Construction of Probe Interval Models

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

Abstract

An *interval graph* for a set of intervals on a line consists of one vertex for each interval, and an edge for each pair of intersecting intervals. A *probe interval graph* is obtained from an interval graph by designating a subset P of vertices as *probes*, and removing the edges between pairs of vertices in the remaining set N of *non-probes*. We examine the problem of finding and representing possible layouts of the intervals, given a probe interval graph. We obtain an $O(n + m \log n)$ bound, where n is the number of vertices and m is the number of edges. The problem is motivated by an application to molecular biology.

1 Introduction

The problem of creating an **interval model** of a graph is defined as follows. The input is an undirected graph G. The output is a set of intervals on a line, with one interval representing each vertex of G, so that the edges of G correspond to those pairs of intervals that intersect. The class of graphs for which the problem has a solution is called **interval graphs**.

An application to molecular biology is the problem of reconstructing the arrangement of fragments of DNA taken from multiple copies of the same genome. The inputs to the problem are results of laboratory tests that tell which pairs of fragments occupy intersecting intervals on the genome. 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 certain virus were an interval graph. This provided strong evidence that its genetic information was physically organized in a linear arrangement.

A linear-time algorithm for creating an arrangement of intervals, given G, appeared in [2]. More recently, a variant that makes more efficient use of laboratory resources has been studied [16, 11], but no linear time bound is known for it. This variant is the basis of a patent [17]. A subset of the fragments is designated as **probes**, and for each probe, one may test all non-probe fragments for intersection with the probe. This results in an incomplete interval graph. The object is to reconstruct an arrangement of fragments on the genome that could have given rise to the test results, under the assumption that the tests are reliable.

In graph-theoretic terms, the input to the problem is a graph $G = (V, E)$ and a subset P of **probe vertices**. The set $N = V - P$ is an independent set. G is a probe-interval graph iff it can be extended to an interval graph by adding edges between non-probe vertices. The object is either to determine that G is not a probe-interval graph, or else produce a set of intervals whose intersections model the adjacencies between members of P and members of V. Such a set is called a **model** or **realizer** of G.

Another application of the problem of constructing a probe-interval model occurs in recognizing circular-arc graphs [9], where an algorithm for it played a key role in obtaining a linear time bound for that problem. Circular-arc graphs have applications to problems involving cyclical schedules, such as traffic-light scheduling.

Let n be the number of vertices and m the number of edges of a graph. An $O(n^2)$ algorithm for constructing probe-interval models is given in [8]. We give an $O(n + m \log n)$ algorithm.

2 Preliminaries

If $G = (V, E)$ is a graph and X is a nonempty subset of V, we let $G|X$ denote the **restriction of G to X**, that is, the subgraph of G induced by X. We let $N(x)$ denote the neighbors of x, and let $N[x] = N(x) + x$ denote the **closed neighborhood** of x.

We treat an undirected graph is a special case of a directed graph where for each directed edge (u, v) there exists a directed edge (v, u). If $G = (V, E)$ is a graph, then its **transpose**, $G^T = (V, E^T)$, is obtained by reversing the direction of each directed edge. A **module** of a graph $G = (V, E)$ is a set $X \subseteq V$ such that for every $y \in V - X$, either all members of $\{y\} \times X$ are directed edges or none of them are, and either all members of $X \times \{y\}$ are directed edges or none of them are. V and its singleton subsets satisfy the requirements,

[*]Dept. of Computer Science and Engineering, University of Colorado at Denver, Denver, CO 80217-3364 USA (corresponding author)

[†]Dept. of Computer Science, Vanderbilt University, Nashville, TN 37235 USA

and are called **trivial** modules. All other modules are **nontrivial**.

A directed acyclic graph is **transitive** if, whenever (a, b) and (b, c) are two edges incident to b in sequence, then (a, c) is also a directed edge. These graphs model poset relations. A **transitive orientation** of an undirected graph is an assignment of directions to its edges that yields a transitive acyclic graph. The class of graphs that can be transitively oriented is called **comparability graphs**. The modules of comparability graphs play an important role in the theory of transitive orientation [6].

Definition 2.1. *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. *If X and Y are properly overlapping members of $\mathcal{F}$, then $X \cup Y$, $X \cap Y$, $X - Y$, and $Y - X$, are members of $\mathcal{F}$.*

The **strong** members of $\mathcal{F}$ are those that properly overlap no other member of $\mathcal{F}$. It is easy to verify that the modules of a graph are a tree-decomposable family.

The **decomposition tree** of a tree-decomposable family is defined as follows. The strong members of a tree-decomposable family are the nodes of the tree, and the tree is the transitive reduction of the containment relation on these members. V is the root, the singleton subsets of V are the leaves, and each internal node's children are a partition of the set that it represents. The tree can be represented with an $O(1)$-space structure for each node, since the set that a node represents can be recovered by visiting its leaf descendants.

Theorem 2.1. *[12, 3, 4]. If $\mathcal{F}$ is a tree-decomposable family, then each internal node of the decomposition tree can then be labeled as* **prime**, **degenerate**, *or* **linear**, *and the children of linear nodes can be ordered, so that the decomposition tree has the following relationship to $\mathcal{F}$:*

- *$X \subseteq V$ is a member of $\mathcal{F}$ iff X is a node of T, a union of children of a degenerate node, or a union of consecutive children of a linear node.*

The decomposition tree for the modules of a graph is called the **modular decomposition**.

We now summarize an abstraction that is due to Moehring [12] and that is useful in the development of our algorithm. The abstraction avoids explicit mention of graphs and modules, while retaining those properties required to prove most of the interesting theorems about modules. Moehring has shown that a variety of interesting structures other than modules in graphs are instances of the abstraction, so a proof that uses only the abstraction is more general than one that makes specific mention of graphs and modules.

In the following definition, $\mathcal{S}$ corresponds to the set of graphs, $V(G)$ corresponds to a set of vertices of G, $G|X$ corresponds to the subgraph induced by X, and $\mathcal{F}(G)$ corresponds to the set of modules of G.

Definition 2.2. *Let $\mathcal{S}$ be some class of structures that can be defined over a set. $\mathcal{S}$ include a definition of what constitutes isomorphism between two members of $\mathcal{S}$. $\mathcal{S}$ also includes a definition of three functions, denoted $V()$, $|$, and $\mathcal{F}()$. Let $G \in \mathcal{S}$.*

- *$V(G)$ returns a set;*
- *For $X \subseteq V(G)$, the* **restriction of** *G* **to** *X, denoted $G|X$, yields an instance G' of $\mathcal{S}$ such that $V(G') = X$;*
- *$\mathcal{F}(G)$ defines a tree-decomposable family on $V(G)$;*

Then $(\mathcal{S}, V(), \mathcal{F}(), |)$ defines a **quotient structure** *if it satisfies the following:*

- *("The Restriction Rule:") For each $Y \subseteq V(G)$ and $X \in \mathcal{F}(G)$, $X \cap Y \in \mathcal{F}(G|Y) \cup \{\emptyset\}$.*
- *("The Substructure Rule:") For each $Y \subseteq X \in \mathcal{F}(G)$, $Y \in \mathcal{F}(G)$ iff $Y \in \mathcal{F}(G|X)$.*
- *Let $\mathcal{P}$ be a partition of V such that each member of $\mathcal{P}$ is a member of $\mathcal{F}(G)$. There exists a* **quotient** *$G' \in \mathcal{S}$, denoted $G/\mathcal{P}$, such that for all ways to select a set A consisting of one representative from each member of $\mathcal{P}$, $G|A$ is isomorphic to G'.*
- *("The Quotient Rule:") Let $\mathcal{P}$ be as in the last condition. If $\mathcal{W} \subseteq \mathcal{P}$, then $\bigcup \mathcal{W} \in \mathcal{F}(G)$ iff $\mathcal{W} \in \mathcal{F}(G/\mathcal{P})$.*

Let $TD(G)$ denote the tree decomposition of $\mathcal{F}(G)$, which exists by Theorem 2.1. In the case where $\mathcal{F}(G)$ denotes the modules of G, we get a quotient structure, and $TD(G)$ is just the modular decomposition.

3 Containment orientations and Δ modules

We may assume without loss of generality that no endpoints of intervals in a realizer of an interval graph coincide, since in any realizer where they do, the endpoints can be moved by small amounts to make this true. We may thus capture all of the relevant combinatorial properties of a realizer by traversing it from left to right, creating a list of identifiers of vertices. In the resulting sequence, each vertex appears twice. Let us call this the **string representation** of the realizer. The interval graph can clearly be reconstructed from this string, and this abstraction ignores irrelevant features of an interval realizer, such as the exact geometric placement of the endpoints that realize the string.

Henceforth, we will mean this representation when we refer to an interval realizer. We consider two realizers to be different iff their string representations differ. All interval graphs have at least two realizers, where one is obtained by reversing the representation of the other. Some interval graphs have a large number of realizers.

Given an interval graph and an interval realizer R, we may partition the edges of the graph into the set E_1 of **overlap** edges, which arise from two intervals that each contain an endpoint of the other, and the set E_c of **containment** edges, which arise from an interval properly containing the other. Let E_n be the edges of the complement of G. $\{E_c, E_1, E_n\}$ is a partition of the edges of the complete graph on V. Let $G_c = (V, E_c)$, $G_1 = (V, E_1)$, and $G_n = \overline{G} = (V, E_n)$. When R is not understood, we let $E_c(R)$, $G_c(R)$, $E_1(R)$ $G_1(R)$, $E_n(R)$, $G_n(R)$, etc., denote these structures. We say that R is a **realizer** of G_c, G_1, and G_n. We will use multiple subscripts to denote the graphs arising from unions of these edge sets. For instance $G_{1n} = (V, E_1 \cup E_n)$.

For $x \in \{c, 1, n\}$, let $A_x = \{(a, b) : ab \in E_x$ and the right endpoint of a occurs before the right endpoint of b in $R\}$, and let $D_x = (V, A_x)$. Let a **containment orientation** of G be $H = (V, A_c(R) \cup E_1(R))$ for some realizer R of G. Note that H serves to represent D_c, G_1, and G_n. When R is not understood, we let $A_x(R)$, $D_x(R)$, and $H(R)$ denote these structures. We say that R is a **realizer** of $D_c(R)$, $D_1(R)$, and $D_n(R)$, and $H(R)$. We may again use multiple subscripts to denote unions of these. For example, $D_{1n}(R)$ denotes the graph $(V, A_1(R) \cup A_n(R))$.

A Δ module of H is a module X of H that satisfies the following additional **Delta requirement**: either $G|X$ is a clique or there exists no $y \in V - X$ such that $\{y\} \times X \subseteq E_1$.

Theorem 3.1. *[9] Let $\mathcal{H}$ denote the containment orientations of interval graphs. For $H \in \mathcal{H}$, let $V(H)$ denote the vertices of H, $\mathcal{F}(H)$ its Δ modules, and $H|X$ denote the subgraph induced by X. Then $(\mathcal{H}, V(), \mathcal{F}(), |)$ is a quotient structure.*

Definition 3.1. *A graph $G = (V, E)$ is* **prime** *if has has only the trivial modules V and $\{\{x\} : x \in V\}$. H is* **Δ-prime** *if it has only these sets as Δ modules.*

Theorems 2.1 and 3.1 imply that the Δ modules can be represented with a decomposition tree, which we will call the **Delta tree of** H, or $\Delta(H)$.

D_c is a transitive orientation of G_c, and $D_{1n} = (V, A_1 \cup A_n)$ is a transitive orientation of G_{1n}. Those transitive orientations that G_{1n} that are given by $D_{1n}(R) = (V, A_1(R) \cup A_n(R))$ for some realizer R of H are called **interval orientations** of G_{1n}. Not every transitive orientation of G_{1n} is an interval orientation.

Theorem 3.2. *R is recoverable from $H(R)$ and $D_{1n}(R)$.*

This can be easily understood by observing that $A_c \cup A_1 \cup A_n$ is a transitive orientation of a complete graph, hence a linear order on V, and this order gives the order of appearance of right endpoints in the realizer. Similarly, $(A_c)^T \cup A_1 \cup A_n$ gives the order of left endpoints. There is a unique way to interleave these two orders to realize H. This can be accomplished in linear time if the orientation D_{1n} is represented implicitly by means of one of its topological sorts of V [9].

Since an undirected graph G is a special case of a symmetric directed graph, it is legitimate to define a relation on the directed edges of an undirected graph, and to view of an orientation of G as a subset of the directed edges of G. The following relation has a similar role with respect to interval orientations and Δ modules as the well-known Γ relation has with respect to transitive orientations and standard modules that is given in [5, 6]:

Definition 3.2. *[9] 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:*

- *$ab, ac \in E_n$ and $bc \in E_{1c}$;*
- *$ab, ac \in E_{1n}$ and $bc \in E_c$;*
- *$ab \in E_n$ and $bc, ac \in E_1$.*

A Δ **implication class** is the equivalence classes of the transitive symmetric closure of Δ. That is, e_a and e_b are in the same Δ implication class iff there is a sequence $(e_a = e_1, e_2, e_3, ..., e_{k-1}, e_k = e_b)$ of directed edges of G_{1n} such that each for each j from 1 to $k - 1$, $e_j \Delta e_{j+1}$. A Δ **color class** is the union of an implication class and its transpose, which, by symmetry, must also be an implication class.

Theorem 3.3. *[9] An orientation of G_{1n} is an interval orientation iff it is an acyclic union of Δ implication classes.*

Theorem 3.4. *[9] Edges $ab, cd \in G_{1n}$ are in the same Δ color class iff there is no Δ module X such that*

$G_{1n}|X$ contains exactly one of ab and cd. If there exist disjoint strong Δ modules Y and Z such that $a, c \in Y$ and $b, d \in Z$, then (a, c) and (b, d) are in the same implication class.

Below, we give an efficient algorithm for finding a containment orientation H of G. Theorems 3.4 and 3.3 imply that $\Delta(H)$ gives a compact representation of all interval orientations of G_{1n} corresponding to H, hence of all interval realizers of H.

Johnson and Spinrad [8] give a related way to represent implicitly all possible realizers of G, which is, in turn, related to Booth and Lueker's earlier PQ representation of possible clique arrangements in the realizer of G [2]. We achieve our improvements to their time bound by working with H instead, as it allows us to adapt and use a nice mathematical framework from the literature on transitive orientations.

4 Using the Δ tree to represent all realizers of H

If R is an interval realizer, then the **restriction** of R to X, denoted $R|X$, is the result of deleting all intervals except those in X.

The modules of a graph are often described in terms of a type of substitution operation on graphs [12]. The definition of Δ modules is motivated by a similar substitution operation on interval realizers:

Definition 4.1. *Let R_1 and R_2 be two interval realizers on disjoint sets V_1 and V_2 of intervals, and let $x \in V_1$. A* **substitution of R_2 for x in R_1** *is the realizer R that is obtained as follows.*

1. *If the two endpoints of x are contiguous in R_1, then these two endpoint are removed from the string representation of R_1, and the string representation of R_2 is substituted in their place.*

2. *If all left endpoints precede all right endpoints in R_2, then the left endpoint of x is replaced in R_1 with the sequence of left endpoints of R_2, and the right endpoint of x is replaced in R_1 with the sequence of right endpoints of R_2.*

For example, if $R_1 = uvwvwxxu$ and $R_2 = abacbc$, then substituting R_2 for x yields $uvwvwabacbcu$. On the other hand, if $R_2 = abcbac$, then it implements a complete interval graph, hence all left endpoints precede all right endpoints. We may substitute R_2 for w, yielding $uvabcvbacxxu$. Note that after a substitution, R_2 becomes a Δ module inside the resulting realizer.

In [9], it is shown that if X is a strong Δ module of H, then in every realizer of H, either the set of

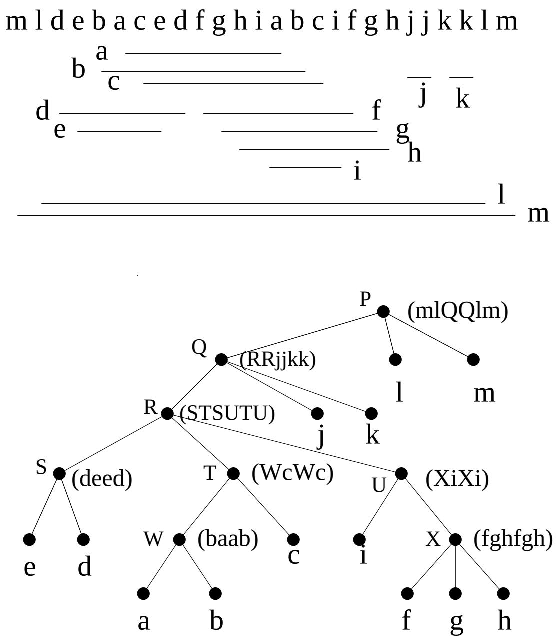

Figure 1: An interval realizer and the Δ tree for the graph $H = (V, A_c \cup E_1)$ given by a realizer R. When M is an internal node and $\mathcal{C}$ is its children, the node is labeled with a **string quotient**, depicted in parentheses. This quotient is a realizer of $(H|M)/\mathcal{C}$. By performing substitution operations in postorder, it is possible to reconstruct R in $O(n)$ time, using elementary data structures.

endpoints of X are consecutive, or else the set of left endpoints are consecutive and the set of right endpoints are consecutive. The remaining Δ modules are those sets for which this is true in some, but not all, realizers of H.

Suppose R is available, M is a node of the decomposition tree, and $\mathcal{C}$ is the children of M. Let a **string quotient** on a node X with children $\mathcal{C}$ be any realizer of the modular quotient $(H|X)/\mathcal{C}$. It is possible to reconstruct R from this labeling of the decomposition tree, by a composition of substitution operations during a postorder traversal of the tree. Figure 1 gives an example.

Because each node of the tree is prime, degenerate, or linear, the string quotient always represents a Δ-prime graph or is of one of the forms given in Figure 2. We have seen that any realizer R of H is represented by a labeling of the decomposition tree with a certain set of string quotients. Conversely, it is easily seen that whenever the tree is labeled with string quotients, it takes $O(n)$ time to assemble the corresponding realizer using substitution operations. To do this, one must

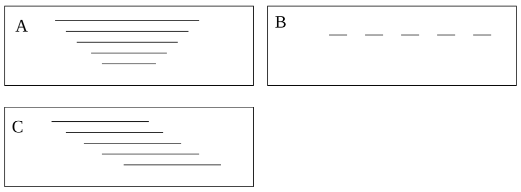

Figure 2: A string quotient is either Δ-prime (if X is prime), has the form of Figure A (if X is linear), or one of the forms of Figures B and C (if X is degenerate). In Figure 1, P, S, and W are linear, Q, T, X, and U are degenerate. R is prime; though $S \cup T$ is a module of H, it fails to be a Δ module. A new realizer of H can be represented by reversing the quotient string at a prime node, or by replacing the quotient at a degenerate node M with one of the other trivial $k!$ realizers of $(H|M)/\mathcal{C}$, where $k = |\mathcal{C}|$. Using a sequence of such replacements of quotient strings, the tree can be made to represent any realizer of H.

implement each string quotient with a doubly-linked list and a pointer to the first instance of a right endpoint in the list. In addition, each child must have pointers to its two occurrences in its parent's string quotient. This gives the following:

Theorem 4.1. *There is a one-to-one correspondence between realizers of H and ways to label the Δ tree with string quotients.*

Clearly, there is only one string quotient for a linear node, and there are $k!$ string quotients for a degenerate node with k children. If X is a prime node, then there are only two string quotients, by the Quotient Rule, Theorem 3.2, Theorem 3.3, and Theorem 3.4, and one can be obtained from the other by reversing its string representation.

These observations and Theorem 4.1 justify our claim that the Δ decomposition tree implicitly models all realizers of H. The string representations of realizers of H are a language over alphabet V, and the Δ tree gives a grammar for the language.

5 Computing the Δ tree incrementally

For finding the Δ tree, we use a generic algorithm for finding the decomposition tree of an instance of a quotient structure H on V that is given in [10]. The algorithm is expressed there as a modular decomposition algorithm, but it is proved there that it is general to all quotient-structures. The algorithm works by repeated application of an **incremental step**. At each iteration, $TD(H|U)$ is known where $U \subset V$. An arbitrary $z \in V - U$ is selected, and the decomposition is expanded to yield $TD(H|(U + z))$. By iterating, U can be expanded incrementally until $U = V$, at which time $TD(H|U) = TD(H)$ is returned.

Let $\mathcal{A}$ be the set $\{X : X \in \mathcal{F}(H|U)$ and $X + z \in \mathcal{F}(H|(U + z)\}$, let $\mathcal{A}'$ be the members of $\mathcal{A}$ that are neither disjoint from nor properly overlap any other member of $\mathcal{A}$, and let $\mathcal{A}'' = \{X + z : X \in \mathcal{A}'\}$. Let X_0 be the unique minimal member of $\mathcal{A}'$. Note that $X_0 + z \in \mathcal{F}(H|(U + z))$. If $X_0 \in \mathcal{F}(H|(U + z))$ and X_0 is not strong in $\mathcal{F}(H|(U+z)$, then $D_0 = \{z\}$; otherwise, $D_0 = X_0$. Let $\mathcal{M}$ be the maximal members of $\mathcal{F}(H|D_0)$ that are also members of $\mathcal{F}(H|(D_0 + z))$. Let $\mathcal{C}$ be the nodes of $\Delta(H|U)$ that are disjoint from D_0 or contained in a member of $\mathcal{M}$.

Theorem 5.1. *Let $\mathcal{A}''$, $\mathcal{M}$, and $\mathcal{C}$ be the sets defined above in terms of $H|U$. The nodes of $TD(H|(U + z))$ are given by $\mathcal{A}'' \cup \mathcal{M} \cup \mathcal{C} \cup \{\{z\}\}$.*

There is a unique minimal node of $TD(H|U)$ that is a member of $\mathcal{A}'$. Let Z_0 denote this node.

Algorithm 5.1. Incremental step in computing the Δ tree of a containment orientation H

1. Find Z_0, D_0, and $\mathcal{M}$.
2. If $D_0 \neq Z_0$, make D_0 a new child of Z_0, and for each child U of Z_0 that is contained in D_0, remove U from the list of children of Z_0.
3. Make $\{z\}$ and the members of $\mathcal{M}$ be children of D_0.
4. For each member X of $\mathcal{M}$, let the maximal nodes of $\Delta(H|U)$ that are proper subsets of X be the children of X.

.

Since the Δ modules define a quotient structure, we do not need to re-prove that the algorithm can be used to compute the Δ tree. Let us now address the implementation details.

During the incremental construction of the Δ tree, we implement the string realizer of each quotient using a splay-tree implementation of a list [15], which is referred to as a "path" in that paper. This allows us to maintain a function $f()$ on each point in the list that gives the number of intervals passing through the point. The data structure supports each of the following operations in $O(\log n)$ time amortized time: accessing the i^{th} element of a list, cutting a list into two lists at a given point, concatenating two lists, reversing a list, adding a constant to $f(x)$ at each element x of a list, and querying the list for the point that minimizes

$f()$, each in $O(\log n)$ time. It is easy to verify that this allows us to carry out each of the substitution operation of Definition 4.1 $O(\log n)$ amortized time, while still maintaining $f()$ in the resulting realizer.

The analysis of the time bound uses the following **credit discipline** [15]. Each node of the decomposition tree carries a credit. Processing of z requires at most $O(|N(z)|)$ new credits, and must take $O((|N(z)| + k) \log n)$ time, where k is the number of credits freed up by nodes of $\Delta(H|U)$ that are deleted in transforming the tree into $\Delta(H|(U + z))$. This clearly gives an $O(n + m \log n)$ bound for the collection of incremental steps required to build the tree.

The first step is to assign an **adjacency labeling** to nodes of $\Delta(H|U)$. This consists of two labels on each node X of $\Delta(H|U)$ such that there is an edge between z and some member of X. An undirected edge ab of H is considered to consist of two directed edges, (a, b) and (b, a). The first label tells whether all members of $\{z\} \times X$ are directed edges of H, and the second tells whether all members of $X \times \{z\}$ are directed edges of H. X is labeled **mixed** if it fails one of these tests, which implies that X is not a module in $H|(U + z)$. The absence of labels on X indicates that there are no edges of H in $\{z\} \times X$ or in $X \times \{z\}$. The adjacency labeling can be accomplished without violating the credit discipline [13, 10]. In contrast to the algorithms of [13, 10], we must also maintain a **cliquehood label** on each internal node of the tree that indicates whether it is a clique of G. This also presents no problem for the time bound.

Steps 2, 3, and 4 of Algorithm 5.1 are implemented as they are in [10] for modular decomposition, except for updating the string quotients when a node is deleted, which can be accomplished efficiently using substitution operations. Finding D_0, given Z_0, requires a straightforward modification that accommodates the additional Δ constraint that must be satisfied by Δ modules. It remains to describe how to find Z_0 and $\mathcal{M}$.

5.1 Finding Z_0.

For finding Z_0, we adopt a strategy similar to that of [13, 10], by starting at the root of $\Delta(H|U)$, and traverse downward through the chain of ancestors of Z_0 until we reach Z_0. This step is a bottleneck in the $O(n^2)$ bound in the implementations for standard modular decomposition. However, because we are implementing this step on the Δ tree rather than on the modular decomposition tree, we are able to implement it more efficiently.

We use the following approach:

Algorithm 5.2. Finding Z_0.

Input: A node W that is a (not-necessarily proper) ancestor of Z_0.

Case 1: W has more than one child labeled **mixed**. Then $Z_0 = W$

Case 2: W has exactly one mixed child Y. Then Y is an ancestor of Z_0 iff $Y + z$ is a Δ module in $H|(U+z)$; otherwise $Z_0 = W$

Case 3: No previous case applies, and W is degenerate. If there is a unique child Y whose labels differ from the label of edges between children of W, then Y is an ancestor of Z_0 iff $Y + z$ is a Δ module of $H|(U+z)$; otherwise $Z_0 = W$.

Case 4: No previous case applies, and W is linear. This is handled with a straightforward variant of the approach for Case 3.

Case 5: No previous case applies, and W is prime. If there is a child Y such that $Y + z$ is a Δ module in $H|(U + z)$, then Y is an ancestor of Z_0; otherwise $Z_0 = W$.

If $W = Z_0$, return W; else recurse on Y.

All of the cases except Case 5 are handled with the techniques from [10]. Case 5 is the actual bottleneck for the time bound of that algorithm, and a data structure for handling it is the subject of much of the paper. However, when working with Δ trees, a string realizer R' of $(H|W)/\mathcal{C}$ is available, where $\mathcal{C}$ is the set of children of W, and this quotient is Δ-prime. The problem reduces to the following:

- Given $H|(X' + z)$ such that $H|X'$ is Δ-prime, find $y \in X'$ such that $\{y, z\}$ is a Δ module in $H|(X'+z)$.

Let R' be a realizer of $H|X'$. By the Quotient Rule, adding an interval corresponding to z to R' and removing y results in a realizer R'' of an isomorphic Δ-prime graph. By Theorems 3.2, 3.3, and 3.4, the placement of endpoints of z among endpoints of R'' is unique, and must be the same positions as those of y in R'. Finding this placement, if it exists, solves the problem. Solving this problem given R' takes time proportional to the number of edges incident to z, except for one case, which requires finding a point in a given section of the string realizer that is covered by a minimum number of intervals in R'. This is accomplished in $O(\log n)$ time, using the splay-tree implementation of R'.

5.2 Finding $\mathcal{M}$.

Lemma 5.1. *[9] Let x be a source or sink in some interval orientation of G_{1n}, and let $\mathcal{P}$ denote $\{x\}$ and the maximal standard modules of $H = (V, A_c \cup E_1)$ that do not contain x. Then every member of $\mathcal{P}$ is a Δ module of H.*

Lemma 5.2. *Let X be a child of Z_0 in $\Delta(H|U)$. In the interval orientation D_{1n} of G_{1n} given by any realizer of $T|(U+z)$, z is a source or sink in $D_{1n}|(X+z)$.*

It follows from these two lemmas that finding $\mathcal{M}$ reduces to finding the maximal standard modules of H that are contained in D_0. This latter problem is solved in [13], and it is easy to implement the solution so that it conforms to the credit discipline.

6 Finding a containment orientation H for a probe-interval graph G

A probe interval graph G is realized with a set R of intervals, and a list P of those intervals that correspond to the probes. Thus, we may let (R, P) denote a **probe-interval realizer**. Just as in the case of interval realizers of interval graphs, a probe-interval realizer partitions the edges of G into a set E_c of containment edges and a set E_1 of proper-overlap edges. We let E_n denote those nonadjacent pairs where at least one member of the pair is a probe. As before, it assigns an orientation A_c of E_c, which tells, for each edge, which interval is contained in which, and an orientation A_1, which tells, for each edge, which interval precedes which. The graph $H = (V, A_c \cup E_1)$ is again a **containment orientation** of G, and H and P can be used to represent D_c, G_1, and G_n.

In this section, we examine the problem of computing a containment orientation of G, given only G.

Let $p = |P|$, $n' = |N|$, $n = p+n'$, m_p be the number of edges in $G|P$, and m_n be the number of edges from P to N, with $m = m_p + m_n$.

To simplify the problem, let us get rid of any isolated or universal vertices, and all but one vertex in any module that is contained in N, and all but one vertex in any module of N, and then all but one vertex x in any module that is a clique, selecting x to be a member of P. Now, no two adjacent vertices have identical closed neighborhoods. It takes linear time to find these and throw them out. These can all be added back in as duplicate intervals once we get a probe interval realizer on the reduced graph.

In the remainder of this section, let G denote this reduced graph, and assume that it has no clique modules or universal vertices.

Lemma 6.1. *[7] If G is an interval graph with no universal vertices or clique modules of size two, then there exists an interval realizer where $N[u] \subset N[w]$ iff u's interval is a proper subset of w's interval in the realizer.*

Lemma 6.2. *If G is a probe interval graph with no universal vertex, no clique modules, and no modules that are subsets of N, then there is a probe interval realizer of G with the following properties:*

1. *If uw is an edge of G and $u, w \in P$, then $N[u] \subset N[w]$ iff u's interval is a proper subset of w's interval;*

2. *If uw is an edge of G and one of u and w is in P and the other is in N, then $N[u] \cap P \subset N[w] \cap P$ iff u's interval is a proper subset of w's interval.*

Proof. (Sketch). No pair of vertices in N is adjacent, so the lemma is vacuously true for these pairs. For adjacent pairs in P, there is a containment orientation H' of an interval graph that is obtained is obtained by adding edges between members of N in H. There is a realizer R of H' that satisfies Lemma 6.1. Since the neighborhood of a vertex in P is the same in H as it is in H', R satisfies the claim for pairs of vertices in P.

We then show how to adjust endpoints of interval of N in R make the claim true for pairs that have one member in P and one in N. We fix the intersection for one edge at a time to make it match the lemma.

The general strategy for each case is illustrated by the case where $u \in P$, $w \in N$, and $N[u] \cap P \subset N[w] \cap P$. If w's interval does not contain u's in R, then since they are neighbors, their intervals overlap. We can stretch w's endpoint that is inside u, moving it just past the other endpoint of u. This cannot cause w to lose neighbors in P since it only grew. It cannot pick up new neighbors in P, since it is already a neighbor of all vertices of P with endpoints in u's interval. This causes the claim to be true for w and u without changing the intersection relationship on any edge that is not incident to w.

We then show that no step reverses an adjustment made in a previous step, which guarantees that the algorithm halts.

Let us create a bipartite graph $H(V, P', E_H)$ where $V = P \cup N$, P' consists of one copy of each vertex in P. To define the edges of H, let $x \in V$, $y \in P$, and y' be the copy of y in P'. Then $xy' \in H$ iff $xy \in G$. By Lemma 6.2, the problem of computing a containment orientation reduces to finding neighborhood containments in H between pairs in V that are adjacent in G, and neighborhood containments in H between pairs in P' whose copies in P are adjacent in G. By the following two facts, this takes $O(n + min\{n^2, m \log n\})$ time:

Lemma 6.3. *H is chordal bipartite.*

Theorem 6.1. *[14] It takes $O(n + k + min\{n^2, m \log n\})$ time to find neighborhood con-*

tainments for k pairs of vertices in a chordal bipartite graph.

When construction of a probe-interval model comes up as a subproblem in [9], it is in a special case where this step can also be carried out in linear time. This lucky circumstance permits linear-time recognition of circular-arc graphs.

7 Constructing probe-interval models

Let H be a containment orientation of a probe-interval graph G. If R is a probe-interval realizer of H, then $R|P$ is an interval realizer of the interval graph $G|P$ and its containment orientation $H|P$. $R|P$ has the property that the adjacencies of each $z_i \in N$ can be represented by adding a single interval to R for z_i. Let an **extensible realizer** of $H|P$ be one that has this property. An **extensible orientation** is the orientation of $G_{1n}|P$ given by an extensible realizer of $H|P$.

Finding a probe-interval realizer clearly reduces to finding an extensible realizer of $H|P$, and our algorithm works by solving this reduced problem.

The difficulty is that not all interval realizers of $H|P$ are extensible realizers. On the other hand, every extensible realizer of $H|P$ is also an interval realizer. Thus, there are additional constraints on extensible realizers that do not apply to interval realizers. We capture the necessary constraints by merging Δ implication classes in $G_{1n}|P$ to obtain more restrictive "extensible implication classes."

Note now that now that we are working with a probe-interval graph rather than an interval graph, $G_{1n} = (V, E_1 \cup E_n)$ contains no edges between members of N. Let $\{z_1, z_2, ..., z_{|N|}\}$ denote the vertices of N. If e_a and e_b are directed edges of $G_{1n}|P$, note that they are in the same implication class of $G_{1n}|P$ iff there is a sequence $(e_a = e_1, e_2, e_3, ..., e_{k-1}, e_k = e_b)$ of directed edges of G_{1n} such that each for each j from 1 to $k-1$, $e_j \Delta e_{j+1}$.

We now give the central insight of this section. For the **extensible implication classes**, *we modify this definition by letting each e_j be an arbitrary edge of G_{1n}, but require that e_j and e_{j+1} both be edges of $G_{1n}|(P+z_i)$ for some i.* Letting the chain of edges roam freely within $G_{1n}|(P+z_i)$ tightens the constraints on possible orientations: it incorporates the constraints imposed by all Δ implication classes in each $G_{1n}|(P+z_i)$. This ensures that $D_{1n}|P$ can be extended an interval orientation in each $G_{1n}|(P+z_i)$. Equivalently, it ensures that each z_i can be added as a single interval to the corresponding interval orientations of $G_{1n}|P$. By requiring that e_j and e_{j+1} be edges in the same $G_{1n}|(P+z_i)$, it we ensure that these are the only additional constraints.

An interval graph is a special case of a probe-interval graph where N is empty. We now generalize the critical theorems from interval graphs to probe interval graphs. The proof of the following is based on the "central insight" above, and generalizes Theorem 3.3:

THEOREM 7.1. *If H is a containment orientation of a probe-interval graph, then an orientation of $G_{1n}|P$ is an extensible orientation for H iff it is an acyclic union of extensible implication classes.*

Let $\mathcal{F}$ be a set family on universe V, and let $U \subseteq V$. The **restriction of $\mathcal{F}$ to U**, denoted $\mathcal{F}|P$, is the set family $\{X \cap P : X \in \mathcal{F}\}$. Let $\mathcal{A}_i$ denote the Δ modules of $H|(P+z_i)$. $\mathcal{F} = \bigcap_{i=1}^{|N|}\{\mathcal{A}_i|P\}$ is a family of Δ modules, which we may call the **extensible modules of $H|P$**.

The following generalizes Theorem 3.4:

THEOREM 7.2. *Edges $ab, cd \in G_{1n}|P$ are in the same extensible color class iff there is no extensible module X such that $G_{1n}|X$ contains exactly one of ab and cd. If there exist disjoint strong extensible modules Y and Z such that $a, c \in Y$ and $b, d \in Z$, then (a, c) and (b, d) are in the same extensible implication class.*

Let H be containment orientation given by a realizer with probe intervals P and non-probe intervals N, and let E be its edges. We can use $H(P, N, E)$ as a shorthand notation to denote this. Let $\mathcal{S}_P$ denote the set of all containment orientations of probe-interval graphs that have the probe vertices indicated. Let us consider $H_1(P_1, N_1, E)$ and $H_2(P_2, N_2, E)$ to be isomorphic members of $\mathcal{S}_P$ only if there is an isomorphism that maps P_1 to P_2 and N_1 to N_2. If $H = H(P, N, E)$ is a probe-interval graph and $X \subset P$, let $H|_P X$ denote the probe-interval graph $H(X, N, E(H|(X \cup N)))$, where $E(H|(X \cup N))$ denotes the edges of $H|(X \cup N)$. Let $V(H) = P$. Let $\mathcal{F}_P(H)$ denote the extensible modules on P in H.

The following generalizes Theorem 3.1:

THEOREM 7.3. *$(\mathcal{S}_P, V(), \mathcal{F}_P(), |_P)$ defines a quotient structure.*

Thus, $\mathcal{F}_P(H)$ has a tree decomposition, which will the **extensible tree decomposition**. Let us denote it by $ET(H)$. Note that $ET(H)$ is built on top of P only, even though it is determined by all the edges of G.

By Theorems 7.1, 7.2, and 7.3 $ET(H)$ can be used to represent all extensible orientations of $G_{1n}|P$, just as the Δ tree can be used to represent all interval orientations for an interval graph. Since all extensible modules are Δ modules, extensible modules reflect substitution operations, just as before. The main difference is that the string quotients can no longer be constrained to be Δ prime when a node is prime in the decomposition tree; the quotient may have Δ modules that are not extensible modules, hence be prime only with respect to extensible modules.

Let $H = H(P, N, E)$, and let $\{z_1, z_2, ..., z_|N|\}$ be a numbering of vertices in N Let N_i denote $\{z_1, z_2, ..., z_i\}$, and let H_i denote $H(P, N_i, E(H|(P \cup N_i)))$, let $\mathcal{F}_i$ denote the extensible modules of H_i, and let $T_i = ET(H_i)$ be their decomposition tree. $T_{|N|}$ is what we need to compute to represent the extensible realizers of $H|P$. We do this by computing the Δ tree of the interval graph $H|P$. We then pass through $|N|$ stages. At stage i, we modify T_{i-1} to get T_i.

The algorithm for producing T_i from T_{i-1} uses the following alterations to the definitions of Algorithm 5.1. Let $\mathcal{F}' = \{W : W - \{z_i\} \in \mathcal{F}_{i-1}$ and W is a Δ module of $H|(P + z_i)\}$. Let $\mathcal{A}$ be the set $\{X : X \in \mathcal{F}_{i-1}$ and $X + z_i \in \mathcal{F}'\}$. Let $\mathcal{A}'$ be the members of $\mathcal{A}$ that are neither disjoint from nor properly overlap any other member of $\mathcal{A}$, and let $\mathcal{A}'' = \{X + z_i : X \in \mathcal{A}'\}$. Let X_0 be the minimal member of $\mathcal{A}'$. $X_0 + z_i \in \mathcal{F}'$. If X_0 is also a member of $\mathcal{F}'$ and overlaps another member of $\mathcal{F}'$, then $D_0 = \{z_i\}$; otherwise, $D_0 = X_0$. Let $\mathcal{M}$ be the maximal subsets of D_0 that are members of $\mathcal{F}'$. Let $\mathcal{C}$ be the nodes of the tree representation of $\mathcal{F}_{i-1}$ that are disjoint from D_0 or contained in a member of $\mathcal{M}$.

Algorithm 7.1. Produce T_i from T_{i-1}

Input: H, z_i, and the decomposition tree T_{i-1}

Incorporate z_i into T_{i-1} using Algorithm 5.1 and the new definitions of $\mathcal{A}''$, $\mathcal{M}$, and $\mathcal{C}$ given in this section, yielding decomposition tree T' on $P + z_i$.

Remove $\{z_i\}$ from the leaves of T', and collapse any chains of duplicate nodes in the tree, to obtain T_i.

One obstacle to efficient implementation is that our implementation of Case 5 of Algorithm 5.2 assumes that the quotient at W is Δ-prime. Now, after $\{z_1, z_2, , ..., z_i\}$ have been added and removed, such a quotient is prime only in the sense that it has no extensible modules in $H|(P \cup \{z_1, ..., z_i\})$.

Lemma 7.1. *Let T_i be the tree on $H|P$ after insertion of $z_i \in N$. Let $\mathcal{B}_i$ be the set of nodes in $\{T_1, T_2, ..., T_k\}$. The transitive reduction of the containment relation on $\mathcal{B}_i$ is a tree.*

Let T_i and $\mathcal{B}_i$ be as in Lemma 7.1. The **refined version** of T_i is the transitive reduction of $\mathcal{B}_i$. A way to compute the refined version of T_i is to simply mark nodes as deleted when they are deleted, but to fail to remove them from the structure. Since we must operate on both T_i and its refined version T'_i, we give each node of T_i a list of children in T_i, and each node T'_i a list of children in T'_i. Similarly, each node of T_i gets a pointer to its parent in T_i, and each node of T'_i gets a pointer to its parent in T'_i. The nodes of T'_i that are not nodes of T_i are **helpers**. Since nodes in T_i are also nodes of T'_i, we distinguish between a node's **children** and **parent** in T_i, and its **helper children** and **helper parent** in T'_i. Each node of T'_i carries a quotient label that applies to its children in T'_i, and each node of T_i carries a second quotient label that applies to its children in T_i. The quotient labels for quotients in T_i may be expanded as before when a node gets deleted or reversed.

For the analysis of the time bound, we use the same credit discipline as before, letting a node release its credit when it becomes a helper, since it is then implicitly deleted from the tree. The parent and child pointers are used to skip over helper nodes on steps that could not charge the cost of touching helper nodes due to the absence of a credit in them.

The advantage of using the helper tree is that the the quotient induced in a prime node W by its helper children is Δ-prime, thus allowing us to carry out and analyze Algorithm 5.2 as before.

References

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

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

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

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

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

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

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

[8] 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 (2001), pp. 477–486.

[9] R. M. McConnell. *Linear-time recognition of circular-arc graphs*, Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS01), 42 (2001), to appear.

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

[11] F.R. McMorris, C. Wang, and P. Zhang. *On probe interval graphs*, Discrete Applied Mathematics, 88 (1998), pp. 315–324.

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

[13] J. H. Muller and J. P. Spinrad. *Incremental modular decomposition*, Journal of the ACM, 36 (1989), pp. 1–19.

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

[15] R. E. Tarjan. *Data structures and network algorithms.* Society for Industrial and Applied Math., Philadelphia, 1983.

[16] P. Zhang. *Probe interval graphs and its applications to physical mapping of DNA*, manuscript, 1994.

[17] P. Zhang. *United states patent: Method of mapping DNA fragments*, available at www.cc.columbia.edu/cu/cie/techlists/patents/5667970.htm. July 3, 2000.