A certifying algorithm for the consecutive-ones property

Ross M. McConnell *

Abstract

We give a forbidden substructure characterization of set families that have the consecutive-ones property, and a linear time algorithm to find the forbidden substructure if a set family does not have the property. The forbidden substructure has size $O(n)$, where n is the size of the domain. The PQ tree is a well-known data structure for representing all consecutive-ones orderings. We show that it is given by a substitution decomposition of arbitrary set families that has not been described previously. This observation gives a generalization of the PQ tree to arbitrary set families, and we give a linear-time algorithm to compute it.

1 Introduction

A 0-1 matrix as the *consecutive-ones property* if it is possible to order the columns so that, in every row, the 1's form a consecutive interval. Equivalently, a family $\mathcal{F}$ of subsets of a domain V has the consecutive-ones property if it is possible to assign a linear order to V where each member of $\mathcal{F}$ is an interval.

The problem of finding a consecutive-ones ordering, when one exists, is the key step of Booth and Lueker's well-known algorithm for recognizing interval graphs [1]. To do this, Booth and Lueker use the *PQ tree*, which gives an implicit representation of all the consecutive-ones orderings of $\mathcal{F}$. Interval graphs and variations of them have applications in many optimization problems and in molecular biology [7, 2, 13]. PQ trees also have applications to finding planar embeddings of planar graphs [10].

The way the PQ tree represents consecutive-ones orderings is illustrated in Figure 1. The nodes are of two types: P nodes and Q nodes. The children of a P node are unordered, but the children of a Q node are linearly ordered. The consecutive-ones orderings of the matrix are the leaf orders that result from assigning an arbitrary order to children of each P node, and reversing the order of children of some of the Q nodes.

Booth and Lueker's algorithm runs in time linear in the size of the input. It either finds a consecutive-ones ordering or determines that the matrix does not have the consecutive-ones property. A practical problem with
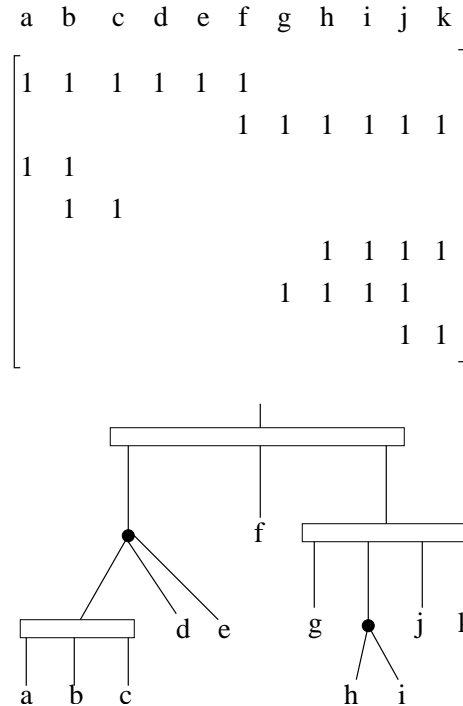

Figure 1: The PQ tree is a way of representing all consecutive-ones orderings of the columns of a matrix. The leaf order of the tree gives a consecutive-ones ordering. Permuting the order of children of a P node (black points) gives a new order. For instance, by permuting the order of the left child of the root, we see that $(d, a, b, c, e, f, g, h, i, j, k)$ is a consecutive-ones ordering. Reversing the order of children of a Q node (horizontal bars) gives a new consecutive-ones order. For instance, by reversing the order of the right child of the root, we see that $(a, b, c, d, e, f, k, j, h, i, g)$ is a consecutive-ones ordering. The consecutive-ones orderings of the matrix are leaf orders that can be obtained by these operations.

---
*rmm@cs.colostate.edu, Computer Science Department, Colorado State University, Fort Collins, CO, 80523-1873 U.S.A.

this is that, although the algorithm has been proven correct, a program that implements it may have bugs. When a program claims that a given matrix does not have the consecutive-ones property, the user cannot tell whether an error has occurred.

A *certifying algorithm* is an algorithm for a decision problem that provides a piece of evidence (a *certificate*) that allows its answer to be easily checked. More discussion of the practical importance of certifying algorithms can be found in [17, 14, 9].

A variety of algorithms for the consecutive-ones problem have previously appeared in the literature. All of these give a certificate in the form of a consecutive-ones ordering when a matrix has the property, but reject it without supporting evidence when it does not.

Tucker [16] gives a forbidden substructure characterization of consecutive-ones matrices in terms of *asteroidal triples*, but does not give an algorithm for finding such a forbidden structure when the family fails to have the consecutive-ones property. We give an alternative characterization, and a linear-time algorithm for finding it. The size of the certificate is $O(n)$.

Our certificate is derived as follows. Given a family $\mathcal{F}$ of subsets of a domain V, a consecutive-ones ordering $(x_1, x_2, ..., x_n)$ of V defines a *consecutive-ones relation* $R = \{(x_i, x_j) | i < j\}$.

Suppose a consecutive-ones ordering is not known. For $a, b \in V$, (a, b) and (b, a) cannot appear in the same consecutive-ones relation – they are incompatible. Suppose that $a, b, c \in V$ and there exists $X \in \mathcal{F}$ such that $a, c \in X$ and $b \notin X$. Then (a, b) and (b, c) cannot appear in the same consecutive-ones relation, since this would imply that b goes between a and c in the corresponding consecutive-ones ordering, which would imply that X is not consecutively ordered, a contradiction. Therefore, (a, b) and (b, c) are incompatible in this case.

We define the *incompatibility graph* to be an undirected graph whose vertices are the elements of $\{(a, b) | a, b \in V \text{ and } a \neq b\}$, and whose edges are the pairs of vertices that are incompatible by the above observations. (See Figure 2).

A consecutive-ones relation must have no incompatible pairs, so it is an independent set in this graph that consists of half of the vertices. The reverse of a consecutive-ones ordering is also a consecutive-ones ordering, so the remaining vertices of the incompatibility graph must also be an independent set. Therefore, the incompatibility graph must be bipartite whenever the matrix has a consecutive-ones ordering. We show that this is also a sufficient condition: the incompatibility graph fails to be bipartite whenever a set family fails to have the consecutive-ones property.
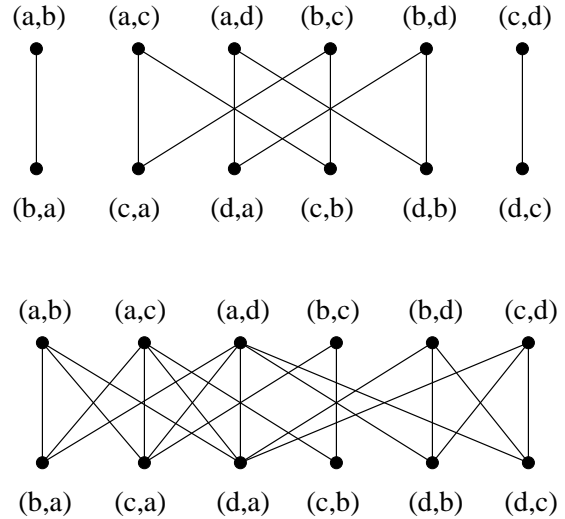
It follows that when a set family does not have the

Figure 2: The incompatibility graph $I_C(\mathcal{F})$ for $\mathcal{F} = \{\{a, b\}, \{b, c, d\}, \{a, b, c\}\}$. The upper figure shows the incompatibilities induced by just the first set, $\{a, b\}$, as well as the rule that (x, y) and (y, x) are incompatible. The lower figure shows the whole graph, and is obtained by adding incompatibilities induced by the other sets. $\mathcal{F}$ has the consecutive-ones property iff its incompatibility graph is bipartite.

consecutive-ones property, the incompatibility graph must have an odd cycle. Our algorithm returns an odd cycle of length $O(n)$. Each edge of the cycle is labeled with a set from the family that documents the incompatibility.

The size of the incompatibility graph is not necessarily linear in the sum of cardinalities of the set family, but it is unnecessary to construct it explicitly, either to find the odd cycle or to verify the certificate.

We also describe a type of *substitution decomposition* of arbitrary 0-1 matrices and set families. This gives a generalization of PQ trees to arbitrary 0-1 matrices or set families. We give a linear-time algorithm to compute this generalized PQ tree. The algorithm is based on elementary operations such as radix sorting, a simple partition-refinement operation, off-line least common ancestors, and a graph traversal algorithm such as DFS.

2 Preliminaries

The problem of consecutively ordering columns of a matrix reduces to that of ordering the domain V of a set family $\mathcal{F}$ so that each member of $\mathcal{F}$ is an interval on the ordering.

Let $n = |V|$ and let $m = Size(\mathcal{F})$ denote the sum of cardinalities of members of $\mathcal{F}$. An algorithm on $\mathcal{F}$ is

linear if it runs in $O(n + m)$ time.

The *symmetric difference* of two sets is $A \Delta B = (A \cup B) - (A \cap B)$. Two sets X and Y *overlap* if they intersect but neither is a subset of the other. That is, they overlap if $X - Y$, $X \cap Y$, and $Y - X$ are all nonempty.

If X and Y are disjoint sets, $a \in X$ and $b \in Y$, then $\{a, b\}$ *goes between* X and Y, and (a, b) *goes from* X *to* Y.

$\mathcal{F}$ is *tree-like* if $\emptyset \notin \mathcal{F}$, $V \in \mathcal{F}$, $\{x\} \in \mathcal{F}$ for all $x \in V$, and for all $X, Y \in \mathcal{F}$, X and Y do not overlap.

LEMMA 2.1. *The Hasse diagram of the subset relation on a tree-like family $\mathcal{F}$ on domain V is a tree, and if X is a nonempty subset of V and does not overlap any member of $\mathcal{F}$, then X is a union of one or more children of a node in this tree.*

Let us call the Hasse diagram of such a family the family's *inclusion tree*. $\mathcal{F}$ is a *weakly partitive family* if:

- $V \in \mathcal{F}$, $\emptyset \notin \mathcal{F}$, and for all $v \in V$, $\{v\} \in \mathcal{F}$

- For all $X, Y \in \mathcal{F}$, if X and Y overlap, then $X \cap Y \in \mathcal{F}$, $X \cup Y \in \mathcal{F}$, $X - Y \in \mathcal{F}$, and $Y - X \in \mathcal{F}$.

A member of a weakly partitive family $\mathcal{F}$ is *strong* if it overlaps with no other member of $\mathcal{F}$. Though $\mathcal{F}$ is not a tree-like family, its strong members are. Let $T(\mathcal{F})$ denote their inclusion tree.

THEOREM 2.1. *[3, 15] Let $\mathcal{F}$ be a weakly partitive family, let X be an internal node of $T(\mathcal{F})$, and let $S_1, S_2, ..., S_k$ be the children of X. Then X is of one of the following three types:*

- **Degenerate:** *For every $I \subset \{1, \ldots k\}$ such that $1 < |I| < k$, $\bigcup_{i \in I} S_i \in \mathcal{F}$*

- **Prime:** *For every $I \subset \{1, \ldots k\}$ such that $1 < |I| < k$, $\bigcup_{i \in I} S_i \notin \mathcal{F}$*

- **Linear:** *There exists an ordering of $\{1, 2, ..., k\}$ such that if $I \subset \{1, \ldots k\}$ and $1 < |I| < k$, then $\bigcup_{i \in I} S_i \in \mathcal{F}$ iff the members of I are consecutive in the ordering.*

By Lemma 2.1 and Theorem 2.1, if $\mathcal{F}$ is a weak partitive family, $Y \subseteq V$ is a member of $\mathcal{F}$ iff it is either a node of $T(\mathcal{F})$, the union of a set of children children of a degenerate node of $T(\mathcal{F})$, or the union of a consecutive set of children in the ordering of a linear node. By $\mathcal{F}$'s *decomposition tree*, we denote $T(\mathcal{F})$, the labeling of its internal nodes as prime, degenerate, or linear, and the linear ordering of children of linear nodes. This gives a representation of $\mathcal{F}$ in $O(n)$ space.

$\mathcal{F}$ is a *strongly partitive family* if:

- $V \in \mathcal{F}$, $\emptyset \notin \mathcal{F}$, and for all $v \in V$, $\{v\} \in \mathcal{F}$

- For all $X, Y \in \mathcal{F}$, if X and Y overlap, then $X \cap Y \in \mathcal{F}$, $X \cup Y \in \mathcal{F}$ and $X \Delta Y \in \mathcal{F}$.

A strongly partitive family is weakly partitive, so it has a decomposition tree. A weakly partitive family is strongly partitive iff its decomposition tree has no linear nodes.

A *transitive orientation* of an undirected graph is an assignment of orientations to its edges such that whenever (a, b) and (b, c) are directed edges into and out of b, then (a, c) is also a directed edge.

The presence of $\emptyset$ in a set family is irrelevant to the consecutive ones property or the decomposition that we describe. For simplicity, we will assume that $\emptyset \notin \mathcal{F}$ except where stated otherwise.

3 Partitive families and the PQ tree

DEFINITION 3.1. *If $\mathcal{F}$ is a family of subsets of a universe V, then $\mathcal{F}$'s non-overlapping family, denoted $\mathcal{N}(\mathcal{F})$ is the family of nonempty subsets of V that do not overlap with any member of $\mathcal{F}$.*

THEOREM 3.1. *If $\mathcal{F}$ is an arbitrary set family, then $\mathcal{N}(\mathcal{F})$ is a strongly partitive set family.*

THEOREM 3.2. *[8] If $\mathcal{F}$ has the consecutive-ones property, the PQ tree is the decomposition tree of $\mathcal{N}(\mathcal{F})$, where the prime nodes are interpreted as the Q nodes and the degenerate nodes are interpreted as the P nodes.*

In [8], Theorem 3.1 is given for sets that have the consecutive-ones property, but the proof is general to arbitrary set families. This characterization of the PQ tree in Theorem 3.2 is not completely satisfactory because it says nothing about the ordering of children of Q nodes. We now give a related characterization that captures this property also.

DEFINITION 3.2. *If $\mathcal{F}$ is an arbitrary set family on domain V such that $\emptyset \notin \mathcal{F}$, let its weak closure, $\mathcal{W}(\mathcal{F})$, and strong closure, $\mathcal{S}(\mathcal{F})$, denote the smallest weakly partitive family and strongly partitive family, respectively, that contain $\mathcal{F}$ as a subfamily.*

LEMMA 3.1. *If $\mathcal{F}$ is an arbitrary set family on domain V, then $\mathcal{W}(\mathcal{F})$ and $\mathcal{S}(\mathcal{F})$ are unique.*

Proof. We prove the lemma for the weak closure; the proof for the strong closure is similar. Let $\mathcal{A} := \mathcal{F} \cup \{V\} \cup \{\{x\} | x \in V\}$. While $\mathcal{A}$ is not a partitive family, execute the following *closure* operation:

- Select overlapping members X and Y of $\mathcal{A}$ such that $\{X \cap Y, X \cup Y, X - Y, Y - X\}$ are not all in $\mathcal{A}$. Let $\mathcal{A} := \mathcal{A} \cup \{X \cap Y, X \cup Y, X - Y, Y - X\}$.

When this procedure halts, the family is weakly partitive, and by induction on the number of iterations, it is contained in every other weakly partitive family that contains $\mathcal{F}$

THEOREM 3.3. *If $\mathcal{F}$ is an arbitrary set family on domain V, then the decomposition tree of $\mathcal{S}(\mathcal{F})$ is obtained from the decomposition tree of $\mathcal{N}(\mathcal{F})$ by relabeling each prime node as degenerate and vice versa.*

An interesting observation is that if $\mathcal{F}$ is itself a strongly partitive family, then $\mathcal{S}(\mathcal{F}) = \mathcal{F}$, hence $\mathcal{N}(\mathcal{N}(\mathcal{F})) = \mathcal{F}$. This associates each strongly partitive family $\mathcal{F}$ with *dual* $\mathcal{N}(\mathcal{F})$, and this relationship is symmetric. A partitive family is its own dual iff its decomposition tree is binary.

THEOREM 3.4. *If $\mathcal{F}$ is an arbitrary set family on domain V, then the decomposition tree of $\mathcal{S}(\mathcal{F})$ is obtained from the decomposition tree of $\mathcal{W}(\mathcal{F})$ by relabeling all linear nodes as degenerate.*

THEOREM 3.5. *If $\mathcal{F}$ is a set family on domain V that has the consecutive-ones property, then its PQ tree is the decomposition tree of $\mathcal{W}(\mathcal{F})$, where the prime nodes are interpreted as P nodes and the linear nodes, together with the linear ordering on their children, are interpreted as Q nodes.*

DEFINITION 3.3. *The* generalized PQ tree *of an arbitrary set family $\mathcal{F}$ is the decomposition tree of $\mathcal{W}(\mathcal{F})$.*

THEOREM 3.6. *A set family has the consecutive-ones property if and only if its generalized PQ tree has no degenerate nodes.*

By Theorem 3.5, the standard PQ tree is just the special case of a generalized PQ tree in the case that the set family has the consecutive-ones property.

Theorems 3.3, 3.4, and 3.5 give an algorithm for finding the generalized PQ tree of an arbitrary set family $\mathcal{F}$: find the decomposition tree of $\mathcal{N}(\mathcal{F})$, swap the roles of prime and degenerate nodes to obtain the decomposition of $\mathcal{S}(\mathcal{F})$, and then determine which of the degenerate nodes in this tree are linear in the decomposition of $\mathcal{W}(\mathcal{F})$, as well as the linear order on children of these nodes. This is the basis of our algorithm.

4   PQ trees as a substitution decomposition

The best-known example of a substitution decomposition is defined by the *modules* of a graph. A *module* of an undirected graph $G = (V, E)$ is a set X of vertices that all have the same neighbors in $V - X$. The modules

form a strongly partitive family, and their decomposition tree is the one given by Theorem 2.1.

Suppose $\mathcal{P}$ is a partition of the vertex set where every partition class is a module, and S_1 and S_2 are two sets, each consisting of one representative from each member of $\mathcal{P}$. It is easily seen that the subgraphs induced by S_1 and S_2 are isomorphic. Let the *modular quotient $G/\mathcal{P}$* denote this induced subgraph; the choice of representatives is irrelevant. The quotient may be interpreted as a graph where each member of $\mathcal{P}$ is a vertex and two members of $\mathcal{P}$ are adjacent iff there are edges between them in G. A subgraph of G induced by a member of $\mathcal{P}$ is a *factor*. G can easily be reconstructed from the quotient and factors by substituting each factor in the place of the corresponding vertex of the quotient. This is the *composition* operation.

This decomposition satisfies the following:

The Autonomy Rule: If $X \subseteq V$ is a module, then the modules of G that are subsets of X are the same as the modules of $G[X]$.

The Quotient Rule: If $\mathcal{P}$ is a quotient and $\mathcal{X} \subseteq \mathcal{P}$, then $\mathcal{X}$ is a module of $G/\mathcal{P}$ iff $\bigcup X$ is a module of G.

Since the modules are a partitive family, they have a decomposition tree. If X is an internal node and $\mathcal{C}$ are its children, then $G[X]/\mathcal{C}$ is a modular quotient. Collectively these quotients give a representation of G, which can be obtained by working inductively from the leaves to the root, performing substitution operations on these quotients.

Möhring [15] surveys substitution decomposition in a variety of domains, such as boolean functions, k-ary relations, and set families. What these substitution decompositions have in common is a definition of module-like structures that form a strongly or weakly partitive family, definitions of the quotients and factors that have properties analogous to the autonomy and quotient rules, and a substitution operation for reconstructing the original structure from quotients and factors.

We now give a new substitution decomposition on set families whose decomposition tree is the generalized PQ tree. The module-like structures are just the members of $\mathcal{N}(\mathcal{F})$. Note that if $\mathcal{P}$ is a partition of V where every member of $\mathcal{P}$ is a member of $\mathcal{N}(\mathcal{F})$, then every member of $\mathcal{F}$ is either a subset of some member of $\mathcal{P}$ or a union of members of $\mathcal{P}$.

DEFINITION 4.1. *Let $\mathcal{F}$ be an arbitrary set family on domain V, and let $\mathcal{P}$ be a partition of V where every member of $\mathcal{P}$ is a member of $\mathcal{N}(\mathcal{F})$.*

1. If $X \in \mathcal{P}$, the factor *induced by X is the family of members of $\mathcal{F}$ that are subsets of X.*

2. *The* quotient $\mathcal{F}/\mathcal{P}$ *is the family* $\mathcal{Q} = \{\mathcal{X}|\mathcal{X}$ *is a nonempty subfamily of* $\mathcal{P}$ *and* $\bigcup \mathcal{X} \in \mathcal{F}\}$.

The quotient can be represented by $\mathcal{Q} = \{X \cap S | X \in \mathcal{F}\}$, where S consists of one representative element from each member of $\mathcal{P}$. The choice of representatives is irrelevant as they all give isomorphic set families. It is easily seen that, given these definitions, the autonomy and quotient rules are valid. Therefore, we may represent an arbitrary set family $\mathcal{F}$ with its decomposition tree, together with a labeling of the nodes of the tree with quotients. Figure 3 gives an illustration on a family that has the consecutive-ones property.

This substitution decomposition we give above differs from previous ones. For instance, Möhring defines one on set families whose composition operation works as follows. Let the quotient $\mathcal{C}' = \{\{a,b\},\{b,c\},\{c,d\}\}$ and the factors be $\mathcal{C}_a = \{\{1,2\}\{2,3\}\}$, $\mathcal{C}_b = \{\{4,5\}\}$, $\mathcal{C}_c = \{\{6\}\}$, and $\mathcal{C}_d = \{\{7,8\},\{8,9,10\}\}$. To perform the composition on $\{a,b\}$ and its factors, one must generate unions derived from the Cartesian product of the factors $\mathcal{C}_a$ and $\mathcal{C}_b$, namely $\{1,2,4,5\}$ and $\{2,3,4,5\}$. Doing this to all members of $\mathcal{C}'$ yields $\{\{1,2,4,5\},\{2,3,4,5\},\{4,5,6\},\{6,7,8\},\{6,8,9,10\}\}$.

The composition operation on most examples of substitution decomposition, including modular decomposition, use a composition operation based on a Cartesian product. By contrast, the one we give above just replaces a set in the quotient with the *union* of domains of the corresponding factors, as well as including each member of a factor in the composition.

5 Constructing the generalized PQ tree

We now describe a linear-time algorithm for finding the generalized PQ tree in linear time. By Theorem 3.6, this gives the PQ tree for a consecutive-ones matrix in linear time, but it also computes the generalized PQ tree when it is not. The algorithm consists of two parts: finding the nodes of the PQ tree and finding the ordering of children at the linear (Q) nodes.

5.1 Finding the nodes of the generalized PQ tree If $\mathcal{F}$ is a set family, let its *overlap graph* $G_o(\mathcal{F})$ be the graph that has one vertex for each member of $\mathcal{F}$ and an edge between two vertices iff the corresponding members of $\mathcal{F}$ overlap.

Given a connected component $\mathcal{C}$ of $G_o(\mathcal{F})$, let $B_{\mathcal{C}}$ be an equivalence relation on $\bigcup \mathcal{C}$, where if $x, y \in \bigcup \mathcal{C}$, then xBy iff the family of members of $\mathcal{C}$ that contain x is the same as the family of members of $\mathcal{C}$ that contains y. Let $\mathcal{C}$'s *blocks* be the equivalence classes of $B_{\mathcal{C}}$.

THEOREM 5.1. *If $\mathcal{F}$ is a set family on domain V such that $\emptyset \notin \mathcal{F}$, then $X \subseteq V$ is a node of the decomposition*
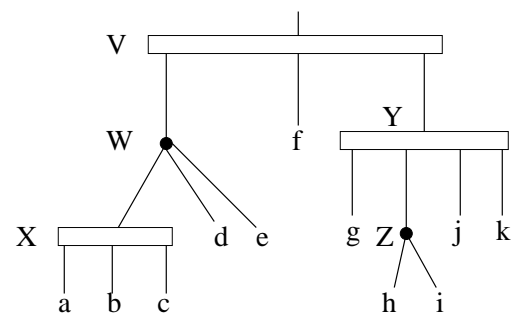
Figure 3: The PQ tree gives a substitution decomposition of an arbitrary set family into quotient families. Here, the family has the consecutive-ones property and is displayed in matrix form.

tree of $\mathcal{N}(\mathcal{F})$ iff it is one of the following:

1. V or a one-element subset of V;

2. $\bigcup \mathcal{C}$ for some connected component $\mathcal{C}$ of $\mathcal{F}$'s overlap graph;

3. A block of a connected component of $\mathcal{F}$'s overlap graph.

The following theorem, due to Dahlhaus, is critical for our time bound:

THEOREM 5.2. *[5] It takes $O(n+m)$ time to find $\mathcal{F}$'s overlap components.*

Dahlhaus's very clever algorithm uses only radix sorting, off-line least-common ancestors [4], and a graph traversal algorithm such as depth-first search.

We may then find the blocks of the connected components as follows. Let k be the number of overlap components. Number them from 1 to k. Number the members of each overlap component $\mathcal{C}$ from 1 to $|\mathcal{C}|$. Number the elements of V from 1 to $|V|$. For each member X of $\mathcal{F}$, and each element $y \in X$, create a triple (y, i, j) where i is the component number of X and j is X's number label. Radix sorting all such triples with y as the primary key, i as the secondary key, and j as the tertiary key gives, for each element y of V and each overlap component, a sorted list $(j_1, j_2, ..., j_k)$ of members of the component that y belongs to. Making y the "owner" of this list and prepending the component number to each of these lists and then putting them in lexical order makes the blocks consecutive, and the blocks are the owners of each group of identical lists in the ordering.

By Theorem 5.1, this gives the set $\mathcal{N}$ of nodes of the PQ tree. To construct the inclusion tree, radix sort the members of $\mathcal{N}$ by size. Then, create a list, for each $x \in V$, of the members of $\mathcal{N}$ that contain x, in ascending order of size. This can be accomplished by visiting each $Y \in \mathcal{N}$ in descending order of size, and for each $x \in Y$, inserting a pointer to Y to the front of x's list. Then, visit each member x of V, putting a parent pointer from each member of x's list to its successor in x's list if there isn't already one, as these are the chain of ancestors of $\{x\}$.

5.2 Finding a spanning tree of each overlap component Dahlhaus's algorithm [5] finds the overlap components without actually computing the overlap graph or even spanning trees of its components. However, it is not hard to modify the algorithm to get a spanning tree of each component, as we show next.

His algorithm, when run on a set family $\mathcal{H}$, begins creating a list L of the members of $\mathcal{H}$, sorted in ascending order of cardinality. Then, for each member X of $\mathcal{H}$, if X overlaps with some member of $\mathcal{H}$ that lies to the right of X in L, he computes the rightmost member Y of L that X overlaps with. Let us denote Y by $Max(X)$. $Max(X)$ is undefined if Y does not exist.

If $C \in \mathcal{H}$ and $Max(C)$ is defined, then C and $Max(C)$ are adjacent in the overlap graph. A key observation is the following:

LEMMA 5.1. *If $C, D \in \mathcal{H}$, $C \cap D \neq \emptyset$, and D lies between C and $Max(C)$ in L, then D overlaps either C or $Max(C)$.*

By this lemma, D can be safely placed in the same connected component as C and $Max(C)$ without finding which of C and $Max(C)$ overlaps D.

His algorithm creates a surrogate graph $G_C(\mathcal{H})$ that is based on this lemma and whose connected components are the same as the connected components of the overlap graph of $\mathcal{H}$. For $x \in V$, let $L_x = C_1^x, C_2^x ... C_k^x$ be the subsequence of L defined by those sets that contain x. For all $x \in V$, C_i^x and C_{i+1}^x such that there exists $j \leq i$ such that either $Max(C_j^x) = C_{i+1}^x$ or $Max(C_j^x)$ lies to the right of C_{i+1}^x in L, then $C_i^x C_{i+1}^x$ is an edge of $G_C(\mathcal{H})$. In this case, let C_j^x be a *witness* for the edge.

Dahlhaus shows that $G_C(\mathcal{H})$ can be constructed in linear time. Let us modify the algorithm so that it labels each edge of $G_C(\mathcal{H})$ with a witness. The algorithm knows the witness when the edge is installed, so this does not affect the time bound.

To find a spanning tree of a component $\mathcal{C}$, we traverse the component of $G_C(\mathcal{F})$ using a graph traversal algorithm such as depth-first search. While doing this, we grow a spanning tree of the overlap component. We add some nodes before we actually visit them in $G_C(\mathcal{F})$, and whenever we visit a node, we add it to the spanning tree if it has not already been added.

An observation that we will use is that if U is a set and W is a set such that $|W| \leq |U|$, it takes $O(|U|)$ time to find whether W and U overlap using an initialized array of size $|V|$ that is left initialized after the operation.

Suppose by induction that we have visited k nodes and successfully added them to a partial spanning tree of the overlap component. It suffices to describe how to add a node D to the partial spanning tree when we travel to it along an edge CD from a node C that is already part of the spanning tree. Let A be the witness on CD and let $B = Max(A)$. Note that $|A| \leq C, D \leq B = |Max(A)|$.

If neither A nor B is in the spanning tree yet, find

whether C overlaps B; if it does, establish CB as a spanning-tree edge, and if it does not, establish CA as a spanning-tree edge. This takes $O(|B|)$ time. Then establish AB as a spanning-tree edge in $O(1)$ time. This brings A and B into the spanning tree. If one of A and B is already in the spanning tree, just establish AB as a spanning-tree edge in $O(1)$ time. Now find whether D overlaps A in $O(|D|)$ time. If it does, establish DA as a spanning-tree edge, and if it does not, establish DB as a spanning-tree edge. The total time is $O(1)$ plus time proportional to the sum of cardinalities of the sets that it brings into the spanning tree. Therefore, the total time to find the spanning tree of the component is $O(k)$, where k is the sum of cardinalities of the members of the component, in addition to a one-time $O(n)$ initialization that does not need to be repeated for each component. This gives the following:

THEOREM 5.3. *It takes $O(n + m)$ time to find a spanning tree of each overlap component of an arbitrary set family $\mathcal{F}$.*

5.3 Finding the orderings of children of Q nodes We now have the decomposition tree of $\mathcal{N}(\mathcal{F})$. By Theorem 3.3, we get the decomposition tree of the strong closure $\mathcal{S}(\mathcal{F})$ of $\mathcal{F}$ by relabeling every prime node degenerate and every degenerate node prime. By Theorem 3.4, we may obtain the generalized PQ tree by finding which degenerate nodes should be relabeled linear in the decomposition tree of $\mathcal{W}(\mathcal{F})$, and the linear order on children of these nodes. This gives the generalized PQ tree.

Let a set family be *simple* if its generalized PQ tree has only one internal node, the root. A simple family is *degenerate*, *linear*, or *prime*, depending on whether this root is degenerate, linear, or prime, respectively. A simple family is prime iff very member is either its domain or a singleton subset, so these families can be recognized in linear time.

Each degenerate node X of the decomposition tree of $\mathcal{S}(\mathcal{F})$ is either a degenerate or a linear node in the decomposition tree of $\mathcal{W}(\mathcal{F})$. We must distinguish these two cases. By the autonomy and quotient rules, X's quotient $\mathcal{Q}$ in the tree is simple and either degenerate or linear. The various quotients are restrictions of disjoint subfamilies of $\mathcal{F}$, so it suffices to give a linear-time algorithm that, given a simple family, determines its type. We solve the problem by giving an algorithm to distinguish a simple degenerate from a simple linear family.

We use a partitioning procedure that is related to one given in [11, 12] for finding a transitive orientation of a graph.

Let $\mathcal{Q}'$ be a simple degenerate or linear family, and

let $\mathcal{Q}$ be the result of removing V and its one-element subsets. Clearly, $\mathcal{Q}$ has a single overlap component. The algorithm performs one *pivot* on each $X \in \mathcal{Q}$. Once a pivot has been performed on X, it has been *processed* and is not used again. The algorithm keeps an linearly ordered family $\mathcal{P}$ of disjoint subsets of V. That is, the members of $\mathcal{P}$ are kept in a linearly ordered list. Let $S = V - \bigcup \mathcal{P}$ be those members of V that are not in any processed set.

By Theorem 5.1, $\mathcal{Q}$ has a single overlap component. By Theorem 5.3, we may compute a spanning tree T of this overlap component in linear time.

The purpose of each pivot is to refine the partition $\mathcal{P}$ and to remove members of S to create a new class of $\mathcal{P}$. It does this while maintaining the invariant that there exists a consecutive-ones ordering such that whenever a is a member of an earlier partition class of $\mathcal{P}$ than b, a is earlier than b in the consecutive-ones ordering. That is, the ordering on $\mathcal{P}$ is *consistent* with a consecutive-ones ordering.

The following are the invariants that the algorithm maintains:

1. The processed members of $\mathcal{Q}$ induce a subtree of the spanning tree T of the overlap component.

2. The members of $\mathcal{P}$ are the blocks of the processed members of $\mathcal{F}$

3. The ordering of members of $\mathcal{P}$ is consistent with a consecutive-ones ordering on the processed members of $\mathcal{Q}$.

4. The ordering of members of $\mathcal{P}$ and its reverse are the only possible orderings of $\mathcal{P}$ that are consistent with a consecutive-ones ordering on the processed members of $\mathcal{Q}$.

The initial pivot occurs on arbitrary $X \in \mathcal{Q}$, and creates an initial partition $\mathcal{P} = (X)$, with $S = V - X$. This establishes the invariants initially.

Suppose that after $i - 1$ pivots, the invariants hold. Let $\mathcal{P} = (X_1, X_2, ..., X_k)$. We select Z to be an unprocessed neighbor of a processed node in the spanning tree T, and let $\mathcal{H}$ be the already processed members of $\mathcal{Q}$. Then, by invariant 4, $\mathcal{H} \cup \{Z\}$ can only have the consecutive-ones property if one of the following cases occurs:

1. S_Z is empty and $X_{i+1}...X_{j-1}$ are contained in Z,

2. S_Z is nonempty, $i = 1$, and $X_1, ..., X_{j-1}$ are contained in Z,

3. S_Z is nonempty, $j = k$, and $X_{i+1}, ..., X_j$ are contained in Z.

If these conditions are not met, we declare $\mathcal{Q}$ degenerate. Otherwise, the pivot on Z in case 2 consists of prepending S_Z as a new partition class to the beginning of $(X_1, X_2, ..., X_k)$ to obtain $(S_Z, X_1, X_2, ..., X_k)$, and then replacing X_j in the ordering with $X_j \cap Z$ and $X_j - Z$, in that order, to obtain $(S_Z, X_1, X_2, ..., X_{j-1}, X_j \cap Z, X_j - Z, ..., X_k)$. Case 3 is handled in a left-right symmetric way. Case 1 is handled by splitting X_i as in Case 3 and X_j as in Case 2. Any empty members of $\mathcal{P}$ are then discarded.

If $\mathcal{Q}'$ is a simple linear family, then after all members have been processed, $S = \emptyset$ and every member of $\mathcal{P}$ contains a single element of V. The linear ordering on members of $\mathcal{P}$ gives the linear ordering of the domain of $\mathcal{Q}$ by Invariant 3. This gives the linear ordering of children of $X = \bigcup \mathcal{Q}$ in the generalized PQ tree.

For finding the generalized PQ tree of a set family $\mathcal{F}$, we spend linear time on each quotient, and since the quotients are restrictions of disjoint sets of members of $\mathcal{F}$, it takes linear time to find the whole tree.

6 A forbidden substructure characterization

Gallai [6] describes a relation Γ on the edges of an undirected graph that models the constraints that determine whether the graph has a transitive orientation. He described a close relationship that it has to the modular decomposition of the graph. We now describe an analogous relation for the consecutive-ones problem on pairs of elements of the domain of $\mathcal{F}$, and show that it has a similar relationship to the generalized PQ tree.

DEFINITION 6.1. *Let $\mathcal{F}$ be an arbitrary set family on domain V. Let $A_{\mathcal{F}} = \{(a, b) | a, b \in V \text{ and } a \neq b\}$. The* incompatibility graph $I_C(\mathcal{F})$ *of $\mathcal{F}$ is the undirected graph that is defined as follows: (See Figure 2.)*

- *$A_{\mathcal{F}}$ is the vertex set of $I_C(\mathcal{F})$.*

- *The edge set of $I_C(G)$ are pairs of the following forms:*

  - *$\{(a, b), (b, a)\}$*
  - *$\{(a, b), (b, c)\} |$ a, b, $c \in V$ and there exists $X \in \mathcal{F}$ such that $a, c \in X$ and $b \notin X$.*

The following is the basis for our certificate when $\mathcal{F}$ does not have the consecutive-ones property.

THEOREM 6.1. *Let $\mathcal{F}$ be an arbitrary set family on domain V. Then $\mathcal{F}$ has the consecutive-ones property if and only if its incompatibility graph is bipartite, and if it does not have the consecutive-ones property, the incompatibility graph has an odd cycle of length at most $n + 2$.*

The introduction explains why the incompatibility graph must be bipartite when $\mathcal{F}$ has the consecutive-ones property. In the rest of this section, we show the converse of this, which is that it fails to be bipartite when $\mathcal{F}$ does not have the consecutive-ones property.

LEMMA 6.1. *If $Y \in \mathcal{N}(\mathcal{F})$, $(a, b) \in Y$ and $c \notin Y$, then (a, b) is compatible with (b, c).*

Proof. Suppose (a, b) and (b, c) are incompatible. Then there exists $X \in \mathcal{F}$ such that $a, c \in X$ and $b \notin X$. But then X overlaps Y, contradicting Y's membership in $\mathcal{N}(\mathcal{F})$.

This means that any connected component of the incompatibility graph consists of pairs that all have the same least-common ancestor in the decomposition tree of $\mathcal{N}(\mathcal{F})$. Moreover, if this ancestor is degenerate, then they all go between some pair of children of this ancestor.

To prove Theorem 6.1, we modify our algorithm for producing the generalized PQ tree so that it returns an odd cycle of length at most $n + 2$ when it discovers that $\mathcal{F}$ does not have the consecutive-ones property. We represent the odd cycle in the form of a cyclic sequence of elements of the form $((a, b), (b, c), X)$, where X is a member of $\mathcal{F}$ that contains a and c but does not contain b. To check the answer, the user may verify that the cycle has odd length and that the witnesses are valid, that is, that a witness X contains a and c but does not contain b. This takes $O(n)$ tests of membership of an element of V in a member of $\mathcal{F}$, which can be carried out in $O(n \log n)$ time if each set is implemented with a sorted array, in $O(n)$ time if the sets are represented with a 0-1 matrix, or in $O(n)$ expected time if they are implemented with a hash table. This is sublinear, so it constitutes what is called a *strong certificate* [9].

Let us make use of a concept that is dual to the incompatibility graph, namely, a *forcing relation*.

DEFINITION 6.2. *If $((a, b), (b, c))$ is an edge of the incompatibility graph, then (a, b) forces (c, b) and (b, a) forces (b, c). Let $\mathcal{F}$'s forcing graph be the undirected graph that has $A_{\mathcal{F}}$ as its vertex set and the forcing relation as its edge set.*

Conceptually, if one ordered pair forces another, then either they both appear in the tournament of a consecutive-ones ordering or neither does.

LEMMA 6.2. *Let X and Y be overlapping members of $\mathcal{F}$. If (a, b) and (c, d) are two pairs that each go from from an earlier to a later member of the sequence $(X - Y, X \cap Y, Y - X)$, then there is a path of length at most three from (a, b) to (c, d) in $\mathcal{F}$'s forcing graph.*

Proof. Suppose $a \in X - Y$, $b \in X \cap Y$. If $d \in Y - X$, then, because of Y, (a, b) forces (a, d) and because of X, (a, d) forces (c, d). If $d \in X \cap Y$, then $c \in X - Y$. Let $e \in Y - X$. Then (a, b) forces (a, e) because of Y, (a, e) forces (c, e) because of X, and (c, e) forces (c, d) because of Y.

The only case that doesn't map to the foregoing by renaming of elements is $a, c \in X - Y$ and $b, d \in Y - X$. Because of X, (a, b) forces (c, b) and (c, b) forces (c, d).

COROLLARY 6.1. *During execution of the partitioning algorithm of Section 5.3, let $\mathcal{F}'$ be the set of processed sets, and let $X_1, X_2, ..., X_k$ be the sequence of blocks that are contained in $\bigcup \mathcal{F}'$. Let $(a, b), (c, d)$ be two pairs that each go from an earlier to a later member of this sequence. Then there is a path of length at most $k - 1$ from (a, b) to (c, d) in the forcing graph.*

Proof. By induction on the number of pivots. The base case is given by Lemma 6.2. The new ordered pairs that must be considered are those that go between two halves of a set that is split by a pivot on Z. Let X_i, X_j, and $\mathcal{P}$ be as in the description of the algorithm. If Z splits set X_i into $X_i - Z$ and $X_i \cap Z$, then there is a path of length 2 in the forcing graph connecting any pair of members of $(X_i - Z) \cap (X_i \cap Z)$ by Lemma 6.2. For $u \in X_i - Z$ and $w \in X_i \cap Z$, $y \in Z \cap X_{i+1}$, (u, w) is forced by (u, y), which has paths of of length k to the other pairs that go from earlier to later classes. Splitting X_i yields $k + 1$ classes and (u, w) has a path of length $k + 1$ to all other pairs that go from earlier to later classes. The same argument applies if Z splits S and $S \cap Z$ is placed at the beginning of the ordering of $\mathcal{P}$. A symmetric argument applies if X_j is split or if S is split and $S \cap Z$ is placed at the end of the ordering. If two sets are split by the pivot, applying this argument in sequence to each of the splits yields $k + 2$ sets and a path of length $k + 2$.

Using the foregoing, it is not hard to show that the connected component of the incompatibility graph is either the pairs of elements of V that go between two children of a prime node of the generalized PQ tree, or the set of pairs that have a single linear or degenerate node as their least-common ancestor. This is analogous to the relationship of Γ and the modular decomposition described by Gallai in [6].

We now give the proof of Theorem 6.1.

Proof. By Lemma 6.1, we may focus on elements of $A_{\mathcal{F}}$ that have the same degenerate least-common ancestor in the decomposition tree of $\mathcal{W}(\mathcal{F})$.

Suppose that the algorithm declares that $\mathcal{Q}$ does not have the consecutive-ones property. Then Z fails to

meet the required conditions, which means that there exist X_p, X_q, X_r with $p < q < r$ such that X_p contains $a \in Z$, X_q contains $b \notin Z$ and X_r contains $c \in Z$. Therefore, $((a, b), (b, c))$ is an edge of the incompatibility graph.

There is a path $((a, b), (x_2, y_2), (x_3, y_3), (x_{k-1}, y_{k-1}), (b, c))$ of length at most k in the forcing graph from (a, b) to (b, c). This corresponds to a path $((a, b), (y_2, x_2), (x_3, y_3), (y_4, x_4), ...)$ in the incompatibility graph, where every other element of the path is reversed. If k is even, the last element of the path is (b, c). If k is odd, the last element is (c, b) and it can be turned into an even-length path of length $k + 1$ by appending (b, c), since (c, b) and (b, c) are adjacent in the incompatibility graph. In either case, we get an even-length path of length at most $k + 1$ in the incompatibility graph between (a, b) and (b, c). Adding the incompatibility $((a, b)(b, c))$ to this gives an odd cycle of length at most $k + 2$. Since $\mathcal{Q}$ is a quotient on $\mathcal{F}$, which has domain of size n, this corresponds to a path of length at most $k + 2$ in $\mathcal{F}$'s forcing relation. Since $k \leq n$, Theorem 6.1 follows.

The constructive proofs of Corollary 6.1 and Theorem 6.1 are easily turned into linear-time algorithms. As in the proof of the theorem, an incompatible pair $(a, b), (b, c)$ is found during partitioning. The reader may easily verify that the partitioning algorithm can be modified so that, given two pairs (a, b) and (b, c) that each go from earlier to later partition classes, it finds a path in the forcing graph between them of the claimed length, using the construction of the proof of Corollary 6.1. Running the partitioning algorithm a second time to find a forcing path from (a, b) to (b, c) gives a forcing path, from which an odd cycle in the incompatibility graph is obtained as in the proof of the theorem.

References

[1] S. Booth and G. S. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms*, J. Comput. Syst. Sci., 13 (1976), pp. 335–379.

[2] A. Brandstaedt and V. B. Le and J. P. Spinrad, *Graph Classes: A Survey*, SIAM Monographs on Discrete Mathematics, Philadelphia, 1999.

[3] M. Chein and M. Habib and M. C. Maurer, *Partitive hypergraphs*, Discrete Mathematics, 37 (1981) pp. 35–50.

[4] T. H. Cormen and C. E. Leiserson and R. L. Rivest and C. Stein, *Introduction to Algorithms*, McGraw Hill, Boston, 2001.

[5] E. Dahlhaus, *Parallel algorithms for hierarchical clustering, and applications to split decomposition and parity graph recognition*, Journal of Algorithms, 36 (2000) pp. 205–240.

[6] T. Gallai, *Transitiv orientierbare Graphen*, Acta Math. Acad. Sci. Hungar., 18, (1967), pp. 25–66.

[7] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[8] W. L. Hsu and R. M. McConnell, *PC trees and circular-ones arrangements*, Theoretical Computer Science, 296 (2003), pp. 59–74.

[9] D. Kratsch and R. M. McConnell and K. Mehlhorn and J. P. Spinrad, *Certifying algorithms for recognizing interval graphs and permutation graphs*, ACM-SIAM SODA, 14 (2003), pp. 866–875.

[10] A. Lempel and S. Even and I. Cederbaum, *An algorithm for planarity testing of graphs*, in Theory of Graphs, P. Rosenstiehl (ed.), Gordon and Breach, New York, 1967.

[11] R. M. McConnell and J. P. Spinrad, *Linear-time modular decomposition and efficient transitive orientation of comparability graphs*, ACM-SIAM SODA, 5 (1994), pp. 536–545.

[12] R. M. McConnell and J. P. Spinrad, *Modular decomposition and transitive orientation*, Discrete Mathematics, 201 (1999), pp. 189–241.

[13] T. A. McKee and F. R. McMorris, *Topics in Intersection Graph Theory*, SIAM, Philadelphia, 1999.

[14] K. Mehlhorn and S. Näeher, *The LEDA Platform for Combinatorial and Geometric Computing*, Cambridge University Press, Cambridge", 1999.

[15] R. H. Möhring, R. H., *Algorithmic aspects of the substitution decomposition in optimization over relations, set systems and Boolean functions*, Annals of Operations Research, 4 (1985), pp. 195–225.

[16] A. Tucker, *A structure theorem for the consecutive 1's property*, J. Comb. Th. (B), 12 (1972), pp. 153–162.

[17] H. Wasserman and M. Blum, *Software reliability via run-time result-checking*, J. ACM, 44 (1997), pp. 826–849.