Linear-Time Transitive Orientation

Ross M. McConnell[*] Jeremy P. Spinrad[†]

Abstract

The **transitive orientation problem** is the problem of assigning a direction to each edge of a graph so that the resulting digraph is transitive. A graph is a **comparability graph** if such an assignment is possible. We describe an $O(n+m)$ algorithm for the transitive orientation problem, where n and m are the number of vertices and edges of the graph; full details are given in [18]. This gives linear time bounds for maximum clique and minimum vertex coloring on comparability graphs, recognition of two-dimensional partial orders, permutation graphs, cointerval graphs, and triangulated comparability graphs, and other combinatorial problems on comparability graphs and their complements.

1 Introduction

A partial order may be viewed as a transitive directed acyclic graph. A **comparability graph** is the graph obtained by ignoring the edge directions of a transitive directed acyclic graph; it gives the comparability relation for a partial order. It is well known that every partial order is the intersection of a set of total orders [8]. A **two-dimensional partial order** is a partial order that is the intersection of two linear orders, and a **permutation graph** is the corresponding comparability graph. If π is a permutation of a set V, the permutation graph corresponding to π is the graph on V where a pair of nodes is adjacent if and only if the relative order of the pair is inverted by the permutation. These classes of graphs and partial orders arise in many combinatorial problems. For a survey, see [12, 19].

The previous algorithms for transitive orientation took $O(n^2)$ [25], $O(\delta m)$ [11, 13, 22], or $O(n+m\log n)$ time [17], where δ is the maximum degree of any vertex in the graph. Our algorithm produces a **linear extension** of the transitive orientation, that is, a total ordering of the nodes such that whenever b is a successor of a in the ordering, (b,a) is not an edge in the transitive orientation. Like the $O(n^2)$ algorithm of [25], the algorithm fails to recognize whether a graph is a comparability graph. The algorithm is useful because it makes it possible to solve a number combinatorial problems on comparability graphs where recognition is not necessary. It either provides a certificate that its answer to the problem is correct, or it demonstrates that the input graph is not a comparability graph. It fails to recognize comparability graphs only because it sometimes provides a solution and certificate even when the input graph is not a comparability graph.

A key element in some of these corollary results is that if G is a graph whose complement is a comparability graph, the algorithm can produce a linear extension of a transitive orientation of the complement of G in time that is linear in the size of G, not in the size of the complement. Given any linear order R on the vertices of a graph G, we may find for any node v the number k of predecessors in R that are adjacent in G. The number of predecessors of v in R that are nonadjacent in G may be found by subtracting k from the number of predecessors of v in R. Symmetric computations can be made for successors. These observations give the following:

PROPOSITION 1.1. *If G is a comparability graph, then it takes $O(n+m)$ time to find the number of predecessors and successors of each vertex of G in a transitive orientation of G. If G is a co-comparability graph, then it takes $O(n+m)$ time to find the number of predecessors and successors of each vertex of G in a transitive orientation of the complement of G.*

The following are problems that may be solved in linear time using the decomposition algorithm. The bounds are new, except in the case of interval graph recognition.

1. **Recognition of permutation graphs and two-dimensional partial orders:** Recognition of partial orders of dimension k, where k is greater than two, is NP complete [29]. Recognition of two-dimensional partial orders clearly reduces to recognition of permutation graphs. Previous $O(n^3)$ and $O(n^2)$ algorithms for the problems have been given [4, 22, 25, 24]. We recognize permutation graphs by finding two total orders, R and R', whose intersection is the partial order given by a transitive

[*]Amherst College. Current address: Department of Computer Science, Willamette University, Salem, OR 97302 USA, rmcconne@willamette.edu. Supported in part by the graduate school "Algorithmische Diskrete Mathematik", which is supported by the Deutsche Forschungsgemeinschaft, grant WE 1265/2-1

[†]Department of Computer Science, Vanderbilt University, Nashville, TN 37235 USA, spin@vuse.vanderbilt.edu

orientation of G. We use the well-known characterization that G is a permutation graph iff G and its complement $\overline{G}$ are both transitively orientable [22]. The rank of a node in R is one plus the number of its predecessors in a transitive orientation of G plus the number of its predecessors in a transitive orientation of its complement. Reversing the linear extension of the transitive orientation of the complement and repeating the operation gives the rank of each node in R'. A linear time bound follows from Proposition 1.1. Verifying that the graph is a permutation graph thus reduces to verifying that the intersection of the computed orders R and R' is, in fact, the assumed transitive orientation of G, which is easily performed in time that is linear in the size of G.

2. **Recognition of cointerval graphs and interval graphs:** A graph is an interval graph if it is the intersection graph of a set of intervals on the line. A cointerval graph is the complement of an interval graph. The bound for interval graph recognition was previously known [2], but our algorithm gives a novel approach. If the graph is a cointerval graph, then we may find the number of predecessors and successors of each node in a transitive orientation of the graph, and if it is an interval graph, we may do the same thing for a transitive orientation of the complement, by Proposition 1.1. Ordering the beginning points by number of predecessors and endpoints by successors, then interleaving these two lists, gives a set of intervals that realize the graph. Verifying that the intervals realize the graph gives the recognition algorithm, and this is easily performed in time linear in the size of G.

3. **Maximum clique and a minimum vertex coloring in a comparability graph.** Suppose an orientation F of the edges of G is given by our algorithm. It is an easy exercise to color each vertex according to the length of the longest directed path that begins at it in (V, F), using a postorder operation during a depth-first search of (V, F) [12]. This gives a coloring of the nodes of the input graph such that no two adjacent nodes have the same color. If G is a comparability graph, then the longest path in the graph (V, F) is a clique of G because of the transitivity of F. That the sizes of the clique and the coloring are the same gives a certificate of correctness. If the longest path is not a clique, then the orientation is not transitive and G is not a comparability graph.

4. **Maximum independent set and minimum clique cover in co-comparability graphs.** A graph is a co-comparability graph if its complement is a comparability graph. (As we have seen, examples of co-comparability graphs are interval graphs and permutation graphs.) Find a linear extension of a transitive orientation of the complement of G. Label each node v with the length of the longest path in this oriented complement that begins at v. To spend $O(n+m)$ time, label nodes in order, starting at the end of the linear extension. When node i is reached, put it in bucket k, where k is the length of the longest path that begins at i in the transitive orientation of the complement. The bucket corresponding to i is one plus the highest bucket number of its non-neighbors that are already in buckets. This is found in time proportional to the number of neighbors of i by marking the neighbors of i, then searching downwards through the buckets, starting at the highest nonempty bucket, for an unmarked node.

5. **Recognition of triangulated comparability graphs** [12]: A graph is *triangulated* if every cycle of size greater than three has a chord. Interval graphs are an example. Triangulated comparability graphs are the class where G is both triangulated and a comparability graph. To recognize triangulated comparability graphs, use the linear procedure of [15], which assumes that a transitive orientation is given.

6. **Recognition of circular permutation graphs** [23]: A circular permutation graph is a graph where each vertex of G corresponds to a chord connecting two concentric circles, and where two vertices are adjacent in the graph if and only if the corresponding chords intersect each other. Using our bounds for transitive orientation and permutation-graph recognition, R. Sritharan has obtained linear bounds for recognition of circular permutation graphs [27].

2 Modular Decomposition

Let $V(G)$ denote the vertices of a graph G. If $X \subseteq V(G)$, then $G|X$ denotes the subgraph of G induced by X. Sets X and Y *overlap* if they intersect, but neither contains the other.

A **module** of an undirected graph G is a set X of vertices such that for any $x \in V(G) - X$, either every element of X is adjacent to x or no element of X is adjacent to x. $V(G)$ and its singleton subsets are **trivial modules**, and a graph is **prime** if it has no other modules. It is **degenerate** if every subset of its vertices is a module. The complete and edgeless graphs are the only degenerate graphs.

If X and Y are disjoint modules in G, then either $X \times Y \subseteq E$ or $(X \times Y) \cap E = \emptyset$. Thus, if $\mathcal{P}$ is a partition of vertices of G, the relationship of the members of $\mathcal{P}$ is itself given by a graph, denoted $G/\mathcal{P}$.

Let T be a tree whose leaves are the nodes of a graph. Think of each node as synonymous with the set consisting of its leaf descendants. The children of a node U are denoted $children_T(U)$. Label the internal nodes of T as prime or degenerate. This is a **partitive tree** corresponding to a family $\mathcal{F}$ of sets of vertices, where the members of $\mathcal{F}$ are the nodes of the tree, as well as those sets that are a union of children of a degenerate node. The appendix gives an illustration.

The family of modules of G is given by a unique partitive tree; this tree is the **modular decomposition**. If a node U is labeled degenerate in the tree, then $(G|U)/children_T(U)$ is a degenerate graph, and if it is labeled prime, then this quotient is a prime graph.

The modular decomposition has been generalized to $k-ary$ relations, hypergraphs, and other structures [20, 28, 10, 1]. Algorithms for computing it have a lengthy history [14, 3, 21, 9, 16, 17, 6, 7]; $O(n+m)$ bounds were obtained recently [17, 6, 7].

An edge (a, b) is **contained** in a module X if both of a and b are members of X. The edges that are contained in X may be oriented without regard to the orientation of those edges that are not contained in X, except that one must ensure that cycles do not form. This allows application of a divide-and-conquer strategy to the transitive orientation problem, using the modular decomposition tree as a guide in breaking up the problem into smaller subproblems. This eventually reduces the problem to that of finding a transitive orientation of the quotient $(G|U)/children_T(U)$ for each node U in the modular decomposition tree. However, if U is labeled degenerate, then any linear ordering of the nodes of this quotient will do, since $(G|U)/children_T(U)$ is complete or edgeless. Thus, the entire problem reduces to that of finding a transitive orientation of a prime graph [19]. Henceforth, we may assume without loss of generality that the graph we need to orient is prime, since modular decomposition may be obtained in linear time.

We will use not just the modular decomposition, but a more general class of partitive trees that we call **M trees**. These are partitive trees where the modules of the graph are a *subfamily* of the set family the tree represents. In particular, we use the following classes of M trees:

M1: Internal nodes are labeled prime or degenerate, and for each degenerate node U, there exists a set of representatives from all members of $children_T(U)$ that induces a complete or an edgeless subgraph in G.

M2: Internal nodes are labeled prime or degenerate, and for each degenerate node U, the members of $children_T(U)$ are modules in $G|U$ and $(G|U)/children_T(U)$ is complete or edgeless.

The linear-time modular decomposition algorithm of [17] first computes a **P_4 tree**, which is an M1 tree. It modifes the M1 tree to yield an M2 tree, and then modifies the M2 tree to get the modular decomposition. These trees have the property that the family of sets represented by each tree is a subfamily of the family represented by its predecessor. The transitive orientation algorithm that we describe here takes an approach that closely parallels the steps of that decomposition algorithm.

3 Overview of the Algorithm

We now describe a procedure called **vertex partitioning** for transitively orienting a comparability graph. We assume without loss of generality that the graph is prime, as described above. We begin with a starting partition $\mathcal{P} = \{\{v\}, V - \{v\}\}$ of the vertices of G. We then refine $\mathcal{P}$ inductively as follows. Select a **pivot vertex x**. Select a set $\mathcal{Q}$ of partition classes *that do not contain* x, and split each member $Y \in \mathcal{Q}$ into two classes Y_a and Y_n, which are the members of Y that are adjacent and nonadjacent to x, respectively. This gives a refinement of $\mathcal{P}$ whenever some member of $\mathcal{Q}$ contains both neighbors and non-neighbors of x. Since the graph is prime, there exists for every non-singleton $Y \in \mathcal{P}$ a pivot $x \in V(G) - Y$ that properly splits Y. Thus, there exists a sequence of operations that refines $\mathcal{P}$ until each partition class is a singleton set.

Here is how the procedure may be used to produce a linear extension of a transitive orientation of G. Select an arbitrary vertex v. Keep the partition classes in an ordered list $\mathcal{L}$, which is initially $(\{v\}, V - \{v\})$. Whenever a partition class Y is split into two classes by a pivot vertex x, let the two new classes occupy consecutive positions at the location in $\mathcal{L}$ previously occupied by Y. If x was in a class that precedes Y in $\mathcal{L}$, then let Y_n occupy the earlier of the two new positions, and if x was after Y, then let Y_n occupy the later of the new positions. The appendix gives an illustration.

Proposition 3.1. *Let G be a prime comparability graph. If v is selected arbitrarily in* **VertexPartition**, *then when the procedure halts, the member of the rightmost partition class in $\mathcal{L}$ is a sink in a transitive orientation of G. If v is selected to be a node that is a sink in a transitive orientation of G, then when the procedure halts, $\mathcal{L}$ gives a linear extension of a transitive* *orientation of G.*

Proof (sketch): If v is arbitrary, the invariant is maintained during the vertex partition that if y is a vertex in the last partition class Y in $\mathcal{L}$, then all edges from y to $V(G) - Y$ are oriented toward y. This is seen with the following inductive argument. Let x be a pivot that splits Y into Y_n and Y_a. Suppose $y \in Y_a$. Then (x, y) is an edge of G. Since (x, y) is oriented toward y then so must any edge (z, y) such that $z \in Y_n$ (otherwise (x, y) and (y, z) give a transitivity violation). When v is a sink in a transitive orientation, a similar inductive argument is used to demonstrate the invariant that all edges connecting different partition classes must be oriented in the direction of the beginning of $\mathcal{L}$. Q.E.D.

By the proposition, a linear extension of a transitive orientation is obtained by performing selecting an arbitrary vertex v, performing a vertex partition on the initial partition $(\{v\}, V(G) - \{v\})$ to identify a sink y, then performing a second vertex partition on the initial partition $(\{y\}, V(G) - \{y\})$.

To produce a linear extension of the transitive orientation of the complement of a graph, keep an ordered list of partition classes as above. Earlier, when a class Y split into Y_a and Y_n, we put Y_n nearer the class containing the pivot vertex in $\mathcal{L}$. Instead, we now put it farther than Y_a. Since this is the only asymmetry in the treatment of edges versus nonedges in the statement of the algorithm, it follows that the modification yields an algorithm for transitively orienting the complement of the graph.

4 Linear-Time Transitive Orientation

To implement each $Y \in \mathcal{P}$, we use a doubly linked list of nodes, where each entry in the list also has a pointer to the head of the list. Using this data structure, it is easily seen that a bound of $O(|N(x)|)$ can be achieved for a pivot, by removing the members of Y_a from Y as they are found in x's adjacency list and letting what remains of Y assume the role of Y_n.

It is also easily seen that after a pivot operation, one may delete the edges from x to members of $\mathcal{Q}$ without affecting the results of any future pivots. The time spent traversing these elements in the adjacency list may be charged to their deletion. The only obstacle to a linear time bound is the time spent traversing elements in x's adjacency list that go from x to vertices that are not in members of $\mathcal{Q}$. Since $\mathcal{Q}$ may not be chosen to contain x's partition class, this obstacle cannot be dealt with in a straightforward way.

A partition class Y is **consistent** with an M tree if it is either a node of the M tree or a union of children of a degenerate node of the M tree. A partition $\mathcal{P}$ of nodes of G is consistent with an M tree if each partition class is consistent with the tree. For an M tree T that appears during the decomposition algorithm, we give a **resolving procedure**, which is a procedure that generates pivot operations until $\mathcal{P}$ is consistent with T. If another pivot is then performed and some classes are split, we may again call the procedure to further refine $\mathcal{P}$ until it is again consistent with T. Here is a summary of how we use this idea to obtain the result.

1. (Sketched in Section 4.2) We give a resolving procedure on the $M1$ tree, called **M1resolve.** Using amortization arguments, it can be shown that the bound on the total time spent in any $O(n+m)$ calls to the procedure is $O(n + m)$.

2. (Omitted for space reasons) Using **M1resolve** as a subroutine, we derive a resolving procedure, **M2resolve**, on the M2 tree. The time bound on any $O(n + m)$ calls to **M2resolve** is $O(n + m)$.

3. (Sketched in Section 4.1) Using **M2resolve** as a subroutine, we derive $O(n+m)$ procedure that halts only when $\mathcal{P}$ is consistent with the modular decomposition tree of the graph. If the graph is prime, then $\mathcal{P}$ must consist of singleton sets. This gives a linear-time implementation of **VertexPartition**, hence a linear-time solution to the transitive orientation problem.

Full details are given in [18].

4.1 Vertex partitioning, given M2resolve.

In this section we show that the existence of **M2resolve** implies that **VertexPartition** may be performed in linear time.

DEFINITION 4.1. *A set W of vertices of G is* **split** *if it intersects more than one partition class of G. It is* **isolated** *if every member of $\mathcal{P}$ that intersects W is contained in W.*

REMARK 4.1. *If T is an M tree, then after a resolving procedure halts on T, every split node of T is isolated.*

PROPOSITION 4.1. *[26] Let T be a partitive tree on prime graph G. In $O(n + m)$ time, we may label each node U of T with a vertex w that splits U.*

The procedure for performing the vertex partition employs a preorder traversal of the internal nodes of the M2 tree, starting at the children of the root. At each node U of the tree, a call is made to **M2resolve**. We then perform a pivot by choosing a pivot vertex x that is known to split U. If U's parent is labeled prime, let the partition classes that intersect U assume the role of $\mathcal{Q}$ in the pivot operation, and if U's parent is labeled degenerate, let the set of partition classes that intersect the parent of U assume the role of $\mathcal{Q}$. After this procedure halts, all members of $\mathcal{P}$ are either singleton sets or unions of leaf children of degenerate nodes, as we will show. A final pivot with each vertex $x \in V(G)$, setting $\mathcal{Q}$ to be the members of $\mathcal{P}$ that do not contain x, ensures that all members of $\mathcal{P}$ are singletons.

To verify the correctness, note that after a call to **M2resolve**, every member of $\mathcal{P}$ is either a node of the tree or a union of children of a degenerate node. By induction on the depth of a node U in the tree, U's parent is split, hence isolated by the call to **M2resolve**, when U is reached. Similarly, U is isolated if its parent is prime. Thus, $\bigcup \mathcal{Q}$ is either U or its parent, depending on whether the parent is prime. To obtain a linear time bound, we preprocess the adjacency lists by numbering the leaves of the M2 tree in depth-first order, and sort the adjacency lists of the nodes in this order. This ensures that each node of the tree, hence $\bigcup \mathcal{Q}$, occupies a contiguous interval in x's adjacency list. The pivot at U may be performed by traversing only that interval, which is then deleted from x's adjacency list.

4.2 A resolving procedure on the P_4 tree.

In this section, we sketch how we derive the resolving procedure on the M1 tree. For an M1 tree, we use the *P_4 tree*, which may be constructed in linear time and which is shown in [26] to satisfy the definition of an M1 tree. Any graph that has no prime induced subgraph has a modular decomposition tree whose nodes are all degenerate. Such a graph is a **Cograph**. The **Cotree** algorithm of [5] gives its modular decomposition. The children of the root of a cotree are the connected components either of the graph or of its complement (exactly one of these is disconnected), and the subtrees are the cotrees on the subgraphs induced by those components. A P_4 is the four-vertex graph on vertex set $\{a, b, c, d\}$ with edge set $\{(a, b), (b, c), (c, d)\}$.

Definition 4.2. *If $(a, b), (b, c), (c, d)$ is an induced P_4 of a graph, then let $A := N(b) - N(c) - N(d)$, $B := (N(a) \cap N(c)) - N(d)$, $C := (N(b) \cap N(d)) - N(a)$, $D := N(c) - N(b) - N(a)$, $U = (N(a) \cap N(b) \cap N(c) \cap N(d)) \cup (\overline{N}(a) \cap \overline{N}(b) \cap \overline{N}(c) \cap \overline{N}(d))$, and let $E = V - A - B - C - D - U$.*

The following procedure defines a P_4-tree for a graph G. Because some steps of the procedure leave choices open, G can have many nonisomorphic P_4-trees.

```
Function Createtree($G$)

if $G$ is a cograph then T := cotree ($G$)
else
    let $(a,b),(b,c),(c,d)$ be a $P_4$ in $G$;
    select $x \in \{a,b,c,d\}$
    $T$ := Createtree($G|(U \cup \{x\})$);
    Let $p$ be the leaf of $T$ that corresponds to $\{x\}$
    Label $p$ prime
    for $X = A,B,C,D,E$ do
        $T_X$ := Createtree($G|X$)
        make $T_X$ a child of $p$;
return $T$;
```

A **universal pivot** on vertex x is one where $\mathcal{Q}$ is selected to be the members of $\mathcal{P}$ other than the one that contains x.

Proposition 4.2. *Let a, b, c, d and A, B, C, D, E, U be as in Definition 4.2, and let e be an arbitrary member of E. Suppose $\{a, b, c, d, e\}$ is not contained in a single partition class of $\mathcal{P}$. Cycling through (a, b, c, d, e) four times, performing a universal pivot on each member of the sequence, refines $\mathcal{P}$ so that each member of the new partition is contained in one of $\{A, B, C, D, E, U\}$.*

To make use of this observation, we use a tree that combines the recursion trees for **Createtree** and for **Cotree**. We will refer to this combined tree as the **local tree**. Let T' be the recursion tree corresponding to an execution of **Createtree**. Next, replace each **Createtree** node with two nodes, p and s, make the former node's children A, B, C, D, E into children of p, label p a **P_4 node**, make p be a child of s, label s as an S node, and make $U \cup \{x\}$ be the other child of s. Replace each leaf of T' with the cotree that was generated at that leaf. Though the data structure representing the local tree uses $O(1)$ space for each node, we may think of each node of the local tree as being the subset of $V(G)$ corresponding to its leaf descendants. The sets of vertices passed to recursive calls are give by pointers in leaf descendants.

We label each node of the local tree with a *representative* vertex from the set that it represents. If the node is a cotree node, the representatives of its children are chosen to have minimum degree. If the node is a P_4 node, the representatives of its children A, B, C, D are a, b, c, d, respectively, and the representatives of children E and $U \cup \{x\}$ are chosen to have minimum degree. This is accomplished in linear time in postorder. We show in [18] that the degree sum of the representatives is $O(n + m)$.

Definition 4.3. *A set X* **coincides with the local tree** *if it satisfies the following conditions:*

1. *For any P_4 node P and its parent S, either $X \cap P$ is empty, $X \cap P = P$, or $X \cap S$ is contained in one of A, B, C, D, E.*

2. *For any cotree node W, $X \cap W$ is either empty, a union of children of W, or contained in a child of W.*

A proof of the following proceeds by induction on the size of G [18]:

LEMMA 4.1. *$\mathcal{P}$ coincides with the local tree if and only if it is consistent with the corresponding P_4 tree.*

After each pivot operation, one may maintain marks on the nodes of the tree that indicate whether they are split by the current partition $\mathcal{P}$. The inductive procedure for doing this adds nothing to the asymptotic running time of the vertex partition.

The resolving procedure for the P_4 tree "processes" each split cotree or P_4 node that has not been processed in the current call or in a previous call to the procedure. To process a P_4 node, it performs four pivots on the representatives of its children, as described in Proposition 4.2. Degenerate nodes are also processed in this way, while S nodes are simply marked as processed after they are split. It halts only when all nodes are either unsplit or processed. Split nodes are processed in a chain of ancestors, beginning at minimal split node in the tree, and ending at a child of a previously processed node. By induction on the height of a node in this chain, it may be shown that when a node is processed, the representatives of its children do not all lie in the same partition class. Using Proposition 4.2, it can then be shown by induction that $\mathcal{P}$ coincides with the conditions of Definition 4.3 at every processed node. The procedure halts only when every split cotree and P_4 node has been processed, either in the current call or a previous call to the procedure, so by Lemma 4.1, it is consistent with the P_4 tree when it halts. This establishes the correctness. The linear time bound for $O(n+m)$ calls to the procedure follows from the $O(n+m)$ bound on the degree of a representative, summed over all representatives, and the fact that no node of the tree is processed more than once.

References

[1] Paola Bonizzoni. The $(n-2)$ property of primitive 2-structures. In M. Venturini Zilli A.M. Marchetti Spaccamela, P. Mentrasti, editor, *Proceedings of the Fourth Italian Conference on Theoretical Computer Science*, pages 96–109. World Scientific, London, 1992.

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

[3] B. Buer and R.H. Möhring. A fast algorithm for the decomposition of graphs and posets. *Mathematics of Operations Research*, 8:170–184, 1983.

[4] C.J. Colbourn. On testing isomorphism of permutation graphs. *Networks*, 11:13–21, 1981.

[5] D.G. Corneil, Y. Perl, and L.K. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 3:926–934, 1985.

[6] A. Cournier and M. Habib. A new linear algorithm for modular decomposition. In Sophie Tison, editor, *CAAP '94: 19th International Colloquium, Lecture Notes in Computer Science*, pages 68–82. Edinburgh, UK, 1994.

[7] E. Dahlhaus, J. Gustedt, and R.M. McConnell. Efficient and practical modular decomposition. *Same volume.*

[8] B. Duschnik and E.W. Miller. Partially ordered sets. *Amer. J. Math.*, 63:600–610, 1941.

[9] A. Ehrenfeucht, H.N. Gabow, R.M. McConnell, and S.J. Sullivan. An $O(n^2)$ divide-and-conquer algorithm for the prime tree decomposition of two-structures and modular decomposition of graphs. *Journal of Algorithms*, 16:283–294, 1994.

[10] A. Ehrenfeucht and R.M. McConnell. A k-structure generalization of the theory of 2-structures. *Theoretical Computer Science*, 132:209–227, 1994.

[11] M.C. Golumbic. The complexity of comparability graph recognition and coloring. *J. Combin. Theory Ser. B*, 22:68–90, 1977.

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

[13] A. Gouilà-Houri. Caracterisation des graphes non orientès dont on peut orienter les arrêtes de manière à obtenir le graphe d'une relation d'ordre. *C. R. Acad. Sci. Paris*, 254:1370–1371, 1962.

[14] L.O. James, R.G. Stanton, and D.D. Cowan. Graph decomposition for undirected graphs. In F. Hoffman and R.B. Levow, editors, *3rd South-Eastern Conf. Combinatorics, Graph Theory and Computing*, pages 281–290. Utilitas Mathematica, Winnipeg, 1972.

[15] T. Ma and J. Spinrad. Cycle-free partial orders and chordal comparability graphs. *Order*, 8:49–61, 1991.

[16] R.M. McConnell. An $O(n^2)$ incremental algorithm for modular decomposition of graphs and 2-structures. *Algorithmica*, 14:229–248, 1995.

[17] R.M. McConnell and J.P. Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 536–545. Arlington, Virginia, 1994.

[18] R.M. McConnell and J.P. Spinrad. Modular decomposition and transitive orientation. Technical Report 475/1995, Technische Universität Berlin, Fachbereich Mathematik, 1995.

[19] R.H. Möhring. Algorithmic aspects of comparability graphs and interval graphs. In I. Rival, editor, *Graphs and Orders*, pages 41–101. D. Reidel, Boston, 1985.

[20] R.H. Möhring. Algorithmic aspects of the substitu-

tion decomposition in optimization over relations, set systems and boolean functions. *Annals of Operations Research*, 4:195–225, 1985/6.

[21] J.H. Muller and J. Spinrad. Incremental modular decomposition. *Journal of the ACM*, 36:1–19, 1989.

[22] A. Pnueli, A. Lempel, and S. Even. Transitive orientation of graphs and identification of permutation graphs. *Canad. J. Math.*, 23:160–175, 1971.

[23] D. Rotem and J. Urrutia. Circular permutation graphs. *Networks*, 12:429–437, 1982.

[24] J. Spinrad and J. Valdes. Recognition and isomorphism of two-dimensional partial orders. In *Proceedings of the 10th Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science*, pages 676–686. Springer-Verlag, Berlin, 1983.

[25] J.P. Spinrad. On comparability and permutation graphs. *Siam J. Comput.*, 14:658–670, 1985.

[26] J.P. Spinrad. P_4 trees and substitution decomposition. *Discrete applied mathematics*, 39:263–291, 1992.

[27] R. Sritharan. A linear time algorithm to recognize circular permutation graphs. *Networks*, page to appear.

[28] D. Wagner. Decomposition of k-ary relations. *Discrete Math.*, 81:303–322, 1990.

[29] M. Yannakakis. The complexity of the partial order dimension problem. *SIAM J. Algebraic and Discrete Methods*, 3:303–322, 1982.

Appendix

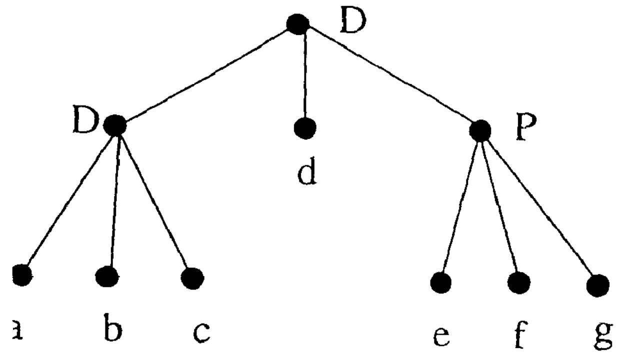

Figure 1. A partitive tree on set $\{a, b, c, d, e, f, g\}$. The labels of the internal nodes stand for "degenerate" and "prime". The "nodes" of the tree are $\{a\}$, $\{b\}$, $\{c\}$, $\{d\}$, $\{e\}$, $\{f\}$, $\{g\}$, $\{a, b, c\}$, $\{e, f, g\}$, $\{a, b, c, d, e, f, g\}$. Additional sets in the set family represented by the tree are unions of children of degenerate nodes: $\{a, b\}$, $\{b, c\}$, $\{a, c\}$, $\{a, b, c, d\}$, $\{a, b, c, e, f, g\}$, $\{d, e, f, g\}$. The modular decomposition is the representation of the modules of a graph with a partitive tree.

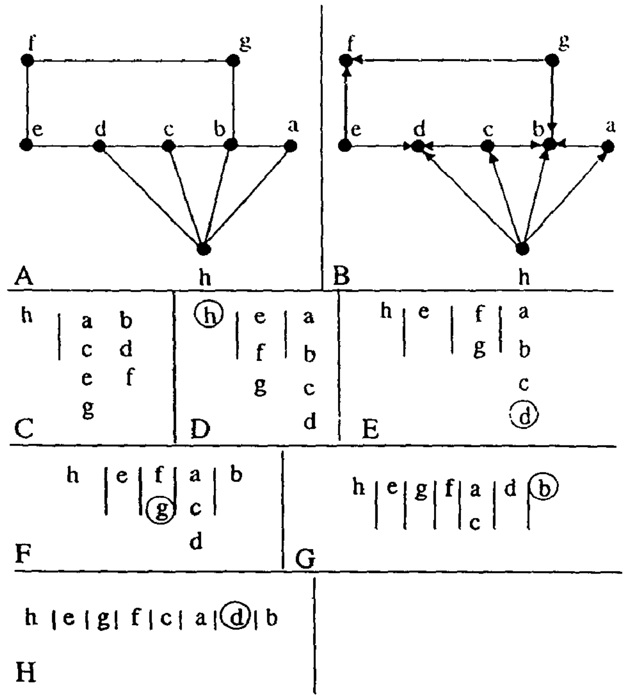

Figure 2. Vertex partitioning. A. A prime comparability graph G. B. A transitive orientation of G. Note that h is a source. C. The initial ordered partition $\mathcal{P} = (\{h\}, \{a, b, c, d, e, f, g, h\})$. D. After a pivot on h. Since $\{a, b, c, d\}$ is adjacent to h and h is earlier in the ordering, this subset goes after $\{e, f, g\}$. E. After a pivot on d. Since e is adjacent to d and d is later in the ordering, e goes before f and g. F. After a pivot on g. G. After a pivot on b. H. After a second pivot on d. This gives a linear extension (topological sort) of the transitive orientation of G in which h is a source. Every prime comparability graph has two transitive orientations; the transpose of this is the one where h is a sink.