Linear-time recognition of circular-arc graphs

Ross M. McConnell *

April 29, 2001

University of Colorado at Boulder
Technical Report CU-CS-914-01
Department of Computer Science
Campus Box 430
University of Colorado
Boulder, Colorado 80309 USA

Abstract

A graph G is a *circular-arc graph* if it is the intersection graph of a set of arcs on a circle. That is, there is one arc for each vertex of G, and two vertices are adjacent in G if the corresponding arcs intersect. We give a linear-time algorithm for recognizing this class of graphs. When G is a member of the class, the algorithm gives a certificate in the form of a set of arcs that realize it.

*Department of Computer Science and Engineering, University of Colorado at Denver, Denver, CO 80217-3364 USA, rmcconne@carbon.cudenver.edu. This research was supported by the University of Metz. The author gratefully acknowledges Dieter Kratsch for extensive inputs to this paper.

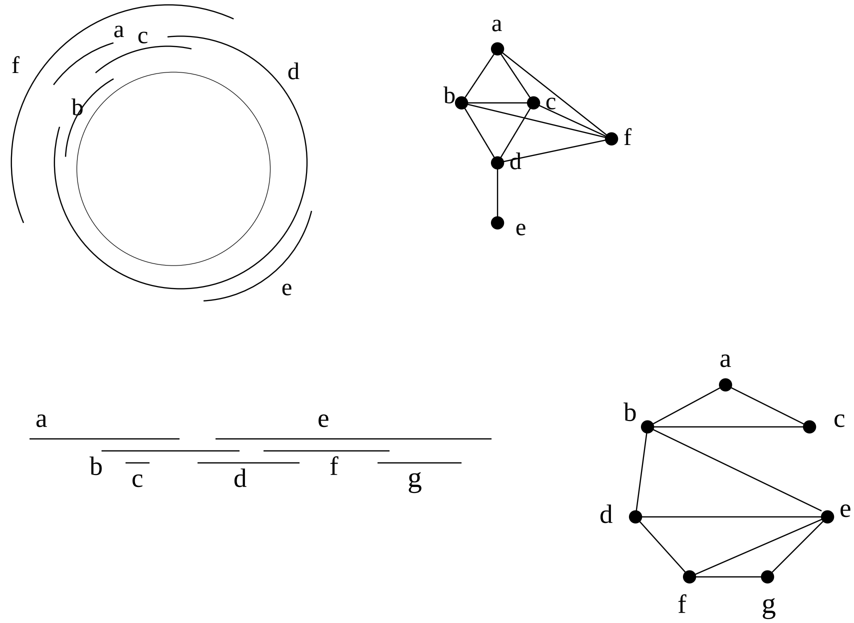


Figure 1: A circular-arc graph is the intersection graph of a set of arcs on the circle, while an interval graph is the intersection graph of a set of intervals on the line.

1 Introduction

The *intersection graph* of a family of n sets is the graph where the vertices are the sets, and the edges are the pairs of sets that intersect. Every graph is the intersection graph of some family of sets [17]. An graph is an *interval graph* if there is a way to order the universe from which the sets are drawn so that each set is consecutive. Equivalently, a graph is an interval graph if it is the intersection graph of a finite set of intervals on a line. A graph is a *circular-arc graph* if it is the intersection graph of a finite set of arcs on a circle. (See Figure 1.)

An interval graph is a special case of circular-arc graphs; it is a circular-arc graph that can be represented with arcs that do not cover the entire circle. Some circular-arc graphs do not have such a representation, so the class of interval graphs is a proper subclass of the class of circular-arc graphs.

Interval graphs and circular-arc graphs arise in scheduling problems and other combinatorial problems. Before the structure of DNA was well-understood, Seymour Benzer [1] was able to show that the set of intersections of a large number of fragments of genetic material in a virus were an interval graph. This provided strong evidence that genetic information was arranged inside a linear structure. Benzer earned the Nobel Prize partly for this work.

Being able to determine whether a graph is an interval graph or circular-arc graph constitutes *recognition* of these graph classes. However, having a representation of a graph with intervals or arcs can be helpful in solving combinatorial problems on the graph, such as isomorphism testing and finding maximum independent sets and cliques [4, 14]. Therefore, a stronger result than just recognizing the class is being able to produce the representation whenever a graph is a member of the class. In addition to its other uses, the representation constitutes a certificate that the graph is a member of the class.

Fulkerson and Gross [10] gave an $O(n^4)$ algorithm for solving this problem on interval graphs. Booth and Lueker later improved this to linear-time [3]. Until now, a linear-time bound for circular-arc graphs has been elusive.

The reason that the problem is harder on circular-arc graphs than on interval graphs is that there are two ways to travel from one point to another on a circle, as opposed to just one on a line. When attempting to construct a realizer of a circular-arc graph, one must choose one of these when joining an adjacent pair of arcs, and the correct choice is not always obvious. The necessity of making these choices is absent in the interval-graph recognition problem. In an interval graph the maximal cliques correspond to regions of maximal overlap among the intervals, and there are therefore $O(n)$ maximal cliques. This plays an important role in Booth and Lueker's algorithm. This is not true of circular-arc graphs. For instance, three arcs can intersect pairwise around the circle, yet have no point in common. The number of maximal cliques in a circular-arc graph can be exponential in n [26].

It was initially conjectured by Booth [2] that recognition of circular-arc graphs was NP-complete. Tucker disproved this with an $O(n^3)$ algorithm [26]. Hsu improved this to $O(nm)$, where m is the number of edges [14], and Eschen and Spinrad further improved this to $O(n^2)$ [9].

In the current paper, we give an $O(n+m)$ time bound. Because this is linear in the size of the graph, further improvements to the time bound are not possible. Like the previous algorithms, this one produces a set of arcs that realize the graph whenever it is a circular-arc graph.

The algorithm is based on modular decomposition, transitive orientation of comparability graphs, and algorithms on permutation graphs and interval graphs. For overviews of these topics, their applications, and their relationships to circular-arc graphs, see the books by Golumbic [12] and Roberts [23], and the survey article by Moehring [20].

2 Preliminaries

Let $G = (V, E)$ be a graph. By $n(G)$ and $m(G)$ we denote the number of vertices and edges, respectively, of G. We use n and m for these if G is understood. Let $E' \subseteq G$ denote that $E' \subseteq E$, and $e \in G$ denote that $e \in E$. $\overline{G}$ denotes the complement of G, whose vertices are V and whose edges are the non-edges of G. If G is a digraph, G^T denotes its transpose, which is obtained by reversing the directions of all of its directed edges. If v is a vertex in G, let $N(G, v)$ denote the neighbors of v in G. When G is understood, we may use $N(v)$ to denote this. $N[G, v]$ denotes the *closed neighborhood*, $N(G, v) \cup \{v\}$, and $N[v]$ denotes this when G is understood. Let $\overline{N}(G, v)$ denote the *non-neighbors*, $V - N[G, v]$, of v, and let $\overline{N}(v)$ denote the same thing when G is understood.

If A is a matrix, we let A_{ij} denote the value in row i and column j of A. If $X \subseteq V$, $G|X$ denotes the *restriction of G to X*, namely, the graph $G' = (X, E \cap (X \times X))$. Similarly, if A is an $n \times n$ matrix and X is a subset of $\{1, 2, ..., n\}$, then $A|X$ is the $|X| \times |X|$ matrix of entries whose rows and columns are both in X.

We may assume without loss of generality that no endpoints of arcs coincide, since in any realizer where they do, the endpoints can be moved by small amounts to make this true. To avoid cumbersome terms, let us call the clockwise endpoint of an arc the *left* endpoint, and the counterclockwise endpoint the *right* endpoint. We can remember this relationship by thinking of the arc as "facing" the center of the circle. If x is a vertex of the graph, $l(x)$ and $r(x)$ denote the left and right endpoints of its arc. In contexts where the realizer is understood, we will find it convenient to let x stand both for a vertex in G and for the arc $[l(x), r(x)]$ in the realizer. If R is an interval realizer, we assume that the first given in the realizer is the left endpoint that is adjacent to the uncovered part of the circle. Let us call this representation the **string realizer** since a string with two occurrences of each arc name suffices to represent it.

If G is a circular-arc graph with realizer R, and X is a subset of the vertices, then $R|X$ is the *restriction of* R *to* X, namely, the result of removing from R those arcs corresponding to vertices not in X. Clearly, $R|X$ is a realizer of $G|X$.

Since combinatorial arguments are often easier than geometric ones, we will sometimes operate on a realizer that is equivalent to a geometric one, namely, a circular ordering of $\{l(x) : x \in V\} \cup \{r(x) : x \in V\}$. The circular ordering represents the order of left and right endpoints as one travels counterclockwise around the circle. The circular ordering can be represented by an ordered list, where an incremental rightward movement is a movement from position i to position $i + 1 \pmod{2n}$. Let us call this the *circular-list* representation of the realizer. A circular-arc graph may have more than one realizer; when counting its realizers, we consider two realizers to be the same if one is a cyclic permutation of the other, and different otherwise. If G is an interval graph, we adopt the convention of circularly permuting the realizer so that an uncovered part of the circle occurs immediately following the last endpoint in the list. Let us call this the *ordered list* representation of an interval realizer. We consider two interval realizers to be different if their ordered-list representations are different.

An undirected graph is a special case of a directed graph where each undirected edge consists of two oppositely directed edges. In this paper, we consider an $n \times n$ matrix A to be synonymous with a complete graph on vertex set $V = \{v_1, v_2, ..., v_n\}$, where each directed edge (v_i, v_j) is labeled with A_{ij}. A *module of a matrix* corresponds to a set X of vertices such that for every vertex $y \notin X$, all directed edges in $X \times \{y\}$ have the same label, and all directed edges in $\{y\} \times X$ have the same label. In this case, y fails to *distinguish* members of X. A module of a graph or digraph $G = (V, E)$ is a module in its boolean adjacency matrix. That is, it is a set of vertices such that for each $y \notin X$, every element of $X \times \{y\}$ is an edge or none is, and every element of $\{y\} \times X$ is an edge or none is. Modules of matrices are studied in [5, 6], where they are called modules on *two-structures*. They were previously known in the special case of graphs, and first studied in [11].

V and its singleton subsets are *trivial modules*. A graph or matrix with only trivial modules is called *prime*. A *universal vertex* is a vertex x with $N[x] = V$. A *clique module* is a module of G that is a (not necessarily maximal) clique in G. Universal vertices are identifiable by their degrees. The clique modules can be found in linear time by lexically sorting adjacency lists.

A *modular partition* of V in a matrix is a partition of V where every partition class is a module. If X and Y are disjoint modules, then every element of $X \times Y$ has the same label. The *modular quotient* induced by the parts is the matrix obtained by making one vertex for each part, and letting the label of each ordered pair (X, Y) of parts be the labels of the edges from X to Y. One way to represent a modular quotient is with the submatrix induced by any set P consisting of one vertex from each part. Similarly, if R is a circular-arc realizer, then the quotient is realized with the smaller realizer $R|P$.

Two sets A and B *overlap* if $A \cap B$, $A - B$, and $B - A$ are all nonempty. A module is *strong* if it overlaps no other module. The transitive reduction of the containment relation on strong modules of a graph $G = (V, E)$ is a tree, and is called the *modular decomposition.* The root of the modular decomposition is V and the leaves are its singleton subsets. At most one of G and its complement, $\overline{G}$, can be disconnected. If one of them is disconnected, the children of the root are the connected components, and V is a *degenerate node.* The remaining nodes are *prime nodes.* The family $\mathcal{F}$ of modules of G consists of those sets that are nodes of the tree, and those sets that are a union of siblings that have a degenerate parent. Let $MD(G)$ denote the modular decomposition of G.

The modular decomposition of a matrix is defined in the same way, except that the root is degenerate if the complement of the graph consisting of those edges with one label is disconnected. If T is a matrix, let $MD(T)$ denote its modular decomposition.

A digraph is a *dag* if it is acyclic. A digraph is *transitive* if, for every pair (a, b) and (b, c) of directed edges, (a, c) is also a directed edge. A transitive dag models a poset relation. If $G = (V, E)$ is a dag, the *transitive closure* of G is the minimum transitive dag on V that contains it. The *transitive reduction* is the minimum subgraph of G on V that has the same transitive closure as G does. A *linear order* is a complete transitive dag. A linear order can be represented by giving its unique topological sort, $(v_1, v_2, ..., v_n)$. A *linear extension* of G is a linear order that contains G.

A *transitive orientation* of an undirected graph is an assignment of directions to its edges so that the resulting digraph is a transitive dag. A graph is a *comparability graph* if there exists a transitive orientation of it. Finding a transitive orientation of a comparability graph takes $O(n + m)$ time [18]. The complement of an interval graph is a comparability graph.

An undirected graph can be viewed as a special case of a directed graph, where each undirected edge ab represents two directed edges, (a, b) and (b, a). If (a, b) and (c, d) are two directed edges, we say that $(a, b)\Gamma(c, d)$ iff $a = c$ and b and d are nonadjacent, or $b = d$ and a and c are nonadjacent. If G is a comparability graph, then $(a, b)\Gamma(c, d)$ implies that in any transitive orientation of G, either both of (a, b) and (c, d) appear as directed edges, or neither does. To understand why, suppose that $(a, b)\Gamma(c, d)$, but (a, b) and (d, c) are directed edges in an orientation of G. Suppose without loss of generality that $a = c$. Then (d, c) and $(c, b) = (a, b)$ require a transitive directed edge (d, b), but d and b are nonadjacent, so this is impossible.

The transitive closure of the Γ relation is an equivalence relation on directed edges, and the equivalence classes are groups of directed edges, called *implication classes*, that must either all appear in or all be absent from any transitive orientation of G. Therefore, if G is a comparability graph, (a, b) and its transpose (b, a) cannot be in the same implication class.

The implication class containing (a, b) consists of the transposes of the directed edges in the implication class containing (b, a). The union of an implication class and its transposes is a *color class.* The color classes are a partition of the undirected edges of G.

There is a type of dual relationship between the modules of G and its color classes.

Lemma 2.1 *(see, for example, [20]) If X and Y are disjoint nodes of the modular decomposition of G, then $X \times Y$ is a subset of a single implication class.*

The vertices spanned by a color class are always a module of G. Moreover, if M is a module, an edge of G that is in $G|M$ is always in a different color class from an edge that is not in $G|M$. Because of this, given a transitive orientation of G and its modular decomposition, one may obtain a new transitive orientation by reversing the directions of all edges inside a node of the modular decomposition tree. In a degenerate node whose children are nonadjacent in $\overline{G}$, a transitive orientation of G determines a linear order on the children. Reversing the orientations of all edges between a pair of children that are adjacent in this linear order gives a new transitive orientation. All transitive orientations of G are obtainable from a single one by applying these two reversal operations in different places in the modular decomposition tree.

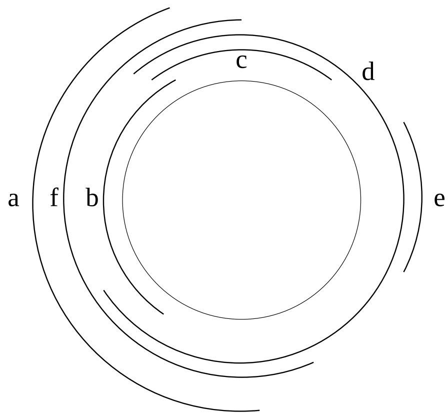
A *permutation graph* is a graph that is a comparability graph, and whose complement is a comparability graph. The union of a transitive orientation of a permutation graph and a transitive orientation of its complement gives a linear order on the vertices. Reversing the orientations in the complement and again taking the union of the two gives another linear order. Two vertices are adjacent in the graph iff their relative order is the same in both of these orderings. This *permutation realizer* gives a way to represent a permutation graph with two orderings of the vertices.

3 Intersection matrices

If G is a circular-arc graph and R is a realizer, we can classify each ordered pair (x, y) of vertices with $x \neq y$ according to the type of intersection of their arcs in R, as follows [26, 14, 9]:

- **single overlap:** Arc x contains a single endpoint of arc y.
- **double overlap:** x and y jointly cover the circle and each contains both endpoints of the other.
- **Arc x is contained in arc y**
- **Arc x contains arc y**

Let $v_1, v_2, ..., v_n$ be a numbering of vertices of V. A *circular-arc matrix* of a realizer R is an $n \times n$ matrix T where T_{ij} is a label that tells the type of the relationship between arc v_i and arc v_j in the realizer. (See Figure 2.) R is a realizer of T if it realizes not just the graph implied by T but also the intersection types claimed by T. A circular-arc matrix is



	a	b	c	d	e	f
a		t	1	2	n	c
b	c		1	2	n	c
c	1	1		c	n	1
d	2	2	t		t	2
e	n	n	n	c		n
f	t	t	1	2	n	

Figure 2: The intersection matrix of a circular-arc realizer gives the types of intersections between arcs. The types consist of two arcs overlapping at one endpoint, G_1 (1), two arcs overlapping at two endpoints, G_2 (2), two arcs not intersecting, G_n (n), one arc being contained in the other D_c (c) and one arc containing the other $(D_c)^T$ (t). The pairs of each type give the graphs G_1, G_2, G_n, D_c, and $(D_c)^T$, respectively. The matrix has a skew symmetry, where t in a row i, column j corresponds to c in row j column i.

an *interval matrix* if there is a realizer for it that does not cover the circle; in this case G is an interval graph, since a circular-arc realizer can be cut at an uncovered point on the circle and rolled out onto a line. An *intersection matrix* is a matrix whose entries give the alleged intersection types in some circular-arc realizer, but where it is not a requirement that the realizer exist. In general, we refer to a circular-arc matrix as an intersection matrix, except when we wish to emphasize that a realizer is required to exist.

From these types, we get a partition of the complete graph into the following undirected graphs on V:

- G_1: single overlaps
- G_2: double overlaps
- G_c: containments
- $G_n = \overline{G}$: non-intersections

We will use multiple subscripts to denote unions of these. For example $G = G_{12c}$, the union of edges of G_1, G_2, and G_c. An edge of G_c can be viewed as two directed edges with the following classifications.

- D_c: edges of the form $\{(x, y) : x \text{ is contained in } y\}$
- $(D_c)^T$: edges of the form $\{(x, y) : x \text{ contains } y\}$.

The *non-neighbors* of a vertex x are its neighbors in G_n, its *overlap neighbors* are its neighbors in G_1, its *double-overlap neighbors* are its neighbors in G_2, and its *containment* *neighbors* are its neighbors in G_c. $G(T)$ denotes the graph that gives the pairs that intersect, hence $G(T) = G_{12c}$. The *neighbors* are the the neighbors in $G(T)$. If M is a module of T, then each vertex in $V - M$ has the same relationship to members of M. The non-neighbors, overlap neighbors, double-overlap neighbors, and containment neighbors are those vertices of $V - M$ that have those relationships to the members of M.

These intersection types have played a significant role in the previous algorithms for recognizing circular-arc graphs. Tucker [26] developed ways of constraining the possible types of intersection in order to facilitate insertion of arcs during an incremental assembly of a realizer. This step was a bottlenecks in Tucker's algorithm. A faster approach for computing the constraints was one of the significant innovations of Eschen and Spinrad [9], and it plays a critical role in the current paper.

A departure of our approach from previous ones is the use of a simple operation on the intersection matrix, which we will call a *flip*. Recall that we consider a realizer to be a circular list giving the order of left and right endpoints of the vertices' arcs around the circle. Let a *geometric flip* on a vertex be an exchange of the positions of $l(x)$ and $r(x)$ in this list. This is equivalent to making x's arc travel between the same endpoints as before, but around the opposite part of the circle as before. (Figure 3). This changes the types of relationships involving x's arc as follows:

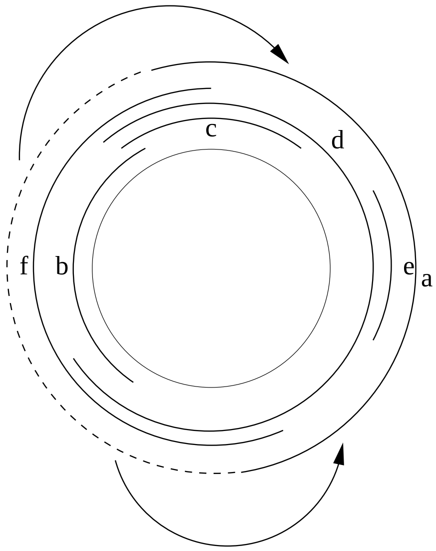
- $(x, y) \in G_2$ becomes $(x, y) \in D_c$ and $(x, y) \in D_c$ becomes $xy \in G_2$
- $(x, y) \in G_n$ becomes $(x, y) \in (D_c)^T$ and $(x, y) \in (D_c)^T$ becomes $(x, y) \in G_n$
- $(x, y) \in G_1$ remains unchanged

If T is an intersection matrix, we can compute the intersection matrix T' for the realizer that results from a flip on x without any knowledge about the realizer, other than what is given directly by T. It requires only a simple relabeling of some of the entries of x's row and column in T. Let us call this relabeling of T an *algebraic flip* on x.

One matrix being obtainable from another by a sequence of algebraic flips is an equivalence relation on intersection matrices. Let us call this relation *flip-equivalence.* Every flip-equivalence class on circular-arc matrices contains an interval matrix: in a circular-arc realizer of a matrix T, if one picks a point on the circle that does not coincide with an endpoint of an arc and flips all arcs containing the point, the resulting set of arcs will fail to cover the circle at that point, and realize the interval matrix T' obtained from T by the equivalent algebraic flips.

It is tempting to try to transpose parts of the problem to another class of graphs. Eschen and Spinrad [9] compute the intersection constraints by transposing this problem into the domain of chordal biparite graphs. Hsu [14] approaches the recognition problem by transposing much of it to the domain of circle graphs. The significance of flip operations is that they allows us to transpose much of the problem to the domain of comparability graphs, where we make use of existing analytical and algorithmic tools for that graph class.

It should be noted that despite superficial differences, there is a non-obvious relationship between the flip operation and Hsu's use of circle graphs to model parts of the problem. This is discussed in greater detail in the concluding section of our paper.



	a	b	c	d	e	f
a		n	1	c	t	2
b	n		1	2	n	c
c	1	1		c	n	1
d	t	2	t		t	2
e	c	n	n	c		n
f	2	t	1	2	n	

Figure 3: A *geometric flip* consists of rerouting an arc's path between its endpoints; in this case arc a is flipped. (Compare to Figure 2.) A flip has a predictable effect on the intersection matrix: it swaps the roles of n and t, and 2 and c in a row, and swaps the roles of n and c, and 2 and t in a column. Whether or not we know the matrix's circular-arc realizer, we may find the effect of a geometric flip on it. This operation on the intersection matrix is an *algebraic flip.*

4 Summary of the circular-arc graph recognition algorithm

Our algorithm produces a circular-arc realizer of G if it is a circular-arc graph. If G is not a circular-arc graph, it cannot return a circular-arc realizer, as no such realizer exists. The algorithm may return a circular-arc realizer for a graph that is not G. To check this, we use the following algorithm.

Algorithm 4.1 *Checking whether a circular-arc realizer realizes G.*

For each vertex x, look up the arc for each neighbor of x in the adjacency-list representation of G and verify that it intersects x's arc. Then count the number of intervals that intersect x's interval, halting and rejecting the realizer if the count exceeds the degree of x in G.

For the correctness of the recognition algorithm, it suffices to verify only that it returns a valid realizer when G is a circular-arc graph. Henceforth, we will assume that G is a circular-arc graph, except in the analysis of the time bound.

Algorithm 4.2 summarizes the approach.

Algorithm 4.2 *Constructing a circular-arc realizer.*

1. *Find an intersection matrix T that can be shown to be realized by some realizer of G;*

2. *Perform a set of algebraic flips on T to obtain an interval matrix T';*
3. *Find an interval realizer R' of T';*
4. *Invert the flips used to obtain T' from T, but apply them as geometric flips to R'. This gives a circular-arc realizer R of T.*

The fourth subproblem is trivial. We summarize the approach to the other three now, and give the full details below.

4.1 Finding an intersection matrix that gives the intersection types for some realizer of G

We use Hsu's characterization of a set of constraints on the intersection types that are sufficient to get an intersection matrix. Eschen and Spinrad [9] give an algorithm for the intersection types in $O(n^2)$ time when the vertices can be partitioned into two cliques. We improve this to $O(n+m)$ for arbitrary circular-arc graphs. This requires a sparse representation of the intersection matrix, consisting of a labeling of the edges of G with the type of intersection that they represent.

4.2 Finding a set of flips to turn T into an interval matrix

With a single flip, we create a vertex v_0 of degree $O(m/n)$ that has no double-overlap relationships. All arcs intersecting v_0 contain exactly one endpoint of v_0. Flipping an arc containing one endpoint of v_0 makes it contain the opposite endpoint of v_0 instead. Flipping all arcs intersecting v_0 so that they contain the same endpoint of v_0 would vacate a region of the circle adjacent to the other endpoint of v_0. This would therefore yield an interval matrix.

While it is possible to do it this way, we find it easier to use the following approach to vacate part of the circle. We find an interval A of the circle such that all arcs that are non-neighbors of v_0 are either contained in A or contain A. Flipping an arc does not affect the positions of its endpoints. We find the set Q of neighbors of v_0 with an endpoint in A, and flip them so that they all cover the same endpoint of v_0. (See Figures 11 and 12.) This ensures that one endpoint p of A is uncovered by any arc in Q. All other arcs in the realizer are either disjoint from A or contain A, and these are easy to distinguish using the intersection matrix. Flipping those that contain A leaves a region of the circle next to p uncovered.

4.3 Finding an interval realizer of an interval matrix

It is easily seen that if T is an interval matrix, G_n is a comparability graph: the order of left endpoints in an interval realizer is a linear extension of a transitive orientation. (See Figure 4.) G_{1n} is a comparability graph for the same reason. In an interval matrix, G_2 is empty, since a double overlap of two arcs covers the entire circle. Therefore, G_c is the complement of G_{1n}. Since it also has a transitive orientation, namely D_c, it follows that G_c and G_{1n} are complementary comparability graphs, hence permutation graphs.

In [18], an algorithm is given for interval-graph recognition that uses a linear extension of a transitive orientation of $\overline{G} = G_n$. Thus, we can get an interval realizer of G in this way. Unfortunately, because of the added constraints in the types of intersections imposed by T, this might fail to be a realizer of T.

The permutation-graph recognition algorithm of [18] uses the transitive orientation algorithm to compute linear extensions of transitive orientations D and D' of a permutation graph H and its complement $\overline{H}$. It then finds the linear orders $D \cup D'$ and $D \cup (D')^T$ in linear time, which gives the permutation realizer. Running this algorithm gives a linear extension of D_c and a linear extension of a transitive orientation G_{1n}, and therefore gives a permutation realizer of G_{1n}. It is easy to see that these two permutations are the order of appearance of left endpoints and the order of appearance of right endpoints in an interval realizer of G. But once again, this may fail to be an interval realizer of T.

The key observation we use is that every realizer of T gives a single orientation of G_{1n} that is *simultaneously* transitive in G_{1n} and G_n. Finding a transitive orientation of G_{1n} that is simultaneously transitive in G_n further constrains the possible solutions, and allows us to get the orders of appearance of left and of right endpoints of a realizer of T. Interleaving these orders to produce the full realizer is a trivial operation. We show how the transitive orientation algorithm of [18] can be modified to produce the simultaneous transitive orientations of G_{1n} and G_n.

5 Creating an intersection matrix that shares a realizer with a given circular-arc graph

In this section, we give the implementation of Step 1 of Algorithm 4.2.

Lemma 5.1 *Let G be a circular-arc graph, and let X be a clique module of G.*

1. *If y is a universal vertex, a circular-arc realizer for G can be obtained from any circular-arc realizer for $G|(V - \{y\})$, in $O(1)$ time.*
2. *If X is a clique module of G and $x \in X$, an intersection matrix for G can be obtained from any intersection matrix of $G|((V - X) \cup \{x\})$ in $O(|X|)$ time if X is given.*

Proof: To obtain a realizer of G from a realizer of $G|(V - \{y\})$, insert $r(y)$ anywhere, and insert $l(y)$ immediately to the right of it. This causes it to cover the circle, except in an interval between $r(y)$ and $l(y)$ that has no endpoints of other arcs in it.

Let $X = \{x_1, x_2, x_3, ..., x_k\}$. A realizer of G can be obtained from a realizer of $G|((V - X) \cup \{x\})$, by replacing $l(x)$ with the sequence $(l(x_1), l(x_2), ..., l(x_k))$, and replacing $r(x)$ with the sequence $(r(x_1), r(x_2), ..., r(x_k))$. $\Box$

The goal of the recognition algorithm is to produce a circular-arc realizer of G if one exists. We apply the reduction Lemma 5.1 initially to G, eliminating all universal vertices and all but one vertex from each remaining clique module, to obtain a graph G'. We find a realizer of G', and use Lemma 5.1 to find a realizer of G.

Clique modules are groups of vertices whose closed neighborhoods are identical. The clique modules are found by creating a list that has directed edges (u, v) and (v, u) for each undirected edge uv in G, and a loop (x, x) for each vertex, and then radix sorting this list to obtain sorted closed neighborhood lists for each vertex. Radix sorting the neighborhood lists makes clique modules consecutive. This takes $O(n + m)$ time.

Thus, we assume in the rest of this section that G has no universal vertices or clique modules, hence no two vertices have identical closed neighborhoods.

Let $T(G)$ denote the $n \times n$ matrix T of labels that is defined as follows:

1. if x and y are nonadjacent then $T_{xy} = G_n$
2. else if $N[x] \subset N[y]$ then $T_{xy} = D_c$
3. else if $N[y] \subset N[x]$ then $T_{xy} = (D_c)^T$
4. else if $N[x] \cup N[y] = V$ and for each $z \in N[x] - N[y]$, $N[z] \subset N[x]$ and for each $z' \in N[y] - N[x]$, $N[z'] \subset N[y]$, then $T_{xy} = G_2$
5. else $T_{xy} = G_1$

Theorem 5.2 *[14] If G is a circular-arc graph with no clique modules or universal vertices, then $T(G)$ is the intersection matrix of a circular-arc realizer of G.*

Hsu calls a realizer of this matrix a *normalized model*, and works exclusively with normalized models. If G is a graph, performing a flip on a vertex in $T(G)$ yields a new matrix T' for a graph G', and T' gives the intersection types of a realizer of G'. It is not necessarily true that $T' = T(G')$, so a geometric flip may or may not give a normalized realizer. The sole significance to us of a normalized model is that it allows us to get an initial intersection matrix.

A *chordal bipartite graph* is a bipartite graph with no chordless cycles of length 6 or greater. Let n_1 and n_2 denote the sizes of its bipartition classes. A *neighborhood containment test* on two vertices x and y consists of evaluating the expression $N(x) \subseteq N(y)$. A *disjoint neighborhood test* consists of evaluating whether $N(x) \cap N(y)$ is empty.

Theorem 5.3 *[25] Neighborhood containments tests between between k pairs of vertices in a chordal bipartite graph can be performed in $O(n_1 n_2 + k)$ time.*

Theorem 5.4 *[7] Disjoint neighborhood tests between between k pairs of vertices in a chordal bipartite graph can be performed in $O(n_1 n_2 + k)$ time.*

Theorem 5.5 *[9]: Let G be an arbitrary circular-arc graph, and let m_0 be a vertex whose arc contains no other in some circular-arc realizer of G. Let $D(N'[m_0], V, E_D)$ be a bipartite graph, where $N'[m_0]$ is a copy of $N[m_0]$, and $xy \in E_D$ iff $x \neq y$ and $xy \notin E$. Then D is chordal bipartite.*

In the statement of Theorem 5.5 in [9], m_0 is only required to be a vertex of minimum degree. However, this is only to ensure for the proof that m_0's arc contains no other in some circular-arc realizer. The proof works without modification for this version of the theorem.

Theorem 5.6 *Let G be a graph and let $U \subseteq V$ such that $G|U$ is an interval graph. Then it takes $O(n+m+k)$ time to evaluate $N[x] \cap U \subseteq N[y] \cap U$ and $(N[x] \cap U) \cup (N[y] \cap U) = U$ at k pairs $\{x, y\}$ of vertices such that that $y \in U$.*

Proof: Compute an interval realizer of $G|U$ in $O(n+m)$ time [3]. Create a list L of the left endpoints in the order in which they appear, and a list R of the right endpoints in the order in which they appear. For each vertex w of G, compute the leftmost right endpoint $L(w)$ and the rightmost left endpoint $R(w)$ among neighbors of w in the realizer of $G|U$. Compute also the leftmost right endpoint $L'(w)$ and the rightmost left endpoint $R'(w)$ among non-neighbors of w in the realizer of $G|U$. Computing $L(w)$ and $R(w)$ clearly takes $O(|N(w)|)$ time. To compute $L'(w)$, mark neighbors of w in the realizer, in $O(|N(w)|)$ time. Then traverse R from left to right until a right endpoint of an unmarked vertex is encountered; this is $L'(w)$. $R'(w)$ can be found by a symmetric operation on L. All but one endpoint encountered in each traversal is a neighbor of w, so this takes $O(|N(w)|)$ time. Then unmark neighbors in $O(|N(w)|)$ time to reinitialize the realizer. The entire operation takes $O(|N(w)|)$ time, hence $O(n+m)$ time over all vertices.

For $\{x, y\}$ such that $y \in U$, $N[x] \cap U \subseteq N[y] \cap U$ iff $l(y) < L(x)$ and $r(y) > R(x)$. $(N[y] \cap U) \cup (N[x] \cap U) = U$ iff $l(y) < L'(x)$ and $r(y) > R'(x)$. □

Theorem 5.7 *[9] Let G be a graph and let $U \subseteq V$ such that $G|U$ is an interval graph. Then it takes $O(n+m+n|V-U|)$ time to evaluate $N[x] \cap U \subseteq N[y] \cap U$ at all pairs $\{x, y\}$ of vertices such that $x \in U$ and $y \in V - U$.*

Nonadjacent pairs always have the same intersection type, so we use a sparse representation of the intersection matrix that consists of labeling the edges of G with their intersection type. The following two theorems suffice.

Theorem 5.8 *It takes $O(n+m)$ time to determine that G is not a circular-arc graph, or else evaluate the following boolean expressions at each edge xy in G:*

1. *$N[x] \subseteq N[y]$ and $N[y] \subseteq N[x]$;*

2. *$N[x] \cup N[y] = V$.*

In their conference paper, Eschen and Spinrad give an $O(n^2)$ bound, but omit the proof of Part 2 due to space considerations. However, it is a straightforward variant of the proof of Part 1. They do not give a tighter bound, presumably because this suffices to get their $O(n^2)$ bound for circular-arc graph recognition. However, it is also straightforward to get an $O(n+m)$ bound using their methods, if one is interested in evaluating these expressions only at adjacent pairs of vertices, which we show now.

Proof: A vertex of minimum degree satisfies the requirements of m_0 in Theorem 5.5, and has degree $O(m/n)$ in G. The size of the bipartite adjacency matrix for the graph D given by the theorem is $O((m/n)n) = O(m)$. Let U denote $V - N[m_0]$.

It takes $O(n+m)$ time to find an interval model of $G|U$ [3].

1. Consider pairs of the form $\{x,y\} \subseteq N[m_0]$. By Theorem 5.3, we may evaluate $N[\overline{G},x] \subseteq N[\overline{G},y]$, which is true iff $N[y] \subseteq N[x]$. Using Theorem 5.4, we test whether the neighborhoods of x and y are disjoint in D, which tells whether $N[x] \cup N[y] = V$ in G.

2. Next, consider adjacent pairs of the form $\{x,y\} \subseteq U$. It takes linear time for all such pairs to evaluate $N[x] \cap U \subseteq N[y] \cap U$, by Theorem 5.6, and linear time to evaluate $N(D,y) \subseteq N(D,x)$ in D, by Theorem 5.3, which gives $N[x] \cap N[m_0] \subseteq N[y] \cap N[m_0]$. The results of these two tests give $N[x] \subseteq N[y]$. To evaluate $N[x] \cup N[y] = V$, $(N[x] \cap N[m_0]) \cup (N[y] \cap N[m_0]) = N[m_0]$ iff $N(D,x) \cap N(D,y)$ is empty. By Theorem 5.4, we can compute this for all these adjacent pairs in linear time. We evaluate $(N[x] \cap U) \cup (N[y] \cap U) = U$ using Theorem 5.6.

3. Pairs of the form $\{x,y\}$ with $x \in N[m_0]$ and $y \in U$ are handled as in Part 2, except that $N[x] \subseteq N[y]$ must be obtained with Theorem 5.7. Since $|N[m_0]|$ is $O(m/n)$, the time bound is still $O(m)$.

□

Theorem 5.9 *It takes $O(n+m)$ time to determine that G is not a circular-arc graph, or else evaluate the following boolean expressions at each edge xy in G:*

$N[x] \cup N[y] = V$, and for each $z \in N[x] - N[y]$, $N[z] \subseteq N[x]$.

An $O(n^2)$ bound for evaluating this in the special case where the vertices can be partitioned into two cliques is given in [9]. (Their circular-arc recognition algorithm works by reducing the general problem to subproblems of this type.) We need a linear bound for arbitrary graphs. In the remainder of this section, we give the proof.

A *probe interval graph* is a graph $G = (V, E)$ and a partition $\{P, N\}$ of V such that $G|P$ is an interval graph, U is an independent set, and there is an interval realizer on V where edges of $P \times V$ can be realized with an interval model. There are no restrictions on adjacencies and nonadjacencies assigned by the realizer among members of N. Such a realizer of G is a *probe interval realizer.*

Theorem 5.10 *If G is a probe interval graph and the partition $\{P, N\}$ of its vertices, it takes $O(n + m + n|N|)$ time to construct a probe interval model.*

After we circulated an earlier draft of this paper with a proof of the following Theorem 5.10, it was pointed out that Johnson and Spinrad have shown an $O(n^2)$ variant of it [16]. Our proof of Theorem 5.10 is given in [15], where we also improve the time bound given by the theorem to $O(n + m \log n)$.

Theorem 5.11 *Let G be a graph, and let $U \subseteq V$ such that $G|U$ is a probe interval graph with partition $\{P, N\}$ of U. Given a probe interval model for $G|U$, it takes $O(n+m+n|V-P|)$ time to evaluate $N[x] \cap P \subseteq N[y] \cap P$ at all adjacent pairs $\{x, y\}$ such that at most one of x and y is in $V - U$.*

Proof: The algorithms of Theorems 5.6 and 5.7 can be trivially adapted to the problem, once a probe interval model R of $G|U$ is available. For Theorem 5.6, compute $L(w)$, $R(w)$, $L'(w)$ and $R'(w)$ in $R|P$, and let $l(y)$ and $r(y)$ denote the position of $y \in U$ relative endpoints of P. The test is now identical. An identical adaptation suffices for the algorithm of Theorem 5.7 given in [9]. □

With these results in hand, let us turn to the problem posed by Theorem 5.9. Flipping of arcs provides the required reductions. We again assume that G has no clique modules or universal vertices. Flipping y causes it to lose precisely those neighbors whose arcs, hence whose neighborhoods, are contained in $N[y]$. If $N[x] \cup N[y] = V$ then flipping y and testing whether its neighborhood is now contained in $N[x]$ tells whether for every $z \in N[y] - N[x]$, $N[z] \subseteq N[y]$.

Algorithm 5.12 *Build an auxilliary graph for Theorem 5.9.*

- *Compute neighborhood containments in G and* dominating neighbors *of the form $xy \in G(T)$ and $N[x] \cup N[y] = V$, using Theorem 5.8.*
- *For each such pair:*
  - *Find one, say, x, with degree at least as large as the other.*
  - *Insert a copy x' into G and make it adjacent to x, thereby making $\{x, x'\}$ a clique module.*
- *Let us call this graph H', and the set of new vertices F. Let R' be the realizer of H' derived from $T(G)$ by letting each added arc $x' \in F$ properly contain its copy x in V. (Since we do not know $T(G)$, we define R' in this way but do not compute it.) Let H be the circular-arc graph that results from flipping each $x' \in F$ in R'.*
- *Find a vertex m_0 that minimizes $|N[H, m_0] \cap V|$.*
- *Return m_0, $H|V$, and the edges of H that go between V and F.*

Lemma 5.13 *Algorithm 5.12 takes $O(n+m)$ time and $|F| = O(m/n)$.*

Proof: The size of H' is $O(n+m)$ and each vertex in F has degree $\Omega(n)$ in H', so the size of H is also $O(n+m)$, and $|F|$ is $O(m/n)$. We may spend $O(n)$ time on each member of F.

The critical observation is that the algorithm does not need to compute R, R', R_H or edges in $H|F$ in order to be able to return the result. $H|V = G$, so these edges are already available. For $x \in V$ and $y' \in F$, let y be the unflipped copy of y' in V. Since G has no clique modules, xy' fails to be an edge of H iff $N[G, x] \subset N[G, y]$. $N[G, x] \subset N[G, y]$ only if x and y are adjacent in G, so the results of this test are known from the first step of the algorithm. Since $|F|$ is $O(m/n)$, the time bound follows for these pairs. □

Lemma 5.14 *Let m_0 be a vertex of H that minimizes $|N[H,x] \cap V|$. If $m_0 \in V$, then there is a realizer R of H such that arc m_0 contains no other.*

Proof: Recall that G has no clique modules or universal vertices. Let R_G be the unknown realizer of $T(G)$ given by Theorem 5.2. Let R' be the realizer of H' obtained from R_G by adding a copy y' of each arc y of R that has a duplicate element of F in H'. To disambiguate what happens to the relationship of y and y' when y' is flipped, let us assume that the endpoints of y' have been stretched slightly so that y' properly contains y. In the circular ordering of endpoints in R', the endpoints of y and y' form two consecutive pairs. Let R_H be the realizer of H obtained by flipping the members of F.

Let z' be an arc such that $N[H,z'] \cap V = N[H,m_0] \cap V$. Since m_0 minimizes $N[H,x] \cap V$, only arcs of this form could be be contained in arc m_0 in R_H. If $z' \in V$, then m_0 and z' are a clique module of G, a contradiction. Thus, z' is the flipped copy of some $z \in V$. Because of the way the copy of z' is created, z and z' are nonadjacent, hence so are m_0 and z. The endpoints of z and z' make two adjacent pairs in the circular ordering of endpoints in R_H, so if arc z' fails to contain arc m_0 in R_H, z is a neighbor of m_0 in G, a contradiction. $\Box$

In what follows, we let $N[H,x]$ denote the neighborhood in H of $x \in V \cup F$ in H, and $N[x]$ denote the neighborhood in G of $x \in V$.

Let m_0 be the vertex returned by Algorithm 5.12. Below, we need for m_0 to be a member of V. If $m_0 \in F$, m_0 is the twin of some $x \in V$, and each $y' \in F$ is the twin of some $y \in V$. If x and y are adjacent and $N[x] \cup N[y] = V$, then we do not know whether m_0 and y' are adjacent, since we do not know whether x and y are a G_1 or a G_2 relationship. However, since we already know neighborhood containments among adjacent pairs in V we can find out in $O(n)$ time for all $z \in N[x] - N[y]$, whether $N[z] \subset N[x]$. Similarly, we can find for all $z' \in N[y] - N[x]$ whether $N[z] \subset N[y]$. (As we see below, however, this second test is unnecessary.) Since $|F| = O(m/n)$, all of these evaluations take $O(m)$ time. This disambiguates G_1 and G_2 edges incident to x, which gives all edges of $H|(V \cup \{m_0\})$ and all edges of H between m_0 and members of $F - \{m_0\}$. At this point, we reset $G = H|(V \cup m_0)$, $V = V \cup \{m_0\}$, $F = F - \{m_0\}$. We rerun Algorithm 5.12, but this time, there is no need to insert a copy of m_0 and flip it, since G already has one. We select the same m_0 to return as the first time. It is still true that $|N[H,m_0] \cap V|$ is minimized. Now, $m_0 \in V$, and Algorithm 5.12 gives us $H|V$ and edges of H from V to F. By Lemma 5.14, m_0's arc contains no other arc in some realizer of H.

Let $X = N[H,m_0] \cap V$, $Y = N[H,m_0] \cap F$. Define bipartite graphs $D_1(X \cup Y, V)$ and $D_2(X, V \cup F)$ as described in Theorem 5.5. The edges of H that are required to compute each of these are now available. Each must be chordal bipartite, as they are induced subgraphs of the unknown chordal bipartite graph on H defined by Theorem 5.5.

For each pair x, y such that $N[x] \cup N[y] = V$, at least one of x and y has a copy in F. Suppose without loss of generality that y has a copy y' in F. Suppose $N[x] \cup N[y] = V$. If for some $z \in N[x] - N[y]$, $N[z] \not\subset N[x]$, then z has a neighbor that x does not have, and this neighbor must be a neighbor z' of y that x does not have. Since $N[x] \cup N[y] = V$, this implies that $z' \in N[y] - N[x]$, and $N[z'] \not\subset N[y]$, since z' has a neighbor, z, that y does not have. Testing for all $z \in N[x] - N[y]$ whether $N[z] \subset N[x]$ suffices for the the seemingly stronger requirement of the fourth condition in the definition of $T(G)$, where the condition on z' is redundant. But the non-neighbors neighbors of y in V are exactly the neighbors of y' in V whose neighborhoods in V are contained in y'. *The condition is true iff* $N[H,y'] \cap V \subseteq N[H,x] \cap V$.

Thus, for Theorem 5.9, it remains to find for each $\{x,y'\}$ with $x \in V$ and $y' \in F$ whether $N[H,y'] \cap V \subseteq N[H,x] \cap V$.

Algorithm 5.15 *Evaluate* $N[H,y'] \cap V \subseteq N[H,x] \cap V$ *for each adjacent pair* xy' *in* H *with* $x \in V$ *and* $y \in F$.

Let $X = N[H,m_0] \cap V$ *and* $Y = N[H,m_0] \cap F$. *Note that* $H|(V-X) = G|(V-N[m_0])$ *is an interval graph.*

Break the problem into two parts: $N[H,y'] \cap X \subseteq N[H,x] \cap X$ *and* $N[H,y] \cap (V-X) \subseteq N[H,x] \cap (V-X)$.

Case 1 : $x \in V-X, y' \in Y$

- **a:** $N[H,y'] \cap (V-X) \subseteq N[H,x] \cap (V-X)$: *Apply the algorithm of Theorem 5.7.*
- **b:** $N[H,y'] \cap X \subseteq N[H,x] \cap X$: *Apply the algorithm of Theorem 5.3 to* D_2.

Case 2 : $x \in V-X,\ y' \in F-Y$

- **a:** $N[H,y'] \cap (V-X) \subseteq N[H,x] \cap (V-X)$: *Apply the algorithm of Theorem 5.7.*
- **b:** $N[H,y'] \cap X \subseteq N[H,x] \cap X$: *Apply the algorithm of Theorem 5.3 to* D_2.

Case 3 : $x \in X, y' \in Y$

- $N[H,y'] \cap V \subseteq N[H,x] \cap V$: *Apply the algorithm of Theorem 5.3 to* D_1.

Case 4 : $x \in X, y' \in F-Y$

- **a:** $N[H,y'] \cap X \subseteq N[H,x] \cap X$: *Apply the algorithm of Theorem 5.3 to* D_2.
- **b:** $N[H,y'] \cap (V-X) \subseteq N[H,x] \cap (V-X)$: $H_I = H|((V-X) \cup (F-Y))$ *is an interval graph, since a realizer of* H *fails to cover* m_0. *Let* H_P *be the probe interval graph obtained by omitting edges of* $H|(F-Y)$ *from* H_I. *Build a probe interval model for* H_P *using the algorithm of Theorem 5.10. Apply the algorithm of Theorem 5.11 on* $H|(V \cup (F-Y))$, *using* $U = (V-X) \cup (F-Y)$.

For the time bound, the matrices D_1 and D_2 have $O(m)$ entries each, since $|X \cup Y|$ is $O(m/n)$. Theorem 5.3 gives a linear bound for the cost of handling pairs in cases 1b, 2b, 3, and 4a. $|Y|$ and $|F-Y|$ are each $O(m/n)$, so Theorem 5.7 gives a linear bound for cases 1a and 2a, Theorem 5.10 gives a linear bound for building the probe model for case 4b, and Theorem 5.11 gives a linear bound for performing the neighborhood containment tests required for Case 4b.

This completes the proof of Theorem 5.9. By Theorems 5.5 and 5.9, we get the following summary, which completes the time bound of Step 1 of Algorithm 4.2:

Theorem 5.16 *If* G *is a circular-arc graph, it takes* $O(n+m)$ *time to to find an intersection matrix for a realizer of* G.

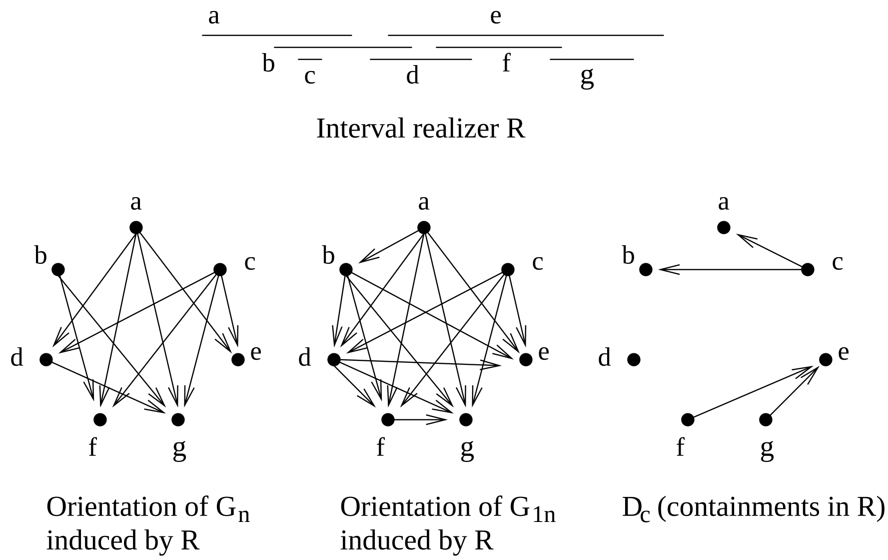


Figure 4: An interval orientation of an interval matrix is an orientation of the non-intersections (G_n) and the union of non-intersections and single-endpoint overlaps (G_{1n}) that is given by the relative order of two arcs in a realizer. The orientation of G_n is a subset of the orientation of G_{1n}; both orientations are transitive. The complement of G_{1n}, G_c, also has a transitive orientation, D_c, which is determined by which interval is contained in which. The union of D_c and the the interval orientation of G_{1n} is a linear order, which gives the order of right endpoints of R. The union of $(D_c)^T$ and the interval orientation of G_{1n} gives the order of left endpoints. R can be obtained from an interval orientation by interleaving these orders of left and right endpoints.

6 Finding an interval realizer of an interval matrix

In this section, we give the implementation of Step 3 of Algorithm 4.2.

When T is an intersection matrix from some realizer R of G, it is not the case that every realizer of G realizes T. This is because its partition of edges of G into D_c, G_1, and G_2 relations given by another realizer of G may not be faithfully reflected by the types of intersections in T. Therefore, T places additional constraints on possible realizers that G does not, and not all realizers of G satisfy the requirements of Step 3.

Let T be an interval matrix and R be an interval realizer of T. If $ab \in G_{1n}$, let us say that a *precedes* b in R if $l(a) < l(b)$, even if they overlap at one endpoint. (Because $ab \in G_{1n}$, this happens iff $r(a) < r(b)$.) Let the *interval orientation* of G_{1n} given by R be the orientation of its edges such that for each edge $ab \in G_{1n}$, (a, b) appears as a directed edge in the orientation iff a precedes b in R. (See Figure 4.) Then R *realizes* the orientation. An *interval orientation* of G_{1n} is any orientation of G_{1n} that has an interval realizer. An *interval orientation* T' of T is obtained by replacing elements of T labeled G_1 with D_1 or $(D_1)^T$, elements labeled G_n with D_n or $(D_n)^T$, so that the elements labeled D_1 and D_n give an interval orientation of G_{1n}, and the elements labeled $(D_1)^T$ and $(D_n)^T$ are its transpose. In this case, we say that R is a realizer of T'.

An interval orientation of G_{1n} gives a simultaneous transitive orientation of both G_n and of G_{1n}. To see why, note that if $ab, bc \in G_n$, then a and b are nonadjacent and b and c are nonadjacent. If a comes before b in a realizer and b comes before c, then a and c are nonadjacent, and a comes before c, providing a transitive edge (a, c) for the orientations (a, b) and (b, c) given by the linear extension. A similar argument applies to G_{1n}.

Therefore, G_n and G_{1n} are comparability graphs. In an interval matrix, G_2 is empty, so G_c is the complement of G_{1n}. G_c is also a comparability graph, since it has D_c as a transitive orientation. Since G_c and G_{1n} are complementary comparability graphs, they are permutation graphs.

Below, we show that it is easy to construct a realizer if an interval orientation of the interval matrix is given. Thus, we reduce the problem of finding an interval realizer of an interval matrix to that of finding an interval orientation of the matrix. This has the advantage that it allows us to apply tools and concepts that were developed for the transitive orientation problem.

A module of a graph can be viewed as the result of a substitution operation that is depicted in Figure 5. As we mentioned in Section 2, the edges that are internal to a module in a comparability graph may be oriented independently of those that are not when one wishes to compute a transitive orientation. (One must be careful not to create directed cycles among edges internal to different modules, however.) The modules completely specify the sets of edges that can be oriented independently of others [11, 12, 20]. There is a structure that has an analogous role in the problem of finding interval orientations of G_{1n}, and which we will call Δ modules.

To get an intuitive notion of a Δ module, let us define an operation on interval realizers that is analogous to the substitution operation on graphs, and which is depicted in Figure 6. Δ modules are always modules of the interval matrix for the set of intervals that result from the substitution operation. However, not all modules of the interval matrix are Δ modules. An example of this is given in Figure 7. The Δ modules completely specify the sets of edges that can be oriented independently when computing interval orientations, just as modules do this for transitive orientations.

6.1 Basic tools

Let the *mirror transpose* R^T of the ordered-list representation of R be the result of reversing the list and swapping the roles of left and right endpoints. It is what is seen if the realizer is held up to a mirror. If the interval graph represented by R is complete, then all left endpoints precede all right endpoints. In this case, let the *sandwich transpose*, R^S, denote the result of reversing separately the order of the left endpoints and the order of right endpoints, without swapping their roles. The next two lemmas are obvious.

Lemma 6.1 *Suppose an interval realizer R realizes the containment relation D_c and the interval orientation D_{1n} of G_{1n}. Then:*

1. *R^T realizes D_c and $(D_{1n})^T$*
2. *If R realizes a complete graph, then R^S realizes $(D_c)^T$ and $(D_{1n})^T$.*

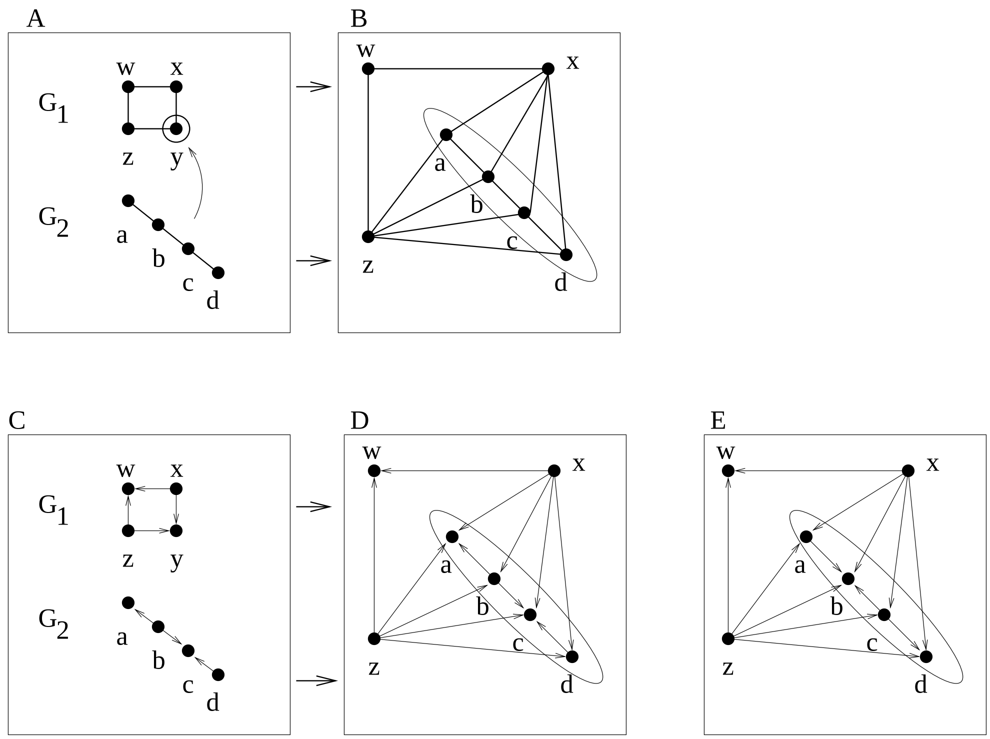


Figure 5: A module of a graph can be viewed as the result of a substitution operation on two graphs G_1 and G_2, where a vertex y of G_1 is replaced with G_2. The edges between vertices of G_2, and between the remaining vertices of G_1 are unaffected, and an edge is installed between a vertex v in G_2 and a remaining vertex u of G_1 if u and y are adjacent in G_1. The vertices of G_2 become a module in the composite graph. Figure B depicts the result of the operation on the two graphs of Figure A. Figure C depicts transitive orientations of G_1 and G_2, while Figure D illustrates that a substitution of transitively-oriented graphs produces a transitively oriented graph. Figure E shows that reversing the transitive orientation of G_2 produces a new transitive orientation of the composite graph that differs only in the orientations of edges internal to the module. In general, a graph with nontrivial modules can be viewed as the result of a substitution operation on smaller graphs, so the modules indicate the sets of edges that can be oriented independently of other edges when computing a transitive orientation of a graph.

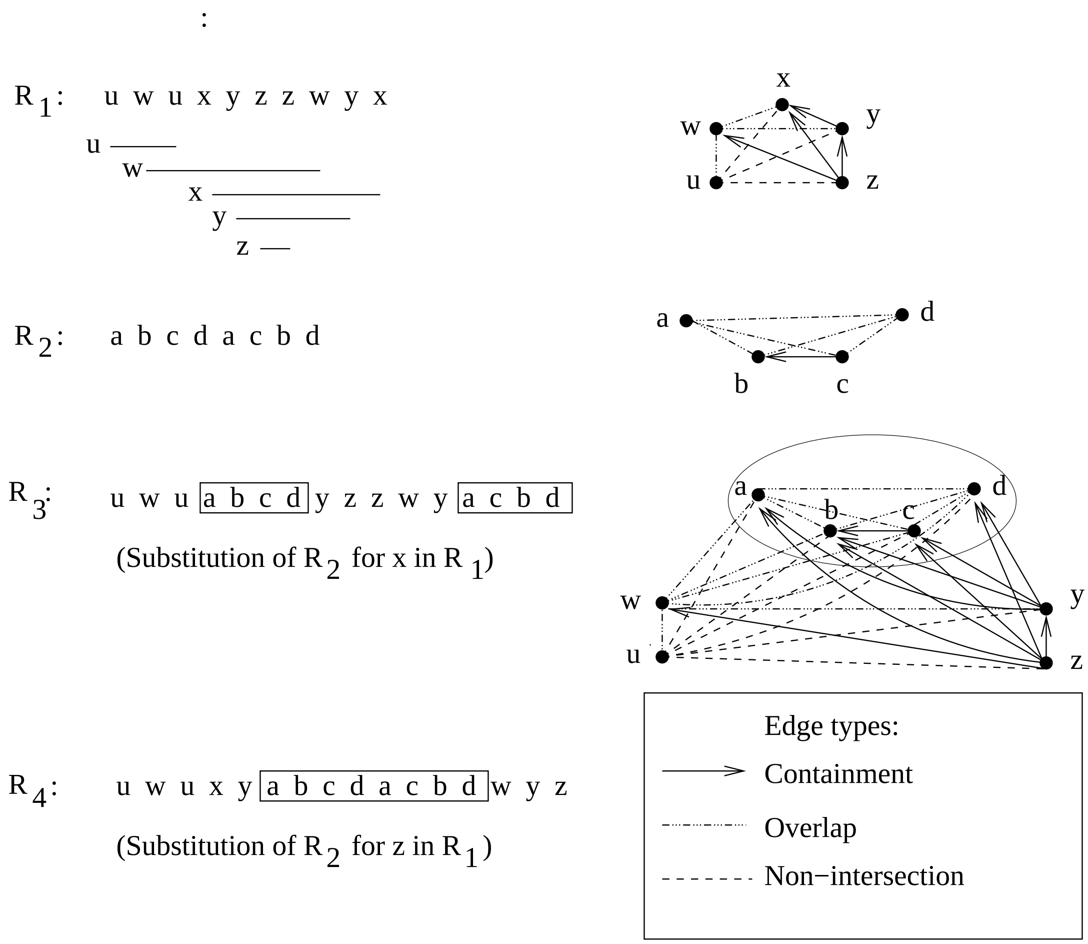


Figure 6: An interval realizer can be represented with a string with two letters in each string, as illustrated by R_1. The edge-labeled graph representing the associated interval matrix is shown to the right, with the labels given according to the legend. R_3 is the result of substituting R_2 for x in R_1, by replacing the left endpoint of x with the left endpoints of intervals in R_2 and the right endpoint of x with the right endpoints of intervals in R_2. A requirement when the two endpoints of x are not consecutive in R_1 is that all left endpoints of R_2 all precede all right endpoints in R_2, hence the interval graph represented by R_2 is a clique. Note that R_2 becomes a module in the resulting interval matrix. A second substitution operation that does not require R_2 to represent a clique is illustrated by R_4; a requirement of this substitution operation is that the two endpoints of the replaced interval in R_1 are consecutive. R_2 once again becomes a module in the resulting interval matrix, which is not depicted. A Δ *module* is a module of an intersection matrix that can result from one of these two allowed substitution operations. In either case, reversing the interval orientations of edges of R_2 before performing the substitution results in a new interval orientation of the final matrix. Thus, the Δ modules give sets of edges that can be oriented independently of other edges when computing an interval orientation of an interval matrix.

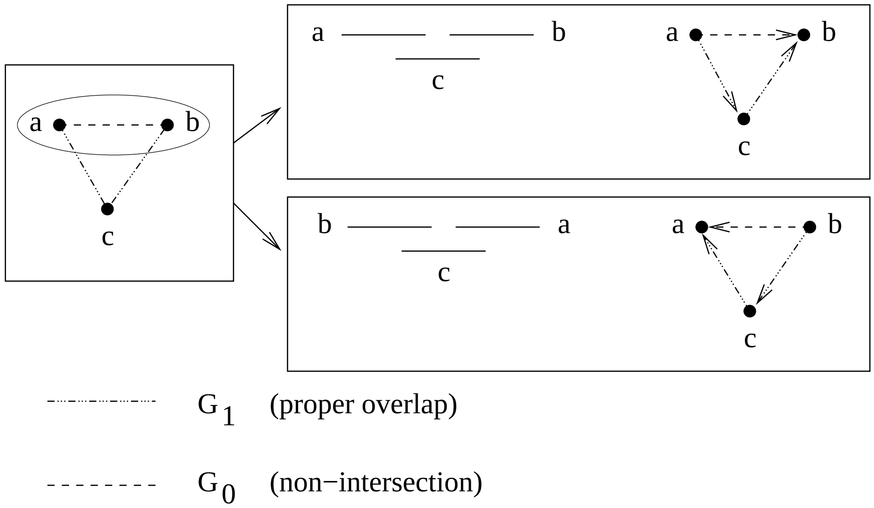


Figure 7: Not all modules of an intersection matrix are Δ modules. The module $\{a, b\}$ of the intersection matrix cannot be the result of either of the substitution operations described in Figure 6. The only two possible realizers and the interval orientations they induce illustrate that the orientations of edges internal to a non-Δ module cannot be oriented independently of other edges when computing an interval orientation.

Lemma 6.2 *If R is an interval realizer on the circle that realizes a complete interval graph, then flipping every arc in R yields R^S.*

Lemma 6.3 *If T is an interval matrix, then for any interval orientation, there is exactly one realizer.*

Proof: There must be at least one realizer by the definition of an interval orientation. It remains to show that there is only one.

Let a and b be two distinct vertices. If $ab \in G_{1n}$, $l(a) < l(b)$ and $r(a) < r(b)$ iff (a, b) is the orientation of ab in the interval orientation. If $ab \in G_c$, then $l(b) < l(a)$ and $r(a) < r(b)$ iff $(a, b) \in D_c$. Since the intersection matrix is an interval matrix, $ab \notin G_2$. The relative order of left endpoints is uniquely constrained by D_c and the interval orientation, and so is the relative order of right endpoints.

Let us now examine the relative order of pairs consisting of one left endpoint and one right endpoint. Let x and y be two vertices. If $x = y$, then $l(x) < r(y)$. Otherwise, suppose without loss of generality that $l(x) < l(y)$. Then $r(x) < l(y)$ iff $xy \in G_n$, and $l(y) < r(x)$ otherwise. The relative order of all pairs consisting of one left endpoint and one right endpoint are uniquely constrained. □

6.2 A Γ-like relation

Let us define an analog Δ of the Γ relation for the problem of finding an interval orientation instead of just a transitive orientation. (The Γ relation involves two edges joined at one end, resembling the letter Γ, while Δ involves three relationships that make a triangle.)

Let Γ_n denote the Γ relation on G_n, let Γ_1 denote the Γ relation on G_1, and let Γ_{1n} denote the Γ relation on G_{1n}.

Definition 6.4 *Let $\{a, b, c\}$ be three vertices. Then $(a, b)\Delta(a, c)$ and $(b, a)\Delta(c, a)$ if one of the following applies:*

- $(a, b)\Gamma_n(a, c)$ (i.e. $ab, ac \in G_n$ *and* $bc \in G_{1c}$);
- $(a, b)\Gamma_{1n}(a, c)$ (i.e. $ab, ac \in G_{1n}$ *and* $bc \in G_c$);
- $ab \in G_n$ *and* $bc, ac \in G_1$.

Let us call the last of the conditions in the definition of Γ a *straddle* relationship, since the edges $ac, bc \in G_1$ "straddle" edge $ab \in G_n$.

By analogy to Γ, we let the Δ **implication classes** be the equivalence classes of the transitive symmetric closure of Δ, and the Δ **color classes** be the union of each equivalence class and its transpose.

Theorem 6.5 *Every interval orientation of G_{1n} in an interval matrix consists of one Δ implication class from each Δ color class.*

Proof: For every edge $ab \in G_{1n}$, exactly one of (a, b) or (b, a) appears in any interval orientation. An interval orientation is a transitive orientation of G_n, so if $(a, b)\Gamma_n(a, c)$ then both or neither appear in any transitive orientation of Γ_n. It is also a transitive orientation of G_{1n}, so if $(a, b)\Gamma_{1n}(a, c)$ then both or neither appear in any transitive orientation of Γ_{1n}. If $(a, b)\Delta(a, c)$ by the straddle condition, with $ab \in G_n$ and $ac, bc \in G_1$, then a and b contain opposite endpoints of c in any interval realizer. The order of left endpoints must be (a, c, b) or (b, c, a). Either both or neither of (a, b) and (a, c) appear in any interval orientation. $\square$

We now show that Δ has the same dual relationship with a certain family of subsets of V as Γ has with modules of a graph. We will call this family the Δ modules. Every Δ module is a module, but some modules are not Δ modules. Like the set of modules, the Δ modules have a decomposition tree where every node is prime or degenerate, and where a set is a member of the family iff it is a node of the tree or a union of children of a degenerate node. The tree is not the modular decomposition tree, however. Just as in the case of Γ, the spans of the equivalence classes induced by Δ are nodes of the tree and pairs of children of degenerate nodes.

Definition 6.6 *Let T be an interval matrix. Let $U(T)$ denote the matrix obtained from T by replacing each instance of D_c or $(D_c)^T$ with G_c, thereby "unorienting" the directed edges in D_c. Let us say that a set M of vertices* **overlaps** *a set E' of edges of T if $T|M$ contains some members, but not all members, of E'. A module of T or of $U(T)$ is a Δ* **module** *if it is a module that is a clique of $G(T)$, or else a module X such that for no $y \in V - X$, $\{y\} \times X \subseteq G_1$.*

It is well-known that no module of a graph overlaps a Γ color color class [12]. The following is an analogous result about Δ modules and Δ color classes:

Lemma 6.7 *No Δ module overlaps a Δ color class.*

Proof: A module M of $U(T)$ is a module of G_n and a module of G_{1n}. Therefore, it cannot overlap a pair of edges that are related by the first two conditions of the definition of Δ. For the third condition, M must be a module of G_1. If it contains $\{a, c\}$, it must contain $\{b, c\}$, which implies that it contains $\{a, b\}$. If it contains $\{a, b\}$, then it is not a clique. If $c \in V - M$, then $\{c\} \times M \subseteq G_1$, and it is not a Δ module of $U(T)$. $\square$

Definition 6.8 *A family $\mathcal{F}$ of subsets of a set V is a* **tree-decomposable family** *if it satisfies the following properties:*

1. *V and the members of $\{\{x\} : x \in V\}$ are members of $\mathcal{F}$.*
2. **Overlap closure:** *If X and Y are properly overlapping members of $\mathcal{F}$, then $X \cup Y$, $X \cap Y$, $X - Y$, $Y - X$, and $(X - Y) \cup (Y - X)$ are members of $\mathcal{F}$.*

The **strong** *members of $\mathcal{F}$ are those members that properly overlap with no other member of $\mathcal{F}$.*

Theorem 6.9 *[21, 22, 5, 6] If $\mathcal{F}$ is a tree-decomposable family, then the transitive reduction of the containment relation on strong members of $\mathcal{F}$ is a tree. There is a unique way to label the nodes of this tree* **prime** *and* **degenerate** *so that a set is a member of $\mathcal{F}$ iff it is a node of the tree or a union of children of a degenerate node.*

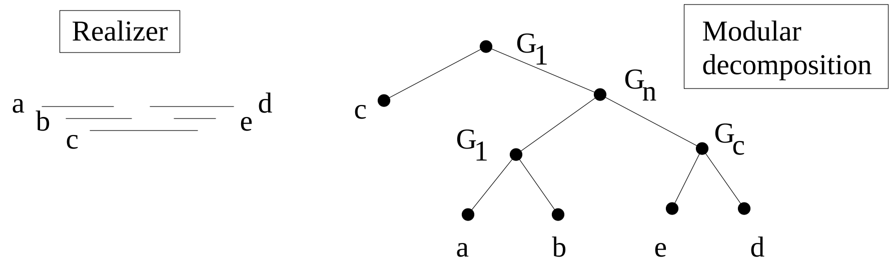
Let us call this tree the **tree decomposition** of $\mathcal{F}$. The modular decomposition of a graph or matrix is just the tree decomposition of its modules, which are a tree-decomposable family.

Theorem 6.10 *The Δ modules of $U(T)$ are a tree-decomposable family.*

Proof: The modules of a symmetric matrix are a tree-decomposable family [5, 6]. Let X and Y be overlapping Δ modules. Since they are modules, $X \cup Y$, $X \cap Y$, $X - Y$, $Y - X$, and $(X - Y) \cup (Y - X)$ are modules. If X and Y are both cliques of $G(T)$, then vertices in $X - Y$ have edges in $G(T)$ to $X \cap Y$, hence to all of Y. $X \cup Y$ is a clique, so all modules that are subsets of $X \cup Y$ are Δ modules, and the claim holds.

Otherwise, suppose without loss of generality that X is not a clique. Then there exists no vertex $w \in V - X$ such that $\{w\} \times X \subseteq G_1$. If $X \cap Y$ is a clique, it is a Δ module. If $X \cap Y$ is not a clique, then neither is Y, so there exists no $w \in V - Y$ such that $\{w\} \cap Y \subseteq G_1$. It follows that there exists no $w \in V - (X \cap Y)$ such that $\{w\} \times (X \cap Y) \subseteq G_1$, and $X \cap Y$ is a Δ module in this case also.

If $X - Y$ is a clique, it is a Δ module, since it is a module. If $X - Y$ is not a clique, then since X is a Δ module, any $w \in V - (X - Y)$ such that $\{w\} \times (X - Y) \subseteq G_1$ must reside in $X \cap Y$. Suppose such a w exists. Since w resides in Y and Y is a module, $(X - Y) \times Y \subseteq G_1$. For $w' \in Y - X$, $\{w'\} \times X \subseteq G_1$, a contradiction, since X is a Δ module.



U(T)

	a	b	c	d	e
a	*	1	1	n	n
b	1	*	1	n	n
c	1	1	*	1	a
d	n	n	1	*	c
e	n	n	1	c	*

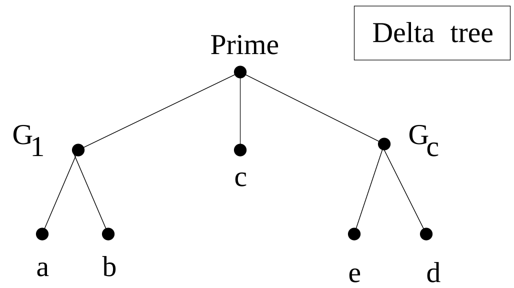


Figure 8: An interval realizer, $U(T)$, the modular decomposition of $U(T)$, and the Δ tree $\Delta(U(T))$.

If $U = (X - Y) \cup (Y - X)$ is a clique, it a Δ module, since it is a module. Otherwise, any vertex $w \in V - U$ such that $\{w\} \times U \subseteq G_1$ must lie inside X, since X is a Δ module. Therefore, $w \in X \cap Y$. Since X is a module and $(Y - X) \times \{w\} \subseteq G_1$, $(Y - X) \times X \subseteq G_1$. $(X - Y) \cup (Y - X)$ can only fail to be a clique if $X - Y$ or $Y - X$ fails to be a clique. Suppose $X - Y$ fails to be a clique. Then X fails to be a clique. $(Y - X) \times X \subseteq G_1$ contradicts the assumption that X is a Δ module. $\square$

Definition 6.11 *Let us call the tree decomposition of $U(T)$ the* **Delta tree** *of $U(T)$, and denote it $\Delta(U(T))$.*

Just like the modular decomposition, $\Delta(U(T))$ tree be represented in $O(n)$ space by creating a node of size $O(1)$ to represent each node $\Delta(T)$, and giving each node a label to indicate whether it is prime, G_n degenerate, G_1 degenerate, or linear, as well as a list of pointers to its children in the transitive reduction of the containment relation. The set U represented by a node can be retrieved in $O(|U|)$ time by visiting its leaf descendants, so there is no advantage to labeling a node with a list of its members. Figure 8 gives an example.

The following Theorems 6.12, 6.13, and 6.14 show that Δ modules satisfy the remaining properties of a class of set system described by Moehring in [21, 22]. In a forthcoming paper, we will deal with similar properties of Δ modules on T and on an interval orientation D_T of T, as well as algorithmic applications of their decomposition tree.

Theorem 6.12 *If X is a Δ module of $U(T)$ and Y is a subset of X, then Y is a Δ module of $U(T)$ iff it is a Δ module of $T|X$.*

Proof: If Y is a clique, the claim holds because of the corresponding theorem about modules of a matrix [5, 6]. If Y is not a clique, then neither is X, and since X is a Δ module, there exists no $y \in V - X$ with an edge of G_1 to any member of X, hence to any member of y. Then Y fails to be a Δ module in $U(T)$ iff there exists $y \in X - Y$ with G_1 edges to Y, which also determines whether it is a module of $U(T)|X$. □

Theorem 6.13 *If X is a Δ module of $U(T)$ and $Y \subseteq V$ intersects X, then $X \cap Y$ is a Δ module of $U(T)|Y$.*

Proof: The analogous theorem for modules of a matrix is given in [5, 6]. Thus, $X \cap Y$ is a module of $U(T)|Y$. X can only fail to be a Δ module of $U(T)|Y$ if $X \cap Y$ is not a clique and there exists $w \in Y - X$ such that $\{w\} \times (X - Y) \in G_1$. But this would imply that X is not a clique in $U(T)$ and $\{w\} \times X \in G_1$ in $U(T)$, contradicting X's status as a Δ module in $U(T)$. □

Theorem 6.14 *If $\mathcal{P}$ is a modular quotient on $U(T)$ consisting of Δ modules of $U(T)$, and $\mathcal{P}' \subseteq \mathcal{P}$, then $\mathcal{P}'$ is a Δ module of $U(T)/\mathcal{P}$ iff $\bigcup \mathcal{P}'$ is a Δ module of $U(T)$.*

Proof: By Lemma 6.7, if $\bigcup \mathcal{P}'$ is a Δ module, then there is no straddle relationship containing edges of $U(T)|\bigcup \mathcal{P}'$ and other edges. By Theorem 6.13, applied to the submatrix induced by one representative from each member of $\mathcal{P}$, this is also true of $\mathcal{P}'$ in $U(T)/\mathcal{P}$. On the other hand, if $\mathcal{P}'$ is a Δ module in $U(T)/\mathcal{P}$, then there exists $y \in V - \bigcup \mathcal{P}'$ such that $\{y\} \times \bigcup \mathcal{P}' \in G_1$ iff this is true for $\mathcal{P}'$ in $U(T)/\mathcal{P}$. If there exists such a y, then $\mathcal{P}'$ is a clique of $U(T)/\mathcal{P}$, and $\bigcup \mathcal{P}'$ can only fail to be a module if it fails to be a clique. This would imply that there is some $X \in \mathcal{P}'$ that is not a clique. But because $y \in V - X$ and $\{y\} \times X \subset G_1$, this would contradict the assumption that X is a Δ module. □

Lemma 6.15 *The set of vertices spanned by a Δ color class in an interval matrix T is a Δ module of $U(T)$.*

Proof: G_2 is empty in T, since T is an interval matrix. Let S be the set of vertices spanned by a color class C. Suppose that S is not a module. Then there exists $x \in V - S$ that has different relationships in T to members of S. C is connected, as only coincident edges are related by Δ. Therefore, x has different relationships between two vertices $y, z \in S$ such that $yz \in C$. At least one edge from x to $\{y, z\}$ is an edge of G_{1n}; without loss of generality, suppose that $xy \in G_{1n}$. If $xz \in G_c$ then $(x, y)\Gamma_{1n}(z, y)$, and $xy \in C$ contradicting x's non-membership in S.

Therefore, xy and xz are edges of G_{1n}. Without loss of generality, suppose $xy \in G_1$ and $xz \in G_n$. If $yz \in G_n$, then $(x, z)\Gamma_n(y, z)$, and we again get the contradiction $x \in S$. If $yz \in G_1$, then $(y, z)\Delta(x, z)$ by the straddle rule, and again, we get the contradiction $x \in S$. Since x does not exist, we conclude that S is a module.

Suppose that S is a module that is not a Δ module. Then it is not a clique, and there exists $w \in V - S$ such that $\{w\} \times S \subseteq G_1$. Suppose there exist $s_1, s_2 \in S$ such that $s_1s_2 \in G_n$ and $s_1s_2 \in C$. Then $(w, s_1)\Delta(s_2, s_1)$, and w is among the vertices spanned by C. Then Lemma 6.7 contradicts our demonstration that S is a module. Otherwise, C consists exclusively of edges of G_1, and there exist $s_3, s_4 \in S$ such that $s_3s_4 \in G_n$ and s_3s_4 is in a different color class C'. Let $P = (s_3 = x_1, x_2, ..., x_k = s_4)$ be a simple path in C from s_3 to s_4. Since s_3s_4 is in a different color class from edges on the path, and the edges on the path are edges of G_1, it follows by induction on i that $s_4x_i \in G_n$ and s_4x_i is a member of C' for each i from 1 to $k-1$. But this contradicts $s_4x_{k-1} \in G_1$. □

Theorem 6.16 *A set of edges of G_{1n} is a Δ color class iff it is the set of edges of G_{1n} connecting children of a prime node in the Δ tree of $U(T)$ or the set of edges of G_{1n} connecting a pair of children of a degenerate node. If X and Y are two children of a node of the tree, then no Δ implication class contains both directed edges in $X \times Y$ and directed edges in $Y \times X$.*

Proof: Let uw be an edge of G_{1n}. If uw connects two children of a prime node A, then all Δ modules that contains u and w contain A. The color class C containing uw must span A by Lemma 6.15. C cannot span a larger set or contain edges that are internal to a child of A, by Lemma 6.7, and the claim follows for C. If uw connects two children B_1 and B_2 of a degenerate node, an identical argument applies.

For the claim about the orientations of edges between X and Y, suppose first that X and Y are two children of a degenerate node Z. If Z is a G_c node, the claim is vacuous. Otherwise, suppose that $a, b \in X$ and $c \in Y$ such that (a, c) and (c, b) are in one implication class. Let A and B be the partition of X such that $A \times \{c\}$ and $\{c\} \times B$ are in the same implication class. If Z is a G_n node, then by transitivity of orientations of G_n in interval orientations, $A \times B \subset G_n$, and $B \cup Y$ is a Δ module of $U(T)$, contradicting X's status as a strong Δ module. If Z is a G_1 node, then X and Y are cliques of $G(T)$. By transitivity of interval orientations of G_{1n}, $A \times B \subseteq G_1$, and $B \cup Y$ is a Δ module, which is again a contradiction. If Z is Δ prime, then by Theorem 6.14, the submatrix induced by one representative vertex from each child of Z has only trivial modules. The orientation of an edge of this substructure cannot be reversed in an interval orientation without reversing all edges in it. If a is the representative of X and c is the representative of Y, then replacing a with b gives an interval orientation of an isomorphic submatrix that differs only in the orientation of a single edge, a contradiction. □

Lemma 6.17 *Let x be a source or sink in some interval orientation of G_{1n}, and let $\mathcal{P}$ denote $\{x\}$ and the maximal modules of $U(T)$ that do not contain x. Then every member of $\mathcal{P}$ is a Δ module.*

Proof: Suppose M is a member of $\mathcal{P}$ that is not a Δ module. Then M is not a clique of $G(T)$, and there exists $y \in V - M$ such that $\{y\} \times M \subseteq G_1$. Let s_1, s_2 be two nonadjacent vertices of M, and assume that s_1's right endpoint precedes s_2's left endpoint in a realizer where x is a source or sink. There exists a sequence $(x = x_1, x_2, ..., x_k = y)$, with $k \geq 1$, such that for each i from 1 to $k-1$, $x_i \in V - M$ and has a different relationship to x_{i+1}

from the relationship it has to M. Since x is a source or sink and has the same relationship to s_1 and s_2, it can have no endpoint between s_1's left endpoint and s_2's right endpoint. By induction on i, the same is true for each x_i in the sequence. But s_1 and s_2 properly overlap $x_k = y$, a contradiction. □

Lemma 6.18 *Let T be an interval matrix, R be an interval realizer of T, and D_{1n} be the interval orientation assigned to G_{1n} by R.*

1. *If Y is a node of $\Delta(T)$ that is a clique of $G(T)$, the left endpoints of Y are consecutive in R and the right endpoints of Y are consecutive in R.*
2. *If Z is a node of $\Delta(T)$ that is not a clique, the endpoints of Z are consecutive in R.*

Proof: The lemma follows almost immediately from Theorem 6.16. For Part 1, the left endpoints of Y precede the right endpoints of Y, since it is a clique. Let $x \in V - Y$. If $\{x\} \times Y \subseteq D_c$, then the endpoints of x lie between the left endpoints of Y and the right endpoints of Y. If $\{x\} \times Y \subseteq (D_c)^T$, then the left endpoint of x precedes all endpoints of Y, and the right endpoint of x follows all endpoints of Y. If $\{x\} \times Y \subseteq G_n$, then x is disjoint from all intervals in Y. If $\{x\} \times Y \subseteq G_1$, then, by Theorem 6.16, one endpoint of x lies in the common intersection of intervals in Y, and the other is disjoint from them; otherwise the interval orientation would contain a mixture of directed edges from x to Y and from Y to x. Since x is an arbitrary member of $V - Y$, no endpoint of a non-member of Y appears among left endpoints of Y or among right endpoints of Y.

For Part 2, if Z is not a clique, then it has no incident overlap edges, since it is a node of $\Delta(U(T))$. If there existed $x \in V - Z$ such that $\{x\} \times Z \subseteq D_c$, this would force all intervals in Z to contain both endpoints of x, making Z a clique. Thus, for every $x \in V - Z$, $Z \times \{x\} \subseteq D_c$ or $Z \times \{x\} \subseteq G_n$. In either case, x does not have any endpoints between two endpoints of Z. □

Theorem 6.19 *Any acyclic union of implication classes gives an interval orientation of G_{1n} in T.*

Proof: If W is a node of the Delta tree, then by Lemma 6.18, the endpoints of intervals realizing W in any interval realizer can be replaced with their mirror transpose. This reverses the orientations of edges of G_{1n} in the interval orientation that the realizer gives. Repeating this operation on the children leaves the net effect of reversing the orientations of edges that go between children of W.

If A and B are two consecutive children in the interval orientation that a realizer assigns to a a degenerate node C labeled G_n or G_1, then either all endpoints of A are consecutive and followed immediately by the consecutive endpoints of B, or the consecutive left endpoints of A are followed immediately by the consecutive left endpoints of B, and the consecutive right endpoints of A are followed immediately by the consecutive right endpoints of B. The orientation of edges between A and B can be inverted without affecting other orientations, by swapping the relative order of these groups of endpoints. By a sequence of such swaps, an arbitrary permutation of children of C can be induced.

By Theorem 6.16, all acyclic unions of implication classes can be obtained by a series of these swaps. By Lemma 6.7, this gives all interval orientations of G_{1n} in T. □

Johnson and Spinrad have independently developed an idea that is related to the Δ tree. Their tree, which they call an MD-PQ tree, has $2n$ leaves instead of n leaves, and each leaf represents an endpoint of an interval in a realizer of G, rather than a vertex. The set of permutations of the leaves represented by a set of allowable reorderings of children of nodes on the tree represents all realizers of G, rather than realizers of an intersection matrix. We jointly give a definition of a Δ tree on T in [15] that allows us to obtain Theorem 5.10, as well as a variant of the theorem that gives a linear time bound.

6.3 An algorithm for finding a realizer of an interval matrix

Given an interval orientation of an interval matrix in the form of a linear extension of the orientation of G_{1n}, it is easy to find the corresponding interval realizer in linear time. The union of D_c and the orientation of G_{1n} is a linear order on the vertices, and gives the order of right endpoints of the intervals, since if x is a vertex, the vertices with earlier right endpoints are those that are either predecessors in the orientation of G_{1n} or vertices whose interval is contained in x's interval. Similarly, the union of $(D_c)^T$ and the orientation of G_{1n} is also a linear order, and gives the order of left endpoints of the intervals.

Let P be the given linear extension of the interval orientation. To find the left-endpoint order L of vertices given by the union of $(D_c)^T$ and the interval orientation of G_{1n}, we find the position of each vertex x in L. This is obtained by adding up the number n_c of predecessors of x in D_c, the number n_1 of neighbors of x in G_1 that are predecessors in the orientation of G_{1n}, and the number n_n of neighbors of x in the orientation of G_n that are predecessors in the orientation of G_n. The first two of these can be easily found in $O(1 + N(G(T), x)$ time. Let $p(x)$ be the position of x in P. We find n_n by evaluating $n_n = (p(x) - 1) - (n_c + n_1)$. Doing this for all vertices takes time proportional to the sum of degrees of the vertices in $G(T)$, or $O(n + m)$ time. This is essentially the trick used in [18] for finding a realizer of a permutation graph, given transitive orientations of the graph and its complement.

We can create the full realizer by zipping these two permutations together in a way that reflects the adjacencies of each vertex. Let $v_1, v_2, ..., v_n$ be the vertices in left-to-right order of right endpoint. We place the right endpoint among the left endpoints $(l(v_1), l(v_2), ..., l(v_n))$, starting with $r(v_1)$, and working up through $r(v_n)$, placing $r(v_i)$ in the first position that is both to the right of $r(v_{i-1})$ and to the right of the rightmost left endpoint of neighbors of v_i.

Finding an interval realizer of an interval matrix therefore reduces to finding a linear extension of an interval orientation. Let us consider the difficulty of finding whether an orientation of G_{1n} given by a linear extension is an interval orientation. The currently known $O(n^2)$ algorithms for transitive orientation fail to recognize that the result that they produce is not transitive when the input graph is not a comparability graph. The resulting orientation always contains a pair (a, b), (b, c) of directed edges such that there is no edge ac. Such a pair is inconsistent with the Γ constraint. The difficulty of recognizing comparability graphs lies in the difficulty of finding such a pair when one exists.

In this paper, we use a variant of the $O(n+m)$ transitive orientation algorithm of [18] to produce an orientation that is consistent with Δ if the matrix is an interval matrix. Fortunately, though Δ resembles Γ, we can quickly detect whether some pair of directed edges in an orientation is inconsistent with Δ (or with the containment relation) by constructing an interval realizer of it, and verifying that the realizer realizes T.

We have given an algorithm for finding an interval realizer from a linear extension of an interval orientation. The problem reduces to finding such a linear extension. We now give an algorithm for this that is based on the transitive orientation algorithm of [18], which uses the Γ relation to constrain the orientation it produces. We adapt the algorithm to use the Δ relation instead of the Γ relation to constrain the orientation.

Let $(X_1, X_2, ..., W, ..., X_{i-1}, Y, X_{i+1}, ..., X_k)$ be an ordering of partition classes on vertices of T. (A requirement is that $W \neq Y$). Let $w \in W$, let $Y_n = N(G_n, w) \cap Y$, $Y_1 = N(G_1, w) \cap Y$, and $Y_c = N(G_c, w) \cap Y$. A *pivot on* Y *with pivot vertex* w consists of the following refinement of the partition: (X_1, X_2, ..., W, ..., X_{i-1}, Y_c, Y_1, Y_n, X_{i+1}, ..., X_k). If W is later than Y in the ordering, a symmetric operation also constitutes a pivot: (X_1, X_2, ..., X_{i-1}, Y, X_{i+1}, W, ..., X_k) becomes (X_1, X_2, ..., X_{i-1}, Y_n, Y_1, Y_c, X_{i+1}, ..., W, ..., X_k). (Notice that the order of Y_n, Y_1, Y_c, is reversed.) We assume that if Y_c, Y_1, or Y_n is empty, it is removed from the refinement. Thus, the refinement is not necessarily a proper one, since as many as two of them might be empty. Finally, let us also allow a pivot to fail to separate Y_n and Y_1 or Y_1 and Y_c, leaving their union in the spot in the ordering that would otherwise have been occupied by the two sets. This is a *lazy pivot.*

This definition is based on the pivot of [18], which divides Y into $N(w) - Y$ and $N(w) \cap Y$. We have defined a lazy pivot on T to help with a reduction to pivoting on G_c and G_{1c} that we use below: a pivot in G_c or G_{1c} is a lazy pivot in T.

We will say that an ordering of partition classes on the vertices is *consistent* with an interval orientation of T if all edges of G_{1n} that go between two partition classes are oriented from an earlier to a later class.

Lemma 6.20 *Let T be an interval matrix.*

1. *If a sequence of partition classes on vertices of T is consistent with some interval orientation D of G_{1n} in T, then they remain consistent with D after a pivot.*
2. *Let X be the rightmost class in a sequence of partition classes before a pivot, let X' be the rightmost class after a pivot and let $x \in X'$. If there exists an interval orientation D' of G_{1n} where all edges of D' between $V - X and \{x\}$ are oriented toward x, then all edges of D' between $V - X'$ and $\{x\}$ are also oriented toward x.*

Proof: For Part 1, let ab be an edge of G_{1n} such that a occurs in a class that is earlier in the sequence (Y_c, Y_1, Y_n) than b does. If $a \in Y_c$ and $b \in Y_1 \cup Y_n$, then $(a,b)\Gamma_{1n}(w,b)$, implying $(a,b)\Delta(w,b)$. Suppose $a \in Y_1$ and $b \in Y_n$. Then if $ab \in G_n$, $(a,b)\Gamma_n(w,b)$, implying again that $(a,b)\Delta(w,b)$, and if $ab \in G_1$, then $(a,b)\Delta(w,b)$ by the straddle condition.

If w is to the left of Y, then (w,b) and (w,a) are directed edges in D, hence so must (a,b). The classes are ordered so that a is in an earlier class than b is, and (a,b) is oriented from an earlier to a later class in the ordering. If w is to the right of Y, then (b, w) and (a, w) are directed edges in D, and $(b, w)\Delta(b, a)$ or $(a, w)\Delta(b, a)$, so (b, a) is also a directed edge in D. The classes are ordered so that b is in an earlier class than a is, and the classes are again ordered so that the orientation (b, a) points to the right. Since ab is an arbitrary edge whose endpoints were split into two classes by the pivot, the condition of Part 1 remains true after the pivot if it was true before.

For Part 2, the proof is identical, except that D' takes the place of D, X takes the place of Y, x takes the place of b, and the pivot w is to the left of X. □

This gives the following algorithm for interval orientation of an interval matrix, which was originally given as a transitive orientation algorithm in [19], using a lemma analogous to Lemma 6.20 for the transitive orientation problem.

Algorithm 6.21 `Orient(T)` *finds an interval orientation of an interval matrix T, using recursive vertex partitioning.*

Partition $(T, \mathcal{P}')$
  While there exists $Y \in \mathcal{P}$ that is not a module
    Perform a pivot that splits Y
  Return $\mathcal{P}'$

RPartition (T) // *v is a vertex; vertices are numbered*
  Let $\mathcal{P} = (V)$
  While not every partition class in $\mathcal{P}$ is a singleton set
    Let Z be non-singleton class of $\mathcal{P}$
    Let $\mathcal{P}'$ be an arbitrary ("capricious") partition of Z
    $\mathcal{Z}$:= `Partition (T|Z,`$\mathcal{P}'$`)`
    Substitute $\mathcal{Z}$ for Z in the ordering on $\mathcal{P}$
  Return the ordering of V given by $\mathcal{P}$

HighLowSplit(T)
  Select v to be the highest-numbered vertex in T
  Return $(\{v\}, V(T) - \{v\})$.

HighLowRPartition(T)
  Run RPartition(T), but each time a capricious partition of a set is required, use HighLowSplit

`Orient` *(T)*
  Number the vertices in any order
  Run `HighLowRPartition` (T) *to get an ordering of the vertices*
  Number the vertices in left-to-right order in this ordering
  Run `HighLowRPartition` (T) *to get a new ordering of the vertices*
  Return this ordering

Lemma 6.22 *If M is a Δ module of $U(T)$, the presence of vertices in $V - M$ has no effect on the relative ordering of members of M produced by* `HighLowPartition`.

Proof: In each call, if a and b are two members of M, they are initially in the same partition class, V, and in the end they are in two different partition classes, $\{a\}$ and $\{b\}$. The event that first splits them into two classes is either a call to `HighLowSplit` or a pivot on some vertex w. In the former case, the presence of non-members of M has no effect on the relative order of $\{a\}$ and $\{b\}$, since this depends only on the order of the numbers assigned to a and to b. In the latter case, $w \in M$, since it must distinguish a and b in order to split them into separate classes. $\square$

Lemma 6.23 *Let M be a module of $U(T)$.*

1. *After the first call to* `RPartition` *from* `Orient`*, the rightmost vertex in M is a sink in an interval orientation of $G_{1n}|M$.*

2. *If M is a Δ module of $U(T)$, then after the second call to* `Rpartition` *from* `Orient`*, the ordering of vertices in M is a linear extension of an interval orientation of $G_{1n}|M$.*

Proof: For Part 1, suppose by induction that the claim is true for modules that are proper subsets of M. M is first split up when a single vertex of M is removed from a partition class Z that contains M. `Partition` returns $\mathcal{Z}$, where all members of $\mathcal{Z}$ are modules of $U(T)$. For each $X \in \mathcal{Z}$ that contains members of M, $X \cap M$ is a module in $U(T)|M$, by Theorem 6.13, and collectively, these intersections partition M into modules of $T|M$. By Lemmas 6.22 and 6.20 (Part 2), and induction on the number of pivots in the while loop of `Partition`, the rightmost class M' in this partition of M is a sink with respect to edges between $M - M'$ and M'. By Lemma 6.22 and the assumption that Part 1 is true for smaller modules than M, the subsequent partition of M' places a sink s in an interval orientation of $G_{1n}|M'$ in the last position of the ordering it returns. If M' is a Δ module, there can be no Δ constraints between edges of $G_{1n}|M'$ and edges of $G_{1n}|M$ that are not edges of $G_{1n}|M'$, by Lemma 6.7. If M' is not a Δ module, there can be vertices of $M - M'$ that have G_1 edges to M', which interact with edges of $G_n|M'$. But these constrain all edges of $G_n|M'$ incident to s to all be directed toward s or all be directed away from s. There is an acyclic union of implication classes where s is a sink, by Theorem 6.19.

Let us now consider the second claim. M is first split up when a source in an interval orientation of $T|M$ is split off from M. The partition $\mathcal{Z}$ returned by `Partition` consists of Δ modules, by Lemma 6.17. Since M is a Δ module, the partition $\mathcal{M}$ that they induce in M consists of Δ modules of $T|M$, by Theorem 6.10. The ordering of $\mathcal{M}$ is consistent with an interval orientation of $T|M$, by Lemmas 6.20 and 6.22. By induction on the size of a Δ module, the remaining pivots produce interval orientations of each member of $\mathcal{M}$, and since they are Δ modules, these orientations can be combined with the ordering on $\mathcal{M}$ to give an interval orientation of $T|M$, by Theorem 6.19. $\square$

Since V is a Δ module, the second call to `RPartition` produces a linear extension of an interval orientation. Once `RPartition` is run, an interval realizer may be constructed and checked in linear time, using Algorithm 4.1. The bottleneck for the time bound is the running time of `RPartition`.

6.4 An $O(n + m \log n)$-time implementation

We have already shown that the time to construct a realizer of an interval matrix is given by the running time of `RPartition` on the matrix.

Algorithm 6.24 gives an implementation of the `while` loop of `RPartition` that allows `RPartition` to run in $O(n+m\log n)$ time. The algorithm is similar to the approach of [19] and [13]. Retrieving E' takes time proportional to the sum of degrees of X. The grouping operations on E' take $O(|E'|)$ time, by distributing the edges to buckets that are initially empty, and maintaining a list of nonempty buckets to allow retrieval of the groups without visiting empty buckets, leaving the buckets empty again. The selection of X ensures that each time a vertex is used as a pivot, the partition class that currently contains it is at most half as large as the class that contained it the last time it was used as a pivot. A vertex is therefore used $O(\log n)$ times as a pivot. A pivot on x requires $O(|N(x)|)$ time, giving the bound.

This clever idea is due to Spinrad, and initially appeared in an unpublished but influential manuscript, which he has freely circulated beginning in 1985 [24]. (This paper was also the first to propose vertex partitioning as an algorithmic tool, and to recognize its importance to the modular decomposition and transitive orientation problems.) Algorithm 6.24 is a variation of his time-bound argument that bounds the cost of partitioning inside all recursive calls to `RPartition`, instead of just in the main loop.

Algorithm 6.24 *Vertex partitioning.*

Partition $(G, \mathcal{P})$
  If $|\mathcal{P}| = 1$ *return* $\mathcal{P}$
  else
    Let X *be a member of* $\mathcal{P}$ *that is not larger than all others*
    Let E' *be the edges of* G *in* $X \times (V(G) - X)$.
    Group members of E' *by endpoint in* X
    Use each $x \in X$ *as a pivot on all partition classes in* $V - X$
    Regroup members of E' *by endpoint in* $V - X$
    Use each $v \in V - x$ *as a pivot on all partition classes in* X
    Let $\mathcal{Q}$ *be the classes of* $\mathcal{P}'$ *now contained in* X
    Let $\mathcal{Q}'$ *be the classes of* $\mathcal{P}'$ *now contained in* $V(G) - X$
    Return $\mathit{Partition}(G|X, \mathcal{Q}) \cup \mathit{Partition}(G - X, \mathcal{Q}')$

Because of its simplicity, it would be the method of choice in a practical implementation of the algorithm of this paper, in our view. This is a bottleneck in the circular-arc graph recognition algorithm. To get a linear time bound for recognizing circular-arc graphs we need a linear bound for `RPartition`. To do this we use a considerably more complicated strategy, which we give below. First, however, we show how `RPartition` can be used to obtain the Δ tree.

6.5 Linear-time implementation

From a theoretical standpoint, an $O(n + m \log n)$ bound for `RPartition` would be a bottleneck in our circular-arc graph recognition algorithm, and would result in an $O(n + m \log n)$ bound for it also.

To get a linear bound, we must make use of the considerably more difficult linear-time approach to vertex partitioning that is given in [18]. In this section, we prove the following:

Theorem 6.25 `RPartition` *can be carried out in* $O(n + m)$ *time.*

This is the most difficult part of the paper, since it requires a summary description of tricks that are explained in greater detail in [18]. There is no loss of continuity if one accepts Theorem 6.25 provisionally on a first reading and skips to Section 7.

The following are given in [18]:

Theorem 6.26 *Let G be a prime graph, and let $\mathcal{P}$ be an arbitrary initial partition of V.* `Partition` *may be implemented so that it takes $O(n + m)$ time to halt with all partition classes singletons.*

Theorem 6.27 *It takes linear time to find the modular decomposition of an undirected graph.*

By Theorem 6.26, it is immediate that we can run `Partition` in linear time on a prime graph; `RPartition` partitions V down to singleton sets. The paper solves the general transitive orientation problem through a reduction to prime graphs. In addition, the algorithm is specific to undirected graphs, and it is not immediate that it can be generalized even to prime interval matrices. For our application, we must be able to run `RPartition` on $U(T)$, where T is an interval matrix that need not be prime, and still get a linear time bound for `RPartition`.

We develop our algorithm in two steps. In the first, we show how to perform `RPartition` on an undirected graph that need not be prime, in linear time, and in the second, we show how to do it on $U(T)$, where T is an interval matrix that need not be prime.

6.5.1 Running `RPartition` on a graph that need not be prime.

Let $G = (V, E)$ be an undirected graph, and let $\mathcal{M}_G$ denote the family of modules of G. Let T be a family of subsets of V where the transitive reduction of the containment relation is a tree, and where V and its singleton subsets are members of T. Moreover, suppose that each node of the tree be labeled *prime* or *degenerate*. Let $\mathcal{F}(T)$ denote the family of sets where $X \in \mathcal{F}(T)$ iff it is a node of T or a union of siblings whose parent is a degenerate node. T is an *M tree* on G if $\mathcal{M}_G \subseteq \mathcal{F}(T)$. If T is the modular decomposition of a graph, then it is an M tree, and $\mathcal{M}_G = \mathcal{F}(T)$.

In general, an M tree looks like the modular decomposition, except that when it is interpreted as the modular decomposition, it exaggerates the number of modules of G.

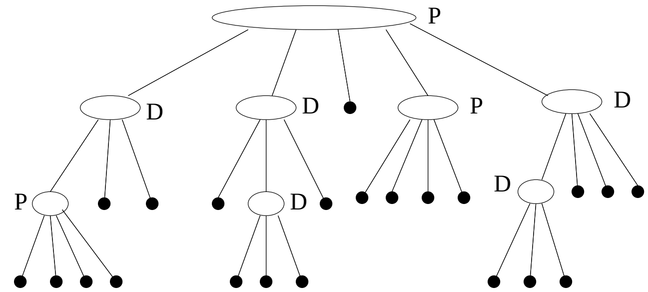


Figure 9: An M tree is a tree of defined on $G = (V, E)$. Each node of the tree is a subset of V. The root is $\{V\}$ and the leaves are vertices of G. Each internal node is the set of vertices given by its leaf descendants. Each node of the tree is labeled "prime" or "degenerate", and every module of G is either a node of the tree or a union of children of a degenerate node. In contrast to the special case of modular decomposition, not every node or union of children of a degenerate node is required to be a module of G. In linear time, one may sort all adjacency lists of G in leaf order, so that members of each internal node are consecutive in each adjacency list. Also in linear time, one may label each internal node U that is not a module with a vertex $s(U) \in V - U$ that has both neighbors and non-neighbors in U, and label U with a pointer to the consecutive region of $s(U)$'s adjacency list that is occupied by $N(s(U)) \cap U$. If U is a prime node, calling the restarting procedure a second time now causes each partition classes in U to be contained in a child of U, and we may repeat the process on the children. A similar set of observations apply when U is a degenerate node.

However, it never implies that a module of G is not a module. If T_1 and T_2 are M trees, then T_2 is *more restrictive* than T_1 if $\mathcal{F}(T_2) \subset \mathcal{F}(T_1)$. The modular decomposition tree is the most restrictive one possible for G. The least restrictive M tree possible has V as a degenerate root and its singleton subsets as its children, since then $\mathcal{F}(T)$ is the power set of V.

The modular decomposition algorithm of [18] constructs a series of length $O(1)$ of M trees, each one more restrictive than the previous, until the modular decomposition is obtained. In constructing T_i from T_{i-1} in the sequence, the algorithm makes use of vertices that demonstrate that each member of $\mathcal{F}(T_{i-1}) - \mathcal{F}(T_i)$ are not modules. Each of these vertices is a non-member of some of the sets that has both neighbors and non-neighbors in them, which discredits them as modules. The main insight behind the transitive orientation algorithm is that these same vertices can be used as pivots during vertex partitioning whenever a class $X \in \mathcal{P}$ arises that is a member of $\mathcal{F}(T_{i-1}) - \mathcal{F}(T_i)$. Let us call these vertices T_i's *pivots*.

Each node of an M tree is represented with an $O(1)$-sized structure that has a pointer to a doubly-linked list of its children. The set that the node represents is given by its leaf descendants.

The members of $\mathcal{P}$ are implemented with doubly-linked lists, and when splitting $Y \in \mathcal{P}$ with a pivot w, we remove neighbors of w from Y, and let what remains of Y stand for the remaining partition class. This avoids the cost of touching non-neighbors of w in Y. Let us call this a *removal operation*.

Lemma 6.28 *[18] (See Figure 9) Given an M tree T on graph G, one may perform the following in time linear in the size of G:*

1. *Sort all adjacency lists so that the neighbors of each vertex x are given in order of their left-to-right appearance as leaves of T. For each node of the tree, the members of the node are therefore consecutive in the adjacency list of x.*

2. *Label each node U of the tree with a sorted list $SN(U)$ of vertices in $V - U$ that are adjacent to all members of U. $SN(U)$ is sorted in left-to-right appearance of its members as leaves of T, and the sum of lengths of these lists is linear in the size of G.*

3. *Label each node U that is not a module with the leftmost vertex $l(U) \in V - U$ and rightmost $r(U) \in V - U$ that is adjacent to some members of U and nonadjacent to others; since U is not a module, at least one of these is defined. Let $s(U)$ denote one of them. U is also equipped with a pointer to the consecutive interval occupied by $N(s(U)) \cap U$ in $s(U)$'s adjacency list.*

The algorithm assigns a private copy of an adjacency-list representation of G to each T_i, and applies Lemma 6.28 to each. It uses a *restarting procedure* on each T_i, which initiates pivots on $\mathcal{P}$, halting only when $\mathcal{P} \subseteq \mathcal{F}(T_i)$. (See Figure 10.) To do this, it makes a call to the restarting procedure on T_{i-1}. When that restarting procedure halts, it makes use of T_i's pivots to split some classes, communicates these splits to T_{i-1} as capricious splits, and then calls the restarting procedure on T_{i-1} to get the partitioning restarted. The procedure fails to find any pivots to restart partitioning only if $\mathcal{P} \subseteq \mathcal{F}(T_i)$, in which case it is allowed to halt.

The restarting procedure is *linear* if it may be restarted $O(n+m)$ times without spending more than $O(n + m)$ total time, including the time to perform the pivots it generates and to maintain the list of unprocessed nodes. All of the restarting procedures in the sequence of M trees are linear ones.

Notice that a linear restarting procedure on the last tree in the sequence, $MD(G)$, would suffice for a linear-time implementation of `RPartition`. The restarting procedure could perform the `while` loop of `Partition`, halting when every partition class is a member of $\mathcal{F}(MD(G))$ (*i.e.* a module). Capricious splits such as `HighLowPartition` could then initiate arbitrary splits of these modules and call the restarting procedure again. By the definition of a linear restarting procedure, the time for all calls to the restarting procedure would be linear.

General linear restarting procedures are given in [18] for all trees in the sequence except for the very last one, which is the modular decomposition of G. This is unfortunate, since doing this requires only a straightforward extension of the techniques used for the other restarting procedures, which must now be resurrected here. The reason that this was not done is that when the modular decomposition is available, transitive orientation of a comparability graph reduces to transitive orientation of prime graphs, and this step was unnecessary for transitive orientation.

The reason pivoting risks being wasteful is that the class that contains a pivot vertex x cannot be split by x, so time might be wasted traversing members of x's class in splitting

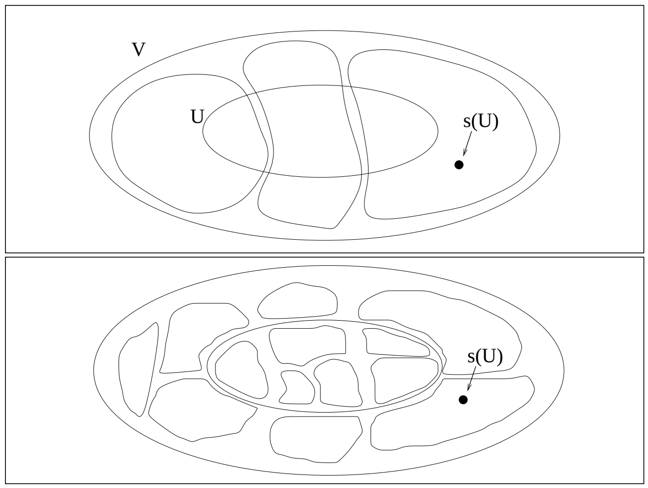


Figure 10: A *restarting procedure* on an M tree performs pivots in G until every partition class is a node or a union of children of of a degenerate node. The top illustration depicts a node U of the M tree that intersects more than partition class in the partition of V. The bottom illustration depicts the situation after the restarting procedure for the tree is called. U now contains all partition classes that it intersects. At this point, U may be "processed", by performing a pivot on classes that U intersects, using the interval occupied by $N(s(U) \cap U)$ in $s(U)$'s adjacency list. By using the pointer to to this region carried by U, this pivoting takes time proportional to $|N(s(U) \cap U)|$. These elements then become irrelevant to future pivots, and may be discarded from $s(U)$'s adjacency list. If U is prime, the partition classes that intersect U are each contained in a child of U. These can now be handled by processing children of U. If U is degenerate, the iteration on children is somewhat more complicated, but similar in principle.

up classes that do not contain x. After a sequence of later pivots, some of these might lie in a different class from x, and these portions of x's adjacency list might have to be re-traversed to look for these elements. Bounding the number of re-traversals is the main challenge.

The basic technique of [18] is to use the restarting procedure on T_{i-1} to impose an organized structure on the partition classes so that a set of classes that are to be split by a pivot x in T_i occupy a consecutive region of x's adjacency list, which is sorted by leaf order in T_i. This region of the list can then be discarded as it can have no effect on the results of future pivots. No elements in the list have to be re-traversed later.

The last tree in the sequence before the modular decomposition is is called an *M2 tree*. An $M2$ tree is an M tree with the following property: for any node U that is labeled degenerate, every union of children of U is a module of $G|U$ (but not necessarily of G).

To get the restarting procedure for the modular decomposition, let us insert a new M tree M_p in the sequence between the M2 tree and the modular decomposition. (M_p is the "penultimate" tree.) We do this by creating a copy of the $M2$ tree, and performing the operations of Lemma 6.28 on it. For each degenerate node U, let $\mathcal{A}$ be its children that are modules of G. $\mathcal{A} = \{W : W$ is a child of U such that $s(W)$ is undefined$\}$. We install a new child A of U representing $\bigcup \mathcal{A}$, and remove the members of $\mathcal{A}$ and their subtrees from the list of children of U and make them the list of children of A. We label A degenerate.

By concurrently traversing the lists $\{SN(A) : A \in \mathcal{A}\}$, we may either conclude that A is a module of G, or else find a first vertex x that is adjacent to a proper subset $\mathcal{A}_a$ of $\mathcal{A}$, and let $\mathcal{A}_n$ be the remaining subset that are nonadjacent to x. When x exists, we let $s(A) = x$, and create two new nodes A_a and A_n standing for $\bigcup \mathcal{A}_a$ and $\bigcup \mathcal{A}_n$. We remove the members of $\mathcal{A}_a$ from the doubly-linked list of children of $\bigcup \mathcal{A}$ in $O(|\mathcal{A}_a|)$ time and make them children of A_a, and then move the remaining list of children of $\bigcup \mathcal{A}$ in $O(1)$ time and make it the list of children of A_n. We then recurse on $\mathcal{A}_a$ and $\mathcal{A}_n$, using what remains of the SN lists in each of these sets. The base case is reached when a recursive call is made on a single member of $\mathcal{A}$ or on a subset $\mathcal{A}_m$ of $\mathcal{A}$ such that $|\mathcal{A}_m| > 1$ and $\bigcup \mathcal{A}_m$ is a module in G. The latter case is recognized when what remains of the $SN()$ lists of members of $\mathcal{A}_m$ are exhausted without discovery of a splitting vertex x. In this case, the corresponding node A_m is labeled degenerate, and its children are $\mathcal{A}_m$.

When we are done, we sort adjacency lists in M_p's private copy of G by leaf order in M_p so that each node of the tree occupies a consecutive portion of each adjacency list.

The restarting procedure given in [18] for the M2 tree is called *M2Resolve()*. As partitioning on $\mathcal{P}$ proceeds, *M2Resolve* a list of "unprocessed" nodes of T_i, each of which contains every partition class that intersects it. Though it is not necessary for the algorithm, it helps to understand it if one assumes that the list is ordered so that ancestors appear before descendants. The list supports permanent removal of a maximal unprocessed node from the front of the list. (This node is used to assist in identifying promising pivots.) When a pivot is performed, it supports identification and insertion of all new nodes that now satisfy the requirement of containing partition classes that intersect them, in an appropriate order at the end of the list. The list can be regarded as an abstract data type that updates itself each time a partition class is split.

This requires an initial overhead of $O(n+m)$ time to set up. After that, the amortized time to update this list when a partition class Y is split by removing a set Y_a from Y is $O(|Y_a|)$, which is the cost of the removal operation. This is true whether the split is the result of a pivot, or is a capricious split of the class that is decided upon by some other process.

To initiate the pivoting, we call *M2Resolve*, beginning with the initial partition $\mathcal{P}$. When it returns, each member P in the refined partition $\mathcal{P}$ is either a node of the $M2$ tree or a union of children of a degenerate node of $M2$.

Let W be a maximal unprocessed node of $M2$. We remove W from the list. If the vertex $s(W)$ that splits W does not exist, W is a module, and we proceed to the next maximal unprocessed node in the list. The reason this is justified is that *M2Resolve* has halted. Each partition class that intersects W is either W itself, or contained in one of its children if W is prime, or is contained in a child or a union of children if W is degenerate. If W is a partition class, the class is a module, and no further work is required on it. A partition class that is contained in one of its children, C, will be dealt with when the unprocessed children of W are processed. If a partition class is a union of children, then since every union of children of W is a module of $G|W$ and W is a module of G, each such union is a module of G, and no further work is required on it.

If $s(W)$ exists, suppose first that W is prime in $M2$. We pivot on $s(W)$ by traversing only the portion of $s(W)$'s adjacency list that contains members of W. We find this portion using the pointer carried by W to $N(s(W) \cap W)$ in $s(W)$'s adjacency list. No members of the partition class containing $s(W)$ are encountered in this region of the list, since $s(W)$ is a vertex outside of W. Every partition class intersecting W is contained in W. These elements may now be discarded from W's adjacency list, as they are irrelevant to future pivots. These pivots ensure that if W is not a module, it now contains more than one partition class. W has now been processed, and we call *M2Resolve* again, which ensures that every partition class that intersects a child C of W is now contained in C. These will be dealt with when C is removed from the list of unprocessed nodes.

The procedure when W is degenerate is slightly more complicated, since W can have a child A in M_p that it does not have in the $M2$ tree. As in the prime case, we begin with a pivot on W with $s(W)$, discarding these members of W from $s(W)$'s adjacency list. Again let $\mathcal{A}$ be those children of W that are modules of G. For each child C not in $\mathcal{A}$, we pivot on the regions occupied by W in $s(C)$. Since C is a module of $G|W$, $s(C)$ lies outside of W. We may then discard members of W from $s(C)$'s adjacency list. A call to *M2Resolve* now returns with each partition class intersecting such a child C contained in C. By default, every partition class intersecting $A = \bigcup \mathcal{A}$ is also contained in A. Pivoting on $s(A)$ and removing members of A from $s(A)$'s adjacency list makes this true for its children A_a and A_n. We may do this recursively, halting when we reach the highest descendants of A that are modules. If such a descendant D contains more than one child of W in the $M2$ tree, then it is a module, and so is every union of its children in the M_p tree. Calling *M2Resolve* now halts only when each partition class is either a union of children of D, in which case it is a module, or contained in a child C of D, in which case it will be dealt with when C is processed.

The procedure can only fail to generate effective pivots when every partition class is a module. That is precisely the halting criterion for a restarting procedure on $MD(G)$.

The total time preparing to restart *M2Resolve* is proportional to the number of elements discarded from adjacency lists, and is therefore linear. There are $O(n + m)$ calls to `M2Resolve()`, so the time these calls require is linear, since `M2Resolve()` is a linear restarting procedure. As explained above, the total amortized cost of maintaining the list of unprocessed nodes is linear. This gives the following:

Lemma 6.29 *A linear restarting procedure may be constructed for the modular decomposition of an undirected graph.*

Corollary 6.30 `RPartition` *takes $O(n + m)$ time on an undirected graph.*

6.5.2 A linear time bound for `RPartition` on an interval matrix.

If T is an interval matrix, then $X \subseteq V$ is a module of T iff it is a module of both G_n and G_{1c}. One problem is that we need to avoid working directly on these graphs, since their size is not linear in the size of $G(T)$. Fortunately, vertex partitioning on G_n can be simulated with vertex partitioning on G_{1c}, and vertex partitioning in G_{1n} can be simulated with vertex partitioning on G_c. The partition of a class induced by a pivot is identical in the simulation; only the interpretation of the classes, hence the order given to them in the linear order on $\mathcal{P}$ is changed. Adjusting the ordering of $\mathcal{P}$ to accommodate this reinterpretation takes $O(1)$ time per split, hence $O(n)$ time total.

Let $M_p(G_c)$ and $M_p(G_{1c})$ denote M_p trees for G_c and G_{1c}. We may use a restarting procedure for each of $MD(G_c)$ and $MD(G_{1c})$. Beginning with our initial partition, we call the restarting procedure for $MD(G_c)$, which refines $\mathcal{P}$ until every partition class is a module of G_c, using pivots in G_c, which are lazy pivots in T.

It may be the case that not every partition class is a module of G_{1c}, so we communicate the list of removal operations that have been performed as capricious splits to $M_p(G_{1c})$, and initiate the restarting procedure for $MD(G_{1c})$. This performs pivots until all members of $\mathcal{P}$ are modules of G_{1c}. At this point, they may not be modules in G_c, so we may communicate the new removal operations back to $M_p(G_c)$ and return control to the restarting procedure for $MD(G_c)$.

Alternating between the restarting procedure of $MD(G_c)$ and $MD(G_{1c})$ continues refining $\mathcal{P}$, halting only when neither of the restarting procedures is able to refine $\mathcal{P}$ further. This occurs only when every member of $\mathcal{P}$ is a module of both G_c and G_{1c}, hence a module of $U(T)$. This is the halting criterion for a restarting procedure $MD(U(T))$.

Since each of the restarting procedures is restarted $O(n)$ times, they take $O(n+m(G_c))$ and $O(n+m(G_{1c})$ time, respectively, for all pivots that they perform. This in linear in the size of $G(T)$. This gives the following:

Lemma 6.31 `RPartition` *on $U(T)$, beginning with arbitrary starting partition, takes $O(n+m)$ time.*

The following is now an immediate corollary;

Theorem 6.32 *It takes $O(n+m)$ time to produce an interval realizer of an interval matrix.*

7 Finding a set of algebraic flips to turn an intersection matrix into an interval matrix.

In this section, we give the implementation of Step 2 of Algorithm 4.2.

We proceed by incrementally performing flips to transform an intersection matrix for the original input graph into an interval matrix. At each point, we will let T denote the current state of the matrix, and let G_c, G_n, G_1, and G_2 refer to those graphs in T. The matrix passes through the following stages:

T_0: T_0 is an intersection matrix of the graph given as the input to the recognition problem.

T_1: T_1 is a matrix that has a vertex v_0 of degree $O(m/n)$, and all members of $N(G(T_1), v_0)$ have a single overlap relation with v_0.

For notational convenience, let $U = N(G(T_1), v_0)$ and $W = \overline{N}(G(T_1), v_0)$. (It will also be the case that $U = N(G(T_2), v_0) = N(G(T_3), v_0)$, and $W = \overline{N}(G(T_2), v_0) = \overline{N}(G(T_3), v_0)$). Let $\mathcal{P}_W = \{D_1, D_2, ..., D_k\}$ be the components of $G_{1n}|W$. All edges between these are edges of G_c. We show that we may assume that all directed edges of D_c between these components are directed from left to right in this list. Let $A(D_1)$ denote the smallest interval on the circle that is disjoint from v_0 and contains all members of D_1. ($A(D_1)$ can be thought of as the region of the circle occupied by D_1). For $2 \le i \le k$, all arcs in D_i contain $A(D_1)$.

T_2: (See Figure 11.) T_2 shares the above properties with T_1, but $G_n|U$ and $G_2|U$ are empty, and all vertices in each component of $G_c|U$ cover a single endpoint of v_0 in any circular-arc realizer of T_2. Because of this, $xy \in G_2$ implies $x, y \in U$, so G_2 is empty.

Let $\mathcal{Q} = \{C_1, C_2, ..., C_h\}$ denote the components of $G_c|U$ that contain an arc with an endpoint in $A(D_1)$. If two members of $\mathcal{Q}$ cover opposite endpoints of v_0, flipping all members of one of them will make them cover the same endpoint.

T_3: (See Figure 12.) The components of $\mathcal{Q}$ are flipped, as necessary, so that they all cover the same endpoint of v_0, and they all, therefore, fail to cover one endpoint, p, of $A(D_1)$. T_3 is the resulting intersection matrix. All arcs in $V - \bigcup \mathcal{Q}$ either contain $A(D_1)$ or are disjoint from it. These can be distinguished from each other by their relationship to any member of D_1 in T_3.

T_4: Flipping the arcs in T_3 that contain $A(D_1)$ leaves the region of the circle adjacent to p and outside of $A(D_1)$ uncovered by any arc. T_4 is therefore an interval matrix.

7.1 Turning T_0 into T_1

For finding the flips we need to perform in T_0 to obtain an interval matrix, we select a vertex v of minimum degree in G. When vertices are tied for minimum degree, we select v to be

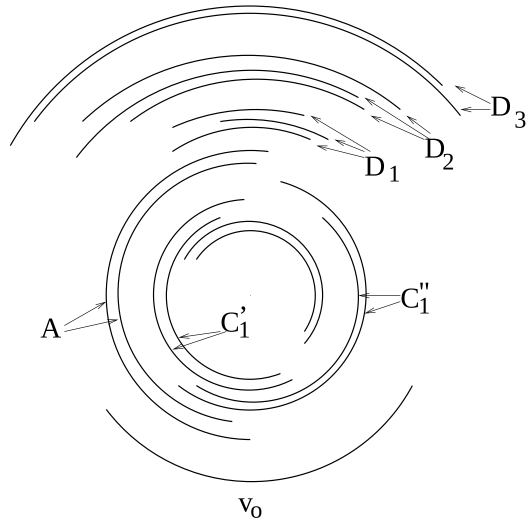


Figure 11: The set W of non-neighbors of v_0 can be divided into components $\{D_1, D_2, \ldots, D_k\}$ of G_{1n}, which are ordered by containment. In T_2, the set U of neighbors of v_0 is a clique and all pairs in U are adjacent in G_c or G_1. Each component $G_c|U$ covers one endpoint of v_0's arc. $\mathcal{Q}$ is the set of components of $G_c|U$ that have a member with an endpoint in the region of the circle occupied by D_1. In this example, $\mathcal{Q} = \{C_1', C_1'', A\}$. C_1' exemplifies $\mathcal{Q}'$, the set of components that have a member with different relationships to members of D_1, and C_1'' exemplifies $\mathcal{Q}''$, the set of components that are not members of $\mathcal{Q}'$, but whose members do not all have the same relationship to D_1. A exemplifies $\mathcal{A}$, the members of $\mathcal{Q}$ that have an overlap relationship with all members of D_1.

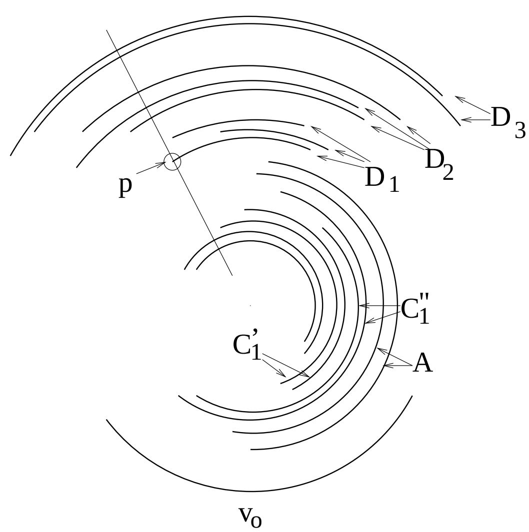


Figure 12: Flipping the members of $\mathcal{Q}$, as needed, so that they cover the right endpoint of v_0, yields T_3. At this point, one endpoint p of the area occupied by D_1 is contained in those arcs that contain all members of D_1 in T_3, and only in those arcs. Flipping these yields T_4, and leaving a region of the circle adjacent to p uncovered. T_4 is therefore an interval matrix.

one whose arc contains no others. An arc cannot contain another that has higher degree, so v always exists. If v has no double-overlap relationships, we select $v_0 = v$. Otherwise, we select a double-overlap neighbor of v that is contained in no other arc, flip it, and set v_0 to be the corresponding vertex. In either case, v_0 now has no double-overlap relations, and has at most the degree of v, which is $O(m/n)$. We then flip every arc that contains v_0. Now all arcs that intersect v_0 have a single-overlap relationship with it. T_1 is the state of T at this point.

Lemma 7.1 *Every edge incident to v_0 in $G(T_1)$ is a single-overlap relationship in a realizer of T_1, and $|U|$ is $O(m/n)$.*

7.2 Turning T_1 into T_2

If for a vertex $x \in U$, x contains v_0's right endpoint, then flipping x will make it contain v_0's left endpoint instead.

If $x, y \in U$ and x and y contain the same endpoint, then they intersect, and since they fail to contain the other endpoint, they do not cover the circle, and they do not have a double intersection. Therefore, if xy is an edge of G_2 or G_n in T_1, then x and y contain opposite endpoints of v_0. $G_{2n}|U$ is bipartite, and the arcs of two vertices in opposite bipartition classes of a component of G_{2n} must contain opposite endpoints. If arc x contains arc y, then they contain the same endpoint of v_0, since they each miss an endpoint, so vertices in the same component of D_c contain the same endpoint of v_0. These observations give a unique bipartition of each component of $(D_c \cup G_{2n})|U$ into sets of vertices containing opposite endpoints. Flipping one of these sets makes them contain the same endpoint, eliminating all edges of G_n and G_2 from the component. Edges of D_c internal to a flipped set are flipped once at each endpoint, and remain edges of D_c, but with their orientations reversed. Edges leading out of the component are edges of G_1 and are unaffected by flipping. This flipping therefore turns the component of $D_c \cup G_{2n}$ into a component of D_c. Doing this for each component of $(D_c \cup G_{2n})|U$ makes $G_n|U$ and $G_2|U$ empty.

Lemma 7.2 *Lemma 7.1 applies to T_2. In addition, all arcs in each of the components of $G_c|U$ contain the same endpoint of v_0 in any realizer of T_2.*

7.3 Turning T_2 into T_3

(See Figure 11.) Let $\mathcal{Q}' = \{C'_1, C'_2, ..., C'_{h'}\}$ be the components of $D_c|U$ that have a member that distinguishes members of D_1, and let $\mathcal{Q}'' = \{C''_1, C''_2, ..., C''_{h''}\}$ be the set $\{C : C$ is a component of $D_c|U$, $C \notin \mathcal{Q}'$ and there is a member of D_1 that distinguishes members of $C\}$. Let $\mathcal{A}$ denote the set set $\{A : A$ is a component of $G_c|U$ such that $A \times D_1 \subseteq G_1\}$. $\mathcal{Q} = \mathcal{Q}' \cup \mathcal{Q}'' \cup \mathcal{A}$.

We obtain T_3 from T_2 as follows:

Case A: If $\mathcal{Q}''$ is nonempty, we use a single rule to decide which members of $\mathcal{Q} = \mathcal{Q}'' \cup \mathcal{Q}'$ to flip to get them all to cover a single endpoint of v_0;

Case B: If $\mathcal{Q}''$ is empty, we use a different rule to decide which members of $\mathcal{Q}'$ to flip to get them all to cover a single endpoint of v_0;

Let T_c denote the resulting matrix. We show that there is a realizer of T_c where all members of $\mathcal{A}$ also cover this same endpoint, so $T_3 = T_c$.

7.3.1 Case A: $\mathcal{Q}''$ is nonempty

Let us say that if $xz \in G_n$ and $yz \in G_1$, then y has a *stronger* relationship to z than x does, since y and z intersect, but x and z do not. If $xz \in G_1$ and $yz \in G_c$, then y again has a stronger relationship with z than x does, since it is a full containment, rather than a partial overlap. Similarly, if $xz \in G_n$ and $yz \in G_c$, then the relationship of y to z is stronger than the relationship of x to z.

For $x, y \in U$, let $x \preceq y$ denote that for every $z \in W$, the relationship of y to z is at least as strong as the relationship of x to z. That is, $x \preceq y$ if $N(G_{1c}, x) \cap W \subseteq N(G_{1c}, y) \cap W$ and $N(G_c, x) \cap W \subseteq N(G_c, y) \cap W$. If A and B are two components of $G_c|U$, then let $A \preceq B$ denote that for every $x \in A$ and $y \in B$, $x \preceq y$. Let $A \sim B$ denote that $A \preceq B$ or $B \preceq A$.

Lemma 7.3 *If A and B are components of $G_c|U$ and cover the same endpoint of v_0 in a circular-arc realizer of T_2, then $A \sim B$.*

Proof: Suppose A and B both cover the right endpoint of v_0 in some circular-arc realizer R. Let $A(W)$ denote the smallest interval of the circle that is disjoint from v_0 and contains every arc in W. (Think of $A(W)$ as the "region occupied by W".) Then for $x \in A$ and $y \in B$, x and y both extend counterclockwise into $A(W)$. If $A \preceq B$ and $B \preceq A$, the lemma holds. Suppose without loss of generality that y extends farther than x does, so that $y \not\preceq x$. Then every arc in W that is contained in x is also contained in y, and every arc in W intersected by x is also intersected by y. Therefore, $x \preceq y$.

Since A and B contain no arc that covers the left endpoint of v_0, $R' = R|(\{v_0\} \cup A \cup B)$ is an interval realizer of $T' = T|(\{v_0\} \cup A \cup B)$. The order of right endpoints in this realizer is a linear extension of an interval orientation of T'. The root node of the modular decomposition of T' is a degenerate node whose children, $\{v_0\}$, A, and B, are components in the complement of G_1. G_{1n} is transitively oriented in the interval orientation, so all edges of G_1 between A and B must either be directed from A to B, or from B to A by Lemma 2.1. Since $x \preceq y$ and $y \not\preceq x$, xy is oriented from x to y. Each arc of B extends at least as far to the right as every arc of A, hence at least as far into the region occupied by W in R. It follows that $A \preceq B$. $\square$

G_2 is empty, D_c is transitive, and $\{D_1, D_2, ..., D_k\}$ are the children of the root of the modular decomposition of $G_c|W$. By Lemma 2.1, we may assume without loss of generality that $D_i \times D_j$ consists exclusively of arcs of D_c whenever $1 \leq i < j \leq k$.

Lemma 7.4 *If $A \in \mathcal{Q}''$, $B \in \mathcal{Q}'' \cup \mathcal{Q}'$ and $A \neq B$, then $A \sim B$ iff A and B cover the same endpoint of v_0 in B.*

Proof: Since $A \in \mathcal{Q}''$, A contains two vertices u and w that each have uniform relationships to all members of D_1, but such that u's relationship to D_1 is different from w's.

Case 1: $\{u\} \times D_1 \subseteq G_n$ and $\{w\} \times D_1 \subseteq G_c$. Flipping A results in $\{u\} \times D_1 \subseteq G_c$ and $w \times D_1 \subseteq G_n$. In one of these cases, A and B cover the same endpoint of v_0, so $A \sim B$ in one of these cases, by Lemma 7.3. In this case, either every member of $B \times D_1 \subseteq G_c$ or every member of $B \times D_1 \subseteq G_n$, but this contradicts membership of B in $\mathcal{Q}$. We conclude that Case 1 cannot occur.

Case 2: $\{u\} \times D_1 \subseteq G_1$ and $\{w\} \times D_1 \subseteq G_c$. Suppose that we flip B as necessary so that it covers the same endpoint as A does. If $A \preceq B$ now applies, then $B \times D_1 \subseteq G_c$, contradicting B's membership in $\mathcal{Q}$. By Lemma 7.3, $B \preceq A$, so no member of $B \times D_1 \in G_c$. By B's membership in $\mathcal{Q}$, there are pairs $(a, b), (c, d) \in B \times D_1$ such that $ab \in G_n$ and $cd \in G_1$ (where it is possible that $a = c$ or $b = d$). Flipping B so that it does not contain the same endpoint of v_0 as A results in $ab \in G_c$, and $cd \in G_1$, and $B \not\sim A$. We conclude that the claim applies in Case 2.

Case 3: $\{u\} \times D_1 \subseteq G_1$ *and* $\{w\} \times D_1 \subseteq G_n$. The proof is identical to that of Case 2, except that the roles of G_c and G_n are swapped. □

Since every circular-arc realizer has a left-right mirror image, we may assume without loss of generality that C_1'' covers the right endpoint of v_0 in R. Lemma 7.4 gives a simple criterion for flipping the remaining members of $\mathcal{Q}$ so that they cover the same endpoint of v_0.

7.3.2 Case B: $\mathcal{Q}''$ is empty

If $\mathcal{Q}''$ is empty, then by mirror symmetry, we can assume without loss of generality that C_1' covers the right endpoint of v_0.

Our strategy is summarized in Figure 13.

At certain points, we will have an interval orientation of $T|Y$ for some $Y \subseteq W$, and a vertex x that we know can be added either as a source or sink to produce an interval orientation of $T|(Y \cup \{x\})$. Which case applies determines whether x must be flipped. The interval realizer of the interval orientation of $T|Y$ is available. Rather than checking the Δ relationships between edges of G_{1n} incident to x and edges of $G_{1n}|Y$ directly, we work with an interval representation of $T|Y$, and use Algorithm 7.5:

Algorithm 7.5 *Checking whether x can be added as a source to an interval orientation of $T|Y$, given the interval realizer R_Y of the orientation.*

In the ordered-list representation of R_Y, find the first point p in the list that is to the right of all left endpoints of neighbors of x in $G(T|Y)$ and to the right of all right endpoints of neighbors of x in $(D_c)^T|Y$. If x can be added as a source, $l(x)$ must be inserted at the front of the list, and $r(x)$ must be inserted at p. The test consists of checking that no non-neighbor of x has a left endpoint to the left of p, and no overlap neighbor of x has a right endpoint to the left of p. The first check may be accomplished by counting the intervals with left endpoint to the left of p and halting if the count exceeds the degree of x in $T|(Y \cup \{x\})$.

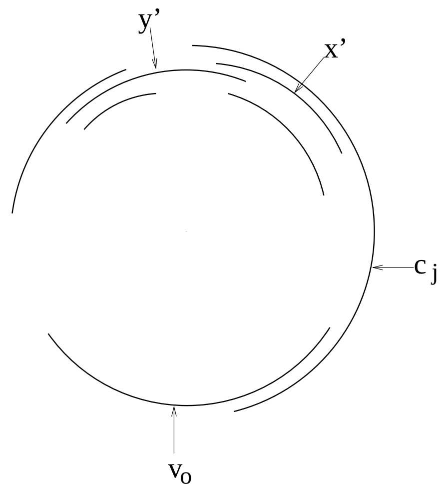


Figure 13: The strategy for finding whether C'_j covers the same endpoint as C'_1 when $\mathcal{Q}''$ is empty. We find a vertex $c_1 \in C'_1$ that distinguishes members of D_1. By mirror symmetry, we may assume that c_1 covers the right endpoint of v_0 in some circular-arc realizer R. Since c_1 is the only vertex in $\{c_1\} \cup D_1$ that intersects v_0, c_1 is a source (extreme leftmost interval) in the corresponding interval realizer $R|(\{c_1\} \cup D_1)$. We use this and Δ relationships involving edges incident to c_1 to find the interval realizer $R|P_i$ for some subset P_i of arcs of D_1. In the illustration, the arcs other than v_0 and c_j represent P_i. In C'_j, there is a vertex c_j that distinguishes members of D_1. We are able to construct P_i so that c_j distinguishes members x' and y' of P_i, where $x'y' \in G_{1n}$. Now c_j is either a source or sink in the interval realizer $R|(\{c_j\} \cup D_1)$, and which of these cases applies is forced by a Δ relationship involving c_j, x', and y'. Whether it is a source or a sink tells whether c_j covers the left endpoint or the right endpoint of v_0. In the illustration, c_j must cover the right endpoint of v_0, because it contains x' and overlaps y'. It could not realize those relationships if it covered the left endpoint of v_0 and entered the top part of the circle clockwise.

The second check is conducted similarly.

To find whether x can be added as a sink, it suffices to use the mirror transpose of R_Y to model the transpose of the orientation of $T|Y$, and test whether x can be added as a source to it. The running time of Algorithm 7.5 is $O(|N(G(T), x))|)$.

Let R be a circular-arc realizer of T_2. Let c_1 be a member of C'_1 that distinguishes members of D_1, let c_j be a member of C'_j that distinguishes members of D_1, let $\mathcal{P}$ be $\{v_0\}$ and the maximal modules of $T_2|(\{v_0, c_1\} \cup D_1)$ that do not contain v_0, and let P be any set consisting of one representative from each member of $\mathcal{P}$. Let $\mathcal{P}' = \mathcal{P} - \{\{v_0\}, \{c_1\}\}$, and let $P' = P - \{v_0, c_1\}$.

We may find $R|P'$ as follows. $T_2|(\{v_0, c_1\} \cup D_1)$ is an interval matrix with interval realizer $R|(\{v_0, c_1\} \cup D_1)$. In $R|(\{v_0, c_1\} \cup D_1)$, v_0 is the interval with the leftmost left endpoint, hence a source in the corresponding interval orientation of $T_2|(\{v_0, c_1\} \cup D_1)$. Let us call `Partition` (Algorithm 6.21) on the initial partition $\{\{v_0\}, \{\{c_1\} \cup D_1\}\}$ in $T_2\ |(\{v_0, c_1\}\ \cup D_1)$, halting when each part is a module. The final partition is $\mathcal{P}$, and by Lemma 6.20, Part 1, the ordering of its parts gives the unique interval orientation of $(T_2|(\{v_0, c_1\} \cup D_1))/\mathcal{P}$ that is consistent with $\{v_0\}$ being a source. By Lemma 6.3, the corresponding interval realizer R_1 is unique. This quotient is isomorphic to $T_2|P$, which is realized by $R|P$. In $R|P$, v_0 is also leftmost, so $R|P$ is isomorphic to R_1. By restriction, this gives $R|P'$.

If there exists $Z \in \mathcal{P}'$ whose members have differing relationships to c_j, let $x_1, x_2 \in Z$ be two vertices whose relationships to c_j differ. Let P_1 be x_1 and one arbitrary representative of each member of $\mathcal{P}' - \{Z\}$, and let P_2 be the result of replacing x_1 in P_1 with x_2. One of these will serve as the P_i mentioned in the caption of Figure 13. If Z does not exist, let $P_1 = P_2$ consist of one representative of each member of $\mathcal{P}'$. The algorithm for finding $R|P$ works identically on $R|P_1$ and $R|P_2$, so $R|P_1$ and $R|P_2$ are isomorphic to $R|P'$, and we do not have to recompute these interval realizers for each C'_j. The relationships of c_j $R|P_1$ and $R|P_2$ may differ, however.

Lemma 7.6 *c_j can be added as a source to both $R|P_1$ and $R|P_2$ iff C'_1 and C'_j cover the same endpoint of v_0.*

Proof: **Case 1:** *Z does not exist.* Since c_j distinguishes members of D_1, it distinguishes $X \in \mathcal{P}'$ from $Y \in \mathcal{P}'$. Since $G_{1n}|D_1$ is connected, there exist $X', Y' \in \mathcal{P}'$ such that $X' \times Y' \subseteq G_{1n}$. Let x' and y' be the representatives of X' and Y' in P_1. Without loss of generality, suppose x' precedes y' in $R|P_1$.

If c_j covers the same endpoint as c_1 in R, then c_j has the leftmost left endpoint in the interval realizer $R|(\{c_j\} \cup P_1)$, so c_j can be added as a source to $R|P_1$, and the relationship of c_j to x' is stronger than its relationship to y'. If c_j is flipped so that it covers the opposite endpoint of v_0 from c_1, $R|(\{c_j\} \cup P_1)$ is still an interval realizer, but with c_j rightmost. Its relationship to x' becomes weaker than its relationship to y'. Since x' is earlier than y' in $R|(\{c_j\} \cup P_1)$, c_j can no longer be added as a source to $R|P_1$.

Case 2: *Z exists.* Then c_j distinguishes $x_1, x_2 \in Z$. Since D_1 is connected and $\mathcal{P}'$ consists of modules of $T_2|(\{v_0, c_1\} \cup D_1)$, there exists Z' in $\mathcal{P}'$ such that $Z \times Z' \subseteq G_{1n}$. For any y in Z', either c_j has different relationships to x_1 and y or to x_2 and y. Without loss of generality, suppose it has different relationships to x_1 and y. Now x_1 and y have the same properties as x' and y' from Case 1, so c_j can be added as a source to $R|P_1$ iff c_j and c_1 cover the same endpoint of v_0.

The results now follows from Lemma 7.2. □

Lemma 7.6 and Algorithm 7.5 provide an efficient test for deciding how to flip members of $\mathcal{Q}' - \{C_1'\}$ so that they cover the same endpoint of v_0 as C_1' does, namely, the right endpoint of v_0.

7.3.3 Showing that T_c serves as T_3

In T_c, all arcs in members of $\mathcal{Q}'$ and $\mathcal{Q}''$ cover the right endpoint of v_0 in some realizer R_c of T_c. With the exception of arcs in $\mathcal{A}$, this is all arcs that have an endpoint in $A(D_1)$.

Lemma 7.7 *There is a circular-arc realizer of T_c where all members of $\mathcal{Q}'$, $\mathcal{Q}''$, and $\mathcal{A}$ cover the right endpoint of v_0.*

Proof: Let A be a member of $\mathcal{A}$ that covers the left endpoint of v_0 in T_c. By the definition of $\mathcal{A}$, $A \times D_1 \subseteq G_1$. Therefore, $A \times D_i \subseteq G_1$ for any $i > 1$, since $D_i \times D_1 \subseteq (D_c)^T$, and D_1 lies in the intersection of all arcs in D_i, which are, in turn, disjoint from both endpoints of v_0. Members of $\mathcal{A}$ are connected components of $\overline{G_{1n}}$. The edges of G_1 are unaffected by flipping, so flipping the members of A modifies $T_c|A$ without affecting other relationships in T_c.

Since they are components of $\overline{G_{1n}}$, they are children of the root of $MD(T)$. Edges of G_1 are unaffected by flipping, so flipping all members of $\mathcal{A}$ to cover the same endpoint of v_0 will not change this fact. By Lemma 6.18, the left endpoints and right endpoints of each $A \in \mathcal{A}$ are consecutive in the realizer after the flipping. However, flipping does not change the order of endpoints, only which endpoints are interpreted as left and which as right endpoints. The left endpoints of A are consecutive and the right endpoints of A are consecutive in R_c.

Since $R_c|A$ is an interval realizer, the flip replaces $R_c|A$ with $(R_c)^S|A$, getting A to cover the right endpoint of v_0, by Lemma 6.2. The sandwich transpose operation on A is possible in the new circular-arc realizer that results, since the left endpoints and right endpoints of A are consecutive. Applying a sandwich transpose operation on A in this circular-arc realizer now undoes all effects on T_c of flipping A, while leaving A on the right endpoint of v_0.

Doing this to each A that covers the left endpoint of v_0 realizes the claim in the lemma. □

Lemma 7.7 justifies letting $T_3 = T_c$. The following is now immediate:

Lemma 7.8 *There exists a circular-arc realizer of T_3 where every member of U with an endpoint inside the region of the circle occupied by D_1 covers the right endpoint of v_0.*

7.4 Turning T_3 into T_4

The following corollary to Lemma 7.8 is now obvious:

Corollary 7.9 *In such a realizer, let p be the far endpoint of the interval $A(D)$ as one travels rightward from the right endpoint of v_0. An arc contains p in its interior iff it is one of the following:*

- *A member of $D_2, ..., D_k$*
- *A member x of U such that $x \times D_1 \subseteq D_c$.*

We may obtain T_4 from T_3 by flipping all arcs that contain p according to this corollary. This vacates the circle just to the right of p, so T_4 is an interval matrix.

7.5 Linear-time implementation

For the analysis of the time bound, we must not assume that the input graph is a circular-arc graph.

Theorems 5.8 and 5.9 ensure that we do not spend more than linear time finding neighborhood containments, even if G is not a circular-arc graph.

We use a labeling of edges of $G(T)$ with their intersection types for a sparse representation of T. We may use an array to represent all intersection types in $U \times V$. Since U has $O(m/n)$ members, the size of this array is $O(m)$.

Turning T_0 into T_1 in linear time is trivial.

For turning T_1 into T_2, we must find the bipartition of the components of $G_{2n}|N[v_0]$. This takes $O(|N[v_0]|^2)$ time using breadth-first search on the array, which is $O(m)$ since $|N[v_0]|$ is $O(m/n)$.

For turning T_2 into T_c, we must compute the $\preceq$ relation in U in linear time in Case A. For $x, y \in U$ in T_2, $x \preceq y$ iff $N[G_c, x] \cap W \subseteq N[G_c, y] \cap W$ and $N[x] \cap W \subseteq N[y] \cap W$ in T_2. Since $U \cup \{v_0\}$ is a clique in $G(T_2)$, $N[x] \cap W \subseteq N[y] \cap W$ iff $N[x] \subseteq N[y]$. Theorem 5.8 gives this in linear time.

To evaluate $N[G_c, x] \cap W \subseteq N[G_c, y] \cap W$ for each pair x, y of neighbors of v_0, note that if all members of $N(v_0)$ are flipped, $N(v_0)$ remains a clique, and the edges of G_n and G_c that are incident to each member of $N(v_0)$ are swapped. $N[G_c, x] \cap W \subseteq N[G_c, y] \cap W$ iff $N[x] \cap W \subseteq N[y] \cap W$ after $N(v_0)$ is flipped. We get the latter with Theorem 5.8 in linear time. Flipping $N(v_0)$ takes $O(n(m/n)) = O(m)$ time.

In Case B, we must perform a vertex partition on a submatrix that is an interval matrix, and find a realizer for the resulting modular quotient. We have already given a linear-time algorithm for this problem, in Section 6. The submatrix is $T|(\{v_0\} \cup \{c_1\} \cup D_1)$. The size of $G(T)|(\{v_0\} \cup \{c_1\} \cup D_1)$ is linear in the size of the input graph to the recognition problem. If the algorithm does not give rise to an interval representation of the quotient, G is not a circular-arc graph, and we halt. It takes linear time to find, for each $X \in \mathcal{P}'$, another member $t(X) \in \mathcal{P}'$ that is adjacent to X in G_{1n}. Finding Z whose members are distinguished by c_j, if Z exists, takes $O(|N(c_j)|)$ time using an operation similar to a pivot on c_j. Then $t(Z)$ then gives the neighbor y of x_1 and x_2 that is needed in Case 2 of the proof of Lemma 7.6.

We also have to evaluate for $c_j \in U$ whether c_j can be added as a leftmost interval to two realizers R_1 and R_2, and repeat this test after we flip c_j. This takes $O(n)$ time using Algorithm 7.5, whether or not c_j is flipped. The total time spent on this test is $O(n)$ times $O(m/n)$ vertices in U. We halt if c_j can be added as neither a leftmost nor a rightmost interval.

Flipping a member of U takes $O(n)$ time, and each member of U is flipped $O(1)$ times. The total time spent flipping neighbors of v_0 is therefore $O(m)$. Since only neighbors of U are flipped to obtain T_3 from the initial intersection matrix, $G(T_3)$ has $O(n(m/n)) = O(m)$ edges.

For turning T_3 into T_4, we must flip members of U that contain $A(D_1)$, which again takes linear time. If $\{D_2, ..., D_k\}$ is nonempty, we also flip every vertex in this set in T_3. If T_3 is a circular-arc matrix, then because $D_i \times D_{i+1} \subseteq D_c$ for each i from 1 to $k-1$, every arc in D_i is contained in the intersection of arcs in D_{i+1}, so D_{i+1} is a clique. Since D_c is transitive, $\bigcup\{D_2, ..., D_k\}$ is a clique in $G(T_2)$, $\bigcup\{D_2, .., D_k\} \times D_1 \subseteq G(T_3)$. The only non-neighbors of these vertices are in $N[v_0]$, so they have degree $\Omega(n - m/n) = \Omega(n - (n-1)/2) = \Omega(n)$. We may check that this is the case in linear time, and halt if it is not. Otherwise, flipping each of them takes time proportional to its degree, hence $O(m)$ for all of them collectively.

The flip of vertices in $D_2, ..., D_k$ can only increase the size of $G(T_4)$ over $G(T_3)$ by adding edges between these vertices and U, and therefore adds $O(m)$ edges to $G(T_4)$. The size of $G(T_4)$ is $O(n + m)$.

8 Relationship to Previous Work

Tucker was the first to recognize the importance of finding neighborhood containments in recognizing circular-arc graphs [26]. This step was a bottleneck in his algorithm. Eschen and Spinrad's innovations for this step in [9] and their improvements given in [8] are obviously critical for our time bound.

There is a strong relationship between the flipping operation and Hsu's use of circle graphs in [14] that is not apparent on first inspection. A circle graph is the intersection graph of chords on a circle. Hsu uses the fact that replacing each arc in a circular-arc representation with a chord connecting the same endpoints yields a circle graph representing G_1. One interpretation of this within the framework developed in our paper is that the circle graph leaves each arc in an "indeterminate" state between its two possible flipped states until its state in the circular-arc graph can be determined. In some sense, it can be viewed as a representation of the entire flip-equivalence class to which G belongs. He describes *consistent modules* of G_1, which share some features of Δ modules of an interval matrix obtainable from G by flipping, though the correspondence between these modules is not perfect.

What makes it possible to exploit flipping as an algorithmic tool is representing the intersection matrix explicitly. Hsu's algorithm does not represent the matrix directly, assuming instead that it is given implicitly by the labeling $T(G)$ described above in Section 5. However, the class of matrices given by $T(G)$ for some circular-arc graphs is a proper subclass of intersection matrices. This subclass is not closed under flipping, or under restriction to induced submatrices. Our approach appears to lead to a simpler algorithm, as it allows us to reduce much of the problem to the well-understood domain of comparability graphs.

References

[1] S. Benzer. On the topology of the genetic fine structure. *Proc. Nat. Acad. Sci. U.S.A.*, 45:1607–1620, 1959.

[2] K.S. Booth. *PQ-Tree Algorithms.* PhD thesis, Department of Computer Science, U.C. Berkeley, 1975.

[3] 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.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Algorithms.* MIT Press, Cambridge, Massachusetts, 1990.

[5] A. Ehrenfeucht and G. Rozenberg. Theory of 2-structures, part 1: Clans, basic subclasses, and morphisms. *Theoretical Computer Science*, 70:277–303, 1990.

[6] A. Ehrenfeucht and G. Rozenberg. Theory of 2-structures, part 2: Representations through labeled tree families. *Theoretical Computer Science*, 70:305–342, 1990.

[7] E.M. Eschen. *Circular-arc graph recognition and related problems.* PhD thesis, Department of Computer Science, Vanderbilt University, 1997.

[8] E.M. Eschen and J.P. Spinrad. An $O(n^2)$ algorithm for circular-arc graph recognition. *Forthcoming.*

[9] E.M. Eschen and J.P. Spinrad. An $O(n^2)$ algorithm for circular-arc graph recognition. *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, 4:128–137, 1993.

[10] D.R. Fulkerson and O. Gross. Incidence matrices and interval graphs. *Pacific J. Math.*, 15:835–855, 1965.

[11] T. Gallai. Transitiv orientierbare Graphen. *Acta Math. Acad. Sci. Hungar.*, 18:25–66, 1967.

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

[13] M. Habib, R. M. McConnell, C. Paul, and L. Viennot. Lexicographic breadth-first search: a partition refining technique. *Theoretical Computer Science*, 234:59–84, 2000.

[14] W. Hsu. O(mn) algorithms for the recognition and isomorphism problems on circular-arc graphs. *SIAM J. Comput.*, 24:411–439, 1995.

[15] J.L. Johnson, R.M. McConnell, and J.P. Spinrad. Construction of probe interval models. *Manuscript.*

[16] J.L. Johnson and J.P. Spinrad. A polynomial time recognition algorithm for probe interval graphs. *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, 12:477–486, 2001.

[17] E. Marczewski. Sur deux proprietés des classes d'ensembles. *Fund. Math.*, 33:303–307, 1945.

[18] R. M. McConnell and J. P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999.

[19] R. M. McConnell and J. P. Spinrad. Ordered vertex partitioning. *Discrete Math and Theoretical and Computer Science*, 4:45–60, 2000.

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

[21] R. H. Möhring. Algorithmic aspects of the substitution decomposition in optimization over relations, set systems and boolean functions. *Annals of Operations Research*, 4:195–225, 1985.

[22] R. H. Möhring and F. J. Radermacher. Substitution decomposition for discrete structures and connections with combinatorial optimization. *Annals of Discrete Mathematics*, 19:257–356, 1984.

[23] Fred S. Roberts. *Graph Theory and Its Applications to Problems of Society.* Society for Industrial and Applied Mathematics, Philadelphia, 1978.

[24] J. P. Spinrad. Graph partitioning. unpublished manuscript, 1985.

[25] J.P. Spinrad. Doubly lexical ordering of dense 0-1 matrices. *Inf. Process. Lett.*, 45:229–235, 1993.

[26] A. Tucker. An efficient test for circular-arc graphs. *SIAM Journal on Computing*, 9:1–24, 1980.