A NEW TEST FOR INTERVAL GRAPHS

Wen-Lian Hsu[1]

Institute of Information Science, Academia Sinica
Taipei, Taiwan, Republic of China
hsu@iis.sinica.edu.tw

Abstract

An interval graph is the intersection graph of a collection of intervals. Interval graphs are a special class of chordal graphs. This class of graphs has a wide range of applications. Several linear time algorithms have been designed to recognize interval graphs. Booth & Lueker first used *PQ*-trees to recognize interval graphs in linear time. However, the data manipulation of *PQ*-trees is rather involved and the complexity analysis is also quite tricky. Korte and Möhring simplified the operations on a *PQ*-tree using an incremental algorithm. Hsu and Ma gave a simpler decomposition algorithm without using *PQ*-trees. All of these algorithms rely on the following fact : a graph is an interval graph iff there exists a linear order of its maximal cliques such that for each vertex *v*, all maximal cliques containing *v* are consecutive. Thus, the precomputation of all maximal cliques is required for these algorithms.

Based on graph decomposition, we give a much simpler recognition algorithm in this paper which directly places the intervals without precomputing all maximal cliques. A linear time isomorphism algorithm can be easily derived as a by-product. Another advantage of our approach is that it can be used to develop an *O(nlog n)* on-line recognition algorithm for interval graphs.

1. Introduction

A graph is *chordal* if it does not contain an induced cycle (a cycle without chords) of length greater than 3. A graph is an *interval graph* if it is the intersection graph of a collection of intervals on a straight line. Interval graphs are a special class of chordal graphs [3]. This class of graphs has a wide range of applications (cf. [3]). Several linear time algorithms have been designed to recognize interval graphs. Booth & Lueker [1] first used *PQ*-trees to recognize interval graphs in linear time as follows: obtain a perfect elimination ordering of the vertices of the given chordal graph. From such an ordering determine all maximal cliques. If the given graph is an interval graph, then a linear order of the maximal cliques satisfying certain consecutive property can be obtained using *PQ*-trees and an interval model can be constructed. However, the data manipulation of *PQ*-trees is rather involved and the complexity analysis is also quite tricky. Korte and Möhring [8] simplified the operations on a *PQ*-tree by reducing the number of templates. Hsu and Ma [7] gave a simpler decomposition algorithm without using *PQ*-trees. All of these algorithms rely on the following fact (cf. [2]): a graph is an interval graph iff there exists a linear order of its maximal cliques such that for each vertex *v*, all maximal cliques containing *v* are consecutive. Thus, the precomputation of all maximal cliques is required for these algorithms.

Based on graph decomposition, we give a new recognition algorithm in this paper which directly places the intervals without precomputing any maximal cliques. Besides its

[1] This research was supported in part by the National Science Council of the Republic of China under grant # NSC 84-2213-E-001-005

simplicity, our approach can also be adapted to develop an $O(n\log n)$ on-line recognition algorithm for interval graphs [6]. A preliminary version of this paper appeared in [4].

Each interval graph has a corresponding interval model in which two intervals overlap if and only if their corresponding vertices are adjacent. Such a representation is usually far from unique. To eliminate uninteresting variations of the endpoint orderings, we shall consider the following block structure of endpoints: Denote the right (resp. left) endpoint of an interval u by $R(u)$ (resp. $L(u)$). In an interval model, define a maximal contiguous set of right (resp. left) endpoints as an *R-block* (resp. *L-block*). Thus, the endpoints can be grouped as a left-right block sequence. Since an endpoint block is a set, the endpoint orderings within a block are ignored. It is easy to see that the overlapping relationship does not change if one permute the endpoint order within each block. Define two interval models for G to be *equivalent* if either their left-right block sequences are identical or one is the reversal of the other. An interval graph G is said to have a *unique model* if all interval models for G are equivalent. We shall apply decomposition in the next section to obtain almost uniquely representable interval graphs.

This paper is arranged as follows. In the next two sections we discuss some properties of interval graphs that are related to the algorithm. Section 2 introduces the decomposition scheme. Section 3 introduces the *ST*-subgraph G'. Our recognition algorithm is described in sections 4, 5 and 6. Section 4 discusses the construction of a special subgraph G'' that is used to find the decomposition tree of G in Section 5 as well as unique interval models in Section 6. The actual interval placement is described in Section 5.

2. The *S*-Decomposition for Interval Graphs

Consider a chordal graph G. Denote its number of vertices by n and its number of edges by m. A vertex u is *simplicial* in G if its neighbors form a clique. In this section we shall show that one can compose larger interval graphs by duplicating a vertex of an interval graph or by substituting a simplicial vertex of an interval graph with another interval graph. Conversely, we shall consider the following *S*-decomposition on chordal graphs, which is very similar to the substitution decomposition. A *module* in G is a set of vertices S such that (a) S is connected; (b) for any vertex $u \in S$ and $v \notin S$, $(u,v) \in E$ iff $(u',v) \in E$ for every $u' \in S$. A module S is *nontrivial* if $1 < |S| < |V(G)|$. A graph is *S-prime* if it has at least four vertices and there exists no nontrivial module. An *S*-decomposition of a graph is to substitute a nontrivial module with a marker vertex and perform this recursively for the module as well as for the reduced graph containing that marker. Note that only (*b*) is required for the module definition in the substitution decomposition. For a more elaborate discussion on the substitution decomposition, the reader is advised to consult the paper of Spinrad [12].

Define $N[u]$ to be the set of vertices including u and those vertices adjacent to u in G; let $N(u) = N[u]-\{u\}$. Two vertices u and v are said to be *similar* if $N[u] = N[v]$. All similar vertices can be located in $O(n + m)$ time by partitioning the vertices based on the neighborhood of each vertex. At the end of the partitioning process, each set with more than one vertex is a set of similar vertices. Thus, we can replace each set of similar vertices by a marker vertex. Hence, from now on, we shall assume G contains no similar vertices.

Let S be any subset of vertices. Define the induced subgraph of G on S by $G[S]$. For any subset M of $V(G)$ let $N(M)$ denote the set of vertices in V-M that are adjacent to some vertex in M. The following lemma has been proved in [7].

Lemma 2.1. *Let G be a chordal graph containing no similar vertices. Let M be a nontrivial module of G. Then $N(M)$ is a clique in G.*

Proof. By the assumption, there exist two nonadjacent vertices *u*, *u'* in *M*. Suppose there also exist two non-adjacent vertices *v*, *v'* in *N*(*M*). Then *u*, *u'*, *v* and *v'* form an induced 4-cycle in *G*, a contradiction. ▮

Corollary 2.2. *Let G be a chordal graph containing no similar vertices. Let M be a module of G. Replace each maximal proper submodule of M by a marker vertex in the graph G*[*M*]*. Denote the resulting graph by G**[*M*]*. Then these marker vertices are simplicial in G**[*M*].

If *G**[*M*] contains at least 4 vertices, then it must be *S*-prime. Each such *S*-prime graph *G**[*M*] is called an *S- prime component* of *G* (and is often referred to as the *representative graph* for *M*). A degenerate case is that *G**[*M*] contains two adjacent vertices, one of which is a marker vertex.

The result of an *S*-decomposition can be represented by a tree, where each subtree represents a nontrivial module marked by its root. For chordal graphs, a module is decomposed into its maximal submodules (these submodules are then replaced by marker vertices in *G**[*M*] as described in Lemma 2.3). For the graph *G* in Figure 1 we illustrate its *S*-decomposition tree in Figure 2. Note that all vertices of *G* are represented as leaves in the decomposition tree.

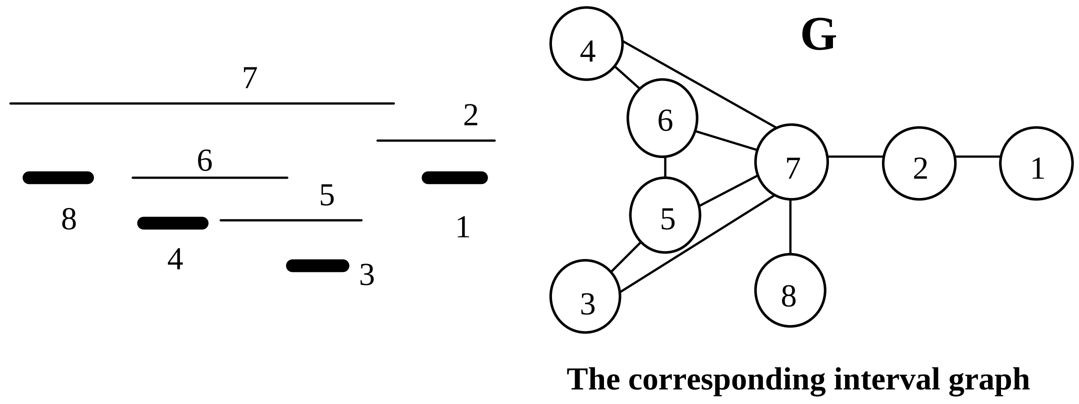

Figure 1. The graph G

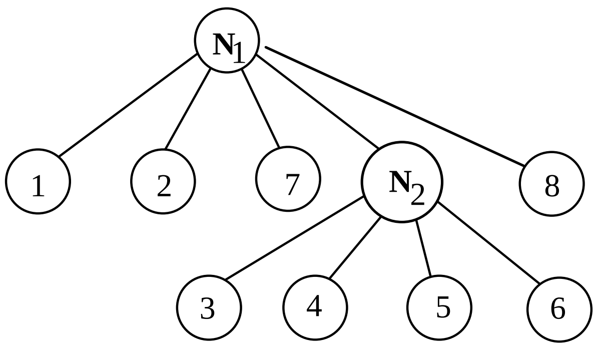

Figure 2. The *S*-decomposition tree of *G*

The following theorem is trivially true since substituting an interval graph for a simplicial vertex in another interval graph gives rise to an interval graph.

Theorem 2.3. *A chordal graph G containing no similar vertices is an interval graph iff every S-prime component of G is an interval graph.*

The main idea of our algorithm is as follows. If the given chordal graph *G* is *S*-prime, then we try to find an interval model for *G* in Section 5. Otherwise, we find an *S*-

decomposition tree for *G* and find an interval model for each *S*-prime component of *G*. The algorithms for tree and model construction are all based on the construction of a special *ST*-subgraph of *G*, which is discussed in Section 3.

3. The *ST*-Subgraph *G'* and the *S*-Decomposition Tree

In this section, we explore more properties of interval graphs that are related to the construction of *S*-decomposition trees. Assume the given graph *G* is chordal and does not have similar vertices. A vertex *i* is said to *contain* another vertex *j* if *N*[*i*] contains *N*[*j*]. Two adjacent vertices *u*, *v* in *G* are said to be *strictly adjacent* if none of *N*[*u*] and *N*[*v*] is contained in the other. Define the *ST-subgraph G'* of *G* to be the subgraph with vertex set *V*(*G*) whose edge set *E*' consists of all those strictly adjacent pairs in *G*. An example of *G'* for the graph *G* in Figure 1 is shown in Figure 4. We shall construct a *S*-decomposition tree based on the connected components of *G'*. Note that, for each edge (*u*,*v*) in *E-E'*, we have either *u* contains *v* or *v* contains *u*. Furthermore, each simplicial vertex in *G* becomes an isolated vertex in *G'* (but not vice versa).

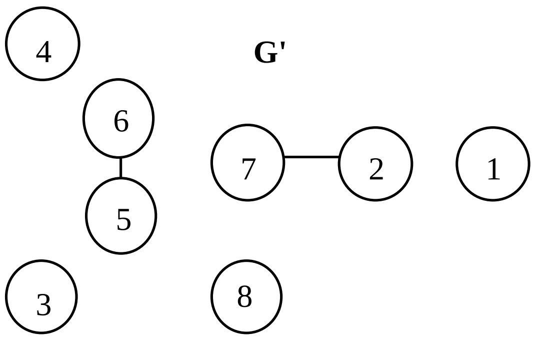

Figure 3. The *ST*-subgraph *G'*

Define the ***skeleton*** of a module *M* of *G* to be the set of vertices in *M* which are not contained in any proper submodule of *M*. The main result of this section is Theorem 3.4, which can be proved from Lemmas 3.1 and 3.3. Let *S* denote the set of simplicial vertices in G.

Theorem 3.4. *The vertex set of each connected component of G'-S gives rise to the skeleton of a submodule of G.*

These connected components are called the ***skeleton components*** of *G*'. As an example, the components {2,7}, {5,6} in Figure 3 are the skeleton components of modules N_1, N_2 respectively.

Lemma 3.1. *Let G be a chordal graph without similar vertices. Let M be a nontrivial submodule in G. Then G'*[*M*] *is disconnected from G'*[*V-M*] in *G*'.
Proof. By Lemma 2.2, *N*(*M*) is a clique in *G*. Since every vertex in *M* must be adjacent to every one in *N*(*M*), there can be no strictly adjacent pairs with one vertex from *M* and another from *V-M* in *G*. ∎

Corollary 3.2. *Let G be a chordal graph without similar vertices. Let M be a nontrivial submodule in G. Let G_M be the graph obtained from substituting M by a marker vertex m in G. Then m is a simplicial vertex in G_M.*

Lemma 3.3. *Let G be an S-prime chordal graph without similar vertices. Let D = V-S. Then G'*[*D*] *is connected.*

Proof. Since G is connected, deleting a simplicial vertex does not disconnect G. Hence, $G[D]$ must also be connected. Suppose $G'[D]$ is disconnected. Let C, C' be two connected components of $G'[D]$ with sizes greater than one such that there exist vertices u, u' in C, C', respectively, with $(u,u') \in E$. Since u and u' are not strictly adjacent in G, we may assume, without loss of generality, that u' contains u in G. Hence, u' is adjacent to every vertex in $N(u)$. We shall derive a contradiction in Claim 2.

Claim 1. u' must contain every vertex in C.

Proof of Claim 1. Suppose u' does not contain a neighbor v of u in C. Then v must contain u' (since u' cannot be strictly adjacent to v). Since v is strictly adjacent to u, there exists a vertex v' adjacent to u but not v. Since u' contains u, u' is adjacent to v', a contradiction to that v contains u'. Hence u' must contain every neighbor of u in C. Through a breadth-first-search from u in C, one can argue iteratively that u' must contain all vertices in C. █

Let Q be the set of vertices in G-C each of which is contained in some vertex in C.

Claim 2. $M = Q \cup C$ is a nontrivial module in G.

Proof of Claim 2. Let v be any vertex in G-M adjacent to a vertex u in M. We shall first argue that v must be adjacent to a vertex in C. By the definition of Q, v cannot be contained in any vertex in C. Hence, v must contain u. If $u \in Q$, then v must be adjacent to the vertex u' in C that contains u. By the above argument, v must also contain u'. Similar to the proof of Claim1, one can argue that v must contain all vertices in C. Hence, v must contain all vertices in M; in particular, v is adjacent to all vertices in M. █

In the following we shall show how to construct the S-decomposition tree of G based on the components of G'. The containment relationships of these components can be obtained as follows.

(1) Form the graph S_G whose vertex set are the connected components of G' (namely, those skeleton components of G' as well as the simplicial vertices of G) and there exists an edge between C_1 and C_2 in S_G iff there exist vertices u, v in C_1, C_2, respectively such that $(u,v) \in E$-E'.

(2) Direct the edges of S_G as follows. Each edge (u,v) in E-E' connecting two components $C(u)$ and $C(v)$ in G' satisfies that $C(u)$ contains $C(v)$ iff $\deg(u) > \deg(v)$. Direct edge $(C(u),C(v))$ from $C(u)$ to $C(v)$. Thus, all edges of S_G can be directed based on the degrees of the incident vertices in G.

(3) The skeleton C^* of the module $V(G)$ is the only component of G' which is not contained in some other component. Find the longest path graph (LPG) of S_G emanating from the vertex representing C^* (every path from C^* to a node C in an LPG is the longest possible). This is exactly the Hasse diagram $T(G)$ of the component containment relationships of G'. In Figure 4, we illustrate the Hasse diagram of the graph G' in Figure 3.

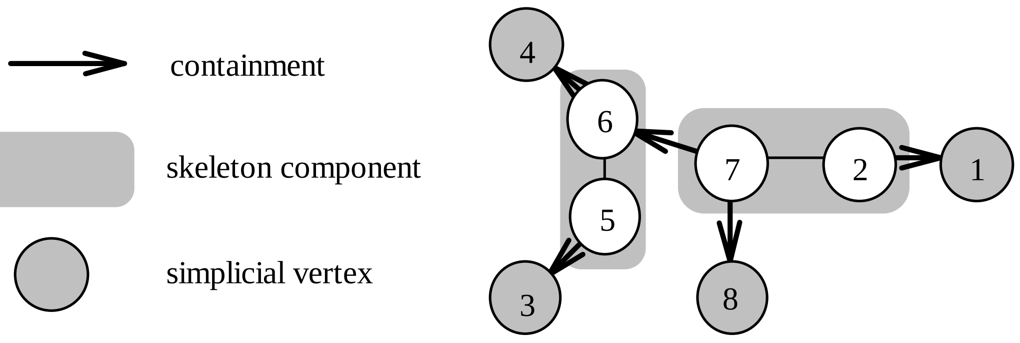

Figure 4. The Hasse diagram of the containment relationships

Lemma 3.5. The Hasse diagram $T(G)$ of S_G satisfies the following properties:

(1) $T(G)$ is a rooted tree with root C^*.

(2) For each internal node C, let T_C be the subtree consisting of node C and its descendants. The set of vertices of G in T_C (denoted by $V_G(TC)$) form a S submodule of G whose skeleton is C.

(3) The S-decomposition tree of G can be obtained from the Hasse diagram $T(G)$ as follows. For each internal node C of $T(G)$, construct a node $N(C)$ for its corresponding S-submodule $V_G(T_C)$. Construct a node for each vertex in C. Let the children of $N(C)$ be the union of the set of children of C in $T(G)$ and the set of vertices in C.

Proof. To show (1) we only have to show that the children of C^* in $T(G)$ other than the simplicial vertices of G are exactly the skeletons of maximal S-submodules of $V(G)$. The rest can be argued by induction. By Theorem 3.4, each such child is the skeleton of an S-module of G. If such an S-module, say M_1, is not maximal, then there is a maximal S-submodule of $V(G)$, say M_2, properly containing M_1. This would imply that there is a path from C^* through the skeleton of M_2 to that of M_1, longer than the direct edge from C^* to the skeleton of M_1, a contradiction. Since the maximal S-submodules of $V(G)$ are disjoint, (1) follows by induction.

Let M be the submodule of G whose skeleton is C. (2) follows from the fact that each vertex in M-C must be contained in some vertex in C. (3) follows by induction using Corollary 2.2. ▮

4. The Special Subgraph *G*”

To find the S-decomposition tree for G it suffices to find the connected components of G’. However, discovering for every adjacent pair in G, whether they are strictly adjacent in G can be very costly. Instead, we shall show how to compute a subset E” of E' efficiently that gives rise to the same set of components of G’.

A necessary and sufficient condition for a graph to be chordal is that it admits a ***perfect elimination ordering***. This is a linear ordering of the vertices such that, for each vertex v, the neighbors of v that are ordered after v form a clique. On chordal graphs, a perfect elimination ordering can be obtained in linear time through a lexicographic breadth-first-search [11] (the resulting ordering is called a ***lexicographic ordering*** **(*LO*)**). This algorithm can be regarded as a partitioning algorithm based on a certain labeling. Imagine there is a label of n digits, initially filled with zeros, associated with each vertex. The n-th vertex is chosen arbitrarily. After the (i+1)-th vertex is chosen, put a "1" in the (n-i+1)-th digit of the labels of the neighbors of the vertex. The i-th vertex is then chosen among the unchosen vertices with the greatest label (with the first digit being the most significant digit). There are two important properties (cf. [2]) of a lexicographic ordering π.

(4.1) *If graph G is chordal,* $\pi^{-1}(u) < \pi^{-1}(v) < \pi^{-1}(w)$ *and* $(u,v), (u,w) \in E$*, then* $(v,w) \in E$*, namely, those neighbors of u ordered after u must form a clique.*

(4.2) *If* $\pi^{-1}(u) < \pi^{-1}(v)$*, then either (a) for each w with* $\pi^{-1}(v) < \pi^{-1}(w)$*, w is adjacent to both of u, v or none of them; or (b) the largest w with* $\pi^{-1}(v) < \pi^{-1}(w)$ *which is adjacent to exactly one of u, v must be adjacent to v.*

A ***cardinality lexicographic ordering*** **(*CLO*)** is an *LO* obtained by breaking ties in favor of the vertex with the maximal degree. Based on a *CLO*, Hsu and Ma [7] designed a linear time algorithm to obtain a complete substitution decomposition of a chordal graph G; and by precomputing all maximal cliques of G, find a linear time recognition algorithm for interval graphs. In this paper, we use a *CLO* to compute a special subgraph G” of a chordal graph G, which allows us to skip the computation of maximal cliques of G and to compute the following in linear time:

(1) the S-decomposition tree of G

(2) an interval model for G.

Let G be a chordal graph. Perform a *CLO* on G. Let π denote the resulting ordering. Let S be the set of simplicial vertices in G and $D = V\text{-}S$. For every vertex u in D, define $f(u)$ to be the vertex in $N(u)$ with the smallest π-index. Since u is not simplicial, u must have a neighbor with a smaller π-index. Hence, we have $\pi^{-1}(f(u)) < \pi^{-1}(u)$. Consider the subgraph $G'' = (V,E'')$ whose edge set E'' is obtained as follows.

$$E'' = \{ (u,v) \mid u, v \in D, \pi^{-1}(u) < \pi^{-1}(v), f(u) \notin N(v) \}$$

Lemma 4.3. *Let G be a chordal graph containing no similar vertices. Then,* $E'' \subseteq E$.

Proof. This follows from the following two facts:

(i) $\pi^{-1}(u) < \pi^{-1}(v)$ implies $N[v] \not\subset N[u]$ by the definition of CLO and the fact that G contains no similar vertices,

(ii) $f(u) \notin N[v]$ implies $N[u] \not\subset N[v]$. ▮

The edge set E'' can be computed in linear time as follows: for every vertex x, determine $U(x) = \{ u \mid f(u) = x \}$ (this can be done for all x through a linear scan). Now, label the neighbors of x. For every u in $U(x)$, connect u to each (unlabeled) vertex w in $N[u]\backslash N[x]$ with $\pi^{-1}(w) > \pi^{-1}(u)$.

Three pairwise nonadjacent vertices in a graph are said to form an *asteroidal triple* if any two of them are connected by a path which avoids the neighborhood of the remaining vertex.

Theorem 4.4 [3]. *An interval graph does not contain an asteroidal triple.*

Theorem 4.5. *If G is an interval graph, then the connected components of G" are identical to those of G'.*

Proof. Let u, v be two strictly adjacent vertices in G with $\pi^{-1}(u) < \pi^{-1}(v)$ but $(u,v) \notin E''$. By the definition of E'', we must have $f(u) \in N(v)$. We shall show that there exists a vertex w with $\pi^{-1}(v) < \pi^{-1}(w)$ such that both (u,w) and (v,w) belong to E''. This would imply that u, v are in the same connected component of G''.

Since u, v are strictly adjacent, there exists a vertex t in $N(u)\text{-}N[v]$. By (4.1), $\pi^{-1}(t) < \pi^{-1}(u)$. Since $t \neq f(u)$ (the latter belongs to $N(v)$), we have $\pi^{-1}(f(u)) < \pi^{-1}(t)$ by the definition of $f(u)$. We now have $\pi^{-1}(f(u)) < \pi^{-1}(t) < \pi^{-1}(v)$. Since v is adjacent to $f(u)$ but not to t, by (4.2), there must exist a vertex ordered after v which is adjacent to t, but not $f(u)$. Let w be such a vertex.

Since $(t,u), (t,w) \in E$, we must have $(u,w) \in E$ by (4.1). Similarly, since $(u,v), (u,w) \in E$, we have $(v,w) \in E$. Since $(w, f(u)) \notin E$, we conclude that $(u,w) \in E''$.

We now show that $(v,w) \in E''$. If $f(v) = f(u)$, then by the definition of E'', we shall have $(v,w) \in E''$. If $f(v) \neq f(u)$, we have $\pi^{-1}(f(v)) < \pi^{-1}(f(u))$ and $(f(v),u) \notin E$. Now, we must have $(f(v),w) \notin E$; otherwise, since $(w, f(u)) \notin E$, we must have $(f(v), f(u)) \notin E$ by (4.1) and we would have an asteroidal triple $(t,f(u),f(v))$ as shown in the subgraph of Figure 5. Hence, $(v,w) \in E''$. ▮

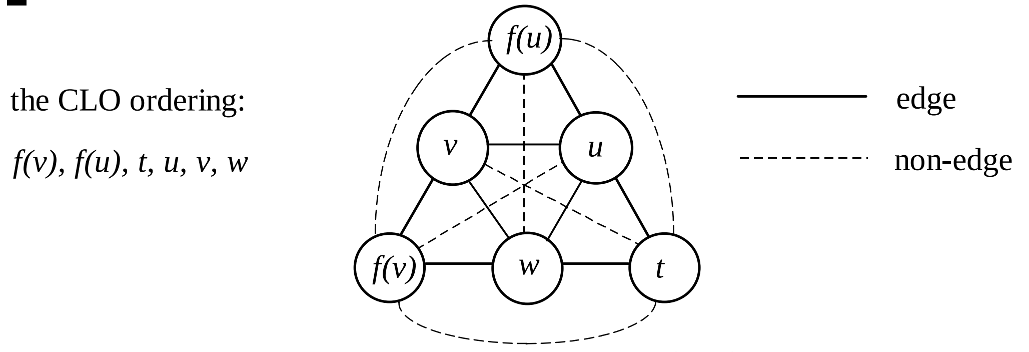

Figure 5. An asteroidal triple (t,f(u),f(v))

The basic scheme of our recognition algorithm is as follows. First, compute a *CLO* for *G*. If *G* is not chordal, stop. Otherwise, construct the subgraph G''. Based on the connected components of G'', compose an *S*-decomposition tree of *G*. Finally, test each *S*-prime component to see if it is an interval graph (Theorem 2.3). This test is the subject of the next section.

5. Constructing an Interval Model for an *S*-Prime Interval Graph

In this section, assume the graph *G* is an *S*-prime interval graph. By constructing a model for *G*, we mean arranging a left-to-right block sequence for *G*. Let *S* be the set of simplicial vertices in *G* and $D = V\text{-}S$. Since *G* is *S*-prime, the block sequence for vertices in *D* is unique. We shall directly place the intervals corresponding to the vertices into the model. The placement is divided into two stages. First, place intervals in *D* one by one into the model. Next place all intervals of *S* in one batch. Each interval to be placed in *D* is known to be strictly adjacent to at least one interval in the current placement. Since we do not have information on all the strictly adjacent relationships, the order of placement is important.

Let u_1 be a vertex in D. Perform a BFS on *D* starting with u_1 by visiting the neighbors according to their ascending degrees. Consider vertices in *D* according to their BFS ordering $\pi' = u_1u_2...u_{|D|}$ in which each u_i with $i \geq 1$ has its children arranged consecutively in π'. Denote by $B_L(u_i)$, the left block containing $L(u_i)$; $B_R(u_i)$, the right block containing $R(u_i)$; and $[B_L(u_i),B_R(u_i)]$ the set of block subsequence from $BL(u_i)$ to $BR(u_i)$.

For the first two intervals u_1 and u_2 of *D*, let $B_L(u_2) = \{L(u_1),L(u_2)\}$, $B_R(u_2) = \{R(u_1),R(u_2)\}$. Find two intervals v_1 and v_2 such that v_1 is adjacent to u_1 but not u_2 in G, and v_2 is adjacent to u_2 but not u_1 in G. Initially, we shall place neighbors of u_1 into the model. Assume we have already placed $u_1, u_2, \ldots, u_{i-1}$. Let u_i be the next neighbor of u_1 to be placed.

An endpoint is said to be *contained in an interval* u_i if it is contained in $[B_L(u_i), B_R(u_i)]$. Without loss of generality, assume $R(u_1)$ is contained in u_2. We first determine (uniquely) whether $R(u_i)$ or $L(u_i)$ is contained in u_1. Consider the following two cases:

Case 1. u_i is not adjacent to u_2 in G. Then we must have that $R(u_i)$ is contained in u_1.

Case 2. u_i is adjacent to u_2 in G. If u_i is also adjacent to v_2 in G, then we have that $L(u_i)$ is contained in u_1. Otherwise, consider two subcases (since $(u_i,u_1) \in E''$, one of them must hold): (i) $f(u_i)$ is not adjacent to u_1. If $f(u_i)$ is adjacent to u_2, then $L(u_i)$ is contained in u_1; otherwise, $R(u_i)$ is contained in u_1. (ii) $f(u_1)$ is not adjacent to u_i. Note that u_i cannot be contained in u_2. If $(f(u_1), u_2) \in$ E, then $R(u_i)$ is contained in u_1; otherwise, $L(u_i)$ is contained in u_1.

Once we know whether $R(u_i)$ or $L(u_i)$ is contained in u_1, the two endpoints of u_i can be uniquely inserted into the current block sequence as follows. Perform two searches starting from the block containing that endpoint of u_j: one to the right of that block, the other to the left. Mark all endpoints of intervals adjacent to u_i. Towards the right, find the first *L*-block B_{L^*} that contains an unmarked left endpoint. If B_{L^*} also contains a marked endpoint, then split B_{L^*} into two *L*-blocks B_1, B_2, where B_1 contains those marked endpoints of *B*, and B_2 contains those unmarked endpoints of *B*. Create a new R-block between B_1 and B_2 containing $R(u_i)$. If all endpoints of B_{L^*} are unmarked, then insert $R(u_i)$ into the *R*-block immediately to the left of B_{L^*}. The insertion of $L(u_i)$ can be done symmetrically to the left. Note that, because of the relations of strict adjacency, certain left-right endpoint relationships within the same R-block or L-block must be maintained. Our main result is Theorem 5.2, which can be proved based on Lemma 5.1.

Theorem 5.2. *The placement for vertices in D at each iteration is unique.*

Now, the placement of u_1 and its children is unique as can be seen from the above description. Hence, we only have to show that the remaining vertices can be placed uniquely into the current model.

Lemma 5.1. *Let u_i be the next interval to be placed, which is not a neighbor of u_1. Let u_j be the parent of u_i in the BFS-tree. It can be determined uniquely whether $R(u_j)$ or $L(u_j)$ is contained in u_i.*
Proof. Let u_k be the parent of u_j. Assume, without loss of generality, $R(u_k)$ is contained in u_j (see Figure 6). Since u_i is not a child of u_k, we must have $(u_i,u_k) \notin E''$. If $(u_i,u_k) \notin E$, then clearly, $R(u_j)$ is contained in i. If $(u_i,u_k) \in E$, then we shall show that $L(u_j)$ is contained in u_i. Note that the test whether u_i is adjacent to u_k or not can be performed for all children of u_j in T in time proportional to $|N[u_k]| + |N[u_j]|$. Consider the following cases:
Case 1. $\pi^{-1}(u_i) < \pi^{-1}(u_j)$ and $\pi^{-1}(u_i) < \pi^{-1}(u_k)$. Since $(u_i,u_k) \notin E''$, we must have $(f(u_i),u_k) \in E$.

Case 2. $\pi^{-1}(u_j) < \pi^{-1}(u_i)$ and $\pi^{-1}(u_j) < \pi^{-1}(u_k)$. Since $(f(u_j),u_k) \notin E$ and $(f(u_j),ui) \notin E$, u_i and u_k must be on the same side, namely $L(u_j)$ is contained in both u_i and u_k.
Case 3. $\pi^{-1}(u_k) < \pi^{-1}(u_i)$ and $\pi^{-1}(u_k) < \pi^{-1}(u_j)$. Since $(u_i,u_k) \notin E''$, we have $(f(u_k),u_i) \in E$. Since $(u_k,u_j) \in E''$, we have $(f(u_k),u_j) \notin E$. ∎

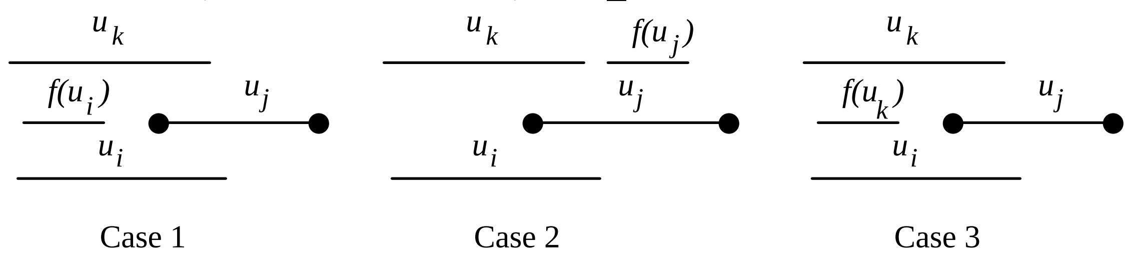

Figure 6. Proofs of the three cases

Finally, all simplicial vertices of G can be placed through a left-to-right scan of the block sequence for D as follows. For each endpoint t scanned, let Q be the set of all intervals whose left endpoints have already been scanned, but whose right endpoints have not been. For each simplicial vertex u, keep a counter $T(u)$ on the number of left-endpoints of $N(u)$ encountered so far. Let $R_L(u)$ be the *right-most left endpoint* of $N(u)$. Let $L_R(u)$ be the *left-most right endpoint* of $N(u)$.

Now, scan the list from left to right and update the counters Q and $T(u)$ for each u. For each left endpoint $L(v)$ scanned increase the counter $T(\cdot)$ of each simplicial vertex of $N(v)$ by 1. As soon as the counter $T(u)$ equals $|N(u)|$ for a simplicial vertex u, the left endpoint currently scanned becomes $R_L(u)$. At this point, we have $Q \supset N(u)$, and $Q\text{-}N(u)$ are those intervals that are not adjacent to u whose right endpoints have not been scanned. Note that endpoints of u must be inserted between $R_L(u)$ and $L_R(u)$ at the place where the current $Q\text{-}N(u) = \varnothing$ (namely $|Q| = |N(u)|$). Hence, when $T(u) = |N(u)|$ for the first time we push u into a stack. When the current $Q\text{-}N(u)$ becomes $\varnothing$, we shall pop element from the stack until u is popped out. Note that any vertex u' popped out the stack together with u will satisfy $Q\text{-}N(u') = \varnothing$. If any of the above conditions is not satisfied, G is not an interval graph. Note that $Q\text{-}N(u)$ does not have to be updated for every u at every endpoint scanning. One only has to evaluate $Q\text{-}N(v)$ for the top vertex v of the stack at each scanning. Since there may be independent simplicial vertices with the same neighbors, the interval orders among them can be arbitrary.

Since all of the above placement can be carried out uniquely (up to the permutation of independent simplicial vertices with the same neighborhood), if any discrepancy occurs, we can immediately conclude that the given graph is not an interval graph; otherwise, an interval model is constructed whose overlapping relationship can be checked against the adjacent relationship of G.

6. Complexity Analysis

The algorithm involves the following steps each of which can be done in $O(m + n)$ time.

(1) The construction of CLO in Section 4.
(2) The construction of the special subgraph G'' in Section 4. This provides the basis for the construction of the decomposition tree as well as the interval models.
(3) The construction of the Hasse diagram for containment relationship of components in G'' (this involves the computation of a longest path graph) in Section 4.
(4) The construction of the *S*-decomposition tree from the Hasse diagram in Section 4.
(5) The construction of interval models for each *S*-prime component of *G* in Section 5.

Hence, the total running time of the recognition algorithm is linear.

Since a by-product of this algorithm is an (unique) *S*-decomposition tree, and each prime component almost has a unique interval model, it would not be difficult to derive a linear time isomorphism algorithm for interval graphs from this recognition algorithm. We should note that a linear time isomorphism algorithm was designed in [9] based on labeled *PQ*-tree.

REFERENCES

1. K. S. Booth and G. S. Lueker, *Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-tree Algorithms*, **J. Comput Syst. Sci.** 13, 1976, 335-379.

2. D. R. Fulkerson and O. A. Gross, *Incidence Matrices and Interval Graphs*, **Pacific J. Math.** 15, 1965, 835-855.
3. M. C. Golumbic, **Algorithmic Graph Theory and Perfect Graphs,** Academic Press, New York, 1980.
4. W. L. Hsu, *A simple test for interval graphs*, **LNCS** 657, (1992), 11-16.
5. W. L. Hsu, *O(mn) algorithms for the recognition and isomorphism problems on circular-arc graphs*, **SIAM J. Comput 24**, 1995, 411-439..
6. W. L. Hsu, *On-line recognition of interval graphs*, **LNCS** 1120, 1996, 27-38.
7. W. L. Hsu and T. H. Ma, *Substitution Decomposition on Chordal Graphs and Applications*, **LNCS** 557, 1991, 52-60; also to appear in **SIAM J. Comput**.
8. N. Korte and R. H. Möhring, *An Incremental Linear-Time Algorithm for Recognizing Interval Graphs*, **SIAM J. Comput.** 18, 1989, 68-81.
9. C. G. Lekkerkerker and J. Ch. Boland, *Representing of a finite graph by a set of intervals on the real line*, **Fund. Math.** 51, 1962, 45-64.
10. G. S. Lueker and K. S. Booth, *A linear time algorithm for deciding interval graph isomorphism*, **J. ACM** 26, 1979, 183-195.
11. D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, **SIAM J. Comput**. 5, 1976, 266-283.
12. J. Spinrad, *On Comparability and Permutation Graphs*, **SIAM J. Comput.** 14, 1985, 658-670.

Mark a block ***full*** (respectively, ***partial***) if all (respectively, part) of its endpoints are marked. In order for G to be an interval graph, there can be at most two partial blocks and those blocks that are either full or partial must form a consecutive subsequence ***S*** with the partial blocks (if any) at the ends. The partial block at the right (respectively, left) end of S must be an L-block (respectively, R-block); the full block at the right (respectively, left) end of S must be an R-block (respectively, L-block).

Assume G is a prime interval graph. Let D be the set of non-simplicial vertices in G. Then G"[D] is connected.

$(f(u_i),u_j)$, $(f(u_j),u_k) \in E$. We now have the following two cases: (i) ui is adjacent to $f(u_k)$. Then $L(u_j)$ is contained in u_i; (ii) ui is not adjacent to $f(u_k)$. Then $(u_k,u_i) \notin E$; for otherwise,

Case 2. $i < k < j$. $(f(u_i),u_k)$, $(f(u_k),u_j) \in E$

Case 3. $j < i < k$. $(f(u_j),u_i)$, $(f(u_i),u_k) \in E$

Case 4. $k < i < j$. $(f(u_k),u_i)$, $(f(u_i),u_j) \in E$

Case 5. $j < k < i$. $(f(u_j),u_k)$, $(f(u_k),u_i) \in E$

Case 6. $k < j < i$. $(f(u_k),u_j)$, $(f(u_j),u_i) \in E$

Let The fact that the block formation and their ordering at each iteration are unique can be easily shown by induction.

Let u_i be the next interval to be placed. Let u_j be the parent of u_i, u_k the parent of u_j. Without loss of generality, assume $L(u_j)$ is in $[BL(u_k),BR(u_k)]$. We shall determine which of $L(u_j)$ and $R(u_j)$ is in $[BL(u_i),BR(u_i)]$ as follows. If (u_i,u_k) Ï E, then $R(u_j)$ is in $[BL(u_i),BR(u_i)]$ (as shown in case (*i*) of Figure 6). If (u_i,u_k) Î E, Mark all endpoints of intervals overlapping with u_i. If there exists an unmarked left endpoint in $[R(u_k),R(u_j)]$, contained in u_i using the fact that $L(u_j)$ is contained in u_i. Starting from the block $BR(u_j)$, traverse to the left. If any unmarked endpoint is encountered before the block $BR(u_k)$, then $L(u_j)$ is in $[BL(u_i),BR(u_i)]$ (as shown in case (ii) of Figure 6); otherwise, $R(u_j)$ is in $[BL(u_i),BR(u_i)]$ (as shown in case (iii) of Figure 6).

Note that the consecutive endpoint property is used here.

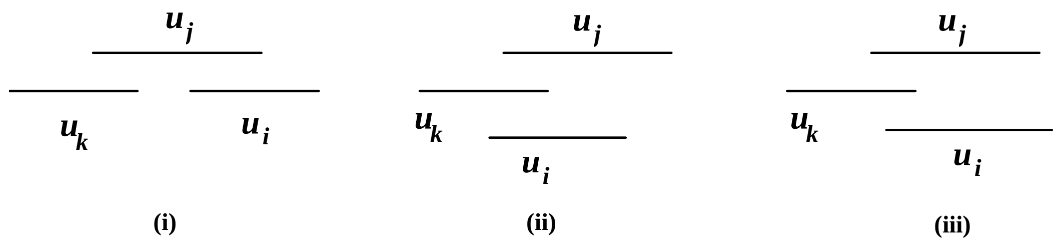

Figure 6. Three possible overlapping relationships

(1) Let u be a vertex in a skeleton C. Let C_1, C_2, …, C_r be those children of C each of whose vertices is contained in u. Then the union of their corresponding modules form a type III module in G. And all type III modules of G are obtained this way.

Note that, in order for the procedure to work, not only the final placement has to be unique, but at each intermediate step the placement has to be unique. Of course, in order for the latter to be uniquely done, adjacency relationships of all intervals have to be taken into consideration, not just those in the current placement.

We illustrate in Figure 6 an example that creates different placements if . Suppose *u*, *v*, *w* have already been placed, and 1, 2 will be placed next. Both 1 and 2 are strictly adjacent to *u*, and are adjcent to *v*, but not to *w*.

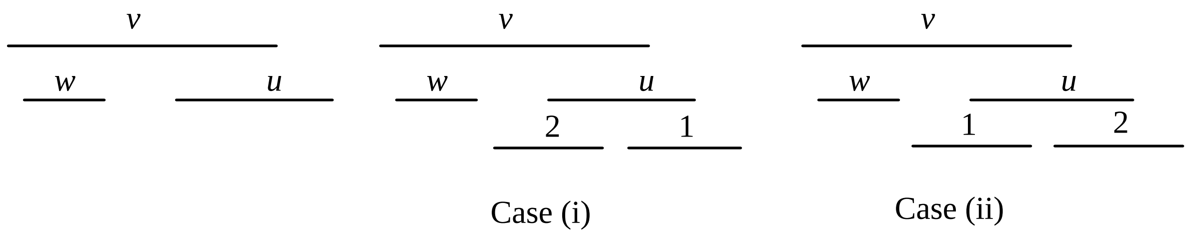

Figure 7. An example of non-unique placement

Since there is a unique substitution decomposition tree for an interval graph and each prime interval graph has a unique left-to-right block sequence, the isomorphism problem for interval graphs can be reduced to that for labelled trees []. In our computation, a type II decomposition can be obtained in linear time assuming the graph does not contain similar vertices. Since there are only type III modules (collections of independent simplicial vertices with the same neighborhood) left in each representative graph of this tree, Finally, if *G* contains similar vertices (must be cliques), one can further modify the above tree by replacing the representatives of similar vertices in the above tree by a new node with those vertices in the module as children.