implemented as a radix sort [16]. The running time of the program is thus $O(e)$ per execution of statement loop, or $O(ne)$ time altogether. LEX M requires $O(n+e)$ storage space.

6. Remarks

This paper has given an $O(n+e)$ algorithm for finding a perfect elimination order on a graph if one exists and a related $O(ne)$ algorithm for finding minimal elimination orderings. The algorithm for finding perfect orderings is optimum to within a constant factor and is asymptotically faster than anything previously published. The minimal ordering algorithm, as implemented here, has the same asymptotic time bound as Ohtsuki's algorithm [18], but his algorithm does not calculate the fill-in produced by the ordering. The approach here solves both the perfect ordering and minimal ordering problems efficiently, reveals certain properties of lexicographic search, and provides a new characterization of minimal triangulations. It is not known whether there is a better algorithm for finding minimal orderings, or whether the problem of finding a minimum ordering is NP-complete.

It is possible to extend the notions of perfect, minimal, and minimum elimination orderings to directed graphs; such orderings are related to trying to minimize the fill-in when performing Gaussian elimination on

sparse asymmetric matrices [10,15]. Lemmas 1, 2, and 3, Corollaries 1 and 2, and Theorem 1 all generalize to directed graphs. However, lexicographic search doesn't seem to help in finding good orderings on directed graphs. We have constructed $O(ne)$ algorithms to compute the fill-in of any ordering and to find a perfect ordering if one exists. We have devised an $O(n^4)$ algorithm for finding a minimal ordering, using the proof of Lemma 2 (Shiloah also claims an $O(n^4)$ algorithm [25]). We can show that testing whether a directed graph has a perfect ordering or computing any ordering's fill-in requires as much time as testing a directed graph for transitivity. These results will be reported in a future paper.

Minimum orderings are desired in practice, but finding them is time-consuming. Minimal orderings are not necessarily close to minimum; for instance, if the lexicographic ordering scheme described here is applied to a graph representing an n by n square grid, the fill-in is $O(n^3)$, while the nested dissection method [9,22] gives an ordering with $O(n^2 \log n)$ fill-in, which is minimum to within a constant factor [12]. The development of good ordering schemes for special cases (such as grid graphs) and the theoretical study of heuristics seems to be fruitful areas for future research.

In particular, two heuristics which seem to work well in practice are the minimum-degree heuristic and the minimum-fill-in heuristic [21]. It seems possible that the minimum degree heuristic produces minimum fill-in to within a constant factor, at least on grid graphs. A proof of such a statement would be extremely interesting.

When performing Gaussian elimination in practice, it might be important to minimize something other than the fill-in, such as the total operation count [1,21]. The problem of finding an ordering which minimizes

the operation count or some other criterion can be formulated graph-theoretically; only the fill-in criterion has been studied extensively.

References

[1] U. Bertele and F. Brioschi, Non-Serial Dynamic Programming, Academic Press, N.Y. (1971).

[2] K. Booth, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, private communication (1974).

[3] E.G. Coffman, Jr. and R.L. Graham, "Optimal scheduling for two processor systems," Acta Informatica, Vol. 1 (1972), pp. 220-213.

[4] S. Cook, "The complexity of theorem-proving procedures," Proceedings Third Annual ACM Symposium on Theory of Computing (1971), pp. 151-158.

[5] G.A. Dirac, "On rigid circuit graphs," Abh. Math. Sem. Univ. Hamburg, Vol. 25 (1961), pp. 71-76.

[6] D.R. Fulkerson and O.A. Gross, "Incidence matrices and interval graphs," Pacific Journal of Mathematics, Vol. 15 (1965), pp. 835-855.

[7] F. Gavril, "Algorithms for minimum coloring, maximum clique, minimum coloring by cliques and maximum independent set of a chordal graph," SIAM Journal of Computing, Vol. 1 (1972), pp. 180-187.

[8] F. Gavril, "An $n^{\log_2 7}$ algorithm for testing chordality of graphs," unpublished manuscript, Syracuse University (1974).

[9] J.A. George, "Nested dissection of a regular finite element mesh," SIAM Journal of Numerical Analysis, Vol. 10 (1973), pp. 345-363.

[10] P.C. Gilmore and A.J. Hoffman, "A characterization of comparability graphs and of interval graphs," Canadian Journal of Mathematics, Vol. 16 (1964), pp. 539-548.

[11] L. Haskins and D. Rose, "Toward characterization of perfect elimination digraphs," SIAM Journal of Computing, Vol. 2 (1973), pp. 217-224.

[12] 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, Vol. 10 (1973), pp. 364-369.

[13] T.C. Hu, "Parallel sequencing and assembly line problems," Operations Research, Vol. 9 (1961), pp. 841-848.

[14] R. Karp, "Reducibility among combinatorial problems," Complexity of Computer Computations, R.E. Miller and J.W. Thatcher, eds., Plenum Press, N.Y. (1972), pp. 85-104.

[15] D.J. Kleitman, "A note on perfect elimination digraphs," SIAM Journal of Computing, to appear.

[16] D. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison Wesley, Reading, Mass. (1973), pp. 170-180.

[17] T. Ohtsuki, L.K. Cheung, T. Fujisawa, "On minimal triangulation of a graph," Memorandum No. ERL-M351, Electronics Research Laboratory, University of California, Berkeley (1972).

[18] T. Ohtsuki, "A graph-theoretic algorithm for optimal pivoting order of sparse matrices," Proceedings Sixth Annual Hawaii International Conference on System Sciences (1973).

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

[20] D. Rose, "On simple characterizations of k-trees," Report #29-72, Center for Research in Computing Technology, Harvard University (1972).

[21] 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), pp. 183-217.

[22] D.J. Rose and G.F. Whitten, "Automatic nested dissection," Proceedings ACM Annual Conference (1974), pp. 82-88.

[23] R. Sethi, "Algorithms for nonpreemptive scheduling on two processors," Computer Science Department, Pennsylvania State University, unpublished manuscript (1974).

[24] V. Strassen, "Gaussian elimination is not optimal," Numerical Mathematics, Vol. 13 (1964), pp. 354-356.

[25] S. Even, private communication.

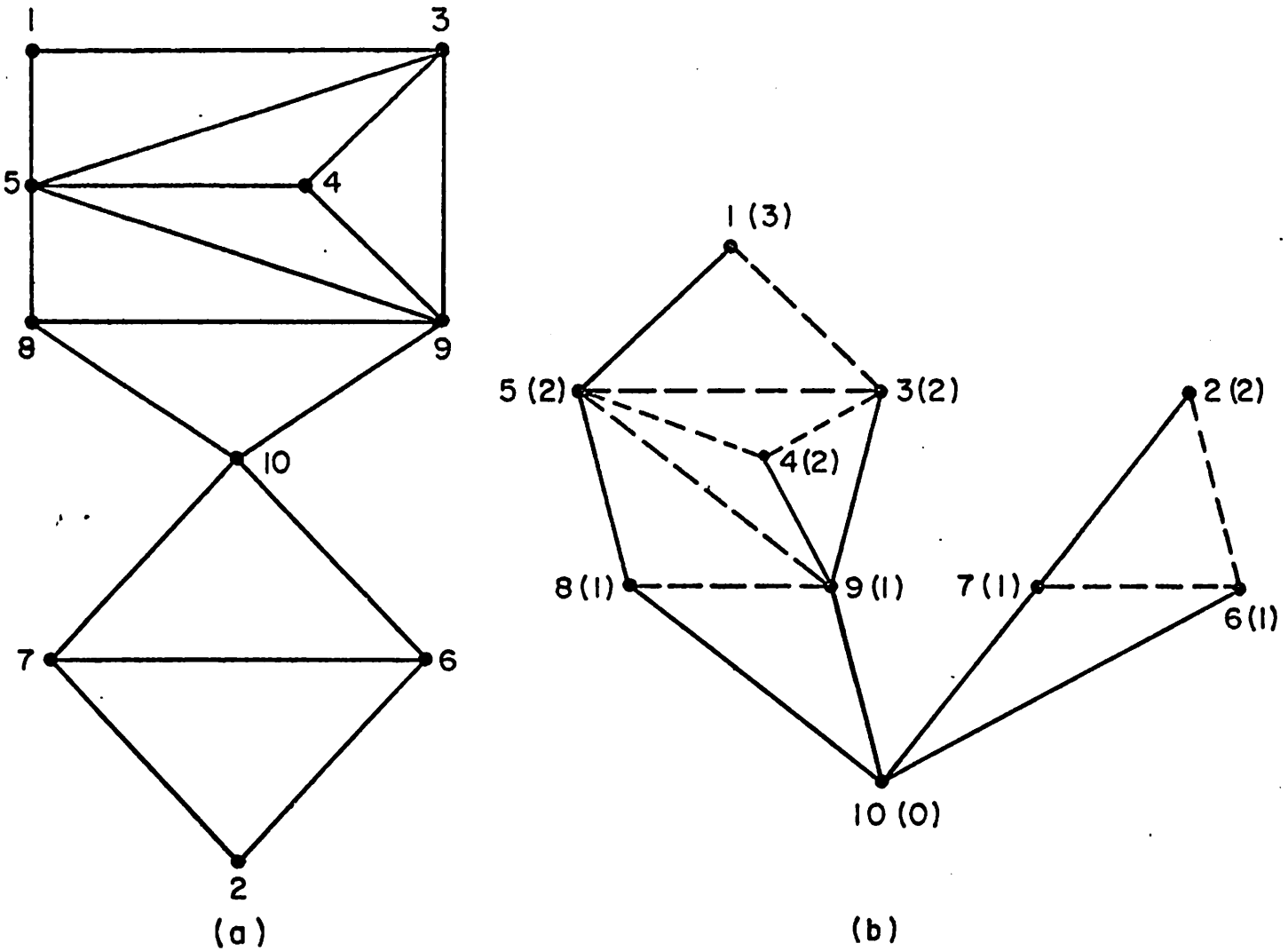

Figure 1: Breadth-first search of a graph starting at vertex 10.
(a) Graph
(b) Tree produced by BFS, with level numbersin parentheses, and non-tree edges dashed. Vertex numbers give a perfect elimination order which is consistent with level order.

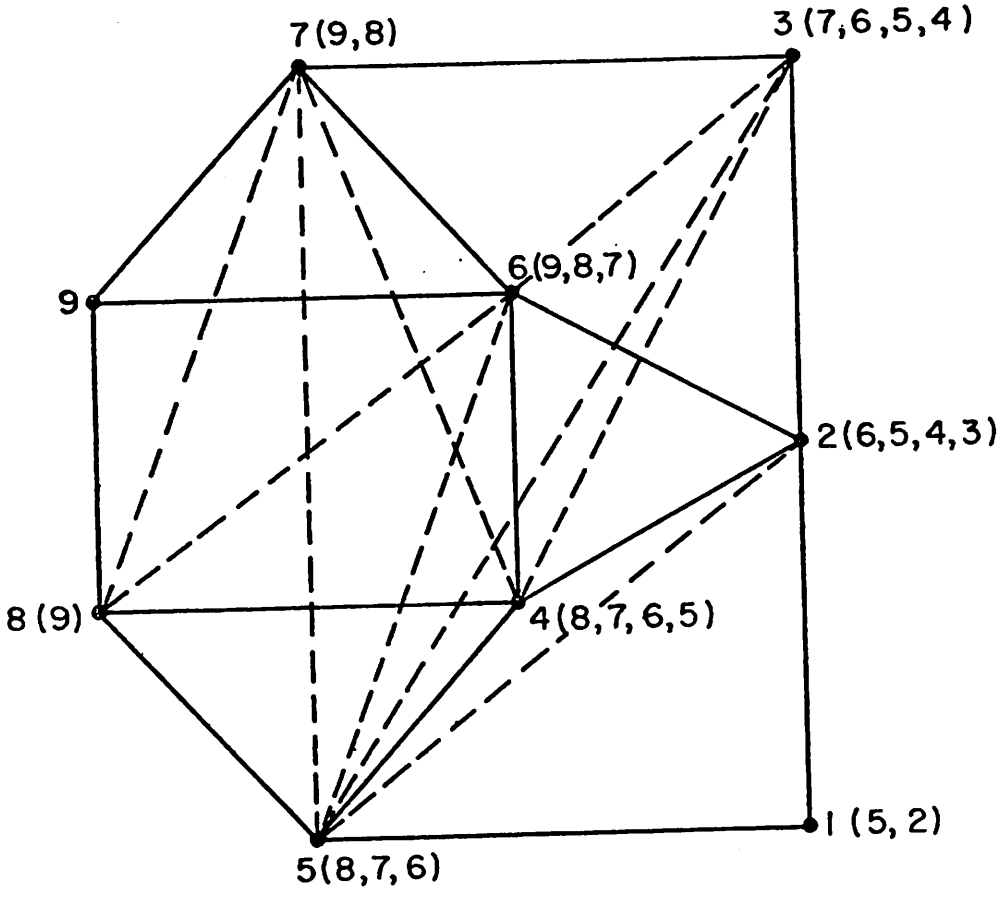

Figure 2: Minimal ordering of a graph generated by LEX M. Final labels are in parentheses, nine fill-in edges are dotted. Ordering is not minimum since there is another ordering with only five fill-in edges.

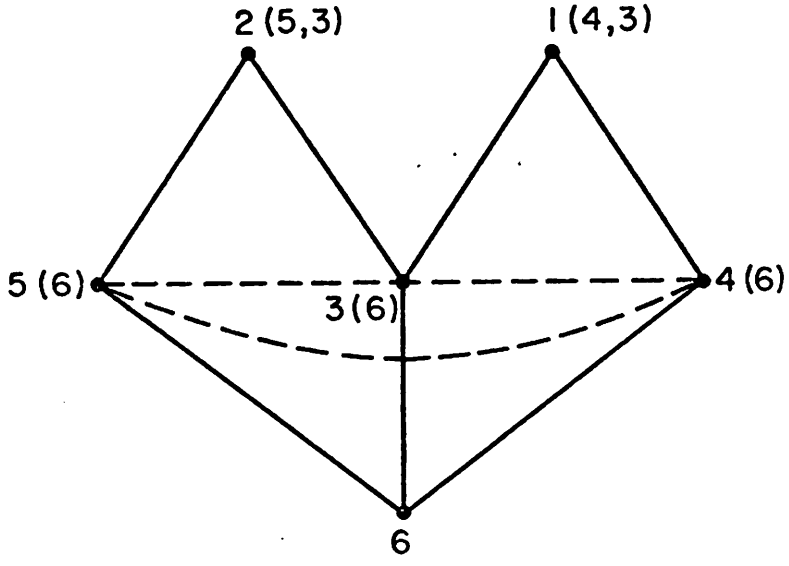

Figure 3: Order generated by LEX P for a non-triangulated graph. Fill-in edges are dotted. Ordering is not minimal since another ordering has fill-in {{3,4},{3,5}}.

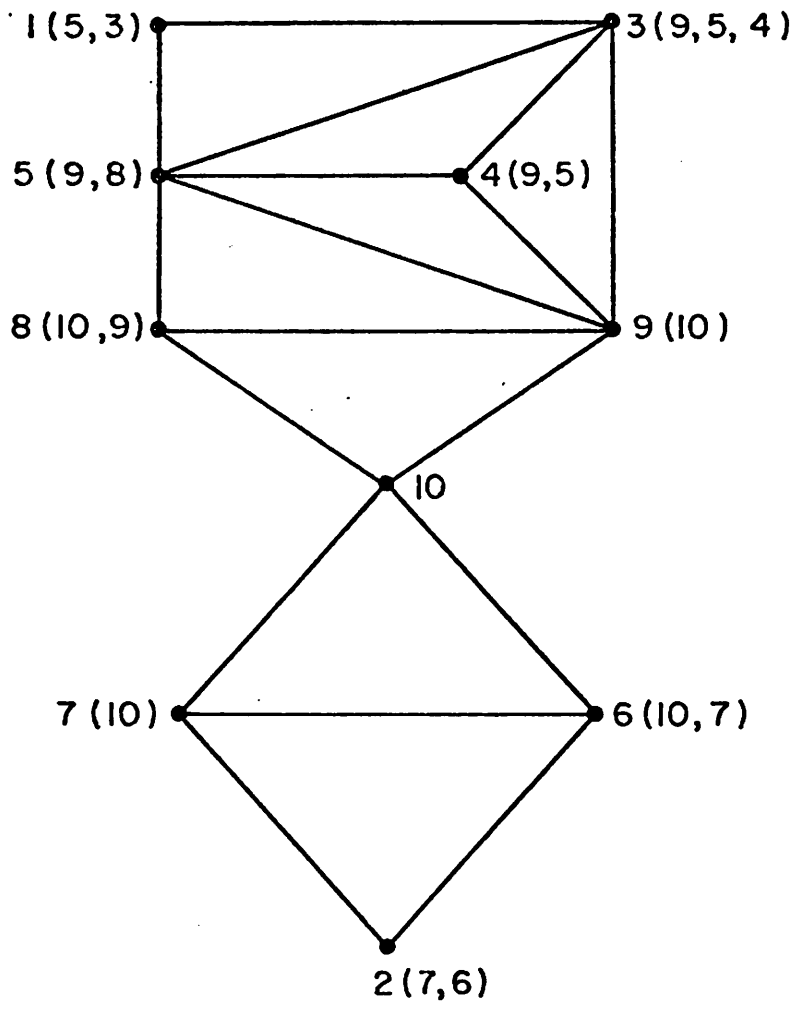

Figure 4: Final labels and perfect elimination order generated when LEX P is applied to graph in Figure 1.

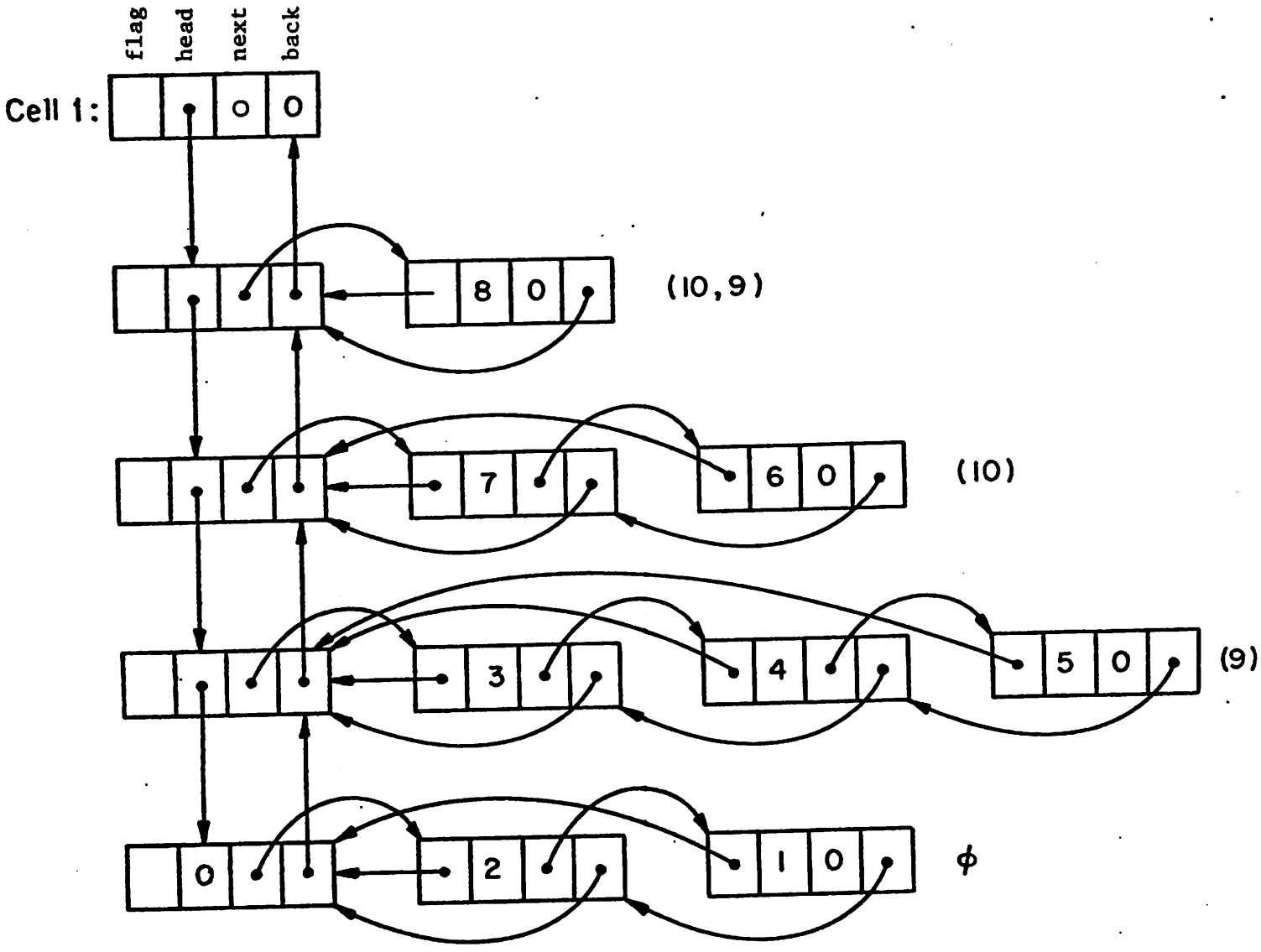

Figure 5: Data structure for Figure 4 example after vertices 10 and 9 have been numbered. For convenience, vertices are assumed to be identified by their elimination number. Implicit labels of the sets are in parentheses.