SIAM J. COMPUT.
Vol. 13, No. 3, August 1984

007

SIMPLE LINEAR-TIME ALGORITHMS TO TEST CHORDALITY OF GRAPHS, TEST ACYCLICITY OF HYPERGRAPHS, AND SELECTIVELY REDUCE ACYCLIC HYPERGRAPHS*

ROBERT E. TARJAN† AND MIHALIS YANNAKAKIS†

Abstract. Chordal graphs arise naturally in the study of Gaussian elimination on sparse symmetric matrices; acyclic hypergraphs arise in the study of relational data bases. Rose, Tarjan and Lueker [SIAM J. Comput., 5 (1976), pp. 266–283] have given a linear-time algorithm to test whether a graph is chordal, which Yannakakis has modified to test whether a hypergraph is acyclic. Here we develop a simplified linear-time test for graph chordality and hypergraph acyclicity. The test uses a new kind of graph (and hypergraph) search, which we call *maximum cardinality search.* A variant of the method gives a way to selectively reduce acyclic hypergraphs, which is needed for evaluating queries in acyclic relational data bases.

Key words. graph algorithm, acyclic data base scheme, sparse Gaussian elimination, graph search, hypergraph

1. Introduction. We shall use more-or-less standard terminology from the theory of graphs and hypergraphs [3], some of which we review here. A *hypergraph* $H = (V, E)$ consists of a set of *vertices* V and a set of *edges* E; each edge is a subset of V. A *graph* is a hypergraph all of whose edges have size two. The *graph* $G(H)$ of a hypergraph H is the graph whose vertices are those of H and whose edges are the vertex pairs $\{v, w\}$ such that v and w are in a common edge of H. Two vertices of a graph G are *adjacent* if they are contained in an edge. A *path* in G is a sequence of distinct vertices $v_0, v_1, \cdots, v_k$ such that v_i and v_{i+1} are adjacent for $0 \leqq i < k$. A *cycle* is a path $v_0, v_1, \cdots, v_k$ such that $k \geqq 2$ and v_0 and v_k are adjacent. Vertices v_i and $v_{(i+1) \bmod (k+1)}$ for $0 \leqq i \leqq k$ are *consecutive* on the cycle. A *clique* of G is a set of pairwise adjacent vertices. A hypergraph H is *conformal* if every clique of $G(H)$ is contained in an edge of H. A graph G is *chordal* if every cycle of length at least four has a chord, i.e., an edge joining two nonconsecutive vertices on the cycle. A hypergraph H is *acyclic* if H is conformal and $G(H)$ is chordal.

Chordal graphs arise in the study of Gaussian elmination on sparse symmetric matrices [12]. Acyclic hypergraphs arise in the study of relational data base schemes [1], [7], [21]; they are powerful enough to capture most real-world situations but simple enough to have many desirable properties [1], [2], [9], [18]. Rose, Tarjan and Lueker [15] have given an $O(n+m)$-time[1] algorithm (henceforth called the RTL algorithm) to test whether a graph is chordal. Yannakakis [19] has extended the algorithm to the problem of testing whether a hypergraph is acyclic. In this paper we propose a simplified version of the RTL algorithm that can be used for testing both chordality of graphs and acyclicity of hypergraphs. In § 2 we develop the algorithm as it applies to graph chordality testing. In § 3 we modify the algorithm for hypergraph acyclicity testing. Besides leading to a method simpler than the RTL test, our analysis provides additional insight into the structure of chordal graphs and acyclic hypergraphs. In § 4 we use this insight to develop a simple linear-time algorithm for selectively reducing acyclic hypergraphs, a problem that arises in evaluating queries in acyclic relational data bases.

* Received by the editors October 7, 1982, and in revised form May 23, 1983.

† Bell Laboratories, Murray Hill, New Jersey 07974.

[1] We shall use n to denote the number of vertices and m to denote the total size of the edges in a hypergraph.

2. Testing chordality of graphs. In this section we shall freely use results of [14], [15]. A discussion of chordal graphs and their importance in Gaussian elimination can be found in [14]. Let $G=(V, E)$ be a graph. Let $\alpha: V \leftrightarrow \{1, 2, \cdots, n\}$ be a numbering defining a total ordering of the vertices of G. We shall use the notation $v <_\alpha w$ to mean $\alpha(v) < \alpha(w)$. The *fill-in* produced by this ordering is the set

$$F(\alpha) = \{\{v, w\} | \{v, w\} \notin E \text{ and there is a path from } v \text{ to } w \text{ containing only } v, w, \text{ and vertices ordered before both } v \text{ and } w\}.$$

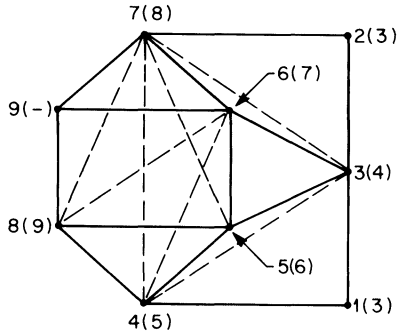

FIG. 1. *A graph numbered by maximum cardinality search and the corresponding elimination graph. Original edges are solid; fill-in edges are dashed. Followers of vertices (to be defined below) are in parentheses.*

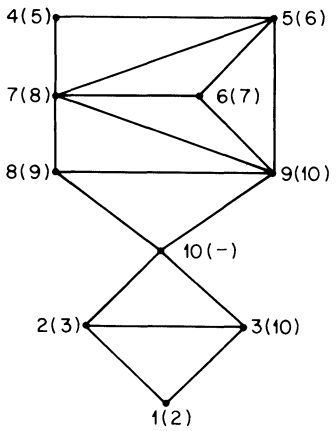

FIG. 2. *A chordal graph numbered by maximum cardinality search. Ordering is zero fill-in. Followers of vertices are in parentheses.*

The *elimination graph* of G with ordering α is $G(\alpha)=(V, E \cup F(\alpha))$. (See Fig. 1.) Note that $G(\alpha)$ is a subgraph of the transitive closure of G. If $F(\alpha)=\varnothing$, α is a *zero fill-in ordering* of G. (See Fig. 2.)

LEMMA 1 [15]. *A pair $\{v, w\}$ is in $E \cup F(\alpha)$ if and only if either $\{v, w\} \in E$ or there is a vertex u such that $\{u, v\}$, $\{u, w\} \in E \cup F(\alpha)$ and u is ordered before both v and w.*

LEMMA 2 [15]. *An ordering α is zero fill-in if and only if for all distinct $\{u, v\}$, $\{u, w\}$ in E such that u is ordered before both v and w, $\{v, w\} \in E$.*

LEMMA 3 [15]. *Any ordering α is a zero fill-in ordering of the corresponding graph $G(\alpha)$.*

THEOREM 1 [14]. *A graph G is chordal if and only if it has a zero fill-in ordering.*

Remark. Lemmas 1 and 2 and Theorem 1 are implicit in the work of Dirac [6] and Fulkerson and Gross [8] on chordal graphs, although they did not consider the notion of fill-in.

The RTL chordality-testing algorithm consists of two steps:

Step 1. Compute an ordering α of G that is zero fill-in if and only if G is chordal.

Step 2. Compute the fill-in produced by α. G is chordal if and only if $F(\alpha)=\varnothing$.

The algorithm we shall describe consists of simplified methods for carrying out steps one and two. Step 1 of the RTL method consists of numbering the vertices from n to 1 in decreasing order using *lexicographic search*, defined as follows: For each unnumbered vertex v, maintain a list of the numbers of the numbered vertices adjacent to v, with the numbers in each list arranged in decreasing order. As the next vertex to number, select the vertex whose list is lexicographically greatest, breaking ties arbitrarily. Although somewhat complicated, lexicographic search can be implemented to run in $O(n+m)$-time.

We shall derive a sufficient condition for an ordering of a chordal graph to be zero fill-in. This condition holds not only for lexicographic search but also for a simpler kind of search that we call *maximum cardinality search*: Number the vertices from n to 1 in decreasing order. As the next vertex to number, select the vertex adjacent to the largest number of previously numbered vertices, breaking ties arbitrarily. (See Figs. 1 and 2.)

LEMMA 4. *Let $G=(V, E)$ be a chordal graph and let α be an ordering of G. If α has the following property, then α is zero fill-in*:

(P) *If $u <_\alpha v <_\alpha w$, $\{u, w\} \in E$, and $\{v, w\} \notin E$, then there is a vertex x such that $v <_\alpha x$, $\{v, x\} \in E$, and $\{u, x\} \notin E$.*

Proof. Suppose α has property P. Let $v_0, v_1, \cdots, v_k$ be an unchorded path for which $\alpha(v_k)$ is maximum, such that $k \geqq 2$ and

(Q) For some i in the interval[2] $[1, k-1]$, the following inequalities hold:

$$v_0 >_\alpha v_k >_\alpha v_1 >_\alpha v_2 > \cdots >_\alpha v_i \quad \text{and} \quad v_i <_\alpha v_{i+1} < \cdots <_\alpha v_k.$$

(A path is *unchorded* if any two nonconsecutive vertices are nonadjacent.) We shall derive a contradiction, thus showing that no unchorded path has property Q.

Since v_0 and v_1 are adjacent but not v_0 and v_k, there is by property P a vertex x such that $v_k <_\alpha x$ and v_k but not v_1 is adjacent to x. Let $j>1$ be minimum such that v_j is adjacent to x. Since G is chordal, x is not adjacent to v_0, for otherwise $v_0, v_1, \cdots, v_j, x$ would be an unchorded cycle. Thus if $v_0 >_\alpha x$, the path $v_0, v_1, \cdots v_j, x$ has property Q; if $x >_\alpha v_0$, the path $x, v_j, v_{j-1}, \cdots, v_0$ has property Q. Either case contradicts the choice of $v_0, v_1, \cdots, v_k$ as the path with $\alpha(v_k)$ maximum having property Q. Therefore no path has property Q.

Now suppose $\{u, v\}$ and $\{u, w\}$ are distinct edges such that u is ordered before both v and w by α. If v and w were not adjacent, either v, u, w or w, u, v would have property Q. Thus v and w must be adjacent. Since this holds for any such $\{u, v\}$ and $\{u, w\}$, it follows from Lemma 2 that α is zero fill-in. □

THEOREM 2. *Any ordering α generated by maximum cardinality search has Property* P *and thus is zero fill-in if G is chordal.*

Proof. Let α be an ordering generated by maximum cardinality search. Suppose $u <_\alpha v <_\alpha w$ and w is adjacent to u but not to v. When v is numbered, v must be

[2] Throughout this paper we shall use the notation $[i_1, i_2]$ to denote the set of integers $\{i | i_1 \leqq i \leqq i_2\}$.

adjacent to at least as many numbered vertices as u. Thus, since u but not v is adjacent to w, v is adjacent to some other numbered vertex x not adjacent to u. Since this holds for all such u, v, w, ordering α has Property P. □

Remark. It is easy to show that any ordering generated by lexicographic search also has Property P.

We can implement maximum cardinality search as follows. We maintain an array of sets *set* (i) for $0 \leqq i \leqq m-1$. We store in *set* (i) all unnumbered vertices adjacent to exactly i numbered vertices. Initially *set* (0) contains all the vertices. We maintain the largest index j such that *set* (j) is nonempty. To carry out a step of the search, we remove a vertex v from *set* (j) and number it. For each unnumbered vertex w adjacent to v, we move w from the set containing it, say *set* (i), to *set* ($i+1$). Then we add one to j and while *set* (j) is empty repeatedly decrement j. If we represent each set by a doubly linked list of vertices (to facilitate deletion) and maintain for each vertex the index of the set containing it, the search requires $O(n+m)$ time. (Since j is incremented n times and is never less than -1, the total time to manipulate j is $O(n)$.)

The following program written in a variant of Dijkstra's guarded command language [5], implements maximum cardinality search. For any unnumbered vertex v, *size* (v) is the number of numbered vertices adjacent to v. If v is a numbered vertex, we define *size* (v) to be -1.

MAXIMUM CARDINALITY SEARCH.

```
local j, v;
for i ∈ [0, n−1] → set (i) := ∅ rof;
for v ∈ vertices → size (v) := 0; add v to set (0) rof;
i := n; j := 0;
do i ≧ 1 →
   v := delete any from set (j);
   α(v) := i; α^{-1}(i) := v; size (v) := −1;
   for {v, w} ∈ E such that size (w) ≧ 0 →
      delete w from set (size (w));
      size (w) := size (w) + 1;
      add w to set (size (w))
   rof;
   i := i − 1;
   j := j + 1;
   do j ≧ 0 and set (j) = ∅ → j := j − 1 od
od;
```

Let us turn to Step 2 of the chordality test. We shall describe an algorithm that computes the fill-in produced by an arbitrary numbering α of an arbitrary graph G in $O(n+m')$ time, where m' is the number of edges in $G(\alpha)$. The algorithm, a simplification of the RTL method, was discovered by Greg Whitten [17], who presented it at the SIAM Symposium on Sparse Matrix Computations, held in Knoxville, Tennessee in 1978, but never published it, even in the conference proceedings.

Let G be a graph, let α be an ordering of G, and let $G(\alpha) = (V, E \cup F(\alpha))$ be the elimination graph of G with ordering α. For any vertex v, let $f(v)$, the *follower* of v, be the vertex of smallest number that is both adjacent to v in $G(\alpha)$ and has number larger than that of v. (See Figs. 1 and 2.) Note that a vertex need not have a follower. We define $f^i(v)$ for $i \geqq 0$ by $f^0(v) = v$, $f^{i+1}(v) = f(f^i(v))$.

LEMMA 5. *If* $\{v, w\} \in E \cup F(\alpha)$ *with* $v <_\alpha w$, *then* $f^i(v) = w$ *for some* $i \geqq 1$.

Proof. We use induction on $\alpha(v)$ from n to 1. If $f(v) \neq w$, then $v <_\alpha f(v) <_\alpha w$ by the definition of f. By Lemma 1, $\{f(v), w\} \in E \cup F(\alpha)$. By the induction hypothesis $f^i(f(v)) = w$ for some $i \geq 1$, i.e., $f^{i+1}(v) = w$. □

THEOREM 3. *A pair $\{v, w\}$ with $v <_\alpha w$ is in $E \cup F(\alpha)$ if and only if there is a vertex x such that $\{x, w\} \in E$ and $f^i(x) = v$ for some $i \geq 0$.*

Proof. Let $\{v, w\}$ be a pair with $v <_\alpha w$. Suppose there is a vertex x such that $\{x, w\} \in E$ and $f^i(x) = v$ for some $i \geq 0$. Then $x \leq_\alpha v$ by the definition of f. Since α is a zero-fill-in ordering for $G(\alpha)$ by Lemma 3, $\{v, w\} \in E \cup F(\alpha)$ by the definition of fill-in.

Conversely, suppose $\{v, w\} \in E \cup F(\alpha)$. We prove by induction on $\alpha(v)$ that there is a vertex x such that $\{x, w\} \in E$ and $f^i(x) = v$ for some $i \geq 0$. If $\{v, w\} \in E$, $x = v$ and $i = 0$ satisfy the theorem. Otherwise, by Lemma 1 there is a vertex u such that $\{u, v\}$, $\{u, w\} \in E \cup F(\alpha)$ and $u <_\alpha v$. By the induction hypothesis there is a vertex x such that $\{x, w\} \in E$ and $f^i(x) = u$ for some $i \geq 0$. By Lemma 4, $f^j(u) = v$ for some $j \geq 0$. Thus x and $i + j$ satisfy the theorem. □

Theorem 3 leads to a fast algorithm for computing fill-in. We process the vertices in order from the vertex numbered 1 to the vertex numbered n. When processing vertex w, we compute the set $A(w)$ of all vertices v such that $\{v, w\} \in E \cup F(\alpha)$ and $v <_\alpha w$; we also find all vertices v such that $f(v)$ should be w, and define $f(v) = w$. To compute $A(w)$ we initialize it to contain all vertices v such that $\{v, w\} \in E$ and $v <_\alpha w$. Then we repeat the following step until it no longer applies: Select a vertex $v \in A(w)$ such that $f(v)$ has been computed (i.e., $f(v) <_\alpha w$) and $f(v) \notin A(w)$, and add $f(v)$ to $A(w)$. After constructing $A(w)$, define $f(v)$ to be w for all vertices $v \in A(w)$ such that $f(v)$ has not yet been defined.

We shall implement a variant of this algorithm to run in $O(n + m')$ time. We use two arrays, f and *index*. When a vertex v with $\alpha(v) = i$ is processed, we initialize $f(v)$ to be v and *index* (v) to be i. The first time we process a higher numbered vertex w adjacent to v, we define $f(v)$ to be w. Every time we process a higher numbered vertex w adjacent to v, we define *index* $(v) = \alpha(w)$. Thus *index* (v) is always the maximum number in the set $\{v\} \cup \{w | \{v, w\} \in E$ and w has been processed$\}$. This idea of an index array is due to Gustavson [11]. To process a vertex w, we repeat the following step for each vertex v such that $\{v, w\} \in E$ and $v <_\alpha w$:

General step. Initialize x to v. While *index* $(x) < \alpha(w)$, set *index* $(x) = \alpha(w)$, add $\{x, w\}$ to $E \cup F(\alpha)$, replace x by $f(x)$, and repeat. When *index* $(x) = \alpha(w)$, if $f(x) = x$ set $f(x) = w$.

The following program implements this method:

```
FILL-IN COMPUTATION.
    local v, w, x;
    for i ∈ [1, n] →
      w := α^{-1}(i); f(w) := w; index (w) := i
      for {v, w} ∈ E such that α(v) < i →
        x := v;
        do index (x) < i →
          index (x) := i;
          add {x, w} to E ∪ F(α);
          x := f(x)
        od;
        if f(x) = x → f(x) := w fi
      rof
    rof;
```

This fill-in algorithm can be used not only in Step 2 of the chordality test, but also in the symbolic factorization step of Gaussian elimination on sparse symmetric matrices [16]. If we only want to test for zero fill-in, as is the case in Step 2 of the chordality test, we can restate the algorithm as follows. Compute $f(v)$ for every vertex v. For every $\{v, w\} \in E$ such that $v <_\alpha w$, verify that either $\{f(v), w\} \in E$ or $f(v) = w$. The following program performs this test:

```
TEST FOR ZERO FILL-IN.
    for i ∈ [1, n] →
        w := α^{-1}(i); f(w) := w; index(w) := i;
        for v such that {v, w} ∈ E and α(v) < i →
            index(v) := i; if f(v) = v → f(v) := w fi
        rof;
        for v such that {v, w} ∈ E and α(v) < i →
            if index(f(v)) < i → reject fi
        rof;
    rof;
    accept;
```

Since the zero-fill-in test terminates as soon as it detects a fill-in edge, it runs in $O(n+m)$ time, rather than in the $O(n+m')$ time needed to actually compute the fill-in.

This completes our implementation of Steps 1 and 2 and gives us a simple $O(n+m)$-time chordality test.

Remark. In both the fill-in computation and the zero-fill-in test we can replace the array *index* by a bit array, say *test*, such that *test* (v) = **true** when vertex w is processed if and only if *index* $(v) = \alpha(w)$. This saves space but requires an extra pass after each vertex is processed, to reset *test* (v) to **false** for each v such that $\{v, w\} \in E \cup F(\alpha)$ and $v <_\alpha w$.

3. Testing acyclicity of hypergraphs. Let $H = (V, E)$ be a hypergraph. We shall assume without loss of generality that every edge is nonempty and every vertex is contained in at least one edge. H is acyclic if and only if either of the following equivalent conditions holds [1]:

(1) All the vertices of H can be deleted by repeatedly applying the following two operations:
   (i) delete a vertex that occurs in only one edge;
   (ii) delete an edge that is contained in another edge.

(2) There is a forest F (called the *join forest*) with the edges of H as vertices, such that for every vertex v of H, the subgraph of F induced by those vertices (edges of H) that contain v is connected.

Condition (1) leads directly to an algorithm for testing the acyclicity of a hypergraph [10], [20]. Another algorithm based on condition (2) appears in [4]. Both of these algorithms run in time quadratic in the size of the hypergraph. Yannakakis [19] has given a linear-time algorithm based on the definition of acyclicity, using techniques from the RTL algorithm. We shall use the techniques of § 2 to obtain a simplified linear-time test.

Our algorithm for testing hypergraph acyclicity consists of two steps, analogous to those of the graph chordality test:

Step 1. Compute an ordering α of $G(H)$ that is guaranteed to be zero fill-in if H is acyclic.

Step 2. Check that α is zero-fill-in and that H is conformal.

Since $G(H)$ is chordal if H is acyclic, we could carry out Step 1 by applying maximum cardinality search to $G(H)$. However, $G(H)$ may have size quadratic in the size of H; consider for example the case of a hypergraph with only a single edge, containing all the vertices. Therefore we shall use the following variant of maximum cardinality search, which operates directly on H: Number the vertices from n to 1 in decreasing order. As the next vertex to number, select any unnumbered vertex in an edge of H containing as many numbered vertices as possible, breaking ties arbitrarily. We call this method *maximum cardinality search on hypergraphs.*

THEOREM 4. *Suppose H is acyclic. Then any numbering α generated by a maximum cardinality search of H can also be generated by a maximum cardinality search of $G(H)$, and is thus zero-fill-in on $G(H)$.*

Proof. Consider applying maximum cardinality search in parallel to H and $G(H)$. We must show that each time we choose a vertex v to number in H, we can choose the same vertex to number in $G(H)$. Suppose i vertices have been numbered identically in H and $G(H)$, for some $i \geq 0$. For any vertex v, let $B(v)$ be the set of numbered vertices adjacent to v.

We first prove that if w is any vertex that can be numbered next in $G(H)$, then there is some edge R of H such that $B(w)$ is exactly the set of numbered vertices in R. Let w be such a vertex, i.e., suppose $|B(w)|$ is maximum among unnumbered vertices. Since $G(H)$ is chordal and maximum cardinality search generates zero-fill-in numberings on chordal graphs, every pair of vertices in $B(w)$ must be adjacent in $G(H)$. Thus $\{w\} \cup B(w)$ is a clique in $G(H)$. Since H is conformal there is some edge R of H such that $\{w\} \cup B(w) \subseteq R$. Furthermore R cannot contain any numbered vertices not in $B(w)$, since all vertices in R are adjacent to w in $G(H)$. This means that $B(w)$ is exactly the set of numbered vertices in R.

Now let w and R be as above and suppose v can be numbered next in H; that is, v is an unnumbered vertex in an edge S of H containing as many numbered vertices as possible. Then S contains at least as many numbered vertices as R, and since every numbered vertex in S is adjacent to v in $G(H)$, the choice of w means that $|B(v)| = |B(w)|$. Thus v can be numbered next in $G(H)$. □

During a maximum cardinality search of a hypergraph, we call an edge *exhausted* if all vertices contained in it are numbered and *nonexhausted* otherwise. If R is a nonexhausted edge containing as many numbered vertices as possible and we number a vertex in R, then if R is still nonexhausted it still contains as many numbered vertices as possible. Thus after selecting a nonexhausted edge having as many numbered vertices as possible, we can number all its unnumbered vertices consecutively before selecting another edge. This restricted form of maximum cardinality search facilitates testing $G(H)$ for chordality and H for conformity. The following program implements this method. (See Figs. 3 and 4.) In addition to numbering the vertices, the program performs the following computations: It numbers the selected edges from 1 to k in order of their selection; if S is the ith edge selected, then $S = R(i)$ and $\beta(S) = i$. It extends this numbering to the vertices by defining $\beta(v) = \min\{\beta(R) | R \text{ is selected and } v \in R\}$. Note that $v <_\beta w$ implies $v >_\alpha w$. Finally, for each edge S it computes $\gamma(S)$, defined to be $\max\{\beta(v) | v \in S\}$ if S is not among the selected edges and $\max\{\beta(v) | v \in S \text{ and } \beta(v) < \beta(S)\}$ if S is among the selected edges. (If $\beta(v) = \beta(S)$ for all $v \in S$, $\gamma(S)$ is undefined.) As an aid to the computation, the program maintains *size* (S) for each edge S, which is the count of numbered vertices in S if S is nonexhausted and minus one if S is exhausted. For $i \in [0, n-1]$, *set* (i) is the set of nonexhausted edges containing i numbered vertices. Index j is the maximum i such that *set* (i) is nonempty; index k counts the number of numbered edges.

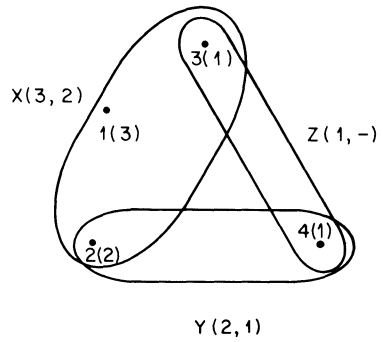

FIG. 3. *Restricted maximum cardinality search of a hypergraph. Numbers on vertices are α-numbers; numbers in parentheses are β-numbers. The first number associated with an edge is its β-number, if any; the second number is its γ-number. Edge Y fails the acyclicity test, since $\gamma(X) = \beta(Y)$ but $X \cap \{v | \beta(v) < \beta(Y)\} = \{3\} \not\subseteq Y$.*

RESTRICTED MAXIMUM CARDINALITY SEARCH ON HYPERGRAPHS.

```
local i, j, k, S;
for i ∈ [0, n−1] → set (i) := ∅ rof;
for S ∈ E → size (S) := 0; γ(S) := undefined; add S to set (0) rof;
i := n+1; j := k := 0;
do j ≧ 0 →
  S := delete any from set (j);
  β(S) := k := k+1; R(k) := S; size (S) := −1;
  for v ∈ S such that v is unnumbered →
    α(v) := i := i+1; β(v) := k
    for S ∈ E such that v ∈ S and size (S) ≧ 0 →
      γ(S) := k;
      delete S from set (size (S));
      size (S) := size (S)+1;
      if size (S) < |S| → add S to set (size (S))
      □ size (S) = |S| → size (S) := −1
      fi
    rof
  rof;
  j := j+1;
  do j ≧ 0 and set (j) = ∅ → j := j−1 od
od;
```

Remark. The assignment "$\alpha(v) := i := i-1$" in this program is a sequential assignment that computes $i-1$ and assigns its value to i and then to $\alpha(v)$.

As it happens, we can test the acyclicity of H using only β and γ; α is unnecessary and its computation can be dropped from the program. The following theorem gives our acyclicity test (see Figs. 3 and 4):

THEOREM 5. *H is acyclic if and only if for each $i \in [1, k]$ and each edge S such that $\gamma(S) = i$, $S \cap \{v | \beta(v) < i\} \subseteq R(i)$. (Since $\beta(v) = i$ implies $v \in R(i)$, this condition is equivalent to $S \cap \{v | \beta(v) \leqq i\} \subseteq R(i)$.)*

Proof. Suppose H is acyclic. Then α is a zero fill-in ordering of $G(H)$. Consider any $i \in [1, k]$. Suppose $\gamma(S) = i$. Then there is a vertex $u \in S$ such that $\beta(u) = i$. Vertex

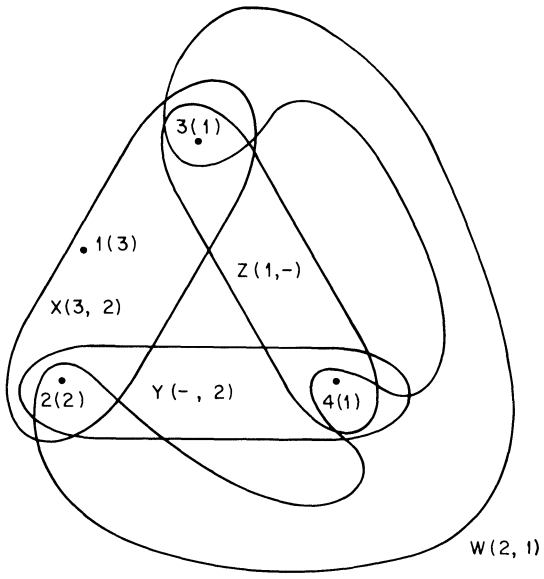

FIG. 4. *Restricted maximum cardinality search of an acyclic hypergraph. Notation is as in Fig. 3. All edges pass the acyclicity test. Note that edge Y is never selected.*

u and its adjacent larger numbered vertices (with respect to α) form a clique in $G(H)$, which since H is conformal must be contained in an edge of E, say T. Since $u \in S \cap R(i)$, $S \cap \{v|\beta(v) < i\} \subseteq S \cap \{v|\alpha(v) \geqq \alpha(u)\} \subseteq T$ and $R(i) \cap \{v|\alpha(v) \geqq \alpha(u)\} \subseteq T$. When u is numbered $R(i)$ contains at least as many numbered vertices as T, which means $R(i) \cap \{v|\alpha(v) \geqq \alpha(u)\} = T \cap \{v|\alpha(v) \geqq \alpha(u)\}$, and $S \cap \{v|\beta(v) < i\} \subseteq R(i)$.

To prove the converse, we use acyclicity condition (1). Suppose that for each $i \in [1, k]$ and each edge S such that $\gamma(S) = i$, $S \cap \{v|\beta(v) < i\} \subseteq R(i)$. We delete vertices and edges from H by processing the sets $R(i)$ from $i = k$ to $i = 1$. To process $R(i)$, we delete every set S such that $\gamma(S) = i$ and every vertex v such that $\beta(v) = i$. This deletion method maintains the following invariants: Just before a set $R(i)$ is processed, every remaining vertex v has $\beta(v) \leqq i$. Every remaining set S such that $\gamma(S) = i$ thus satisfies $S = S \cap \{v|\beta(v) \leqq i\} \subseteq R(i)$, and by rule (ii) S can be deleted when $R(i)$ is processed. Once all such sets are deleted, a vertex v with $\beta(v) = i$ is contained only in $R(i)$ and can be deleted by rule (i). (A set $S \neq R(i)$ containing v has $\gamma(S) \geqq i$; if $\gamma(S) > i$, S was deleted previously.) Thus we can delete all the vertices of H, and H is acyclic. □

We obtain a linear-time algorithm from Theorem 5 by performing together all the set inclusion tests involving a given $R(i)$.

ACYCLICITY TEST.
for $i \in [1, k] \rightarrow$
  for $S \in E$ **such that** $\gamma(S) = i \rightarrow$
    if $S \cap \{v|\beta(v) < i\} \not\subseteq R(i) \rightarrow$ *reject* **fi**
  rof
rof;
accept;

The following program fills in the details of this method. During testing of a set $R(i)$, $index(v) = i$ if $v \in R(i)$; $index(v) < i$ if $v \notin R(i)$.

```
for v ∈ V → index(v) := 0 rof;
for i ∈ [1, k] →
  for v ∈ R(i) → index(v) := i rof;
  for S ∈ E such that γ(S) = i →
    for v ∈ S → if β(v) < i and index(v) < i → reject fi rof
  rof
rof;
accept;
```

This gives an $O(n+m)$-time acyclicity test.

Remark. As in the fill-in computation, we can replace the array index by a bit array if we are willing to reset it after processing each $R(i)$.

If the hypergraph is acyclic we can construct a join forest for it (acyclicity condition (2)) from the parameters γ. Beeri et al. [2] show that if a hypergraph H passes the acyclicity test (1) then a join forest F for H can be constructed as follows: The vertices of F are the edges of H; if an edge R was deleted by operation (ii) of the acyclicity test (1) because it was contained in some other edge S, then F has an edge $\{R, S\}$. (If there were several edges containing R when R was deleted, then we arbitrarily pick one such edge S.) From the proof of Theorem 5 is follows that the forest F with the edges of H as vertices and with edges $\{S, R(\gamma(S))\}$ for all S with $\gamma(S)$ defined is a join forest for the acyclic hypergraph H.

4. Selectively reducing an acyclic hypergraph. We conclude this paper by considering the following problem. Suppose we are given a hypergraph H and a set of marked vertices. We wish to *selectively reduce* H by repeating the following two operations until neither is applicable:

(i) delete an unmarked vertex that occurs in only one edge;

(ii) delete an edge that is contained in another edge.

Selective reduction is necessary in computing queries in acyclic relational data bases. A relational data base is a collection of relations; each relation is a table over some set of attributes. A relational data base can be modeled by a hypergraph whose vertices are the attributes and whose edges are the relations (see [1], [21] for more information). Suppose now that we want to compute the relationship among the attributes in a given set X. If the hypergraph is acyclic this is done as follows: First, we mark the vertices of X and selectively reduce the hypergraph H. After the reduction we are left with a collection of partial edges (i.e. subsets of some of the original edges) $Y_1, \cdots, Y_k$. (It is easy to see that the selective reduction process has the Church–Rosser property; that is, the final collection of partial edges is the same, regardless of the order in which the operations (i), (ii) are applied.) For each Y_i we find all the edges of H that contain Y_i. We project the relations corresponding to such edges onto the attributes Y_i and take the union of the resulting relations, giving a single relation for each Y_i. Then we take the join of the relations obtained for $Y_1, \cdots, Y_k$ (see [13], [18]).

As an example of selective reduction, consider a hypothetical data base for storing information about research papers. Figure 5 illustrates the relations, which correspond to the schemes $R_1 = \{AU, AF\}$, $R_2 = \{PT, AU\}$, $R_3 = \{PT, Y, JN, VN\}$, $R_4 = \{PT, Y, CN\}$, $R_5 = \{CN, CL, Y\}$, $R_6 = \{PT, S\}$, where the abbreviations are as indicated

(a)

AU = AUTHOR
AF = AFFILIATION
CN = CONFERENCE NAME
JN = JOURNAL NAME
CL = CONFERENCE LOCATION
VN = JOURNAL VOLUME, PAGE NUMBERS
Y = PUBLICATION YEAR
PT = PAPER TITLE
S = SUBJECTS

(b)

AU	AF
FRED	IBM
SUE	BELL

PT	AU
NICE RESULTS	FRED
NICER RESULTS	SUE

PT	Y	JN	VN
NICE RESULTS	1982	SICOMP	10, 820-830

PT	Y	CN
NICER RESULTS	1983	STOC

CN	CL	Y
STOC	BOSTON	1983

PT	S
NICE RESULTS	SORTING
NICER RESULTS	SORTING

FIG. 5. *A relational data base for research papers.* (a) *Abbreviations of attributes.* (b) *Relations. Only the entries for two papers are shown.*

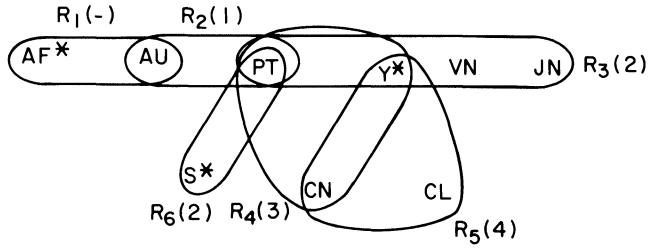

FIG. 6. *Hypergraph corresponding to relational schemes in Fig. 5. Relations are indexed in β-order of a restricted maximum cardinality search. Numbers in parentheses are γ-numbers. Applying the selective reduction algorithm, we delete VN, JN and CL (which are unmarked and in only one edge), then* R_5 *(now contained in* R_4*), then CN, then* R_3*, leaving* R_1, R_2, R_6 *and* $\{PT, Y\}$.

in Fig. 5. Figure 6 shows the hypergraph corresponding to this relation scheme. Suppose we wish to know whether any Bell Laboratories authors published any papers on graph theory in 1983. To answer this query, we must compute the relationship among the attributes in $X = \{AF, S, Y\}$, select those tuples with $AF =$ "Bell Laboratories," $S =$ "graph theory," and $Y =$ "1983," and check whether the result is empty.

After carrying out the appropriate selective reduction, the edges remaining are $R_1 = \{AU, AF\}$, $R_2 = \{PT, AU\}$, $R_6 = \{PT, S\}$, and $\{PT, Y\}$, the last of which is contained in two original edges, R_3 and R_4. Thus we join R_1, R_2, and R_6 with the union of projections of R_3 and R_4 on $\{PT, Y\}$.

Kuper describes a nonlinear algorithm for the selective reduction of an acyclic hypergraph [12]. We shall present a simple linear-time algorithm. In order to solve this problem, we first consider a related but simpler problem: given an acyclic

hypergraph, determine which of its edges are maximal (in the set-theoretic sense). The edges that are not maximal can always be deleted from H in any selective reduction of H, by operation (ii). It is easy to see that if H is acyclic, all its maximal edges are selected during a maximum cardinality search. (Thus any nonselected edge can be deleted during the selective reduction process.) However, not all selected edges need be maximal. We can ensure that only maximal edges are selected by breaking ties according to cardinality (largest edge preferred). However, there seems to be no easy way to incorporate this into the algorithm while preserving the $O(n+m)$ time bound. The following theorem gives an alternative characterization of maximal edges. If R is a selected edge, let R' be the set $\{v \in R | \beta(v) < \beta(R)\}$. (Note that $\gamma(R) = \max\{\beta(v) | v \in R'\}$.)

THEOREM 6. *Suppose we carry out a restricted maximum cardinality search on an acyclic hypergraph H, selecting edges $R(1), R(2), \cdots, R(k)$. Then the maximal edges of H are exactly the selected edges $R(i)$ such that $i = k$ or $|R(i)| > |R'(i+1)|$.*

Proof. Suppose S is a nonselected edge. Let $\gamma(S) = i$. Then $S = S \cap \{v | \beta(v) \leqq i\} \subseteq R(i)$ by Theorem 5, and S is not maximal.

Suppose $R(i)$ for some $i < k$ is maximal. If $\gamma(R(i+1)) = i$, then $R'(i+1) \subseteq R(i)$ by Theorem 5, which means $|R'(i+1)| < |R(i)|$, since $R'(i+1) = R(i)$ would contradict the maximality of $R(i)$. If $\gamma(R(i+1)) \neq i$, then $|R'(i+1)| = |R(i+1) \cap \{v | \beta(v) < i\}| \leqq |R'(i)| < |R(i)|$ by the definition of maximum cardinality search.

Conversely, suppose $R(i)$ is not maximal. Then there is some maximal edge containing $R(i)$, which must be $R(j)$ for some $j > i$. Consider $R(i+1)$. Just before $R(i)$ is selected, $R(i+1)$ contains at most $|R'(i)|$ numbered vertices; just after $R(i)$ is selected, $R(i+1)$ contains at least as many numbered vertices as $R(j)$, and hence at least $|R(i)|$. Since exactly $|R(i) - R'(i)|$ new vertices are numbered when $R(i)$ is selected, $R(i+1)$ contains every vertex in $R(i) - R'(i)$, and $|R'(i+1)| = |R(i)|$. □

We can compute the maximal edges of an acyclic hypergraph in $O(n+m)$ time by carrying out a restricted maximum cardinality search and applying the test in Theorem 6 to each selected edge. Suppose we have done this. Let $R(1)$, $R(2), \cdots, R(k)$ be the list of selected edges according to the restricted maximum cardinality search on the acyclic hypergraph H, and suppose that $R(i)$ is not maximal. By the proof of Theorem 6, just before $R(i)$ is selected, both $R(i)$ and $R(i+1)$ contain the same number of numbered vertices, and $R(i+1)$ contains all unnumbered vertices in $R(i)$. Thus we could have selected $R(i+1)$ in place of $R(i)$ during the search. After selecting $R(i+1)$, $R(i)$ would be exhausted and not eligible for later selection, and the rest of the edges $R(i+2), \cdots, R(k)$ could be chosen in the same order as before. Thus $R(1), \cdots, R(i-1), R(i+1), \cdots, R(k)$ is also a valid selection order (according to maximum cardinality search) for the hypergraph. This means that if we delete from the list of selected edges all nonmaximal edges we are left with a list that corresponds to a valid selection order. The selective reduction process can always begin by deleting all edges not in this list, since each is contained in some edge in the list.

Redefine $R(1), \cdots, R(k)$ to be the remaining (selected) edges, all incomparable, and suppose that some vertex v occurs in only one edge $R(i)$. Consider what happens if we delete v. The remaining hypergraph H' is also acyclic [2]. If the new $R(i)$ (without v) contains only vertices that appear in earlier sets $R(j)$ (i.e. with $j < i$), then $R(i)$ is exhausted when its turn comes, and therefore cannot be selected. However, since v occurs only in $R(i)$ no other edge is affected and therefore $R(1), \cdots, R(i-1)$, $R(i+1), \cdots, R(k)$ is a valid selection order for H'. Since $R(i)$ is not selected and H' is acyclic we know from the previous section that the new $R(i)$ (without v) is contained in some other edge $R(j)$.

If the new $R(i)$ contains some vertex that does not appear in any earlier $R(j)$, then the same order $R(1), \cdots, R(i), \cdots, R(k)$ continues being a valid selection order for H'. From Theorem 6 we know that all these edges are incomparable, unless $|R(i)| = |R'(i+1)|$, in which case $R(i) \subseteq R(i+1)$ and $R(i)$ can be dropped from the list of selected edges without violating the validity of the selection order.

Our discussion suggests the following algorithm for the selective reduction of an acyclic hypergraph. (See Fig. 6.)

Step 1. Carry out a restricted maximum cardinality search of the hypergraph. Apply Theorem 6 to discard the selected edges that are not maximal. Let $R(1), \cdots, R(k)$ be the list of remaining edges, and let $|R'(i)|$ be as defined before Theorem 6. For each unmarked vertex v, compute the number of remaining (selected) edges in which v appears.

Step 2. Repeat the following operation until it does not apply:

Delete any unmarked vertex v that occurs in exactly one edge $R(i)$; decrement $|R(i)|$. If $|R(i)| = |R'(i)|$ or $|R(i)| = |R'(i+1)|$ then do the following: Delete $R(i)$. For each vertex $v \in R(i)$, decrement the count of edges in which v appears. Decrease $|R'(i+1)|$ by $|R(i)| - |R'(i)|$. For $j \geqq i$, replace $R(j)$ by $R(j+1)$.

The selected edges that remain after the execution of these steps are exactly those that cannot be deleted by the selective reduction process. If we maintain a list of the unmarked vertices occurring in exactly one edge and maintain the set of remaining edges $R(1)$, $R(2) \cdots$ as a doubly linked list, we can carry out this computation in $O(n+m)$ time. We leave the implementation of this algorithm as an exercise. The correctness of the method follows from our previous discussion.

Note that in the data base application, Step 1 will be executed only once (when the data base is set up), and only Step 2 will be executed to answer a query. Let $Y_1, \cdots, Y_k$ be the remaining sets of vertices when the algorithm terminates. We have to find now for each Y_i the edges of the original hypergraph H that contain Y_i. Let W be the set of remaining vertices, i.e. the union of the Y_i's. Let $\hat{H}$ be the hypergraph with set of vertices W and an edge $\hat{S} = S \cap W$ for each edge S of H. Then $\hat{H}$ is acyclic [2]. (Strictly speaking, $\hat{H}$ is a multihypergraph; i.e., it might have edges that consist of exactly the same vertices. However, everything we have said about hypergraphs also holds for multihypergraphs.) A valid selection order for $\hat{H}$ according to maximum cardinality search is $Y_1, \cdots, Y_k$. For a vertex v in W, let $\hat{\beta}(v) = \min\{i | v \in Y_i\}$, and for an edge S of H let $\hat{\gamma}(S) = \max\{\hat{\beta}(v) | v \in \hat{S}\}$. From the last section we know that $\hat{S} = S \cap W \subseteq Y_{\hat{\gamma}(S)}$ for each edge S of H. If S contains some Y_i then this can be only $Y_{\hat{\gamma}(S)}$ (since all the Y_j's are incomparable), and this is true if and only if $|S \cap W| = |Y_{\hat{\gamma}(S)}|$. Therefore, all we have to do is compute $\hat{\beta}(v)$ for each $v \in W$, compute $\hat{\gamma}(S)$ for each edge S of H, and compare $|S \cap W|$ to $|Y_{\hat{\gamma}(S)}|$.

REFERENCES

[1] C. BEERI, R. FAGIN, D. MAIER, A. MENDELZON, J. D. ULLMAN AND M. YANNAKAKIS, *Properties of acyclic database schemes*, in Proc. 13th Annual ACM Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1981, pp. 355–362.

[2] C. BEERI, R. FAGIN, D. MAIER AND M. YANNAKAKIS, *On the desirability of acyclic database schemes*, J. Assoc. Comput., 30 (1983), pp 479–513.

[3] C. BERGE, *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1973.

[4] P. A. BERNSTEIN AND N. GOODMAN, *The power of natural semijoins*, this Journal, 10 (1981), pp. 751–771.

[5] E. W. DIJKSTRA, *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

[6] G. A. DIRAC, *On rigid circuit graphs*, Abh. Math. Sem. Univ. Hamburg, 25 (1961), pp 71–76.
[7] R. FAGIN, A. O. MENDELZON AND J. D. ULLMAN, *A simplified universal relation assumption and its properties*, ACM Trans. Database Systems, 7 (1982), 343–360.
[8] D. R. FULKERSON AND O. A. GROSS, *Incidence matrices and interval graphs*, Pacific J. Math., 15 (1965), pp. 835–855.
[9] M. GOODMAN AND O. SHMUELI, *Syntactic characterizations of tree database schemes*, J. Assoc. Comput., Mach., 30 (1983), 767–786.
[10] M. H. GRAHAM, *On the universal relation*, Technical Report, Univ. of Toronto, Toronto, Ontario, Canada, 1979.
[11] F. G. GUSTAVSON, *Some basic techniques for solving sparse systems of linear equations*, in Sparse Matrices and Their Applications, D. S. Rose and R. A. Willoughby, eds., Plenum Press, New York, 1972, pp. 41–52.
[12] G. KUPER, *An algorithm for reducing acyclic hypergraphs*, unpublished manuscript, Stanford Univ., Stanford, CA, 1982.
[13] D. MAIER AND J. D. ULLMAN, *Connections in acyclic hypergraphs*, in Proc. ACM Symposium on Principles of Database Systems, Association for Computing Machinery, New York, 1982, pp. 34–39.
[14] D. J. ROSE, *Triangulated graphs and the elimination process*, J. Math. Anal. Appl., 32 (1970), pp. 597–609.
[15] D. J. ROSE, R. E. TARJAN AND G. S. LUEKER, *Algorithmic aspects of vertex elimination on graphs*, this Journal, 5 (1976), pp. 266–283.
[16] R. E. TARJAN, *Graph theory and Gaussian elimination*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 3–22.
[17] G. WHITTEN, private communication, 1978.
[18] M. YANNAKAKIS, *Algorithms for acyclic database schemes*, Proc. International Conference on Very Large Data Bases, 1981, pp. 82–94.
[19] M. YANNAKAKIS, *A linear-time algorithm for recognizing acyclic hypergraphs*, unpublished manuscript, 1982.
[20] C. T. YU AND M. Z. OZSOYOGLU, *An algorithm for tree-query membership of a distributed query*, in Proc. 1979 IEEE COMPSAC, Institute of Electrical and Electronic Engineers, New York, 1979, pp. 306–312.
[21] C. ZANIOLO, *Analysis and design of relational schemata for database systems*, Ph.D. thesis, Univ. of California at Los Angeles, Los Angeles, CA, 1976.