Discrete Applied Mathematics 45 (1993) 51-62
North-Holland

Cycle structure of edge labelled graphs

James S. Diamond* and Alberto O. Mendelzon

Department of Computer Science, University of Toronto, Toronto, Ont., Canada M5S 1A4

Received 5 September 1984
Revised 2 November 1990

Abstract

Diamond, J.S. and A.O. Mendelzon, Cycle structure of edge labelled graphs, Discrete Applied Mathematics 45 (1993) 51-62.

In this paper we examine the cyclic structure of graphs with edges labelled by elements of a partial order under the operation of deleting any edge whose label is less than or equal to all labels of edges of some cycle containing that edge. We show that all graphs obtained after repeating the above operation as many times as possible have similar structure with respect to the number of edges remaining and thus, in particular, the presence or absence of cycles. To show this, we apply some results of matroid theory. We also give an efficient algorithm to determine whether or not the resulting graph is acyclic. As an application to relational database theory, we show how this algorithm can be combined with a new characterization of database acyclicity to determine the cyclicity of a relational database scheme.

1. Introduction

The subject of cycles is one of the better-studied aspects in graph theory. In this paper we examine the effect of an edge deletion operation upon the cyclic structure of graphs. Specifically, we start with a graph whose edges are labelled with elements from a partially ordered set. We then find an edge e and a cycle C containing e such that e's label is less than or equal to all edge labels in C, and we delete e. We continue this process until there are no more such e and C. Notice that, in general, at any step the choice of e will not be unique. We prove that the particular edge

Correspondence to: Dr. J.S. Diamond, Defence Research Establishment Atlantic, P.O. Box 1012, Dartmouth, N.S., Canada B2Y 3Z7.

* Author's current address is: Defence Research Establishment Atlantic, P.O. Box 1012, Dartmouth, N.S., Canada B2Y 3Z7.

chosen does not affect the size of the resulting graph, and thus, in particular, its acyclicity. Given the above fact, one may wish to be able to efficiently test whether a labelled graph can be reduced to an acyclic graph using this edge deletion operation; we give an algorithm which answers this question in $O(nm)$ time, where n is the number of vertices in the graph and m is the number of edges.

The motivation for this problem comes originally from the theory of relational databases. A relational database scheme is a finite set of relation schemes. Each relation scheme is a finite set of symbols called attributes. A database scheme models the structure of a set of tables, where each relation scheme corresponds to a table, and each attribute of a relation scheme corresponds to the label of some column of the table. A tuple for a relation scheme R is a function mapping each attribute of R to some domain; tuples represents rows in the database tables. A relation is a finite set of tuples. A database is an assignment of a relation to each relation scheme.

For example, consider the database scheme with the following relation schemes: $R = \{Professor, Student\}$, $S = \{Student, Topic\}$ and $T = \{Professor, Topic\}$. Relation R is intended to store information about who is whose advisor, S lists each student's thesis topic, and T stores each professor's topics of interest. Suppose the following relations constitute the database:

R	
Professor	*Student*
Hilbert	Courant
Church	Rogers

S	
Student	*Topic*
Courant	Applied Math.
Rogers	Logic

T	
Professor	*Topic*
Hilbert	Applied Math.
Hilbert	Logic
Church	Logic

Suppose we want to test whether the database is *consistent* in the following sense: for every professor P, student S, and topic T, P supervises S if and only if S's topic is T and P is interested in T. Note that the database given above is not consistent in this sense. For example, Hilbert is interested in Logic, and Rogers' topic is Logic, but Hilbert does not supervise Rogers.

How hard is it to test for consistency of the database? If there are only two relation schemes in the database scheme, then consistency amounts to checking that the two relations have the same projections on their common attributes. For example, R and S above are consistent, since their projections on the *Topic* attribute are both equal to {Applied Math., Logic}. In fact, any two of R, S, and T are consistent in the example above. Consistency of two relations with m and n tuples respectively can obviously be tested in time $O(nm)$. However, consistency of an arbitrary number of relations is NP-complete [5].

Note that consistency of the database always implies pairwise consistency of all relation pairs in the database, but the converse is not true, as the example shows. If we could characterize the class of database schemes such that pairwise consistency implies consistency, we would be guaranteed an easy test for consistency for members of that class. Such a class can be described as follows. Let each attribute be a node in a hypergraph and each relation scheme be a hyperedge. We apply the following algorithm to the hypergraph. Let R and S be two hyperedges, and suppose that the attributes in $R-S$ appear in no other hyperedge. Then we remove R from the hypergraph. The *GYO-reduction* of a hypergraph is the result of applying this process until it is no longer possible. If the result is the empty hypergraph, we say the original hypergraph is *acyclic*. Note that this is not equivalent to the traditional definition of hypergraph acyclicity given by Berge [2]. It can be viewed as a generalization of the idea that a connected graph is acyclic if it can be reduced to the empty graph by removing leaves. Acyclic hypergraphs enjoy many remarkable properties; one of them is that pairwise consistency is equivalent to consistency for arbitrary databases if and only if the database scheme hypergraph is acyclic.

In this paper we provide a new characterization of acyclicity. Construct the intersection graph of the database scheme, i.e., a graph in which the vertices are the relation schemes and there is an edge labelled with $R_i \cap R_j$ between any two schemes R_i and R_j that have a nonempty intersection. Then the scheme is acyclic if and only if the cycle deletion algorithm described at the beginning of this introduction reduces the graph to a tree.

The organization of this paper is as follows: we define needed graph and matroid theoretic terms in Section 2. In Section 3 we prove our graph reduction result using the tools of matroid theory; a purely graph theoretic approach can be found in [4]. In Section 4 our algorithm for efficient determination of the cyclicity of the graph resulting from a maximal set of edge deletions (under the above rule) is given. In Section 5, we give a new characterization of hypergraph acyclicity in terms of a certain graph that represents the pairwise consistency constraints on the database. We then show how our result on edge deletion can be applied to this graph to test for database acyclicity. This is followed by the conclusions in Section 6.

2. Definitions

We use the usual definitions in graph theory; a graph G is a (finite) set of vertices

$V(G)$ and a (finite) set of edges $E(G)$. Our graphs have no loops or multiple edges, and all the cycles considered are simple.

The *symmetric difference* of two subgraphs H_1 and H_2 of G is denoted by $H_1 \oplus H_2$ and is defined by $V(H_1 \oplus H_2) = V(H_1) \cup V(H_2)$ and $E(H_1 \oplus H_2) = E(H_1) \oplus E(H_2)$, where, of course, $E_1 \oplus E_2 = (E_1 - E_2) \cup (E_2 - E_1)$.

We define $L(e)$ to be the label of edge e; all labels are elements of some partially ordered set (poset). Further, we define $L(G)$ to be $\{L(e)\text{: } e \in E(G)\}$ and for any $F \subseteq E(G)$, $L(F) = \{L(e)\text{: } e \in F\}$. A *least edge e* in a set $F \subseteq E(G)$ is any edge such that $\forall e' \in F$, $L(e) \leq L(e')$. An edge $e \in F$ is a *greatest edge* of F if there is no edge $e' \in F$ such that $L(e) < L(e')$.

A *valid deletion sequence* (VDS) $S = (e_1, e_2, \ldots, e_k)$ of a graph G is a sequence of edges of G such that $\forall e_i \in S$, $\exists$ a cycle $C_i \subseteq (G - \bigcup_{j=1}^{i-1} \{e_j\})$ with e_i a least edge of C_i. Given a graph G and a VDS S for G, define G_i to be $G - \bigcup_{j=1}^{i} \{e_j\}$. Given a graph G and a VDS S for G, define $E_k(U) = \{e \in E(G_k)\text{: } l(e) \in U\}$ for $U \subseteq L(G)$.

A *matroid* is a set of elements E and a collection $\mathscr{C}$ of subsets (called *circuits*) of E which satisfy

(1) If $X \neq Y \in \mathscr{C}$, then $X \not\subseteq Y$.

(2) If C_1 and C_2 are distinct members of $\mathscr{C}$ and $z \in C_1 \cap C_2$ there exists $C_3 \in \mathscr{C}$ such that $C_3 \subseteq (C_1 \cup C_2) - \{z\}$.

Given $F \subseteq E$, the *rank* of F, $r(F)$, is defined to be the size of the largest circuit-free subset of F.

A subset U of a partially ordered set P is said to be an *up-set* if $p \in U$ and $q > p$ implies $q \in U$. Define $U_p = \{q \in P\text{: } q \geq p\}$ and $V_p = \{q \in P\text{: } q > p\}$.

3. The edge deletion process

In this section we prove that the ability to obtain an acyclic graph by repeatedly deleting least edges is independent of both the edges chosen and the order in which they are deleted.

Since the set of cycles of a graph forms the set of circuits of a matroid, we shall give a matroid theoretic proof for the above claim. We begin with the following well-known facts of matroid theory.

Lemma 3.1. *Let $B \subseteq E$ and $e \in B$. Then $r(B) = r(B - \{e\})$ iff some circuit contained in B contains e.*

Lemma 3.2. *Let $B \subseteq E$ and $F \subseteq B$. Then*

(1) *$|F| \geq r(B) - r(B - F)$, and*

(2) *$|F| = r(B) - r(B - F)$ iff every circuit in B is disjoint from F.*

Two more lemmas provide us with a simple proof of our edge deletion result.

Lemma 3.3. *After deleting a maximal VDS from a labelled graph G, the remaining edges satisfy $|E_k(\{p\})| = r(E_k(U_p)) - r(E_k(V_p))$ for each $p \in P$.*

Proof. This follows directly from Lemma 3.2 by setting $F = E_k(\{p\})$ and $B = E_k(U_p)$, since $E_k(V_p) = E_k(U_p) - E_k(\{p\})$. □

Lemma 3.4. *For any up-set U, $r(E_i(U)) = r(E(U))$ for $i \in \{0, \ldots, k\}$.*

Proof. Clearly, this is true for $i = 0$. Assume it is true for some $i < k$ and consider the case of $i+1$. If $l(e_{i+1}) \notin U$, the equality holds since $E_{i+1}(U) = E_i(U)$ and $r(E_i(U)) = r(E(U))$. If $l(e_{i+1}) \in U$, then the fact that e_{i+1} is deletable from the graph whose edge set is E_i implies that $E_i(U)$ contains a circuit through e_{i+1}. Thus, by Lemma 3.1, $r(E_i(U)) = r(E_{i+1}(U))$, giving the desired result. □

We can now present our edge deletion result.

Theorem 3.5. $|E_k(\{p\})| = r(E(U_p)) - r(E(V_p))$.

Proof. Lemma 3.3 states $|E_k(\{p\})| = r(E_k(U_p)) - r(E_k(V_p))$. Further, Lemma 3.4 tells us that $r(E_k(U_p)) = r(E(U_p))$ and $r(E_k(V_p)) = r(E(V_p))$. Combining these three facts gives us the theorem. □

Interpretation: Theorem 3.5 shows that if a particular graph can be reduced to a tree through some valid deletion sequence, then regardless of the edges chosen to be deleted or the order in which they are deleted, the graph can still be reduced to a tree. In other words, in attempting to reduce a graph to a tree, it is not possible to make a "bad choice" at any stage: any edge which can be legally deleted can be safely deleted.

Corollary 3.6. *Every maximal VDS of a graph G has the same length.*

4. An algorithm for testing cyclicity

We now present an algorithm which determines whether there is a VDS which will reduce a given graph to a tree. This algorithm is a modification of the greedy algorithm for the minimum spanning tree problem. We then examine the time complexity of the algorithm and prove the algorithm correct.

Algorithm 4.1.
Input: A graph G whose edges are labelled with elements of some poset.
Output: The cyclicity of G after any maximal VDS has been removed.

```
        Algorithm:
1       Set $\hat{E}=E(G)$.
2       Set $F=\emptyset$.
3       Set $VDS=nil$.
4       **While** $|\hat{E}|>0$ **do**
5       **begin**
6         Choose any greatest edge of $\hat{E}$.
          Call it $e$, and remove it from $\hat{E}$.
7         **If** adding $e$ to $F$ would form a cycle $C$
          **then**
8           **if** $e$ is not a least edge of $C$
            **then**
            **begin**
9             output "Cyclic".
10            halt.
            **end**
            **else**
11            Append $e$ to $VDS$.
          **else**
12          Add $e$ to $F$.
        **end**
13      output "Acyclic".
14      halt.
```

Description of the algorithm. A useful way to view this algorithm is to consider it as a process which examines edges of G one at a time. Edges e that can be deleted in the graph $F\cup e$ are appended to VDS; otherwise they are added to the (initially empty) forest F. As shown below in the proof of correctness, each edge need only be considered as a candidate for deletion once; if e is not a least edge of some cycle in $F\cup e$ when it is first examined, it need never be considered for deletion again. Thus when a cycle is formed in F, one can conclude that there is no VDS reducing G to a tree.

Timing analysis. The body of the while loop can be executed at most m times, where $m=|E(G)|$. Choosing a greatest edge can be done in $O(\log m)$ time by storing the edges in a priority queue (see [1]), provided that comparisons of elements in the partial order can be done in constant time. Each iteration of the while loop asks once whether the end vertices of e are in the same tree of the forest F. Further, at most one edge is added to F in each pass through the loop. The union-find algorithm in [1] will process cm such requests in time at most $c'mH(m)$, where c and c' are constants, c' depending on c, and H is defined as follows. Define a function F by $F(0)=1$ and $F(i)=2^{F(i-1)}$ for $i>0$. Define $H(n)$ to be the smallest integer k such that $F(k)\geq n$. To quote [1], "the function H grows extremely slowly". For more details see [1, Chapter 4]. Appending e to VDS can be done in constant time using a linked list representation for *VDS*. If a cycle is found, the newest edge can be checked for minimality in $O(n)$ time, where $n = |V(G)|$. Assuming that $n \leq m$ (and noting that this implies $\log m \leq n$), we see that the overall time bound is $O(mn)$.

Proof of correctness. Clearly, if the algorithm outputs "Acyclic", G can be reduced to an acyclic graph with the valid deletion sequence *VDS*. Consider the case in which the algorithm outputs "Cyclic". We must show that there will be at least one cycle left after removing the edges of any maximal VDS from G. Consider the graph G' defined to be $G - VDS$, i.e., all of G save the edges deleted by the algorithm when the unbreakable cycle C is found. Considering the interpretive comment following Theorem 3.5, we see that G can be reduced to a tree iff G' can be reduced to a tree. Consider the cycle C. As it has no least edge, there must be some edge $e' \in C$ whose label is incomparable to that of e. Notice that any cycle in G' in which e (respectively, e') is a least edge must have at least one other edge not in C with the same label as e (respectively, e'); this follows from the fact that both e and e' are members of only one cycle in F, and edges in G' that are not in F cannot have labels greater (in the partial order) than those of e or e'. Keeping the interpretive comment in mind, for any such cycles, delete the least edge which is not in C. After all such cycles have been broken, the cycle C remains. Thus G' cannot be reduced to a tree, and hence G cannot be reduced to a tree. Thus the algorithm terminates with the correct answer. □

5. An application to relational database theory

5.1. Definitions

A *database scheme* $\mathcal{R}$ is a finite collection of finite sets of symbols called *attributes*. Each such set of symbols is a *relation scheme*. The union of all the attributes is the *universe* U. A *relation* r on relation scheme R is a finite set of *tuples*; a tuple on R is a mapping from R into, say, the integers. For r a relation on R, the *projection* of r on $X \subseteq R$, denoted by $\pi_X r$, is obtained by restricting each tuple in r to X. A *database state* ϱ of $\mathcal{R}$ is an assignment of a relation on R to each $R \in \mathcal{R}$. We write $\varrho = \langle r_1, \ldots, r_n \rangle$, where r_i is a relation on relation scheme $R_i \in \mathcal{R}$.

As mentioned in the introduction, a database scheme $\mathcal{R}$ can be viewed naturally as a hypergraph whose vertices are elements of U and whose hyperedges are the R_i. There is no unique way to generalize the concept of cyclicity from graphs to hypergraphs; database theorists have found a notion called α-acyclicity (acyclicity for short in this paper) that has many remarkable properties. (See [6] for details.) The *GYO reduction* [6] of a database scheme is computed by applying the following operation as often as possible:

delete any relation scheme R such that there exists a relation scheme S such that the attributes in $R-S$ appear in no other relation scheme.

A database scheme is *acyclic* if its GYO reduction is the empty relation scheme.

Example 5.1. The database scheme $\mathscr{R}1=\{\{ABC\},\{CDE\},\{AEF\},\{ACE\}\}$ is acyclic, whereas the scheme $\mathscr{R}2=\{\{ABC\},\{CDE\},\{AEF\},\{ACD\}\}$ is cyclic.

A *typed inclusion dependency* (id from now on) on database scheme $\mathscr{R}$ is a statement of the form $R_i \subseteq_X R_j$, where $R_i, R_j \in \mathscr{R}$ and $X \subseteq R_i \cap R_j$. A state ϱ *satisfies* this id if $\pi_X r_i \subseteq \pi_X r_j$. A *typed equality dependency* (ed from now on) on database scheme $\mathscr{R}$ is a statement of the form $R_i =_X R_j$, with the same restrictions as above. A state ϱ satisfies this ed if $\pi_X r_i = \pi_X r_j$. Thus every ed can be expressed as the conjunction of two id's. A set of ed's E *implies* another set F if every database state that satisfies E also satisfies F. A set of ed's is *nonredundant* if it contains no proper subset that implies the whole set. We say F is a *nonredundant cover* of E if F is nonredundant and F implies E.

We associate with each set of ed's E an undirected graph $G(E)$. $G(E)$ has a node for each R_i that appears in E. For each ed $R_i =_X R_j$, $G(E)$ contains an edge between R_i and R_j labelled with X.

Given a database scheme $\mathscr{R}$, we let

$$S_{\mathscr{R}} = \{R_i =_{R_i \cap R_j} R_j \colon 1 \le i < j \le n\}.$$

5.2. *A characterization of acyclicity*

To prove our acyclicity result, we will use the following characterization of acyclic database schemes. Let a *join graph* for $\mathscr{R}$ be any undirected graph G with a node for each R_i in $\mathscr{R}$ such that, for every attribute A, the subgraph of G induced by all the nodes that contain A is connected.

Example 5.2. Consider the relation schemes given in Example 5.1. Two join graphs for $\mathscr{R}1$ are given in Fig. 1, and two join graphs for $\mathscr{R}2$ are given in Fig. 2. Notice

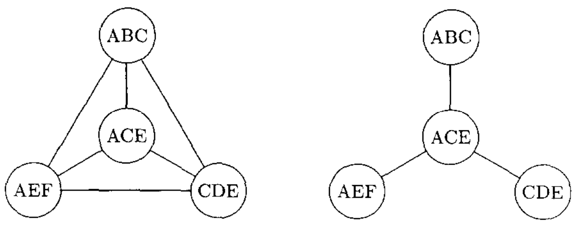

Fig. 1.

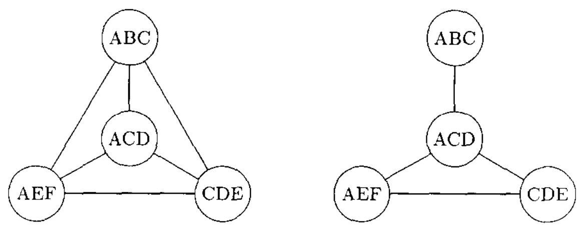

Fig. 2.

that the second join graph for $\mathscr{R}1$ is a tree. On the other hand, no subgraph of the first join graph in Fig. 2 (which is a maximal join graph) is both a tree and a join graph for $\mathscr{R}2$.

Fact 5.3. *$\mathscr{R}$ is acyclic if and only if there exists an acyclic join graph for $\mathscr{R}$.*

Proof. See [3]. □

Lemma 5.4. *Let G be the graph for a set of ed's E. Then G implies an ed $R =_X S$ if and only if there is a path from R to S in G such that every edge label on the path contains X.*

Proof. See [7]. □

Theorem 5.5. *$\mathscr{R}$ is acyclic if and only if there exists a nonredundant cover $N_{\mathscr{R}}$ for $S_{\mathscr{R}}$ such that $G(N_{\mathscr{R}})$ is acyclic.*

Proof. (Only if) If $\mathscr{R}$ is acyclic, by Lemma 5.4 there is a join graph J for $\mathscr{R}$ that is acyclic. Let $N_{\mathscr{R}}$ be the set of all ed's $R_i =_X R_j$ such that there is an edge between R_i and R_j in J labelled with X. We claim that $N_{\mathscr{R}}$ is a nonredundant cover for $S_{\mathscr{R}}$. To show that $N_{\mathscr{R}}$ implies $S_{\mathscr{R}}$, let $R =_Y S$ be any ed in $S_{\mathscr{R}}$, with $Y = R \cap S$. Let $Y = A_1, \ldots, A_p$. Since J is a join graph, there are p paths between R and S such that A_i appears in every node of the ith path. Since J is acyclic, all these paths are the same. That is, there is a path between R and S in which every label contains Y; by Lemma 5.4, it follows that $N_{\mathscr{R}}$ implies $R =_Y S$. To see that $N_{\mathscr{R}}$ is nonredundant, suppose we delete any ed from $N_{\mathscr{R}}$. Because J is acyclic, the two relation schemes that were related by that ed are now in different connected components, so, by Lemma 5.4, the resulting set of ed's cannot imply the whole set.

(If) We show that $G(N_{\mathscr{R}})$ is an acyclic join graph for $\mathscr{R}$. In proof, let A be an attribute and S_1, S_2 be any two relation schemes containing A. Clearly $S_{\mathscr{R}}$ implies $S_1 =_A S_2$, and therefore, so does $N_{\mathscr{R}}$. By Lemma 5.4, there is a path between S_1 and S_2 containing A. Therefore $G(N_{\mathscr{R}})$ is an acyclic join graph for $\mathscr{R}$, hence $\mathscr{R}$ is acyclic by Fact 5.3. □

Our next result is that the graphs $G(N_{\mathscr{R}})$ for $\mathscr{R}$ are exactly the graphs that result when a maximal valid deletion sequence is removed from $G(N_{\mathscr{R}})$.

Theorem 5.6. *Let H be a subgraph of $G(S_{\mathscr{R}})$. H is $G(N_{\mathscr{R}})$ for some nonredundant cover $N_{\mathscr{R}}$ of $S_{\mathscr{R}}$ if and only if H can be obtained by removing a maximal VDS from $G(S_{\mathscr{R}})$.*

Proof. (If) It follows from Lemma 5.4 that each edge that gets deleted from $G(S_{\mathscr{R}})$ corresponds to an ed that is implied by the remaining edges of the graph. Hence, by induction, the ed's in the resulting graph imply all the ed's in the original graph. To see that H corresponds to a nonredundant cover, suppose there is some edge in H, corresponding to the ed $R =_X S$, that is implied by the other edges in H. Again by Lemma 5.4, this implies there is a path between R and S such that every edge label on the path contains X. Hence this edge cannot be in the resulting graph, since it would have been deleted by the algorithm.

(Only if) We show first that $G(N_{\mathscr{R}})$ is always a subgraph of $G(S_{\mathscr{R}})$. Indeed, suppose $G(N_{\mathscr{R}})$ contains some edge corresponding to the ed $R =_X S$, and this edge is not in $G(S_{\mathscr{R}})$. By definition of $S_{\mathscr{R}}$, this implies that X is a proper subset of $R \cap S$. But since $G(N_{\mathscr{R}})$ must imply the ed $R =_{R\cap S} S$, by Lemma 5.4 there must be some path between R and S where every edge label contains $R \cap S$. This would imply that the ed $R =_X S$ is redundant in $N_{\mathscr{R}}$, a contradiction. Since $G(N_{\mathscr{R}})$ is now known to be a subset of $G(S_{\mathscr{R}})$, it is clear that every edge deleted from $G(S_{\mathscr{R}})$ to obtain $G(N_{\mathscr{R}})$ is redundant and thus, by Lemma 5.4, will be deleted by the edge deletion algorithm. Furthermore, the edge deletion algorithm cannot delete any edges from $G(N_{\mathscr{R}})$. It follows that $G(N_{\mathscr{R}})$ is a result of applying the edge deletion algorithm. □

There is one last important fact to be stressed. Theorem 5.6 shows that if there is an acyclic (and thus nonredundant) cover $N_{\mathscr{R}}$ for $S_{\mathscr{R}}$ then this cover can be found by removing a maximal VDS from $G(S_{\mathscr{R}})$. However, Theorem 3.5 shows that if there exists one VDS reducing $G(S_{\mathscr{R}})$ to a tree, then any maximal VDS of $G(S_{\mathscr{R}})$ reduces it to a tree. Thus to test a database scheme for acyclicity we merely form $G(S_{\mathscr{R}})$ and run Algorithm 4.1 on this graph.

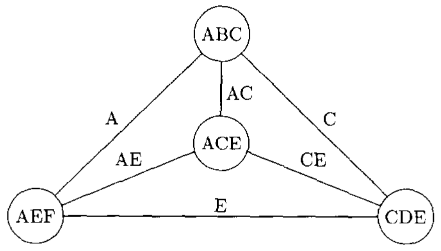

Fig. 3.

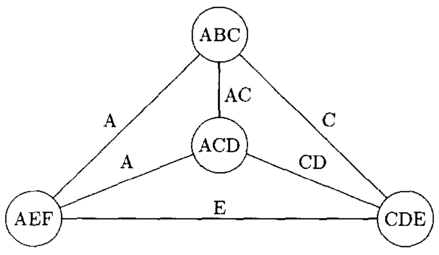

Fig. 4.

Example 5.7. Consider the graph in Fig. 3; this is the complete join graph of $\mathcal{R}1$ previously given in Fig. 1, except that edge labels corresponding to the intersection of the end vertex labels have been added.

Let us refer to an edge by its label. In arbitrary order, we can move the edges "AC", "AE" and "CE" (which are the three greatest edges) from $\hat{E}$ to F without creating a cycle. Now consider, say, the edge "A". This forms a cycle C with the edges "AC" and "AE", but as "A" is a least edge of C, it is appended to *VDS*. Similarly, we can append the edges "C" and "E" to *VDS*. This exhausts $\hat{E}$, and the algorithm terminates, announcing "Acyclic"; thus $\mathcal{R}1$ is an acyclic database scheme.

Now consider the graph in Fig. 4, which is the complete join graph of $\mathcal{R}2$ with edge labels added. The greatest edges of $\hat{E}$ are "AC", "CD" and "E"; these can be added to F without creating a cycle. A greatest remaining edge in $\hat{E}$ is the edge labelled "C". Adding this edge to F creates a cycle, but as this edge is a least edge of the cycle it creates, it can be instead appended to *VDS*. Next choose (say) the edge labelled "A" joining the vertices labelled "ABC" and "AEF". Adding this edge to F would create a cycle with edge labels "A", "AC", "CD" and "E". As "A" is not a least edge of this cycle, the algorithm outputs "Cyclic" and halts. Thus $\mathcal{R}2$ is a cyclic database scheme.

6. Conclusions

We have shown that the order of deletion of least edges in cycles is not important with respect to the number of such possible deletions. We have also shown that there exists an efficient algorithm to decide whether or not a graph can be reduced to a tree by using this edge deletion operation. These two results add to the body of knowledge concerning cycles in graphs.

The intersection graph is a natural representation of the structure of a collection of sets. By viewing the intersection graph of a relational database scheme as a set of statements called equality dependencies, it is natural to consider nonredundant covers of such sets. We have shown that the database scheme is acyclic if and only if some such cover is an acyclic graph. As the graph theory result points out, if one cover is acyclic, then all covers are.

We should point out that efficient tests for acyclicity of database schemes are already known. Bernstein and Goodman have a test for the existence of a join tree [3] in time better than ours, and Tarjan and Yannakakis [8] have given a linear time acyclicity test using a different characterization. However, neither of these methods seem to generalize readily to the edge deletion process on general graphs, that is, graphs not derived from database schemes.

Acknowledgement

We would like to thank Lorna Stewart for a careful reading of the paper, and Derek Corneil, Kent Laver and Eric Mendelsohn for many useful discussions. The authors would also like to thank the referee for pointing out the matroid theoretic proof of Theorem 3.5 which is more succinct than the authors' original graph theoretic proof.

References

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

[2] C. Berge, Graphs and Hypergraphs (North-Holland, Amsterdam, 1973).

[3] P.A. Bernstein and N. Goodman, Power of natural semijoins, SIAM J. Comput. 10 (1981) 751-771.

[4] J.S. Diamond, Edge deletion in labelled graphs, Ph.D. Dissertation, Tech. Rept. No. 189, Department of Computer Science, University of Toronto, Toronto, Ont. (1986).

[5] P. Honeyman, R.E. Ladner and M. Yannakakis, Testing the universal instance assumption, Inform. Process. Lett. 10 (1980) 14-19.

[6] D. Maier, The Theory of Relational Databases (Computer Science Press, Rockville, MD, 1983).

[7] E. Sciore, Inclusion dependencies and the universal instance, in: Proceedings Second ACM SIGACT-SIGMOD Conference on Principles of Database Systems (1983) 48-57.

[8] R.E. Tarjan and M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, SIAM J. Comput. 13 (1984) 566-579.