A fully dynamic graph algorithm for recognizing interval graphs [*]

Louis Ibarra[†]

January 30, 2009

Abstract

We present the first dynamic graph algorithm for recognizing interval graphs. The algorithm runs in $O(n \log n)$ worst-case time per edge deletion or edge insertion, where n is the number of vertices in the graph. The algorithm uses a new representation of interval graphs called the *train tree*, which is based on the clique-separator graph representation of chordal graphs. The train tree has a number of useful properties and it can be constructed from the clique-separator graph in $O(n)$ time.

1 Introduction

1.1 Dynamic graph algorithms

A dynamic graph algorithm maintains a solution to a graph problem as the graph undergoes a series of small changes, such as single edge deletions or insertions. For every change, the algorithm updates the solution faster than recomputing the solution from scratch, i.e., with no previously computed information. Typically, a dynamic graph algorithm has a preprocessing step to compute a solution for the initial graph, along with some auxiliary information. For example, the World Wide Web may be modeled by a dynamic graph whose vertices and edges represent network nodes and links that unpredictably may lose or gain functionality as the network becomes congested, equipment fails, or new equipment is introduced.

We consider dynamic graph algorithms that support the following two operations: a *query* is a question about the solution being maintained, e.g., "Are vertices u, v connected?" or "Is the graph planar?", and an *update* is an edge deletion or edge insertion. A *fully* dynamic graph algorithm supports both deletions and insertions. A *deletions-only* or *insertions-only* dynamic graph algorithm supports only deletions or only insertions, respectively. Some problems are easier when only deletions or only insertions are allowed. For example, there is a fast insertions-only dynamic algorithm for connectivity that uses disjoint-set union [Tar75], but fast fully dynamic algorithms are considerably more complicated [HdT01].

[*]Most of the results in this paper were part of the author's Ph.D. thesis at the University of Victoria.

[†]School of Computing, DePaul University, 243 S. Wabash Ave., Chicago, IL 60604, U.S.A.

Fully dynamic algorithms have been developed for numerous problems on undirected graphs, including connectivity, biconnectivity, 2-edge connectivity, bipartiteness, minimum spanning trees, and planarity [EGI98, FK99]. There are also algorithms that support vertex insertions and vertex deletions, e.g., [HSS01].

1.2 Previous work

Chordal graphs can be recognized in $O(m + n)$ time using a graph search such as Lex-BFS or Maximum Cardinality Search [RTL76, TY84]. Several well-known NP-complete problems can be solved on chordal graphs in $O(m+n)$ time [Joh85]. There is a fully dynamic algorithm that maintains a clique tree of a chordal graph in $O(n)$ time per update [Iba08a]. This algorithm supports updates that yield a chordal graph and queries that ask whether a particular update is supported. (All the running times in this paper are worst-case.)

Interval graphs can be recognized in $O(m + n)$ time using PQ-trees [BL76], MPQ-trees [KM89], a substitution decomposition computed with Lex-BFS [HM99], or a vertex ordering computed with multiple passes of Lex-BFS [COS98]. Many well-known NP-complete graph problems can be solved on interval graphs in polynomial time [Joh85]. There is an incremental algorithm for interval graphs that runs in $O(\log n)$ time per vertex insertion and $O(m + n \log n)$ total time [Hsu96]. This algorithm supports vertex insertions that yield an interval graph and queries that ask whether a particular insertion is supported.

Proper interval graphs can also be recognized in $O(m+n)$ time [CKN$^+$95, DHH96]. There is a fully dynamic algorithm that maintains a straight enumeration of a proper interval graph in $O(\log n)$ time per edge update and $O(d + \log n)$ time per vertex update, where d is the degree of the vertex [HSS01]. This algorithm supports updates that yield a proper interval graph and queries that ask whether a particular update is supported.

The clique-separator graph of a chordal graph G is a graph $\mathcal{G}$ whose nodes are the maximal cliques and minimal vertex separators of G and whose (directed) arcs and (undirected) edges represent the containment relations between the sets corresponding to the nodes [Iba06]. The clique-separator graph reflects the structure of G and it has various structural properties when G is an interval graph, proper interval graph, or split graph. The clique-separator graph can be constructed in $O(n^3)$ time if G is a chordal graph, in $O(n^2)$ time if G is an interval graph, and in $O(m + n)$ time if G is a proper interval graph [Iba06].

1.3 Our results

We introduce the *train tree* of an interval graph G, which is based on its clique-separator graph $\mathcal{G}$. The train tree is also based on the PQ-tree [BL76]. In the Booth-Lueker algorithm for recognizing interval graphs [BL76], the PQ-tree represents valid orderings of the maximal cliques and it is built with a complicated set of templates. The train tree represents valid orderings of the maximal cliques *and* minimal vertex separators, and it is built with a simpler algorithm that uses none of the PQ-tree templates. In addition, the train tree specifies the vertices separated by each minimal vertex separator and it has a number of other properties that may be useful for future work with interval graphs.

We use the clique-separator graph and the train tree to develop the first dynamic graph algorithm for interval graphs. The algorithm maintains both representations and supports edge updates that yield an interval graph and queries that ask whether a particular update is supported. We show that an update in G requires only a small, local change in $\mathcal{G}$ because at most two new maximal cliques or minimal vertex separators are created, each of which differs from an old maximal clique or minimal vertex separator by at most two vertices. Each operation updates the clique-separator graph $\mathcal{G}$ in $O(n \log n)$ time and then builds the new train tree from the new $\mathcal{G}$ in $O(n)$ time.

One potential application for this algorithm is in physical mapping of DNA. Molecular biologists often break DNA into fragments and then attempt to reconstruct the positions of the fragments on the original DNA. Some fragments overlap and new experimental data may change the list of fragments that are known to overlap [HSS01]. We create a graph where the vertices are the fragments and there is an edge between every pair of vertices corresponding to overlapping fragments. Since there may be many fragments that overlap, the graph is potentially dense. In order for the physical mapping problem to be solvable, the graph must be an interval graph. The dynamic graph algorithm would determine whether the graph is an interval graph as edges are added or deleted because of new experimental data.

Since interval graphs $\subset$ chordal graphs and we will use results for these graph classes, we review them in Section 2. In Section 3, we review properties of the clique-separator graph $\mathcal{G}$ when G is a chordal graph or an interval graph. In Section 4, we present the train tree and its properties. In Section 5, we present the train tree algorithm that constructs a train tree of an interval graph from its clique-separator graph in $O(n)$ time. In Section 6 and Section 7, we present the delete and insert operations that run in $O(n \log n)$ time. In Section 8, we present conclusions and open problems.

2 Preliminaries

Throughout this paper, let $G = (V, E) = (V(G), E(G))$ be a simple, undirected graph and let $n = |V|$ and $m = |E|$. For a set $S \subseteq V$, the subgraph of G *induced* by S is $G[S] = (S, E(S))$, where $E(S) = \{\{u, v\} \in E \mid u, v \in S\}$. For a set $S \subset V$, $G - S$ denotes $G[V - S]$. A *clique* of G is a set of pairwise adjacent vertices of G. A *maximal clique* of G is a clique of G that is not properly contained in any clique of G. Figure 1a shows a graph with maximal cliques $\{x, y, z\}, \{y, z, w\}, \{u, z\}$, and $\{v, z\}$.

Let $S \subset V$. S is a *separator* of G if there exist two vertices that are connected in G and not connected in $G - S$. S is a *minimal separator* of G if S is a separator of G that does not properly contain any separator of G. For $u, v \in V$, S is a *uv-separator* of G if u, v are connected in G and not connected in $G - S$. S is a *minimal vertex separator* of G if for some $u, v \in V$, S is a uv-separator of G that does not properly contain any uv-separator of G. Every minimal separator is a minimal vertex separator, but not vice versa. Figure 1a shows a graph with minimal separator $\{z\}$ and minimal vertex separators $\{z\}$ and $\{y, z\}$. Notice $\{y, z\}$ is a minimal xw-separator and $\{z\}$ is a minimal xv-separator, yv-separator, etc. For $u, v \in V$, S *separates* u, v if S is a uv-separator of G. For $X, Y \subset V$, S *separates* X, Y if S is a uv-separator of G for every $u \in X$ and $v \in Y$.

A graph is *chordal* (or *triangulated*) if every cycle of length greater than 3 has a *chord*, which is an edge joining two nonconsecutive vertices of the cycle. A graph G is chordal if and only if G has a *clique tree*, which is a tree T on the maximal cliques of G with the *clique intersection property*: for any two maximal cliques K and K', the set $K \cap K'$ is contained in every maximal clique on the K–K' path in T [BP93, Gol04]. The clique tree is not necessarily unique. Blair and Peyton [BP93] discuss various properties of clique trees, including the following correspondence between the edges of T and the minimal vertex separators of G. (In [BP93], G is a connected chordal graph and then the clique intersection property implies that $K \cap K' \neq \emptyset$ for every $\{K, K'\} \in E(T)$.)

Theorem 1 ([HL89, Lun90]) *Let G be a chordal graph with clique tree T. Let $S \neq \emptyset$ be a set of vertices of G.*

1. *S is a minimal vertex separator of G if and only if $S = K \cap K'$ for some $\{K, K'\} \in E(T)$.*
2. *If $S = K \cap K'$ for some $\{K, K'\} \in E(T)$, then S separates $K - S$ and $K' - S$ and moreover, S is a minimal uv-separator for any $u \in K - S$ and $v \in K' - S$.*

It follows that for any clique tree T of a chordal graph G, the nodes and edges of T correspond to the maximal cliques and minimal vertex separators of G, respectively. A maximal clique corresponds to exactly one node of T and a minimal vertex separator may correspond to more than one edge of T. Furthermore, since a chordal graph G has at most n maximal cliques [FG65], G has at most $n - 1$ minimal vertex separators.

An *interval graph* is a graph where each vertex can be assigned an interval on the real line so that two vertices are adjacent if and only if their assigned intervals intersect; such an assignment is an *interval representation*. A graph is an interval graph if and only if it has a clique tree that is a path, which is a *clique path* [FG65]. Interval graphs are a proper subclass of chordal graphs. Since interval graphs model many problems involving linear arrangements, interval graphs have applications in many areas, including archeology, computational biology, file organization, partially ordered sets, and psychology [BLS99, Gol04, MM99].

Golumbic [Gol04] and McKee and McMorris [MM99] discuss these classes in the context of perfect graphs and intersection graphs, respectively. Johnson [Joh85] discusses them with respect to the hardness of problems.

We will use some terms from partially ordered set theory. Let $\mathcal{F}$ be a family of sets. For $S \in \mathcal{F}$, S is a *minimal element of* $\mathcal{F}$ if no $S' \in \mathcal{F}$ satisfies $S' \subset S$, and S is a *maximal element of* $\mathcal{F}$ if no $S' \in \mathcal{F}$ satisfies $S \subset S'$. $\mathcal{F}$ is a *chain* if for any $S, S' \in \mathcal{F}$, we have $S \subseteq S'$ or $S' \subseteq S$. $\mathcal{F}$ is an *antichain* if for any distinct $S, S' \in \mathcal{F}$, we have $S \not\subseteq S'$ and $S' \not\subseteq S$.

3 The clique-separator graph

In this section, we review the properties of the clique-separator graph $\mathcal{G}$ when G is a chordal graph, including (a) the edges of $\mathcal{G}$ induce a forest, i.e., deleting the arcs of $\mathcal{G}$ produces a forest, (b) contracting every tree of this forest into a single node yields a directed acyclic graph, and (c) for every minimal vertex separator S of G, the vertices of G separated by S are specified by the nodes of $\mathcal{G}$ separated by the node corresponding to S. We then review the properties of $\mathcal{G}$ when G is an interval graph, including (d) every node in $\mathcal{G}$ has at most two predecessors and (e) every tree of the forest (with certain leaves removed) is the unique clique path on those nodes. We will use these properties to construct the train tree and to perform updates.

3.1 The clique-separator graph of a chordal graph

Let G be a chordal graph. The *clique-separator graph* $\mathcal{G}$ of G has the following nodes, (directed) arcs, and (undirected) edges.

- $\mathcal{G}$ has a *clique node* K for each maximal clique K of G.
- $\mathcal{G}$ has a *separator node* S for each minimal vertex separator S of G.
- Each arc is from a separator node to a separator node. $\mathcal{G}$ has arc (S, S'') if $S \subset S''$ and there is no separator node S' such that $S \subset S' \subset S''$.
- Each edge is between a separator node and a clique node. $\mathcal{G}$ has edge $\{S, K\}$ if $S \subset K$ and there is no separator node S' such that $S \subset S' \subset K$.

Throughout this paper, we refer to the *vertices* of G and the *nodes* of $\mathcal{G}$ and we use lowercase variables for vertices and uppercase variables for nodes. We identify "maximal clique" and "clique node" and identify "minimal vertex separator" and "separator node". Figure 1 shows a chordal graph G and its clique-separator graph $\mathcal{G}$, which has clique nodes $K_1 = \{x, y, z\}, K_2 = \{y, z, w\}, K_3 = \{u, z\}, K_4 = \{v, z\}$ and separator nodes $S_1 = \{y, z\}$ and $S_2 = \{z\}$.

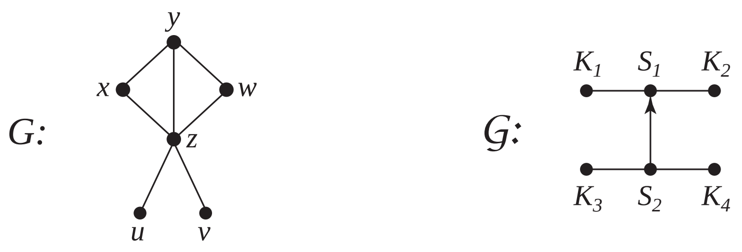


Figure 1: A chordal graph and its clique-separator graph.

Let S and S' be distinct separator nodes of $\mathcal{G}$. If (S, S') is an arc of $\mathcal{G}$, then S is an *immediate predecessor* of S' and S' is an *immediate successor* of S, denoted $S \rightarrow S'$. If there is a directed path from S to S' in $\mathcal{G}$, then S is a *predecessor* of S' and S' is a *successor* of S, denoted $S \rightsquigarrow S'$. $S \rightsquigarrow S'$ if and only if $S \subset S'$. Let K be a clique node of $\mathcal{G}$. If $\{S, K\}$ is an edge of $\mathcal{G}$, then S is a *neighbor* of K and S is *adjacent* to K and vice versa. The number of neighbors of a node is its *degree.* If S is a neighbor of K or there is a directed path from S to a neighbor of K in $\mathcal{G}$, then S is a *predecessor* of K and K is a *successor* of S, denoted $S \rightsquigarrow K$. $S \rightsquigarrow K$ if and only if $S \subset K$. The set of immediate predecessors of S, the set of immediate successors of S, and the set of neighbors of K are antichains.

We can readily show (see [Iba06]) that for every separator node S, there are at least two edges, or two arcs, or an edge and an arc, from S. The following is a simple property of the clique-separator graph that follows from the clique intersection property of clique trees.

Lemma 2 ([Iba06]) *Let G be a chordal graph with clique-separator graph $\mathcal{G}$. For any clique node K and any node $N \neq K$, if $N \cap K \neq \emptyset$, then $N \cap K$ is contained in some neighbor of K in $\mathcal{G}$.*

For a set of nodes $\mathcal{S}$ of $\mathcal{G}$, let $V(\mathcal{S}) = \{v \in V \mid v \in N \text{ and } N \in \mathcal{S}\}$. For a subgraph $\mathcal{H}$ of $\mathcal{G}$, let $V(\mathcal{H}) = \{v \in V \mid v \in N \text{ and } N \text{ is a node of } \mathcal{H}\}$. $\mathcal{H}$ is *connected* if the underlying undirected graph of $\mathcal{H}$ (obtained by replacing every arc with an edge) is connected. $\mathcal{G}$ is connected if and only if G is connected. If a subgraph $\mathcal{H}$ of $\mathcal{G}$ is connected, then $G[V(\mathcal{H})]$ is connected, but the converse is not true in general.

A *box* of $\mathcal{G}$ is a connected component of the undirected graph obtained by deleting the arcs of $\mathcal{G}$. By Lemma 2, a box of $\mathcal{G}$ contains exactly one clique node K and no separator nodes if and only if K is a complete connected component of G. We will show that every box is a tree and so we extend the definition of the clique intersection property to a tree T on a set of nodes in the natural way: for any two nodes N and N' of T, the set $N \cap N'$ is contained in every node on the N–N' path in T.

Figure 2 shows a chordal graph G and Figure 3 shows its clique-separator graph $\mathcal{G}$, where separator nodes S_0, S_1, S_2 have size 1, S_8 has size 3, and every other S_i has size 2. The boxes of $\mathcal{G}$ are B_1, B_2, B_3, B_4, B_5, and B_6.

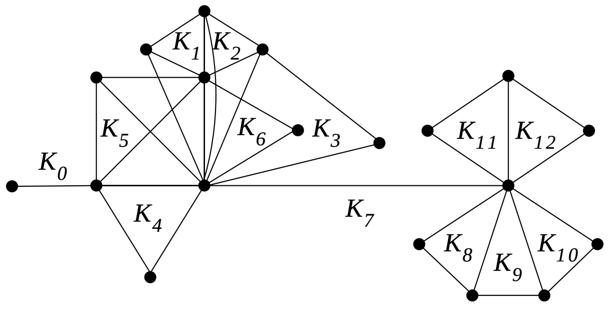


Figure 2: A chordal graph G.

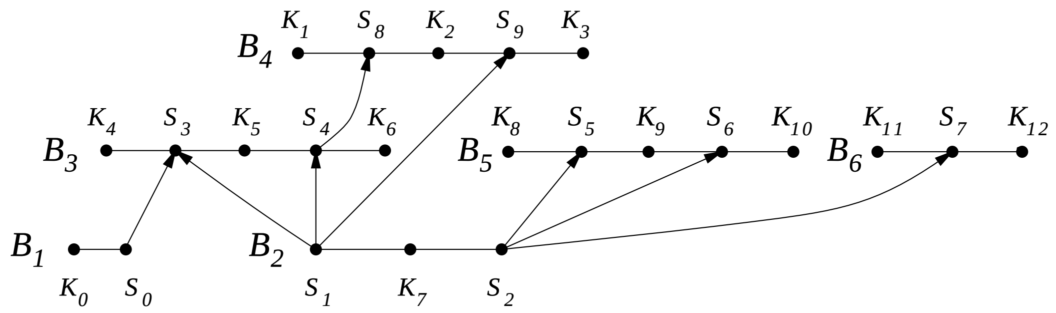


Figure 3: The clique-separator graph $\mathcal{G}$ of the chordal graph in Figure 2.

The graph $\mathcal{G}^c$ is obtained from $\mathcal{G}$ by contracting each box into a single node and replacing multiple arcs by a single arc. We will denote a box of $\mathcal{G}$ and the corresponding node in $\mathcal{G}^c$ by the same variable. We will show that $\mathcal{G}^c$ is a directed acyclic graph, which means $\mathcal{G}^c$ has a topological sort σ. Given σ, we will assume the boxes of $\mathcal{G}$ are indexed so that $\sigma = (B_1, B_2, \ldots, B_{b(\mathcal{G})})$, where $b(\mathcal{G})$ is the number of boxes of $\mathcal{G}$. Let $\mathcal{G}(\sigma, i)$ be the subgraph of $\mathcal{G}$ induced by the nodes in $B_i, B_{i+1}, \ldots, B_{b(\mathcal{G})}$. We will write $\mathcal{G}(\sigma, i)$ as $\mathcal{G}(i)$ if the topological sort σ is clear and we will refer to σ as a topological sort of $\mathcal{G}$. In Figure 3, $\sigma = (B_1, B_2, B_3, B_4, B_5, B_6)$ is a topological sort of $\mathcal{G}$.

For a separator node S of $\mathcal{G}$, let $Preds(S) = \{S' \mid S' = S \text{ or } S' \rightsquigarrow S\}$. We say S *divides* nodes N and N' if N and N' are contained in the same connected component of $\mathcal{G}$ and in different connected components of $\mathcal{G} - Preds(S)$. In Figure 3, S_4 divides K_5 and K_7 and S_5 divides K_8 and K_9. We use "separates" when referring to G and "divides" when referring to $\mathcal{G}$.

3.2 The Main Theorem

Theorem 3 ([Iba06]) *Let G be a chordal graph with clique-separator graph $\mathcal{G}$ and let $\mathcal{G}$ have topological sort σ. Let S be a separator node of $\mathcal{G}$.*

1. *$G - S$ has connected components $G_1, G_2, \ldots, G_k$ and $\mathcal{G} - Preds(S)$ has connected components $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$ such that $k > 1$ and for every $1 \le i \le k$, $V(G_i) = V(\mathcal{G}_i) - S$.*

2. *Every box is a tree with the clique intersection property and the set of its separator nodes is an antichain. $\mathcal{G}^c$ is a directed acyclic graph.*

3. *For every $1 \le i < j \le b(\mathcal{G})$, if S divides nodes N and N' in $\mathcal{G}(j)$, then S divides N and N' in $\mathcal{G}(i)$.*

4. *For every $1 \le i < j \le b(\mathcal{G})$ and every connected component $\mathcal{H}$ of $\mathcal{G}(j)$, at most one separator node of box B_i is a predecessor of any node in $\mathcal{H}$.*

5. *For every $1 \le j \le b(\mathcal{G})$ and every connected component $\mathcal{H}$ of $\mathcal{G}(j)$, $G[V(\mathcal{H})]$ is a connected chordal graph with clique-separator graph $\mathcal{H}$.*

We extend the definition of divides to sets of nodes $\mathcal{S}$ and $\mathcal{S}'$ of $\mathcal{G}$ as follows: *S divides $\mathcal{S}$* and $\mathcal{S}'$ if $\mathcal{S} - Preds(S)$ and $\mathcal{S}' - Preds(S)$ are contained in the same connected component of $\mathcal{G}$ and in different connected components of $\mathcal{G} - Preds(S)$. We extend the definition of divides to subgraphs $\mathcal{H}$ and $\mathcal{H}'$ of $\mathcal{G}$ in the same way. We need one more definition: *S strongly divides* nodes N and N' if $S \rightsquigarrow N$ and $S \rightsquigarrow N'$ and S divides N and N'. In Figure 3, S_4 divides K_5 and K_7 but does not strongly divide K_5 and K_7, whereas S_4 strongly divides K_1 and K_5.

Corollary 4 ([Iba06]) *Let G be a chordal graph with clique-separator graph $\mathcal{G}$. Let S be a separator node of $\mathcal{G}$.*

1. *Let N and N' be nodes of $\mathcal{G}$. Then S divides N and N' if and only if S separates $N - S$ and $N' - S$.*

   Let $\mathcal{S}$ and $\mathcal{S}'$ be sets of nodes of $\mathcal{G}$. Then S divides $\mathcal{S}$ and $\mathcal{S}'$ if and only if S separates $V(\mathcal{S}) - S$ and $V(\mathcal{S}') - S$.

   Let $\mathcal{H}$ and $\mathcal{H}'$ be subgraphs of $\mathcal{G}$. Then S divides $\mathcal{H}$ and $\mathcal{H}'$ if and only if S separates $V(\mathcal{H}) - S$ and $V(\mathcal{H}') - S$.

2. If S has distinct neighbors N and N', or S has neighbor N and successor N', then S strongly divides N and N'.

3. S strongly divides nodes N and N' if and only if $S = N \cap N'$ and S separates $N - S$ and $N' - S$.

4. S strongly divides clique nodes K and K' if and only if $S = K \cap K'$ for some edge $\{K, K'\}$ in a clique tree of G. There are clique nodes that satisfy these statements.

3.3 The clique-separator graph of an interval graph

We now review properties of the clique-separator graph $\mathcal{G}$ when G is an interval graph. A path P on a set of nodes of $\mathcal{G}$ (not necessarily a subgraph of $\mathcal{G}$) is a *c.i.p.* path if P has the clique intersection property. A graph G is an interval graph if and only if G has a *clique path* [FG65], which in our terminology is a c.i.p. path on the clique nodes of $\mathcal{G}$. It is straightforward to show that this characterization implies the following characterization.

Observation 5 *Let G be a chordal graph with clique-separator graph $\mathcal{G}$. Then G is an interval graph if and only if there is a c.i.p. path on the nodes of $\mathcal{G}$.*

A path P on a set of nodes of $\mathcal{G}$ (not necessarily a subgraph of $\mathcal{G}$) is *constraining* if P is the unique c.i.p. path on the set of nodes in P. For paths P and Q on sets of nodes of $\mathcal{G}$, P is a *subsequence* of Q if, when the paths are viewed as sequences, P or its reverse is a subsequence of Q. Observe that a constraining path P is a subsequence of every c.i.p. path that contains the nodes of P. A *leaf* clique node is a clique node that has degree 1, or equivalently, a clique node that is a leaf of some box. The *body* of a box B, denoted *body*(B), is the subgraph of B obtained by deleting the leaf clique nodes of B. The following theorem summarizes the main properties of $\mathcal{G}$ when G is an interval graph.

Theorem 6 ([Iba06]) *Let G be an interval graph with clique-separator graph $\mathcal{G}$. Then for every box B, body(B) is a constraining path whose endpoints are separator nodes. Moreover, every separator node has at most two immediate predecessors and there are $O(n)$ nodes, edges and arcs in $\mathcal{G}$.*

4 The train tree

In this section, we present the train tree of an interval graph and describe its properties. Since the train tree is based on the PQ-tree, we first review the PQ-tree. For a rooted tree T, *leaves*(T) is the set of leaves of T and *path*(T) is the left to right sequence of leaves of T. For a node x of T, *leaves*(x) and *path*(x) are defined similarly for *subtree*$_T(x)$, the subtree of T rooted at x.

4.1 PQ-trees

Booth and Lueker invented the PQ-tree, which represents a set of permutations, for their algorithms to recognize interval graphs and planar graphs [BL76]. Booth and Lueker constructed a PQ-tree using a set of templates, but we do not use templates in our algorithms.

A *PQ-tree* is a rooted tree where the leaves are the elements of a set and each internal node is either a *P-node* or a *Q-node*. A P-node is drawn as a circle and a Q-node is drawn as a rectangle. Two PQ-trees T and T' are *equivalent*, denoted $T \equiv T'$, if T can be transformed into T' by any sequence of the following operations: (1) arbitrarily permute the children of a P-node, (2) reverse the order of the children of a Q-node. A PQ-tree T represents the set of permutations $\{path(T') \mid T' \equiv T\}$ of $leaves(T)$ and a node x of T represents the set of permutations $\{path(T') \mid T' \equiv subtree_T(x)\}$ of $leaves(x)$.

A PQ-tree is *proper* if every P-node has at least two children and every Q-node has at least three children. A proper PQ-tree is unique up to equivalence: if T and T' are proper PQ-trees that represent the same set of permutations, then $T \equiv T'$ [BL76]. Figure 4 shows a proper PQ-tree that represents 4 permutations of $\{1, 2, 3, 4\}$: $(1, 2, 3, 4)$, $(1, 3, 2, 4)$, $(4, 2, 3, 1)$, $(4, 3, 2, 1)$.

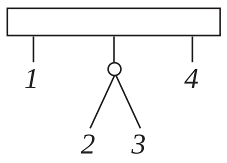


Figure 4: A proper PQ-tree. The circle is a P-node and the rectangle is a Q-node.

For a PQ-tree T with node x, we view $path(T)$ and $path(x)$ as paths. Consequently, T represents a set of paths on $leaves(T)$, x represents a set of paths on $leaves(x)$, and we can refer to the endpoints of $path(T)$ and the endpoints of $path(x)$.

4.2 Train trees

Let G be a chordal graph with clique-separator graph $\mathcal{G}$ and let $\mathcal{G}$ have connected subgraph $\mathcal{H}$. A *train tree* of $\mathcal{H}$ is a proper PQ-tree T such that $leaves(T)$ is the set of nodes of $\mathcal{H}$ and each node x of T represents all c.i.p. paths on $leaves(x)$. Then $path(T)$ is a c.i.p. path on the nodes of $\mathcal{H}$. Figure 5 shows a train tree of the clique-separator graph in Figure 1b.

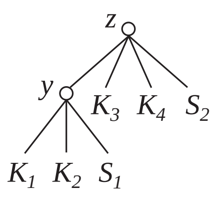


Figure 5: A train tree of the clique-separator graph in Figure 1b.

Let S be any separator node of $\mathcal{G}$ (not necessarily in $\mathcal{H}$) and let x be an internal node of T. If S is contained in both endpoints of $path(x)$, then S *meets* x, and if S is contained in exactly one endpoint of $path(x)$, then S *semi-meets* x. If S meets $root(T)$, then S *meets*

T, and if S semi-meets $root(T)$, then S *semi-meets* T. Since $path(x)$ is a c.i.p. path, the following three statements are equivalent: S meets x, $S \subset N$ for every $N \in leaves(x) - \{S\}$, $S \rightsquigarrow N$ for every $N \in leaves(x) - \{S\}$. In Figure 5, for example, S_2 meets y and z.

We will later show that in any train tree, every P-node y has exactly one separator node child S_y and y is closely associated with S_y. An *offspring* of a Q-node x is a child of x that is a clique node or separator node, or a grandchild of x that is the separator node S_y of a P-node child y of x. Notice that the offspring of a Q-node are leaves of T. Figure 6 shows an interval graph, its clique-separator graph, and its train tree T, where S_1 and S_2 have size 3, S_3, S_4, and S_5 have size 2, and S_6, S_7, S_8, and S_9 have size 1. The offspring of $root(T)$ are $S_6, S_7, K_{11}, S_8, K_{12}$, and S_9, and the offspring of the Q-node child of $root(T)$ are S_1, K_2, and S_2.

We need a bit more notation. If y is a leaf of T and y is a separator node, then S_y denotes this separator node. If y is a P-node of T, then S_y denotes its separator node child, as before. Then whether y is a separator node or P-node, S_y is a separator node.

We will show that any train tree T of $\mathcal{H}$ has the following properties, which can be verified on Figure 6.

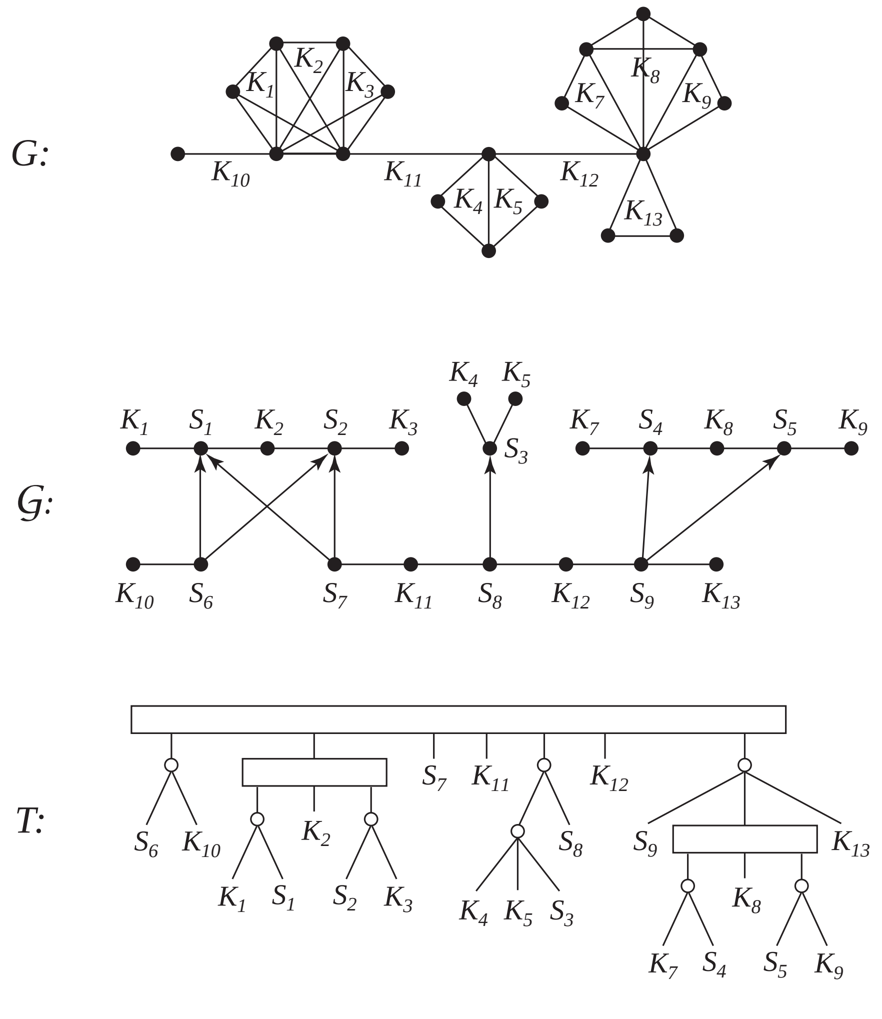


Figure 6: An interval graph G, its clique-separator graph $\mathcal{G}$, and a train tree T of $\mathcal{G}$.

Property 1. For every P-node x, S_x meets x and the following holds.

a. S_x divides $leaves(y)$ and $leaves(y')$ in $\mathcal{H}$ for any other children y and y' of x.

b. S_x divides $leaves(x)$ and $leaves(T) - leaves(x)$ in $\mathcal{H}$.

Property 2. For every Q-node x with children $(y_1, y_2, \ldots, y_k)$, the following holds.

a. If any y_i is a separator node or P-node, then S_{y_i} divides $leaves(y_1) \cup \cdots \cup leaves(y_{i-1})$ and $leaves(y_{i+1}) \cup \cdots \cup leaves(y_k)$ in $\mathcal{H}$.

b. If any y_i is a Q-node, then y_{i-1} and y_{i+1} are separator nodes or P-nodes and y_i has *meeting siblings* $S_{y_{i-1}}$ and $S_{y_{i+1}}$ that meet y_i and that are incomparable, where $1 < i < k$.

c. y_1 and y_k are P-nodes and no separator node in $leaves(x)$ is a predecessor of S_{y_1} or S_{y_k}.

d. If x has consecutive offspring N and N', then $N \subset N'$ or $N \supset N'$.

Properties 1 and 2 imply various train tree properties that we will use to prove the correctness of the algorithms. We present these properties in the rest of this section. In each result, the interval graph H is connected so that its clique-separator graph $\mathcal{H}$ is connected. (If $\mathcal{H}$ is disconnected, then the root of T must be a P-node r whose children are the roots of the train trees of the connected components of $\mathcal{H}$. But no separator node S_r meets T and thus Property 1 cannot hold.)

In the next lemma, part 2 holds for any clique C, whether or not C is maximal.

Lemma 7 *Let H be a connected interval graph with clique-separator graph $\mathcal{H}$ and let $\mathcal{H}$ have train tree T with Properties 1 and 2.*

1. *For any two consecutive nodes N_1 and N_2 of path(T), there is a separator node $S \in leaves(T)$ such that $S = N_1 \cap N_2$.*

2. *For any clique C of H, if separator nodes S_1 and S_2 are minimal elements of the set $\{N \mid N \in leaves(T) \text{ and } C \subseteq N\}$, then S_1 and S_2 are offspring of the same Q-node.*

Proof Let x be a node of T with consecutive children y_1 and y_2 such that $N_1 \in leaves(y_1)$ and $N_2 \in leaves(y_2)$. We claim that if x is a P-node, then $S_x = N_1 \cap N_2$, and if x is a Q-node, then x has a separator node offspring S such that $S = N_1 \cap N_2$. We will prove the claim and then use it to prove the lemma.

Suppose x is a P-node. Then $S_x \in leaves(x)$ may be N_1 or N_2. We use Property 1 in both of the following cases. If $S_x = N_1$ or $S_x = N_2$, then $N_1 \subset N_2$ or $N_2 \subset N_1$, respectively, and so $S_x = N_1 \cap N_2$. If $S_x \neq N_1$ and $S_x \neq N_2$, then $S_x \rightsquigarrow N_1$ and $S_x \rightsquigarrow N_2$ and S_x divides N_1 and N_2. Then by definition, S_x strongly divides N_1 and N_2. By Corollary 4-3, $S_x = N_1 \cap N_2$.

Suppose x is a Q-node. Then y_1 and y_2 are not both Q-nodes by Property 2b. Assume exactly one of y_1, y_2 is a Q-node, say y_2 (Figure 7). Then y_2 has meeting sibling S_{y_1}. If $S_{y_1} = N_1$, then $N_1 \subset N_2$ and so $S_{y_1} = N_1 \cap N_2$. If $S_{y_1} \neq N_1$, then y_1 is a P-node and so S_{y_1} divides N_1 and N_2. Since S_{y_1} meets y_1 and y_2, S_{y_1} strongly divides N_1 and N_2. Then by Corollary 4-3, $S_{y_1} = N_1 \cap N_2$. Assume neither of y_1, y_2 is a Q-node. Then each of y_1, y_2 is a P-node or separator node or clique node; both cannot be clique nodes by Property 2d. Suppose y_1 and y_2 are both P-nodes. Since S_{y_1} and S_{y_2} are consecutive offspring of x, either $S_{y_1} \subset S_{y_2}$ or $S_{y_1} \supset S_{y_2}$, say $S_{y_1} \subset S_{y_2}$. If $S_{y_1} = N_1$, then $N_1 \subset N_2$ and so $S_{y_1} = N_1 \cap N_2$. If $S_{y_1} \neq N_1$, then S_{y_1} is a P-node and as before, it follows that $S_{y_1} = N_1 \cap N_2$. The other cases for y_1 and y_2 are similar.

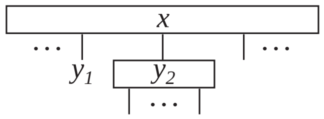


Figure 7: Proof of Lemma 7.

1. Let x be the lowest common ancestor of N_1 and N_2. Then we are done by the claim.

2. Let x be the lowest common ancestor of S_1 and S_2, where S_1 and S_2 are minimal elements of the specified set. Suppose x is a P-node. By the claim, $S_x = S_1 \cap S_2$. Then $C \subseteq S_1 \cap S_2 = S_x$. Since S_x cannot equal both S_1 and S_2, suppose $S_x \neq S_1$. Then $S_x \subset S_1$ and thus S_1 is not a minimal element of the given set, a contradiction. Hence, x must be a Q-node. Let y_1 and y_2 be the children of x such that $S_1 \in \mathit{leaves}(y_1)$ and $S_2 \in \mathit{leaves}(y_2)$. If y_1 and y_2 are consecutive children of x, then by the claim, x has a separator node offspring S such that $S = S_1 \cap S_2$. Then $C \subseteq S_1 \cap S_2 = S$ and as before, we have a contradiction. Hence, y_1 and y_2 are not consecutive children of x.

Suppose at least one of y_1, y_2 is a Q-node, say y_2. Let S be the meeting sibling of y_2 between y_1 and y_2. By Property 2a, S divides S_1 and S_2 and thus S separates $S_1 - S$ and $S_2 - S$. This implies $(S_1 \cap S_2) \subseteq S$ and thus $C \subseteq S$. But since S is a meeting sibling of y_2, $S \subset S_2$. Then S_2 is not a minimal element of the given set, a contradiction. Hence, neither of y_1, y_2 is a Q-node. Now S_1 and S_2 are descendants of x. Suppose at least one of S_1, S_2 is not an offspring of x, say S_1. Then y_1 is a P-node and $S_{y_1} \neq S_1$. Then S_{y_1} divides S_1 and S_2 and thus S_{y_1} separates $S_1 - S_{y_1}$ and $S_2 - S_{y_1}$. This implies $(S_1 \cap S_2) \subseteq S_{y_1}$ and thus $C \subseteq S_{y_1} \subset S_1$. Then S_1 is not a minimal element of the given set, a contradiction. Hence, S_1 and S_2 are both offspring of x. □

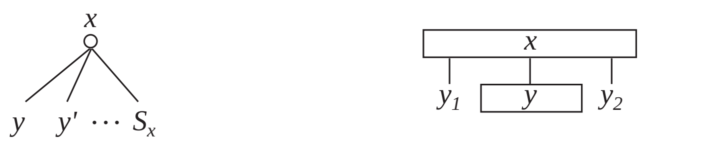
Intuitively, the next lemma shows that there are no "upward" arcs in a train tree.

Lemma 8 *Let H be a connected interval graph with clique-separator graph $\mathcal{H}$ and let $\mathcal{H}$ have train tree T with Properties 1 and 2.*

1. *For every P-node x with a child y, there are no arcs in $\mathcal{H}$ from nodes in $\mathit{leaves}(y)$ to nodes in $\mathit{leaves}(T) - \mathit{leaves}(y)$ and vice versa, except for arcs from S_x to nodes in $\mathit{leaves}(y)$.*

2. *For every Q-node x with a Q-node child y, there are no arcs in $\mathcal{H}$ from nodes in $\mathit{leaves}(y)$ to nodes in $\mathit{leaves}(T) - \mathit{leaves}(y)$ and vice versa, except for arcs from the meeting siblings of y to nodes in $\mathit{leaves}(y)$.*

Proof



a. P-node x has child y b. Q-node x has Q-node child y

Figure 8: Proof of Lemma 8.

1. Let P-node x have child y (Figure 8a). By Property 1a, there are no arcs from nodes in $leaves(y)$ to nodes in $leaves(y')$, where $y' \neq y$ is any other child of x. Since S_x meets y, there are no arcs from nodes in $leaves(y)$ to S_x. It remains to show there are no arcs from nodes in $leaves(y)$ to nodes in $leaves(T) - leaves(x)$ and vice versa. By Property 1b, S_x divides $leaves(y)$ and $leaves(T) - leaves(x)$. Since S_x meets y, $Preds(S_x) \subseteq leaves(T) - leaves(x)$. Then there are no arcs from nodes in $leaves(y)$ to nodes in $(leaves(T) - leaves(x)) - Preds(S_x)$ and vice versa. Since S_x meets y, there are no arcs from nodes in $leaves(y)$ to nodes in $Preds(S_x)$ and vice versa, so we are done.

2. Let Q-node x have Q-node child y (Figure 8b). By Property 2b, y has meeting siblings S_{y_1} and S_{y_2} that meet y and that are incomparable. An argument similar to the one in part 1, except using Property 2a, shows there are no arcs from nodes in $leaves(y)$ to nodes in $leaves(x) - leaves(y)$ and vice versa, except for arcs from S_{y_1} or S_{y_2} to nodes in $leaves(y)$. It remains to show there are no arcs from nodes in $leaves(y)$ to nodes in $leaves(T) - leaves(x)$ and vice versa. Let w be the parent of x. We claim there are no arcs from $leaves(y)$ to nodes in $leaves(w) - leaves(x)$ and vice versa. If w is a P-node, then part 1 proves the claim except for arcs from S_w to nodes of $leaves(y)$. But no such arcs exist because S_w is a predecessor of S_{y_1} and S_{y_2}. If w is a Q-node, then Property 2a proves the claim except for arcs from the meeting siblings of x to nodes in $leaves(y)$. But as before, no such arcs exist. Similar arguments for the other ancestors of x complete the proof. □

4.3 The Ordering Lemma

Let G be a chordal graph with clique-separator graph $\mathcal{G}$ and let $\mathcal{G}$ have topological sort σ. A *spanning set of train trees of* $\mathcal{G}(i)$ is a set of train trees with Properties 1 and 2 such that there is one train tree for each connected component of $\mathcal{G}(i)$, where $1 \leq i \leq b(\mathcal{G})$. If $\mathcal{G}$ has a spanning set of train trees, then by Observation 5, G is an interval graph. We will prove the converse by showing that if G is an interval graph, then the train tree algorithm in the next section constructs a spanning set of train trees of $\mathcal{G}$.

The train tree algorithm works incrementally by processing the boxes of $\mathcal{G}$ in reverse topological order. After building a spanning set of train trees $\mathcal{T}$ of $\mathcal{G}(i+1)$, the algorithm processes box B_i. If B_i contains a separator node S such that $S \rightarrow S'$, then S' is a leaf of some train tree $T \in \mathcal{T}$, which is a *successor tree* of S and B_i. The algorithm merges B_i and its successor trees to obtain a train tree T^{new} of the connected component of $\mathcal{G}(i)$ containing B_i. It adds T^{new} to $\mathcal{T}$ to obtain a spanning set of train trees of $\mathcal{G}(i)$.

The correctness proof for the train tree algorithm is inductive. A successor tree of B_i is a train tree T of some connected component $\mathcal{H}$ of $\mathcal{G}(i+1)$. By the Main Theorem-5,

$H = G[V(\mathcal{H})]$ is a connected chordal graph, and since it has a train tree, it is an interval graph. So H is a connected interval graph with clique-separator graph $\mathcal{H}$ and $\mathcal{H}$ has train tree T with Properties 1 and 2. Then we can apply the results from the previous subsection to $H, \mathcal{H}, T$. Thus, every successor tree of B_i has all the train tree properties we have presented. In the correctness proof, we use these properties to show that T^{new} is the required train tree.

The Ordering Lemma, shown below, specifies the successor trees that B_i can have and also specifies the general structure of T^{new}. A box is *short* if it contains exactly one separator node and *long* if it contains two or more separator nodes. In Figure 3, for example, boxes B_1 and B_6 are short and boxes B_2, B_3, B_4, and B_5 are long. The *outer nodes* of box B are the endpoints of *body*(B), which are separator nodes, and the *inner nodes* of B are the remaining separator nodes of B. For a path P containing nodes N and N', $P(N, N')$ denotes the N–N' subpath of P. In this paper, *disjoint* means *node-disjoint*.

Lemma 9 *Let G be a chordal graph with clique-separator graph $\mathcal{G}$ and let $\mathcal{G}$ have topological sort σ. Let $\mathcal{T}$ be a spanning set of train trees of $\mathcal{G}(i+1)$, where $1 \leq i < b(\mathcal{G})$.*

1. *Suppose B_i has successor trees T and T'. Let P be any c.i.p. path containing the nodes in B_i and leaves$(T) \cup$ leaves(T'). Then leaves(T) and leaves(T') are contained in disjoint subpaths of P and no node of B_i is between any two nodes of leaves(T) in P.*

2. *Suppose B_i is a long box with inner node S and suppose S has successor tree T. Let P be any c.i.p. path containing the nodes in B_i and leaves(T). Let K_1 and K_2 be the neighbors of S on body(B_i). Then every leaf clique node adjacent to S and every node in leaves(T) is on $P(K_1, K_2)$. Furthermore, S meets T.*

3. *Suppose B_i is a long box with outer node S and suppose S has successor tree T. Let P be any c.i.p. path containing the nodes in B_i and leaves(T). Let K be the neighbor of S on body(B_i) and N be the endpoint of P such that $P(K, N)$ contains S. Then every leaf clique node adjacent to S and every node in leaves(T) is on $P(K, N)$. Furthermore, S meets T or semi-meets some $T' \equiv T$. In the latter case, the nodes in B_i and the nodes in leaves(T) are in disjoint subpaths of P.*

4. *Suppose B_i is a box with outer node S. If B_i is a long box, then S semi-meets at most one successor tree. If B_i is a short box, then S semi-meets at most two successor trees.*

Proof Suppose B_i has a successor tree T. Let P be any c.i.p. path containing the nodes in B_i and *leaves*(T). Since *body*(B_i) is a constraining path by Theorem 6, *body*(B_i) is a subsequence of P. Since T represents every c.i.p. path on *leaves*(T), there exists $\bar{T} \equiv T$ such that *path*$(\bar{T})$ is a subsequence of P. Since T can be transformed into $\bar{T}$, for simplicity we assume henceforth that *path*(T) is a subsequence of P.

1. Suppose *leaves*(T) and *leaves*(T') are not contained in disjoint subpaths of P. Then some node N' of *path*(T') appears between two consecutive nodes of *path*(T) in P. Then by Lemma 7-1, some separator node $N \in$ *leaves*(T) is contained in N' and thus $N \rightsquigarrow N'$ in $\mathcal{G}$. Then there is a directed path from a node in *leaves*(T) to a node in *leaves*(T') in $\mathcal{G}(i+1)$. But *leaves*(T) and *leaves*(T') are the nodes of different connected components of $\mathcal{G}(i+1)$, a contradiction. Suppose some node N'' of B_i appears between two nodes of *leaves*(T) in P.

Then a similar argument shows that there is a directed path from a node in $leaves(T)$ to a node in B_i. But σ is a topological sort, a contradiction.

2. Since B_i is long and S is an inner node, $body(B_i)$ has a subpath (S_1, K_1, S, K_2, S_2). Suppose K^l is a leaf clique node adjacent to S. If K^l is not on $P(S_1, S_2)$ or K^l is on $P(S_1, K_1)$ or $P(K_2, S_2)$, then one separator node of B_i is contained in another, contradicting the Main Theorem-2. Thus, K^l is on $P(K_1, K_2)$.

Suppose S has successor tree T and S does not meet T. Then $path(T)$ contains nodes N and N' such that $S \subset N$ and $S \not\subset N'$. Now N is on $P(S_1, S_2)$, or else $S \subset S_1$ or $S \subset S_2$, contradicting the Main Theorem-1. If N is on $P(S_1, K_1)$ or $P(K_2, S_2)$, then S_1 or S_2 is contained in N, contradicting the Main Theorem-4. Thus, N is on $P(K_1, K_2)$, which means (S_1, K_1, N, K_2, S_2) is a subsequence of P. Now N' is not on $P(S_1, S_2)$, or else S_1, S_2, or S is contained in N', a contradiction in every case. Then at least two nodes of B_i are between N and N', contradicting part 1. Hence, S meets T. We showed that $S \subset N$ implies N is on $P(K_1, K_2)$, so we are done.

3. Since B_i is long and S is an outer node, K has a neighbor $S' \neq S$ in B_i. Then (S', K, S, N) is a subsequence of P. Suppose K^l is a leaf clique node adjacent to S. If K^l is not on $P(S', N)$ or K^l is on $P(S', K)$, then one separator node of B_i is contained in another, contradicting the Main Theorem-2. Thus, K^l is on $P(K, N)$.

Suppose S has successor tree T. Let N_1 and N_2 be the endpoints of $path(T)$. By part 1, S is not on $P(N_1, N_2)$. Since S is contained in at least one node on $P(N_1, N_2)$, S is contained in N_1 or N_2 or both. Thus, S meets or semi-meets T. Now by part 1, K is not between any two nodes of $leaves(T)$ in P. This implies that if any node of $leaves(T)$ is not on $P(K, N)$, all nodes of $leaves(T)$ are not on $P(K, N)$, which leads to contradictions. Thus, every node of $leaves(T)$ is on $P(K, N)$. If S meets T, we are done. If S semi-meets T, then the nodes in B_i and at least one node of $leaves(T)$ are in disjoint subpaths of P. Then part 1 implies that the nodes in B_i and the nodes in $leaves(T)$ are in disjoint subpaths of P.

4. Suppose B_i is long and S semi-meets successor trees T_1 and T_2. Let $S' \neq S$ be the other outer node of B_i. By part 1, the nodes in $leaves(T_1), leaves(T_2)$, and $body(B_i)$ are contained in disjoint subpaths P_1, P_2, and P_3 of P, respectively. If P contains P_1, P_2, P_3 in this order, then S is contained in every node of P_2 and so S meets T_2, a contradiction. If P contains P_1, P_3, P_2 in this order, then S is contained in S', contradicting the Main Theorem-1. The other cases are symmetric. Hence, S semi-meets at most one successor tree. A similar argument shows that if B_i is short, then S semi-meets at most two successor trees. □

5 Computing the train tree

In this section, we present the train tree algorithm that constructs a spanning set of train trees $\mathcal{T}$ of $\mathcal{G}$ or rejects because G is not an interval graph. The algorithm processes the boxes of $\mathcal{G}$ in reverse topological order and merges each box of $\mathcal{G}$ and its successor trees. If G is connected, then $\mathcal{G}$ is connected and the algorithm finishes with one train tree.

Let G be a chordal graph with clique-separator graph $\mathcal{G}$ and let $\mathcal{G}$ have topological sort σ. The algorithm builds a spanning set of train trees of $\mathcal{G}(b(\mathcal{G})), \mathcal{G}(b(\mathcal{G})-1), \ldots, \mathcal{G}(1) = \mathcal{G}$ in this order. After building a spanning set of train trees $\mathcal{T}$ of $\mathcal{G}(i+1)$, the algorithm processes box B_i. For a separator node S of B_i, let $\mathcal{T}_S$ be the set of its successor trees and let $\mathcal{T}_S^m$

and $\mathcal{T}_S^{sm}$ be the sets of successor trees that S meets and semi-meets, respectively. Then $\mathcal{T}_S^m \cup \mathcal{T}_S^{sm} \subseteq \mathcal{T}_S \subseteq \mathcal{T}$. By the Main Theorem-4, $\mathcal{T}_S \cap \mathcal{T}_{S'} = \emptyset$ for distinct separator nodes S and S' of B_i. The algorithm processes B_i in three main steps, as follows.

Step 1 calls the subroutine $Scan(S)$ for each separator node S of B_i. $Scan(S)$ computes $\mathcal{T}_S^m$ and $\mathcal{T}_S^{sm}$ and rejects if any of the following holds: (a) $\mathcal{T}_S^m \cup \mathcal{T}_S^{sm} \neq \mathcal{T}_S$, (b) $|\mathcal{T}_S^{sm}| > 0$ and S is an inner node of B_i, (c) $|\mathcal{T}_S^{sm}| > 1$ and S is an outer node of B_i and B_i is a long box, (d) $|\mathcal{T}_S^{sm}| > 2$. If it rejects, then G is not an interval graph by the Ordering Lemma. This step is the only one that can reject. Then Step 1 creates a train tree T^{new} such that $path(T^{new})$ is $body(B_i)$.

Step 2 merges the $\mathcal{T}_S^{sm}$ trees into T^{new}. Let S_1 and S_2 be the outer node(s) of B_i, where $S_1 = S_2$ if B_i is a short box. If $T_1 \in \mathcal{T}_{S_1}^{sm}$ and $T_2 \in \mathcal{T}_{S_2}^{sm}$, the subroutine $SemiMeets$ merges T_1 and T_2 into T^{new} so that $path(T^{new}) = path(T_1) + body(B_i) + path(T_2)$, where $+$ is concatenation.

Step 3 merges the $\mathcal{T}_S^m$ trees into T^{new}. For a separator node S of B_i, the subroutine $Meets$ replaces S with a P-node whose children are S, the leaf clique node(s) adjacent to S, and the root(s) of the tree(s) in $\mathcal{T}_S^m$. We will show that T^{new} is now a train tree of the connected component of $\mathcal{G}(i)$ containing B_i and that T^{new} has Properties 1 and 2. The algorithm adds T^{new} to $\mathcal{T}$ to obtain a spanning set of train trees of $\mathcal{G}(i)$. Hence, after all boxes of $\mathcal{G}$ are processed, $\mathcal{T}$ is a spanning set of train trees of $\mathcal{G}$.

In the train tree algorithm, Q-nodes are never split. Instead, Q-nodes are merged to form larger Q-nodes. So when the nodes in $body(B_i)$ become consecutive offspring of a Q-node, they remain consecutive offspring of some Q-node for the rest of the algorithm.

5.1 The data structure

Each node of $\mathcal{G}$ has a characteristic vector of the vertices it contains. Each clique node has pointers to its neighbors in $\mathcal{G}$. Each separator node has pointers to its immediate predecessors, immediate successors, and neighbors in $\mathcal{G}$. Each P-node has a linked list of its children and every child has a parent pointer. Each Q-node has an ordered linked list of its children and, to achieve the desired running time, only the first and last child has a parent pointer.

Every separator node S has the following attributes, which are used by the update operations. $components_{\mathcal{G}}(S)$ is the number of connected components of $\mathcal{G} - Preds(S)$ that contain a successor of S. By Theorem 1-1 and Corollary 4-4, $components_{\mathcal{G}}(S) \geq 2$. If S is an offspring of a Q-node x, then $left(S)$ and $right(S)$ are pointers to the leftmost and rightmost offspring N_l and N_r of x such that $S \subseteq N_l$ and $S \subseteq N_r$. If S is not an offspring of any Q-node, then $left(S) = right(S) = S$.

5.2 The train tree algorithm

The input is the clique-separator graph $\mathcal{G}$ of a chordal graph G. If $\mathcal{G}$ does not have the properties in Theorem 6, then reject because G is not an interval graph. Otherwise, compute the directed acyclic graph $\mathcal{G}^c$ and a topological sort σ of $\mathcal{G}^c$. Index the boxes so that the boxes with no successors in $\mathcal{G}^c$ are $B_j, B_{j+1}, \ldots, B_{b(\mathcal{G})}$, where $1 \leq j \leq b(\mathcal{G})$.

For each $k = j, j+1, \ldots, b(\mathcal{G})$, create a train tree as follows. If B_k is a short box with separator node S, create a P-node whose children are S and the leaf clique nodes adjacent to S. If B_k is a long box, create a Q-node whose children are the nodes in *body*(B_k) in the same order; then for every separator node S' in B_k adjacent to one or more leaf clique nodes, replace S' with a P-node whose children are S' and the leaf clique node(s) adjacent to S'. The set of these train trees is the initial $\mathcal{T}$. Figure 9 shows the initial $\mathcal{T}$ for the clique-separator graph in Figure 6.

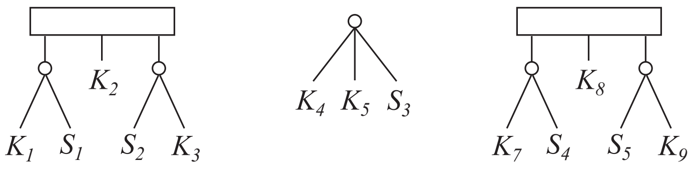


Figure 9: The initial $\mathcal{T}$ for the clique-separator graph in Figure 6.

For each $i = j-1, \ldots, 2, 1$ in decreasing order, process B_i with the following three steps.

Step 1. [Create $\mathcal{T}_S^{sm}$ and $\mathcal{T}_S^m$.] For each separator node S of B_i, call $Scan(S)$ to compute $\mathcal{T}_S^{sm}$ and $\mathcal{T}_S^m$. Create the following train tree T^{new}. If B_i is a short box with separator node S and $\mathcal{T}_S^{sm} = \emptyset$, then T^{new} is S. Otherwise, T^{new} is a Q-node whose children are the nodes in *body*(B_i) in the same order.

Step 2. [Merge $\mathcal{T}_S^{sm}$ trees.] For each separator node S in B_i, if $\mathcal{T}_S^{sm} \neq \emptyset$, then for each $T \in \mathcal{T}_S^{sm}$, call $SemiMeets(S, T)$.

Step 3. [Merge $\mathcal{T}_S^m$ trees.] For each separator node S in B_i, if $\mathcal{T}_S^m \neq \emptyset$ or S is adjacent to one or more leaf clique nodes, then call $Meets(S)$. Add T^{new} to $\mathcal{T}$.

$Scan(S)$:

$Scan(S)$ computes $\mathcal{T}_S^{sm}$ and $\mathcal{T}_S^m$ in three steps. For each successor tree T of S, the first step marks the lowest common proper ancestor of the immediate successors of S in $leaves(T)$. (The immediate successors of S are in no particular order. By Lemma 7-2, if there are several immediate successors of S, they are offspring of the same Q-node of T.) For each marked node x, the second step decides whether S meets or semi-meets the train tree that contains x. The third step checks the rejection conditions and sets the attributes of S.

1. [Find LCAs.] For each immediate successor S' of S, mark the parent of S' in its train tree and if S' is an offspring of a Q-node, then mark the Q-node too. Then for each marked Q-node y in any train tree, if y has exactly one marked child and this child is a P-node, then unmark y, and otherwise, unmark all marked children of y.

2. [Does S meet or semi-meet tree?] For each marked node x in any train tree, do the following. If x is the root of its train tree T and S meets x (because x is a P-node or x is a Q-node with leftmost and rightmost children l and r such that $S \rightarrow S_l$ and $S \rightarrow S_r$), then add T to $\mathcal{T}_S^m$ and go to step 3. Otherwise, place x on the leftmost path

from $root(T)$ by following parent pointers from x to $root(T)$ and applying the PQ-tree operations permute and reverse. Reject if this is not possible, which means x has an ancestor y that is a child of a Q-node but is neither the leftmost nor the rightmost child. If x is a Q-node, then if necessary reverse x so that $S \to S_y$, where y is the leftmost child of x; reject if this is not possible. Add T to $\mathcal{T}_S^{sm}$. Reject if $|\mathcal{T}_S^{sm}| > 2$.

3. [Set attributes.] Reject if $|\mathcal{T}_S^{sm}| > 0$ and S is an inner node, or if $|\mathcal{T}_S^{sm}| > 1$ and S is an outer node of a long box. Set $components_{\mathcal{G}}(S) = degree(S) + |\mathcal{T}_S^{m}| + |\mathcal{T}_S^{sm}|$. If S has no neighbors that are internal nodes of B_i, then set $left(S) = S$ and $right(S) = S$. If S has exactly one such neighbor K, then set $left(S) = K$ and $right(S) = S$. If S has two such neighbors K and K', then set $left(S) = K$ and $right(S) = K'$.

$SemiMeets(S, T)$:

Let x_T be the lowest common proper ancestor of the immediate successors of S in $leaves(T)$. We assume that x_T is on the leftmost path from $root(T)$ and that the root r of T^{new} is a Q-node whose rightmost child is S. $SemiMeets$ merges T into T^{new} by following parent pointers from x_T and merging x_T and its ancestors into T^{new}.

1. [Merge x_T.] If S meets x_T, then make x_T into a child of root r (Figure 10). If S semi-meets x_T, then delete x_T, which must be a Q-node, and make the children of x_T into children of root r in the same order (Figure 11). Now S is to the immediate left of the new child(ren) of root r.

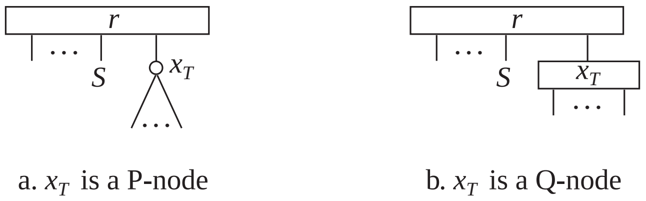


Figure 10: S meets x_T.

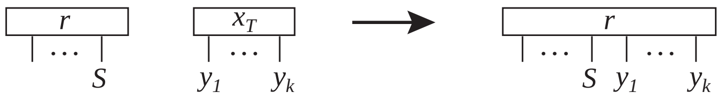


Figure 11: S semi-meets x_T.

2. [Merge ancestors of x_T.] If x_T was the root of T, then go to step 3. Otherwise, $x_0 = x_T$ had proper ancestors $x_1, x_2, \ldots, x_k$ in T, where each x_j is the leftmost child of x_{j+1} and $k \geq 1$. For each $j = 1, 2, \ldots, k$ in this order, do the following.

   If x_j is a P-node, then make x_j into a child of root r and if the only child of x_j is S_{x_j}, replace x_j with S_{x_j}. If x_j is a Q-node, then delete x_j and make the children of x_j into children of root r in the same order.

   Now x_{j-1} (or its former children) is to the immediate left of x_j (or its former children).

3. [Set $right(S)$.] If S met x_T and x_T was a P-node, then set $right(S) = right(S_{x_T})$. If S met x_T and x_T was a Q-node, then set $right(S) = S$. If S semi-met x_T, then set $right(S) = right(S')$, where S' was the rightmost offspring of x_T such that $S \to S'$.

4. [Set $left(S_{x_j})$.] For each x_j that was a P-node, set $left(S_{x_j}) = S'$, which is the leftmost offspring of root r that is to the right of S. (If S met x_T and x_T was a P-node, then $S' = S_{x_T}$. If S met x_T and x_T was a Q-node, then S' is the meeting sibling to the right of x_T. If S semi-met x_T, then S' is the leftmost offspring of x_T.)

$Meets(S)$:

Replace S with a P-node whose children are S, the leaf clique node(s) adjacent to S, and the root(s) of the tree(s) in $\mathcal{T}_S^m$.

Figure 12 shows the train tree algorithm applied to the clique-separator graph in Figure 3. Figure 12a shows the initial $\mathcal{T}$, which consists of B_4, B_5, and B_6. Figure 12b shows $\mathcal{T}$ after B_3 is processed by calling *Semi-Meets*. Figure 12c shows $\mathcal{T}$ after B_2 is processed by calling *Meets* twice. Figure 12d shows $\mathcal{T}$ after B_1 is processed by calling *Semi-Meets*. In that last call, S is S_0, T is the train tree in Figure 12c, x_T is the Q-node with six children, x_1 is the P-node with child S_1, and x_2 is $root(T)$. As examples of the attributes in the final $\mathcal{T}$, $components_{\mathcal{G}}(S_0) = 2$, $left(S_0) = S_0$, $right(S_0) = S_3$, and $components_{\mathcal{G}}(S_1) = 2$, $left(S_1) = S_3$, $right(S_1) = K_7$, and $components_{\mathcal{G}}(S_4) = 3$, $left(S_4) = K_5$, $right(S_4) = K_2$.

5.3 Correctness and complexity

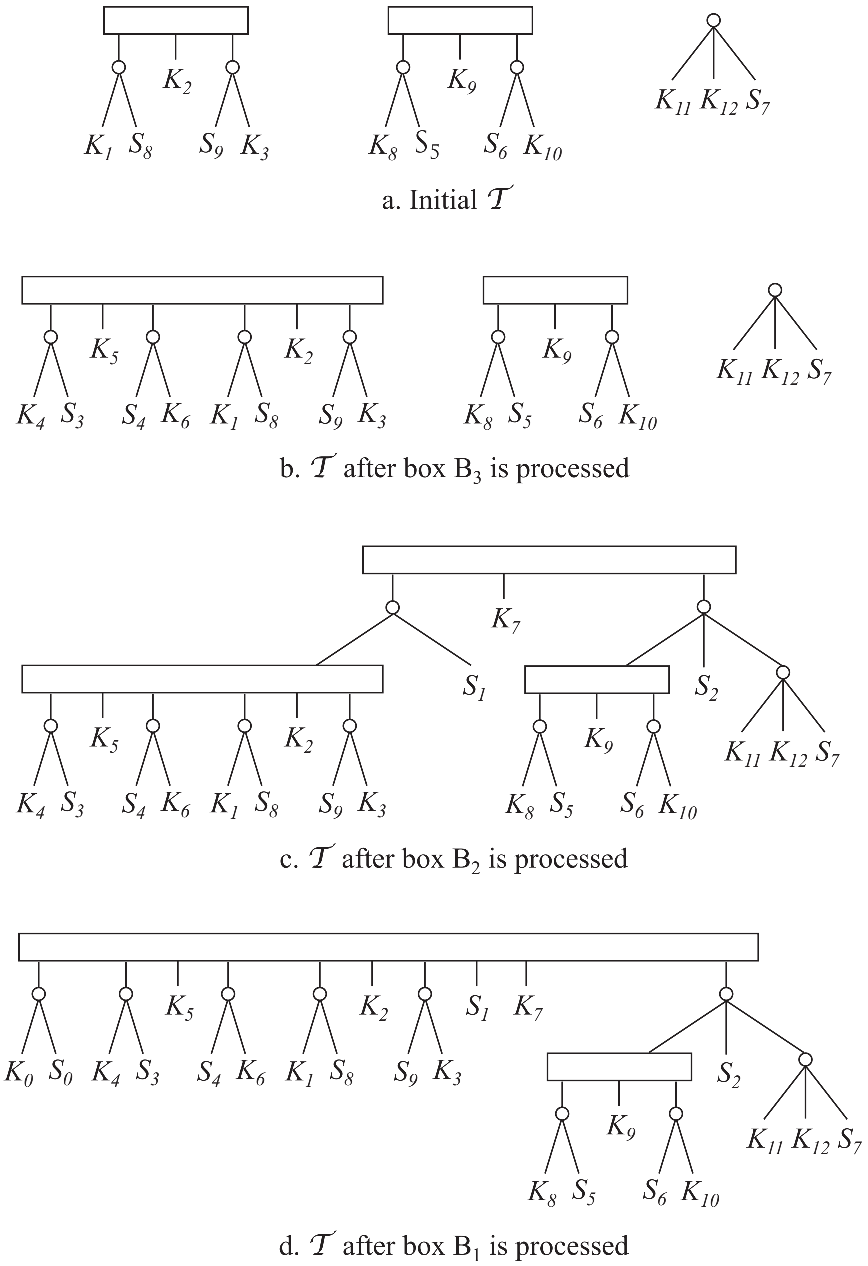
Lemma 10 *If Scan(S) rejects, then G is not an interval graph, and otherwise, Scan(S) correctly computes $\mathcal{T}_S^m$ and $\mathcal{T}_S^{sm}$. Furthermore, after step 1 of Scan(S), each successor tree T of S has a unique marked node, which is the lowest common proper ancestor of the immediate successors of S in leaves(T).*

Proof Consider step 1. Let T be any successor tree of S. If S has exactly one immediate successor S' in $leaves(T)$, then after step 1, the parent of S' (whether P-node or Q-node) is the unique marked node of T. Otherwise, S has at least two immediate successors S'_1 and S'_2 in $leaves(T)$. Since S'_1 and S'_2 are minimal elements of the set $\{N \mid N \in leaves(T) \text{ and } S \subseteq N\}$, S'_1 and S'_2 are offspring of the same Q-node y by Lemma 7-2. Then step 1 marks y and unmarks the marked children of y. Then T contains a unique marked node x, which is the lowest common proper ancestor of the immediate successors of S in $leaves(T)$.

Consider step 2. Let x be the marked node of some successor tree T. We examine whether S meets x. Suppose x is a P-node. Then by step 1, $S \to S_x$, and by Property 1, S meets x. Suppose x is a Q-node with leftmost and rightmost children l and r, which are P-nodes by Property 2c. Then S meets x if and only if S is a predecessor of every node in $leaves(x)$ if and only if $S \rightsquigarrow S_l$ and $S \rightsquigarrow S_r$ if and only if $S \to S_l$ and $S \to S_r$. The last equivalence follows by Property 2c. Hence, S meets x if and only if x is a P-node or x is a Q-node with leftmost and rightmost children l and r such that $S \to S_l$ and $S \to S_r$.

Now we examine whether S meets T. Suppose x is $root(T)$. Then S meets T if and only if S meets x, which step 2 decides correctly. Suppose x is not $root(T)$. We claim that at least one node in $leaves(T) - leaves(x)$ is not a successor of S, which implies S does not meet

T. Assume x is a Q-node. Then by Lemma 8, there are no arcs from nodes in $leaves(x)$ to nodes in $leaves(T) - leaves(x)$. Then every successor of S is contained in $leaves(x)$, which proves the claim. Assume x is a P-node. Then again Lemma 8 proves the claim, except in the case that the parent of x is a Q-node and $root(T)$. In that case, $root(T)$ has leftmost and rightmost children l and r and no separator node in $leaves(x)$ is a predecessor of S_l or S_r. Then at least one of S_l, S_r is not a successor of S, which proves the claim.



K_2
K_1 S_8 S_9 K_3
K_9
K_8 S_5 S_6 K_{10}
K_{11} K_{12} S_7
a. Initial $\mathcal{T}$
K_5 K_2
K_4 S_3 S_4 K_6 K_1 S_8 S_9 K_3
b. $\mathcal{T}$ after box B_3 is processed
K_7
S_1
S_2
c. $\mathcal{T}$ after box B_2 is processed
K_0 S_0
S_1 K_7
d. $\mathcal{T}$ after box B_1 is processed

Figure 12: The train tree algorithm applied to the clique-separator graph in Figure 3.

We have shown that step 2 adds T to $\mathcal{T}_S^m$ if and only if S meets T. Suppose step 2 does not add T to $\mathcal{T}_S^m$. If x cannot be placed on the leftmost path from $root(T)$, then x has an ancestor y (we may have $y = x$) that is a child of a Q-node z but is neither the leftmost nor the rightmost child of z. Since the immediate successors of S in $leaves(T)$ are contained in $leaves(y)$, Property 2c implies that S does not semi-meet z, and moreover, S does not semi-meet any $T' \equiv T$. Then G is not an interval graph by the Ordering Lemma-3. Thus, step 2 adds T to $\mathcal{T}_S^{sm}$ if and only if S semi-meets T. If $|\mathcal{T}_S^{sm}| > 2$, then G is not an interval graph by the Ordering Lemma-4. (We test whether $|\mathcal{T}_S^{sm}| > 2$ in step 2 so that it follows parent pointers for at most three successor trees of S.) Lastly, if step 3 rejects, then G is not an interval graph by the Ordering Lemma-2 or the Ordering Lemma-4. $\square$

Lemma 11 *Suppose the train tree algorithm makes a SemiMeets(S, T) call, where S is a separator node of B_i and T is a train tree of a connected component $\mathcal{H}$ of $\mathcal{G}(i+1)$. Let $\mathcal{C}^{semi}$ be the graph containing body(B_i) and $\mathcal{H}$ and the arcs from nodes of body(B_i) to nodes of $\mathcal{H}$.*

1. If a separator node divides two nodes in $\mathcal{H}$, then it divides these two nodes in $\mathcal{C}^{semi}$.
2. Let T^{semi} be the train tree resulting from the call. For every node w of T^{semi} that is neither the root nor a child of the root, $subtree_{T^{semi}}(w)$ is the same as $subtree_T(w)$.

Proof Since $\mathcal{H}$ is a connected component of $\mathcal{G}(i+1)$ and $\mathcal{C}^{semi}$ is a subgraph of a connected component of $\mathcal{G}(i)$, the Main Theorem-3 implies part 1. When $SemiMeets(S, T)$ runs, each $x_j, 0 \leq j \leq k$, is made a child of root r or x_j is a Q-node whose children are made children of root r. Part 2 follows. $\square$

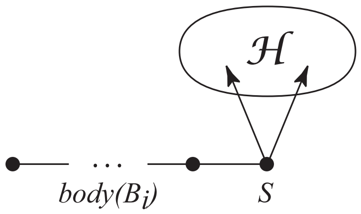


Figure 13: The graph $\mathcal{C}^{semi}$.

Theorem 12 *Let G be a chordal graph with clique-separator graph $\mathcal{G}$. Then G is an interval graph if and only if the train tree algorithm computes a spanning set of train trees of $\mathcal{G}$.*

Proof The $\Leftarrow$ direction follows by Observation 5 and it remains to prove the $\Rightarrow$ direction. For the rest of the proof, assume G is an interval graph.

The algorithm begins by creating a train tree for each box $B_k, k = j, j+1, \ldots, b(\mathcal{G})$, and these trees form the initial $\mathcal{T}$. If B_k is short, then any ordering of its nodes is a c.i.p. path. If B_k is long, then Theorem 6 states that $body(B_k)$ is a constraining path and the Ordering Lemma-2-3 specifies where the leaf clique nodes of B_k may appear in a c.i.p. path on the nodes of B_k. It remains to show that every train tree has Properties 1 and 2. Property 1 and Property 2a hold by Corollary 4-2. Property 2c holds by the Main Theorem-1. The other properties are trivial. Therefore, the initial $\mathcal{T}$ is a spanning set of train trees of $\mathcal{G}(j)$.

Suppose the algorithm has built a spanning set of train trees $\mathcal{T}$ of $\mathcal{G}(i+1), 1 \leq i \leq j-1$. Then the algorithm processes box B_i. If it rejects, it does so in Step 1 and by Lemma 10,

G is not an interval graph, a contradiction. Let $\mathcal{C}$ be the connected component of $\mathcal{G}(i)$ that contains B_i. The rest of the proof shows that the algorithm computes a train tree of $\mathcal{C}$ that has Properties 1 and 2. Therefore, when the algorithm adds this train tree to $\mathcal{T}$, it obtains a spanning set of train trees of $\mathcal{G}(i)$.

The algorithm processes B_i in three steps. Step 1 creates T^{new} with root r. Assume Step 2 makes no *SemiMeets* calls. Then Step 3 makes at least one call of *Meets* because B_i has at least one successor tree. After Step 3, we have the following. Suppose B_i is a short box with separator node S. Then root r is a P-node whose children are S, the leaf clique node(s) adjacent to S, and the roots of the successor tree(s) of S. By the Ordering Lemma-1, T^{new} is a train tree of $\mathcal{C}$. Suppose B_i is a long box. Then root r is a Q-node and for every separator node S of B_i, root r has child S or a P-node child whose children are S, the leaf clique node(s) adjacent to S, and the roots of the successor tree(s) of S. By Theorem 6 and the Ordering Lemma, T^{new} is a train tree of $\mathcal{C}$. In either case, we can readily show that T^{new} has Properties 1 and 2. Therefore, T^{new} is the required tree and we are done.

Assume Step 2 makes at least one *SemiMeets* call, one of at most two calls. Consider the first *SemiMeets*(S, T) call, where S is a separator node of B_i and T is a train tree of a connected component $\mathcal{H}$ of $\mathcal{G}(i+1)$. Let $\mathcal{C}^{semi}$ be the graph containing *body*(B_i) and $\mathcal{H}$ and the arcs from nodes of *body*(B_i) to nodes of $\mathcal{H}$ (Figure 13). By Corollary 4-2, S divides *body*(B_i) and $\mathcal{H}$ in $\mathcal{C}^{semi}$. In the rest of the proof, T denotes the train tree passed to the call and T^{semi} denotes the train tree resulting from the call. Here is a summary of the notation.

- S is a separator node of B_i and T is a train tree of a connected component $\mathcal{H}$ of $\mathcal{G}(i+1)$.
- T^{semi} is the train tree resulting from the *SemiMeets*(S, T) call.
- $\mathcal{C}^{semi}$ is the graph containing *body*(B_i) and $\mathcal{H}$ and the arcs from nodes of *body*(B_i) to nodes of $\mathcal{H}$.

The major part of the proof shows that T^{semi} is a train tree of $\mathcal{C}^{semi}$ and T^{semi} has Properties 1 and 2. We first show that T^{semi} is a proper PQ-tree, i.e., every P-node has at least two children and every Q-node has at least three children. By Lemma 11-2, it suffices to consider root r and its children. Consider *SemiMeets*(S, T). If S meets x_T, then x_T is not the root of T and so step 1 and step 2 add at least two children to root r. If S does not meet x_T, then x_T is a Q-node and so step 1 adds at least three children to root r. In step 2, if any x_j becomes a P-node whose only child is S_{x_j}, then x_j is replaced by S_{x_j}. Thus, T^{semi} is a proper PQ-tree.

Next we show that every internal node y of T^{semi} represents all c.i.p. paths on *leaves*(y). As before, it suffices to consider root r and its children. For any node w of T, *oldleaves*(w) denotes the set of leaves in *subtree*$_T(w)$, and if w is a node of T^{semi}, *leaves*(w) denotes the set of leaves in *subtree*$_{T^{semi}}(w)$. Consider the children of root r. Every child that is a node of *body*(B_i) is a leaf and every other child was x_j or a child of x_j in T, for some $0 \leq j \leq k$. Before *SemiMeets*(S, T) runs, each x_j represents all c.i.p. paths on *oldleaves*(x_j). After *SemiMeets*(S, T) runs, x_j is a child of root r (possibly with one fewer child) or x_j's children are children of root r. So for every child y of root r, y represents all c.i.p. paths on *leaves*(y).

Consider root r. We will show that as *SemiMeets*(S, T) runs, it maintains the invariant that root r represents all c.i.p. paths on *leaves*(r) such that the nodes in *body*(B_i) and the nodes in $leaves(r) \cap leaves(T)$ are contained in disjoint subpaths of the path. Initially $leaves(r)$ consists of the nodes in $body(B_i)$ and so $leaves(r) \cap leaves(T) = \emptyset$. Since $body(B_i)$ is a constraining path by Theorem 6, the invariant holds. After step 1, $leaves(r)$ consists of the nodes in $body(B_i)$ and $oldleaves(x_T)$. Since x_T represented every c.i.p. path on $oldleaves(x_T)$, the invariant holds after step 1.

Consider step 2. If x_T was the root of T before step 1, then S semi-met x_T and $leaves(r)$ now consists of the nodes in $body(B_i)$ and $leaves(T)$, so the invariant holds after step 2. Otherwise, $x_0 = x_T$ had proper ancestors $x_1, x_2, \ldots, x_k$ in T, $k \geq 1$. By Lemma 10, S was not contained in any node in $oldleaves(x_1) - oldleaves(x_T)$. Since x_1 represented every c.i.p. path on $oldleaves(x_1)$, it follows that after step 2 processes x_1, $leaves(r) \cap leaves(T) = oldleaves(x_1)$ and the invariant holds. This holds whether x_1 was a P-node or a Q-node. Similarly, after step 2 processes $x_i, 2 \leq i \leq k$, $leaves(r) \cap leaves(T) = oldleaves(x_i)$ and the invariant holds. Then after step 2, $leaves(r) \cap leaves(T) = oldleaves(x_k) = leaves(T)$ and the invariant holds.

Now in any c.i.p. path containing the nodes in B_i and $leaves(T)$, the nodes in B_i and the nodes in $leaves(T)$ are contained in disjoint subpaths of the path. This follows by the Ordering Lemma-3 if B_i is a long box and by the Ordering Lemma-1 if B_i is a short box. We conclude that after $SemiMeets(S, T)$ runs, root r represents all c.i.p. paths on $leaves(r)$. Therefore, T^{semi} is a train tree of $\mathcal{C}^{semi}$.

Next we show that T^{semi} has Properties 1 and 2 with an exception (Property 2c) that we will discuss. We write each property with the variables renamed appropriately.

Property 1: For every P-node y, S_y meets y and the following holds.
a. S_y divides $leaves(z)$ and $leaves(z')$ in $\mathcal{C}^{semi}$ for any children z and z' of y.
b. S_y divides $leaves(y)$ and $leaves(T^{semi}) - leaves(y)$ in $\mathcal{C}^{semi}$.

Let y be a P-node of T^{semi}. Then y was a P-node of T, although y may have one fewer child in T^{semi}. By Lemma 11-1, y satisfies Property 1a in T^{semi}, and moreover, S_y divides $leaves(y)$ and $leaves(T) - leaves(y)$ in $\mathcal{C}^{semi}$. To show that y satisfies Property 1b in T^{semi}, we must show that S_y divides $body(B_i)$ and $leaves(y)$ in $\mathcal{C}^{semi}$. Recall S divides $body(B_i)$ and $leaves(T)$ in $\mathcal{C}^{semi}$. Then whether $S \rightsquigarrow S_y$ or $S \not\rightsquigarrow S_y$, $body(B_i)$ and $leaves(y)$ are in different connected components of $\mathcal{C}^{semi} - Preds(S_y)$. Then y satisfies Property 1b in T^{semi}.

Property 2: For every Q-node y with children $(z_1, z_2, \ldots, z_k)$, the following holds.
a. If any z_i is a separator node or P-node, then S_{z_i} divides $leaves(z_1) \cup \cdots \cup leaves(z_{i-1})$ and $leaves(z_{i+1}) \cup \cdots \cup leaves(z_k)$ in $\mathcal{C}^{semi}$.
b. If any z_i is a Q-node, then z_{i-1} and z_{i+1} are separator nodes or P-nodes and z_i has *meeting siblings* $S_{z_{i-1}}$ and $S_{z_{i+1}}$ that meet z_i and that are incomparable, where $1 < i < k$.
c. z_1 and z_k are P-nodes and no separator node in $leaves(y)$ is a predecessor of S_{z_1} or S_{z_k}.
d. If y has consecutive offspring N and N', then $N \subset N'$ or $N \supset N'$.

Other than root r, every Q-node of T^{semi} was a Q-node of T and thus satisfies Property 2 in T^{semi}. It remains to show that root r, which is a Q-node, satisfies Property 2 in T^{semi}.

Property 2a: We consider the children of root r from left to right. Every separator node of B_i is a child of root r and by Corollary 4-2, it satisfies Property 2a in T^{semi}. Consider x_T in T. If x_T was a P-node of T (Figure 10a), then x_T is a P-node of T^{semi} and $S \rightarrow S_{x_T}$.

Then since S and x_T are siblings, S_{x_T} satisfies Property 2a in T^{semi}. If x_T was a Q-node of T and S met x_T (Figure 10b), then x_T became a Q-node of T^{semi} and we may ignore it.

Suppose x_T was a Q-node of T and S semi-met x_T (Figure 11). Then the children of x_T in T became children of root r in T^{semi}. By Property 2c, the leftmost child of x_T was a P-node with separator node child S'. Since $S \to S'$, S' satisfies Property 2a in T^{semi}. Let S'' be any separator node child of x_T; the argument is the same if S'' is an offspring of x_T. If x_T was the root of T, then by Lemma 11-1, S'' satisfies Property 2a in T^{semi}. If x_T was not the root of T, then its parent x_1 was a P-node because a Q-node cannot be the leftmost child of another Q-node by Property 2b. Since S_{x_1} met x_T, $S_{x_1} \rightsquigarrow S''$. Since S_{x_1} satisfied Property 1b in T and S'' satisfied Property 2a in T, S'' satisfies Property 2a in T^{semi}.

For any $x_j, 1 \leq j \leq k$, S did not meet x_j. If x_j was a P-node of T, then since S_{x_j} satisfied Property 1b in T, S_{x_j} satisfies Property 2a in T^{semi}. If x_j was a Q-node of T, then an argument similar to the one for x_T shows that every separator node child of x_j satisfies Property 2a in T^{semi}.

Property 2b: Let z be a Q-node child of root r. Suppose $z = x_T$. Then x_T was a Q-node of T and S met x_T, which implies x_T had parent x_1 in T. Then x_1 was a P-node and S_{x_1} met x_T. Since S and S_{x_1} are incomparable, S and S_{x_1} are the meeting siblings of x_T in T^{semi}. Suppose $z \neq x_T$. Then z was a Q-node child of some Q-node $x_j, 1 \leq j \leq k$, in T that was deleted. Since z had meeting siblings in T, z has the same meeting siblings in T^{semi}.

Property 2c: Let z and z' be the leftmost and rightmost children of root r in T^{semi}, respectively. Consider z'. The root of T was $x_k, k \geq 0$. Case 1: x_k was a Q-node. Then z' was the rightmost child of x_k in T. Since S semi-met T and T had Property 2c, z' is a P-node and no separator node in $leaves(r)$ is a predecessor of $S_{z'}$. Case 2: x_k was a P-node. Then S_{x_k} was in a short box and $Meets(S_{x_k})$ was called when that box was processed. Since S_{x_k} had (at least) two adjacent leaf clique nodes, or two successor trees, or an adjacent leaf clique node and a successor tree, x_k had (at least) two children besides S_{x_k}. Then *Semi-Meets*(S,T) does not replace P-node x_k with S_{x_k}, which means $x_k = z'$. Then as before, no separator node in $leaves(r)$ is a predecessor of $S_{z'}$. Therefore, z' satisfies Property 2c.

Consider z. Since z is an outer node of B_i, S_z is a separator node. If B_i is a long box, $S_z \neq S$, and if B_i is a short box, $S_z = S$. By the Main Theorem-2, no separator node in $leaves(r)$ is a predecessor of S_z. Since S_z has (at least) an adjacent leaf clique node or a successor tree $T' \neq T$, there will be a call of $Meets(S_z)$ or *Semi-Meets*(S_z, T'). In the first case, S_z will be replaced with a P-node with child S_z, and in the second case, S_z will no longer be the leftmost child of root r. Therefore, the leftmost child of root r will satisfy Property 2c when the processing of B_i is complete.

Property 2d: We consider the offspring of root r from left to right. Clearly, Property 2d holds for the nodes in $body(B_i)$. Consider x_T. If x_T was a P-node of T, then S met S_{x_T} and so $S \subset S_{x_T}$. Suppose x_T was a Q-node of T. If S met x_T, then x_T is a Q-node in T^{semi} and we may ignore it. If S semi-met x_T, then the children of x_T became children of root r in T^{semi}; the leftmost child of x_T was a P-node with child S' such that $S \subset S'$. We do not need to consider the other children of x_T because T had Property 2d.

Consider x_1. Case 1: x_1 was a Q-node of T. Then the children of x_1 became children of root r in T^{semi}. Moreover, x_T was a P-node of T and since S_{x_T} was an offspring of x_1 in T, S_{x_T} is an offspring of root r in T^{semi}. Case 2: x_1 was a P-node of T. Suppose x_1 becomes a P-node child of root r; the argument is the same if x_1 is replaced with S_{x_1}. Let z be the left sibling of x_1 in T^{semi}. What is z? If x_T was a P-node of T, then $z = x_T$ is a P-node in T^{semi} and $S_z \supset S_{x_1}$. If x_T was a Q-node of T, then either S met x_T and we may ignore x_T, or S semi-met x_T and z was x_T's rightmost child in T, in which case z was a P-node and $S_z \supset S_{x_1}$. A similar argument applies for each of $x_2, \dots, x_k$. Thus, T^{semi} has Property 2d.

This concludes the discussion of $SemiMeets(S, T)$. Now there may be a $SemiMeets(S', T')$ call for a successor tree $T' \neq T$, where S' is an outer node of B_i and T' is a train tree of a connected component $\mathcal{H}'$ of $\mathcal{G}(i+1)$. Let $T^{semi'}$ denote the train tree resulting from this call. A similar argument shows that $T^{semi'}$ is a train tree of the graph containing $body(B_i)$ and $\mathcal{H}$ and $\mathcal{H}'$ and the arcs from nodes of $body(B_i)$ to nodes of $\mathcal{H}$ or $\mathcal{H}'$, and that $T^{semi'}$ has Properties 1 and 2. This completes Step 2 of the processing of B_i.

Step 3 calls *Meets* for certain separator nodes of B_i. For a separator node S, $Meets(S)$ replaces S with a P-node whose children are S, the leaf clique node(s) adjacent to S, and the root(s) of the tree(s) in $\mathcal{T}_S^m$. Informally, Step 3 adds the leaf clique nodes of B_i and the successor trees in $\mathcal{T}_S^m$ to the train tree. Let T^{new} be the new train tree. Recall that $\mathcal{C}$ is the connected component of $\mathcal{G}(i)$ that contains B_i. The leaves of T^{new} are the nodes of $\mathcal{C}$ and T^{new} is a proper PQ-tree. By the Ordering Lemma and the preceding discussion, every internal node y of T^{new} represents all c.i.p. paths on $leaves(y)$. So T^{new} is a train tree of $\mathcal{C}$. We can readily show T^{new} has Properties 1 and 2. This completes the processing of B_i. $\square$

Remark. Theorem 12 implies that every train tree of a connected component of $\mathcal{G}$ has Properties 1 and 2 because a proper PQ-tree is unique up to equivalence [BL76] and Properties 1 and 2 are invariant under the PQ-tree operations permute and reverse.

Theorem 13 *Given a chordal graph G and its clique-separator graph $\mathcal{G}$, the train tree algorithm runs in $O(n)$ time, where n is the number of vertices of G.*

Proof If $\mathcal{G}$ has more than $2n$ arcs, then the train tree algorithm rejects before it processes any boxes. Otherwise, $\mathcal{G}$ has size $O(n)$ and thus computing $\mathcal{G}^c$ and a topological sort of $\mathcal{G}^c$ requires $O(n)$ time. We consider the cost of all *Meets*, *Scan*, and *SemiMeets* calls in turn. Since there are at most $2n$ arcs in $\mathcal{G}$, there are at most $2n$ *Meets* calls. Each *Meets* call takes $O(1)$ time so the total time for all *Meets* calls is $O(n)$. Since there are at most n separator nodes, there are at most n *Scan* calls. Then the total time for step 1 and step 3 in all *Scan* calls is $O(n)$ because step 1 examines each arc from S in $O(1)$ time and step 3 runs in $O(1)$ time. Now in a particular $Scan(S)$ call, step 2 places each T in $\mathcal{T}_S^m$ in $O(1)$ time, or places T in $\mathcal{T}_S^{sm}$ after following parent pointers from x to $root(T)$ in $O(n)$ time, or rejects. If step 2 follows parent pointers for more than two successor trees, it rejects after taking $O(n)$ time. Then the total cost of following parent pointers in all *Scan* calls is bounded by the total cost of following parent pointers in all *SemiMeets* calls, which we analyze next.

The train tree algorithm creates the following train tree nodes. For each box B_i, at most one Q-node is created (in Step 1 or before Step 1). For each separator node of B_i, at most one P-node is created (in Step 3). Then the total number of train tree nodes created is bounded by the number of boxes in $\mathcal{G}$ plus the number of separator nodes in $\mathcal{G}$, which at most $2n$. Then the total number of parent pointers from train tree nodes is at most $2n$. Now in any call of *SemiMeets*, each parent pointer followed from x_j to x_{j+1} is deleted. This means that as the train tree algorithm runs, each parent pointer is followed at most once. Thus, the total cost of following parent pointers in all *SemiMeets* calls is $O(n)$.

We now bound the cost of making the children of each Q-node $x_j, 0 \leq j \leq k$, into children of root r in *SemiMeets*. Each Q-node has a linked list of its children and only the first and last child has a parent pointer. Then we can concatenate two lists and update the parent of the first and last child in $O(1)$ time. Since this is done at most once for each parent pointer followed, the total cost in all *SemiMeets* calls is $O(n)$. (If $Scan(S)$ finds an ancestor y of x that is not the first or last child of its Q-node parent, then y does not have a parent pointer. But in this case, $Scan(S)$ rejects.) Hence, the train tree algorithm runs in $O(n)$ time. □

Theorem 14 *The train tree algorithm correctly computes the $components_{\mathcal{G}}(S)$, $left(S)$, and $right(S)$ attributes for every separator node S.*

Proof Let S be a node of box B_i and consider the processing of B_i. Step 1 calls $Scan(S)$, which sets $components_{\mathcal{G}}(S) = degree(S) + |\mathcal{T}_S^m| + |\mathcal{T}_S^{sm}|$. This is the number of connected components of $\mathcal{G}(i) - Preds(S)$ that contain a successor of S. By the Main Theorem-3, this equals the number of connected components of $\mathcal{G} - Preds(S)$ that contain a successor of S, so $components_{\mathcal{G}}(S)$ is set correctly. Then $Scan(S)$ initializes $left(S)$ and $right(S)$ after determining whether B_i is a short box or long box.

Step 2 may call $SemiMeets(S, T)$, which sets *right* and *left* for some nodes. First, we show that $SemiMeets(S, T)$ sets $right(S)$ correctly. Suppose S met x_T (Figure 10). If x_T was a Q-node, then x_T is a Q-node in T^{new}. If x_T was a P-node, then consider the parent x_1 of x_T. If x_1 was a P-node, then $right(S_{x_T}) = S_{x_T}$ and $S \subset S_{x_T}$ and $S \not\subset S_{x_1}$, and if x_1 was a Q-node, then $right(S_{x_T})$ was already set. In every case, $right(S)$ is set correctly.

Suppose S semi-met x_T (Figure 11). The algorithm sets $right(S) = right(S')$, where S' was the rightmost offspring of Q-node x_T such that $S \rightarrow S'$. Let $N = right(S')$. Since $S \subset S' \subseteq N$ and S semi-met x_T, N had a right sibling N'. Then (S', N, N') was a subsequence of $path(x_T)$. Suppose $S \subseteq N'$, that is, $S \rightsquigarrow N'$. Then there exists a separator node $S'' \in leaves(x_T)$ such that $S \rightarrow S''$ and $S'' \rightsquigarrow N'$. By Lemma 7-2, S'' is an offspring of x_T. Then S'' must be to the left of S'. Then since $path(x_T)$ was a c.i.p. path, $S'' \subset N'$ implies $S'' \subset S'$. But then $S \subset S'' \subset S'$, which contradicts $S \rightarrow S'$. Then $S \not\subseteq N'$, which implies $right(S)$ is set correctly.

Next we show that $SemiMeets(S, T)$ sets *left* correctly. For each x_j that was a P-node, it sets $left(S_{x_j}) = S'$, which is the leftmost offspring of root r that is to the right of S. Since x_j was an ancestor of x_T, $S_{x_j} \subset S'$ and $S_{x_j} \not\subset S$, which implies $left(S_{x_j})$ is set correctly. For each x_j that was a Q-node, it does not set any *left* attributes for the following reason. By Property 2c, the leftmost child of x_j is P-node x_{j-1} and $left(S_{x_{j-1}})$ is set correctly as just discussed. Let $y \neq x_{j-1}$ be a child of x_j such that y is a separator node or P-node. By Property 2c, $S_y \not\subset S_{x_{j-1}}$ and thus $left(S_y)$ does not need to be changed. A similar argument shows that we do not need to change the *right* attributes of the nodes in $leaves(T)$ that become offspring of root r. □

6 Inserting an edge

In this section, we present the operation that inserts an edge into G and uses $\mathcal{G}$ to decide whether the new G is a chordal graph. If it is, the operation computes the new $\mathcal{G}$ by making a small, local change. Subsequently, the train tree algorithm computes the new $\mathcal{T}$ or rejects because the new G is not an interval graph.

6.1 Preliminaries

Let G be a chordal graph (not necessarily an interval graph) with clique-separator graph $\mathcal{G}$. Throughout this section, $\{v_1, v_2\} \notin E$. Then no clique node or separator node of $\mathcal{G}$ contains both v_1 and v_2; we will use this fact implicitly. Given $v_1 \in V$ and $v_2 \in V$, $G + \{v_1, v_2\}$ denotes $(V, E \cup \{\{v_1, v_2\}\})$. The next two theorems give necessary and sufficient conditions for $G + \{v_1, v_2\}$ to be chordal. Theorem 15 uses the clique tree of G and Theorem 16 uses the clique-separator graph of G.

Theorem 15 ([Iba08a]) *Let G be a chordal graph and let $\{v_1, v_2\} \notin E(G)$. Then $G + \{v_1, v_2\}$ is chordal if and only if G has a clique tree T and maximal cliques K_1 and K_2 such that $v_1 \in K_1$ and $v_2 \in K_2$ and $\{K_1, K_2\} \in E(T)$.*

Theorem 16 *Let G be a connected chordal graph with clique-separator graph $\mathcal{G}$ and let $\{v_1, v_2\} \notin E(G)$. Then $G + \{v_1, v_2\}$ is chordal if and only if $\mathcal{G}$ has a separator node S and clique nodes K_1 and K_2 such that $v_1 \in K_1$ and $v_2 \in K_2$ and S strongly divides K_1 and K_2. Moreover, S is unique if it exists.*

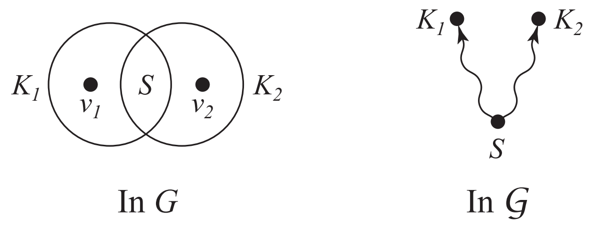


Figure 14: $S = K_1 \cap K_2$ and S strongly divides K_1 and K_2.

Proof The equivalence follows by Theorem 15 and Corollary 4-4. The condition is illustrated in Figure 14. Notice that G must be connected because otherwise, we may have $K_1 \cap K_2 = \emptyset$ and then no separator node equals $K_1 \cap K_2$. To prove uniqueness, suppose separator nodes S and S' and clique nodes K_1, K_2 and K'_1, K'_2, respectively, satisfy the right-hand side of the theorem. Then G has a clique tree T such that $\{K_1, K_2\} \in E(T)$ and $v_1 \in K_1$ and $v_2 \in K_2$. Now edge $\{K_1, K_2\}$ is on the K'_1-K'_2 path in T, or else by the clique intersection property $\{v_1, v_2\}$ is contained in K_1 or K_2, contradicting $\{v_1, v_2\} \notin E(G)$. Then $K'_1 \cap K'_2 \subseteq K_1 \cap K_2$. Similarly, $K'_1 \cap K'_2 \supseteq K_1 \cap K_2$. Then $S = K_1 \cap K_2 = K'_1 \cap K'_2 = S'$. □

For example, consider the chordal graph G in Figure 6. Let v_1 and v_2 be the vertices with degree 2 in K_7 and K_9; these vertices are contained in no other clique nodes. Inserting edge $\{v_1, v_2\}$ into G yields a non-chordal graph because G has no clique tree where K_7 and K_9 are adjacent, or equivalently, $\mathcal{G}$ has no separator node that strongly divides K_7 and K_9.

If $G^+ = G + \{v_1, v_2\}$ is a chordal graph, then we must update $\mathcal{G}$ to obtain the new clique-separator graph $\mathcal{G}^+$. The following theorem shows how to update the nodes of $\mathcal{G}$. We will later consider the edges and arcs of $\mathcal{G}$.

Theorem 17 *Let G be a connected chordal graph with clique-separator graph $\mathcal{G}$ and let $\{v_1, v_2\} \notin E(G)$. Suppose $G^+ = G + \{v_1, v_2\}$ is a chordal graph with clique-separator graph $\mathcal{G}^+$. Let S, K_1, K_2 be the nodes in Theorem 16. Then the following changes to the nodes of $\mathcal{G}$ produce the nodes of $\mathcal{G}^+$.*

1. *Add clique node $K = S \cup \{v_1, v_2\}$.*
2. *For $i = 1$ and $i = 2$, do the following. If $|K_i - S| = 1$, then delete clique node K_i. If $|K_i - S| > 1$, then add separator node $S_i = S \cup \{v_i\}$ unless it already exists.*
3. *If $components_{\mathcal{G}}(S) = 2$, then delete separator node S.*

Proof Since G^+ is chordal, Theorem 15 states that G has a clique tree T and maximal cliques K_1 and K_2 such that $\{K_1, K_2\} \in E(T)$ and $v_1 \in K_1$ and $v_2 \in K_2$. Ibarra [Iba08a] showed that for $i = 1$ and $i = 2$, K_i is not a maximal clique in G^+ if and only if $|K_i - S| = 1$. Ibarra also showed that the following steps update T so that it becomes a clique tree of G^+.

> Delete edge $\{K_1, K_2\}$. Add maximal clique K and edges $\{K_1, K\}$ and $\{K, K_2\}$ (Figure 15). If K_1 and K_2 are both maximal cliques of G^+, then stop. If K_1 is not a maximal clique of G^+, then contract edge $\{K_1, K\}$ and replace K_1 with K. If K_2 is not a maximal clique of G^+, then contract edge $\{K, K_2\}$ and replace K_2 with K. So K_1 and K_2 are replaced by 1, 2, or 3 maximal cliques.

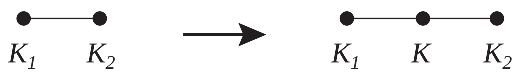


Figure 15: Updating the clique tree T.

It follows that the changes to $\mathcal{G}$ produce the clique nodes of $\mathcal{G}^+$. Next we show that the changes to $\mathcal{G}$ produce the separator nodes of $\mathcal{G}^+$. Since the edges of a clique tree correspond to the minimal vertex separators of the chordal graph by Theorem 1-1, the edges $\{K_1, K\}$ and $\{K, K_2\}$ correspond to $S_1 = K_1 \cap K$ and $S_2 = K \cap K_2$, respectively. By the steps that update T, if $|K_i - S| > 1$, then S_i is a minimal vertex separator of G^+ (we can readily show that the converse holds). In this case we add separator node S_i unless it already exists in $\mathcal{G}$, which occurs when a different edge of T corresponds to S_i. Consider $S = K_1 \cap K_2$. Since the edge $\{K_1, K_2\}$ is the only edge deleted from T, S is the only separator node that might be deleted from $\mathcal{G}$. But S should not be deleted from $\mathcal{G}$ if a different edge of T corresponds to S. We show that S is a separator node of $\mathcal{G}^+$ if and only if $components_{\mathcal{G}}(S) > 2$.

($\Leftarrow$) Suppose $components_{\mathcal{G}}(S) > 2$. Then there exists a clique node K_3 such that K_1, K_2 and K_3 are in different connected components of $\mathcal{G} - Preds(S)$ and S is contained in K_1, K_2, and K_3. By Corollary 4-1, S separates $K_1 - S$, $K_2 - S$, and $K_3 - S$ in G. Then $S = K_1 \cap K_3$ and S separates $K_1 - S$ and $K_3 - S$ in G, and moreover, in G^+. Then S is a minimal vertex separator of G^+ by Corollary 4-3, 4-4, and Theorem 1.

($\Rightarrow$) Suppose $components_{\mathcal{G}}(S) = 2$. Then $\mathcal{G} - Preds(S)$ has exactly two connected components that contain a clique node containing S. By the Main Theorem-1, $G - S$ has exactly two connected components that contain a vertex in a maximal clique containing S;

let V_1 and V_2 be the vertex sets of the components. Then $v_1 \in V_1$ and $v_2 \in V_2$, or vice versa. Since $G^+ = G + \{v_1, v_2\}$, the vertices in $V_1 \cup V_2$ are in one connected component of $G^+ - S$. Then for any maximal clique K' of G^+ that contains S, $K' - S \subseteq V_1 \cup V_2$. It follows that G^+ does not contain two maximal cliques K' and K'' such that $S = K' \cap K''$ and S separates $K' - S$ and $K'' - S$. Then by Theorem 1, S is not a minimal vertex separator of G^+. □

6.2 The insert operation

Let G be an interval graph with clique-separator graph $\mathcal{G}$ and let $\mathcal{G}$ have a spanning set of train trees $\mathcal{T}$. The *Insert* operation computes the new clique-separator graph $\mathcal{G}^+$ or rejects because G^+ is not a chordal graph. Subsequently, the train tree algorithm computes the new $\mathcal{T}$ and the new attributes, or rejects because G^+ is not an interval graph. We maintain the connected components of G, so every vertex can access the name of its connected component.

If v_1 and v_2 are in different connected components of G, then G^+ is always a chordal graph so the operation merges the connected components of G and merges their clique-separator graphs. For example, let G be the graph in Figure 2 with edge K_7 deleted, so that its clique-separator graph $\mathcal{G}$ is the graph in Figure 3 with nodes S_1, K_7, S_2 deleted. Inserting edge K_7 into G yields the graph in Figure 2 with the clique-separator graph in Figure 3.

If v_1 and v_2 are in the same connected component of G, then the operation finds the separator node S and clique nodes K_1 and K_2 in Theorem 16; it rejects if they are not found. The operation updates the nodes as specified in Theorem 17 and adds the appropriate edges and arcs. The main difficulty is that if a separator node S_i is created, the operation must find the arcs to S_i, which is done with a graph search in $\mathcal{G}$. If a node is deleted, any edges or arcs incident on it are deleted.

$Insert(v_1, v_2)$:

v_1 and v_2 are in different connected components of G:

Suppose v_1 and v_2 are in connected components G_1 and G_2 of G with clique-separator graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. Merge the connected components G_1 and G_2. Add clique node $K = \{v_1, v_2\}$ and let $S_1 = \{v_1\}$ and $S_2 = \{v_2\}$. For $i = 1$ and $i = 2$, do the following. If S_i is a separator node of $\mathcal{G}_i$, then add edge $\{S_i, K\}$ and stop. Otherwise, if v_i is not the only vertex in G_i, then add S_i and edge $\{S_i, K\}$ and do the following to compute the other edges and arcs incident to S_i.

Compute the set $\mathcal{M}$ of minimal elements of $\{N \mid N \text{ is a node of } \mathcal{G}_i \text{ and } v_i \in N\}$ as follows: for every node N of $\mathcal{G}_i$ that contains v_i, if no predecessor of N contains v_i, then add N to $\mathcal{M}$. (It suffices to examine the immediate predecessors of a separator node and the neighbors of a clique node. Theorem 18 will show that $\mathcal{M}$ contains either one clique node or only separator nodes.) If $\mathcal{M}$ contains a clique node K', then add edge $\{S_i, K'\}$, and otherwise, add an arc from S_i to every separator node in $\mathcal{M}$.

v_1 and v_2 are in the same connected component of G:

1. [Find S, K_1, K_2.] Find the predecessors of the clique nodes that contain v_1 and the predecessors of the clique nodes that contain v_2. Compute a topological sort $\sigma = (B_1, B_2, \ldots, B_{b(\mathcal{G})})$ of $\mathcal{G}$. Find the separator node S that is a predecessor of some clique nodes K_1 and K_2 containing v_1 and v_2, respectively, and that is in box B_j where

j is as large as possible. If S does not exist or if S exists and K_1 and K_2 are in the same connected component of $\mathcal{G}(j+1)$, then reject.

2. [Process K_1, K_2.] Add clique node $K = S \cup \{v_1, v_2\}$. Do the appropriate case and then repeat with K_1, S_1, v_1 replaced by K_2, S_2, v_2.

Case 1: $|K_1 - S| = 1$. Delete K_1. If K_1 formerly had a neighbor $S' \neq S$, then add edge $\{S', K\}$ (Figure 16).

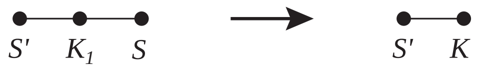


Figure 16: Case 1.

Case 2: $|K_1 - S| > 1$. Let $S_1 = S \cup \{v_1\}$. Then S_1 is a node of $\mathcal{G}$ if and only if (S, S_1) is an arc of $\mathcal{G}$. If S_1 is a node of $\mathcal{G}$, then add edge $\{S_1, K\}$ and go to step 3. Otherwise, add S_1 and edge $\{S_1, K\}$ and do steps (a) and (b) to compute the other edges and arcs incident to S_1.

(a) [Arcs to S_1.] Add arc (S, S_1). (If step 3 deletes S, it adjusts this arc appropriately.) Compute the chain $C = \{S' \mid S' \rightsquigarrow K_1 \text{ and } S' \neq S \text{ and } S' \not\rightsquigarrow S \not\rightsquigarrow S'\}$ by finding the predecessors of K_1 and the predecessors and successors of S (Figure 17). Use binary search to find the largest set S^h in C such that $S^h \subset S_1$. If C is empty or S^h does not exist or $v_1 \notin S^h$, then go to step (b). Otherwise, add arc (S^h, S_1) and delete $\{S^h, K_1\}$ if it is an edge of $\mathcal{G}$.

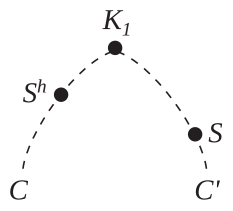


Figure 17: The predecessors of K_1.

(b) [Arcs and edges from S_1.] Compute the set $\mathcal{M}$ of minimal elements of $\{N \mid N$ is a node of $\mathcal{G}$ and $S_1 \subset N\}$ as follows: for every successor N of S that contains v_1, if no predecessor of N is a successor of S that contains v_1, then add N to $\mathcal{M}$. (It suffices to examine the immediate predecessors of a separator node and the neighbors of a clique node. Theorem 18 will show that either $\mathcal{M} = \{K_1\}$ or $\mathcal{M}$ contains only separator nodes.) If $\mathcal{M} = \{K_1\}$, then add edge $\{S_1, K_1\}$ and delete $\{S, K_1\}$ if it is an edge of $\mathcal{G}$. Otherwise, for every separator node S' in $\mathcal{M}$, add arc (S_1, S') and delete (S, S') if it is an arc of $\mathcal{G}$.

3. [Process S.] Do the appropriate case.

Case 1: *components*$_{\mathcal{G}}(S) > 2$. If $|K_1 - S| = 1$ and $|K_2 - S| = 1$, then add edge $\{S, K\}$.
Case 2: *components*$_{\mathcal{G}}(S) = 2$. Delete S. For each former immediate predecessor S^p of S, do the following. If $|K_1 - S| > 1$, then add arc (S^p, S_1). If $|K_2 - S| > 1$, then add arc (S^p, S_2). If $|K_1 - S| = 1$ and $|K_2 - S| = 1$, then add edge $\{S^p, K\}$ unless S^p is

contained in a neighbor of K, in which case add arc (S^p, S') for each neighbor S' of K such that $S^p \subset S'$ and (S^p, S') is not an arc of $\mathcal{G}$.

End Insert

Figure 18 shows an interval graph G, its clique-separator graph $\mathcal{G}$, and the new clique-separator graph $\mathcal{G}^+$ after edges e and f are individually inserted into G. For each insertion, we will indicate which three nodes correspond to the nodes in Theorem 16 and what the operation does. (Note that when we have a choice between two clique nodes, neither clique node is ever deleted.)

When edge e is inserted, the three nodes are S_2, K_2, and K_3. Since $|K_2 - S_2| = 1$ and $|K_3 - S_2| = 1$, step 3 deletes K_2 and K_3 and adds edge $\{S_1, K\}$. Since $components_{\mathcal{G}}(S_2) = 2$, step 4 deletes S_2 and adds edge $\{S_5, K\}$; edge $\{S_4, K\}$ is not added since (S_4, S_1) is an arc.

When edge f is inserted, the three nodes are S_5, K_5, and either K_6 or K_7; we choose K_6. Since $|K_5 - S_5| = 1$, step 3 deletes K_5. Since $|K_6 - S_5| > 1$, step 3 adds separator node S_{new} and edge $\{S_{new}, K\}$. Step 3(a) adds arc (S_5, S_{new}) and no other arcs because C is empty. Step 3(b) computes $\mathcal{M} = \{S_6\}$ and thus adds arc (S_{new}, S_6) and deletes arc (S_5, S_6).

When edge g is inserted, the three nodes are S_4, K_4, and either K_1 or K_2; we choose K_1. Since $|K_4 - S_4| = 1$, step 3 deletes K_4. Since $|K_1 - S_4| > 1$ and (S_4, S_1) is an arc of $\mathcal{G}$, step 3 adds edge $\{S_1, K\}$ and goes to step 4. Since $components_{\mathcal{G}}(S_4) = 2$, step 4 deletes S_4. We omit the figure since the operation only deletes K_4 and S_4 and adds edge $\{S_1, K\}$.

When edge h is inserted, the three nodes are S_5, K_2, and either K_6 or K_7; we choose K_6. The operation adds edges $\{K_2, S_{new}\}$ and $\{S_{new}, K\}$ and other edges and arcs. Since K_2 has three neighbors in $\mathcal{G}^+$, the train tree algorithm will reject.

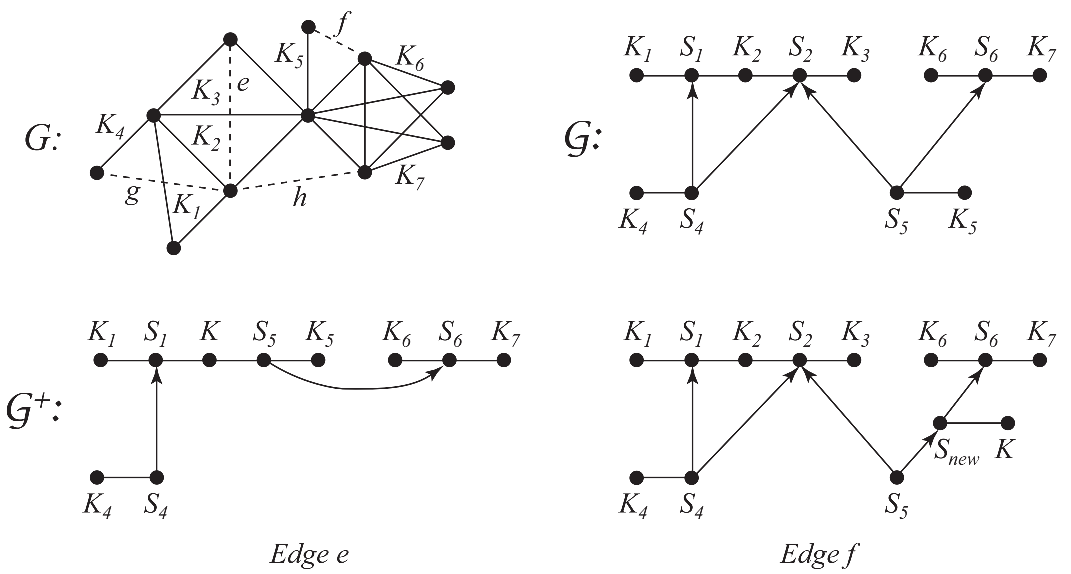


Figure 18: Insert examples.

Theorem 18 *If $G^+ = G + \{v_1, v_2\}$ is a chordal graph, the Insert operation computes its clique-separator graph $\mathcal{G}^+$, and otherwise, the operation rejects.*

Proof Suppose v_1 and v_2 are in distinct connected components G_1 and G_2 of G with clique-separator graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. Then $\{v_1, v_2\}$ is a cut edge of G^+ and $K = \{v_1, v_2\}$ is a clique node of $\mathcal{G}^+$ and the following holds for $i = 1$ and $i = 2$. If v_i is not the only vertex in G_i, then $\{v_i\}$ is a separator of G^+ and $S_i = \{v_i\}$ is a separator node of $\mathcal{G}^+$. Then $\{S_i, K\}$ is an edge of $\mathcal{G}^+$. If S_i is a separator node of $\mathcal{G}_i$, then any other edges and arcs incident on S_i exist in $\mathcal{G}_i$ and we are done. Otherwise, consider $\mathcal{M}$. If exactly one clique node K' of $\mathcal{G}_i$ contains v_i, then $\mathcal{M} = \{K'\}$ because every separator node is contained in at least two clique nodes. Otherwise, the clique nodes of $\mathcal{G}_i$ that contain v_i induce a subtree T_i in any clique tree of G_i. Since v_i is contained in every separator node corresponding to an edge of T_i, $\mathcal{M}$ contains only separator nodes. Thus, the operation computes the arcs and edges incident to S_i. Suppose v_1 and v_2 are in the same connected component of G. We consider each of the three steps in turn.

1. We will show that this step finds the nodes in Theorem 16 if G^+ is a chordal graph, and rejects if G^+ is not a chordal graph. Suppose the operation does not find S and rejects. Then no separator node strongly divides a clique node containing v_1 and a clique node containing v_2. Then G^+ is not a chordal graph by Theorem 16. Suppose the operation finds S. Assume at least one of K_1, K_2 is in B_j, say K_1. Then K_1 is a neighbor of S, or else S is contained in a neighbor of B_j, contradicting the Main Theorem-2. Similarly, if K_2 is in B_j, then K_2 is a neighbor of S, and otherwise, K_2 is a successor of S in some box $B_k, k > j$. In either case, S strongly divides K_1 and K_2 by Corollary 4-2. Assume neither of K_1, K_2 is in B_j.

Suppose K_1 and K_2 are in different connected components of $\mathcal{G}(j+1)$. Since K_1 and K_2 are in the same connected component of $\mathcal{G}(j)$ and in different connected components of $\mathcal{G}(j) - S$ by the Main Theorem-4, S divides K_1 and K_2 in $\mathcal{G}(j)$. Then S divides K_1 and K_2 in $\mathcal{G}$ by the Main Theorem-3. Therefore, S strongly divides K_1 and K_2.

Suppose K_1 and K_2 are in the same connected component of $\mathcal{G}(j+1)$ and the operation rejects. We claim that no separator node of $\mathcal{G}$ strongly divides a clique node containing v_1 and a clique node containing v_2. By the choice of j, no separator node in a box $B_k, k > j$, is a predecessor of a clique node containing v_1 and a clique node containing v_2. Let S' be any separator node in a box $B_i, i \leq j$. Since K_1 and K_2 are in the same connected component of $\mathcal{G}(j+1)$, K_1 and K_2 are in the same connected component of $\mathcal{G} - Preds(S')$. Then by the Main Theorem-1, all clique nodes containing v_1 or v_2 are contained in the same connected component of $\mathcal{G} - Preds(S')$. The claim follows. Then G^+ is not a chordal graph.

2. We will use Theorem 17 implicitly in the rest of the proof. By symmetry, the following statements hold when K_1, S_1, v_1 are replaced by K_2, S_2, v_2. Case 1: $|K_1 - S| = 1$. Then K_1 is not a node of $\mathcal{G}^+$. If $\{S', K_1\}$ is a edge of $\mathcal{G}$ for some $S' \neq S$, then $\{S', K\}$ is a edge of $\mathcal{G}^+$ because $K_1 = S \cup \{v_1\}$ and $K = S \cup \{v_1, v_2\}$. Case 2: $|K_1 - S| > 1$. Then K_1 and S_1 are nodes of $\mathcal{G}^+$ and $\{S_1, K\}$ is a edge of $\mathcal{G}^+$. If S_1 is not a node of $\mathcal{G}$, then steps (a) and (b) compute the other edges and arcs incident to S_1 in $\mathcal{G}^+$, as follows.

(a) (S, S_1) is an arc of $\mathcal{G}^+$ unless S is deleted in step 3. Suppose there exists $\bar{S} \not\subseteq S$ such that $(\bar{S}, S_1)$ is an arc of $\mathcal{G}^+$. Then $v_1 \in \bar{S}$ and $\bar{S} \rightsquigarrow K_1$. For any node of $\mathcal{G}$, the set of its predecessors can be partitioned into at most two chains [Iba06, Lemma 5-1]. Then the predecessors of K_1 can be partitioned into two chains C and C', where S is in C' (Figure 17). Then $\bar{S}$ is in C, which is computed by step (a). Let S^h be the largest set in C such that $S^h \subset S_1$. Note C is nonempty, S^h exists, and $v_1 \in S^h$. Then $S^h = \bar{S}$. Thus, step (a) finds $\bar{S}$ if it exists.

(b) Apart from K, every successor of S_1 in $\mathcal{G}^+$ is a successor of S in $\mathcal{G}$. For any successor N of S, $S_1 \subset N$ if and only if N contains v_1. Therefore, the operation correctly computes $\mathcal{M}$. Since $S_1 \subset K_1$, if there is exactly one maximal clique of G containing S_1, then $\mathcal{M} = \{K_1\}$. Otherwise, an argument similar to the one for the previous $\mathcal{M}$ shows that $\mathcal{M}$ contains only separator nodes. Thus, step (b) computes the rest of the edges and arcs from S_1.

3. Case 1: $\mathit{components}_{\mathcal{G}}(S) > 2$. Then S is a node of $\mathcal{G}^+$. If $|K_1 - S| = 1$ and $|K_2 - S| = 1$, then S_1 and S_2 are not nodes of $\mathcal{G}^+$ and $\{S, K\}$ is an edge of $\mathcal{G}^+$. Otherwise, (S, S_1) is an arc and $\{S_1, K\}$ is an edge of $\mathcal{G}^+$, or (S, S_2) is an arc and $\{S_2, K\}$ is an edge of $\mathcal{G}^+$, or both, so no more changes are required. Case 2: $\mathit{components}_{\mathcal{G}}(S) = 2$. Then S is not a node of $\mathcal{G}^+$. Consider any immediate predecessor S^p of S. If $|K_1 - S| > 1$, then S_1 is a node of $\mathcal{G}^+$ and (S^p, S_1) is an arc of $\mathcal{G}^+$. If $|K_2 - S| > 1$, the argument is symmetric. If $|K_1 - S| = 1$ and $|K_2 - S| = 1$, then $\{S^p, K\}$ is an edge of $\mathcal{G}^+$ unless S^p is contained in a neighbor S' of K in $\mathcal{G}^+$. (If S' exists, the edge $\{S', K\}$ was added in step 2, Case 1.) □

Theorem 19 *The Insert operation runs in $O(n \log n)$ time, where n is the number of vertices of G.*

Proof By Theorem 6, every separator node has at most two immediate predecessors, every clique node has at most two neighbors, and the size of $\mathcal{G}$ is $O(n)$. Since each node has a characteristic vector of the vertices it contains, each step requires $O(n)$ time except the binary search in step 3. Testing whether one separator node is contained in another requires $O(n)$ time, so the binary search requires $O(n \log n)$ time. □

7 Deleting an edge

In this section, we present the operation that deletes an edge from G and uses $\mathcal{G}$ and $\mathcal{T}$ to decide whether the new G is a chordal graph. If it is, the operation computes the new $\mathcal{G}$ by making a small, local change. Subsequently, the train tree algorithm computes the new $\mathcal{T}$ or rejects because the new G is not an interval graph.

7.1 Preliminaries

Let G be a chordal graph (not necessarily an interval graph) with clique-separator graph $\mathcal{G}$. Throughout this section, $\{v_1, v_2\} \in E$. Then $\{v_1, v_2\}$ is contained in at least one clique node of $\mathcal{G}$; we will use this fact implicitly. Given $v_1 \in V$ and $v_2 \in V$, $G - \{v_1, v_2\}$ denotes $(V, E - \{\{v_1, v_2\}\})$. The next theorem gives a necessary and sufficient condition for $G - \{v_1, v_2\}$ to be chordal.

Theorem 20 ([Iba08a]) *Let G be a chordal graph and let $\{v_1, v_2\} \in E(G)$. Then $G - \{v_1, v_2\}$ is chordal if and only if G has exactly one maximal clique that contains both v_1 and v_2.*

Suppose K is the unique maximal clique that contains both v_1 and v_2. Then K is not a maximal clique in $G - \{v_1, v_2\}$, whereas $K - \{v_1\}$ and $K - \{v_2\}$ may or may not be maximal cliques in $G - \{v_1, v_2\}$. For example, consider the chordal graph in Figure 1. Deleting edge $\{y, z\}$ yields a non-chordal graph because two maximal cliques contain both y and z. Deleting edge $\{x, y\}$ yields a chordal graph and $\{x, z\}$, but not $\{y, z\}$, is a maximal clique in the new chordal graph.

If $G^- = G - \{v_1, v_2\}$ is a chordal graph, then we must update $\mathcal{G}$ to obtain the new clique-separator graph $\mathcal{G}^-$. The following theorem shows how to update the nodes of $\mathcal{G}$. We will later consider the edges and arcs of $\mathcal{G}$.

Theorem 21 *Let G be a chordal graph with clique-separator graph $\mathcal{G}$ and let $\{v_1, v_2\} \in E(G)$. Suppose $G^- = G - \{v_1, v_2\}$ is a chordal graph with clique-separator graph $\mathcal{G}^-$. Let K be the unique clique node of $\mathcal{G}$ that contains both v_1 and v_2. Let $N_1 = K - \{v_2\}$ and $N_2 = K - \{v_1\}$. Then the following changes to the nodes of $\mathcal{G}$ produce the nodes of $\mathcal{G}^-$.*

1. *Delete clique node K.*
2. *Add separator node $\bar{S} = K - \{v_1, v_2\}$ unless it already exists or $\bar{S} = \emptyset$.*
3. *For $i = 1$ and $i = 2$, do the following. If N_i is not a neighbor of K, then add clique node N_i. If N_i is a neighbor of K and $components_{\mathcal{G}}(N_i) = 2$, then delete separator node N_i.*

Proof Let T be any clique tree of G. Ibarra [Iba08a] showed that that the following steps update T so that it becomes a clique tree of G^+.

> Delete maximal clique K. Add N_1 and N_2 and add edge $\{N_1, N_2\}$. For every former neighbor K' of K, if $v_1 \in K'$, then add edge $\{K', N_1\}$, and otherwise, add edge $\{K', N_2\}$ (The maximal cliques containing neither v_1 nor v_2 could be made neighbors of either N_1 or N_2. In Figure 19, $K_1, \ldots, K_i$ are the maximal cliques containing v_1 and $K'_1, \ldots, K'_i$ are the remaining maximal cliques.) If N_1 and N_2 are both maximal cliques, then stop. If N_1 is contained in some maximal clique K', then contract edge $\{K', N_1\}$ and replace N_1 with K'. If N_2 is contained in some maximal clique K', then contract edge $\{K', N_2\}$ and replace N_2 with K'. So K is replaced by 0, 1, or 2 maximal cliques.

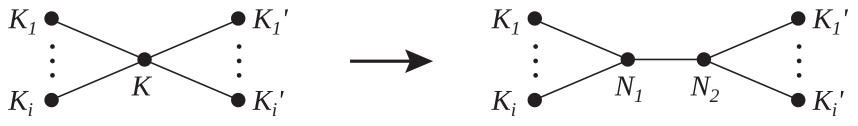


Figure 19: Updating the clique tree T.

Since $N_1 \cap N_2 = K - \{v_1, v_2\} = \bar{S}$, we add separator node $\bar{S}$ unless it already exists or $\bar{S} = \emptyset$. Consider N_1; the argument for N_2 is symmetric. Suppose N_1 is not a clique node of $\mathcal{G}^-$. Then N_1 is contained in a clique node K' of $\mathcal{G}$. Then $K \cap K' = N_1 \neq \emptyset$. Then by Lemma 2, N_1 is contained in a neighbor of K in $\mathcal{G}$. Since $|K - N_1| = 1$, N_1 must be a neighbor of K in $\mathcal{G}$. Thus, if N_1 is not a neighbor of K in $\mathcal{G}$, then N_1 is a clique node of $\mathcal{G}^-$.

Suppose N_1 is a neighbor of K in $\mathcal{G}$. Then $S_1 = N_1$ is a separator node of $\mathcal{G}$. We show that S_1 is a separator node of $\mathcal{G}^-$ if and only if $components_{\mathcal{G}}(S_1) > 2$.

($\Leftarrow$) Suppose $components_{\mathcal{G}}(S_1) > 2$. Then there exist clique nodes $K' \neq K$ and $K'' \neq K$ such that S_1 strongly divides K' and K'' in $\mathcal{G}$. Then by Corollary 4-3, $S_1 = K' \cap K''$ and S_1 separates $K' - S_1$ and $K'' - S_1$ in G; the same is true in G^- since neither K' nor K'' contains both v_1 and v_2. Then S_1 is a minimal vertex separator of G^- by Corollary 4-3 and 4-4.

($\Rightarrow$) Suppose $components_{\mathcal{G}}(S_1) = 2$. Then $\mathcal{G} - Preds(S_1)$ has exactly two connected components that contain a clique node containing S_1. By the Main Theorem-1, $G - S_1$ has exactly two connected components that contain a vertex in a maximal clique containing S_1; let V_1 and V_2 be the vertex sets of these two components. Let G_1 and G_2 be the subgraphs of G induced by $V_1 \cup S_1$ and $V_2 \cup S_1$, respectively. Without loss of generality assume $v_2 \in V_2$ (Figure 20). Note $K = S_1 \cup \{v_2\}$.

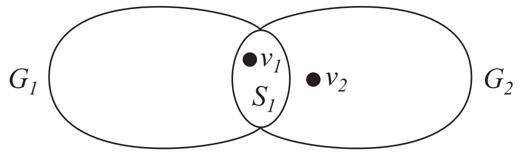


Figure 20: The subgraphs G_1 and G_2.

Now v_2 is the only vertex in V_2 that is adjacent to every vertex of S_1 in G. Otherwise, there is a clique node $K' \neq K$ in the same connected component of $\mathcal{G} - Preds(S_1)$ as K and $S_1 \rightsquigarrow K'$. But since S_1 has neighbor K, this contradicts Corollary 4-2. Then no vertex in V_2 is adjacent to every vertex of S_1 in G^-. It follows that G^- does not contain two maximal cliques K' and K'' such that $S_1 = K' \cap K''$ and S_1 separates $K' - S_1$ and $K'' - S_1$. Then by Theorem 1, S_1 is not a minimal vertex separator of G^-. $\Box$

7.2 The delete operation

Let G be an interval graph with clique-separator graph $\mathcal{G}$ and let $\mathcal{G}$ have a spanning set of train trees $\mathcal{T}$. The *Delete* operation computes the new clique-separator graph $\mathcal{G}^-$ or rejects because G^- is not a chordal graph. Subsequently, the train tree algorithm computes the new $\mathcal{T}$ and the new attributes, or rejects because G^- is not an interval graph. We maintain the connected components of G, so every vertex can access the name of its connected component.

The operation begins by finding the clique node K that contains both v_1 and v_2. There are two main cases. In the first, K is a leaf with neighbor S and in the second, K is an internal node with neighbors S and S'. Each case has subcases that depend on whether v_1 or v_2 is contained in a neighbor of K. (No neighbor of K contains both v_1 and v_2 because K is unique.) The main difficulty is that if separator node $\bar{S}$ is created, the operation must find the arcs from $\bar{S}$, which is done with a search in a train tree of $\mathcal{T}$. The operation uses two subroutines: $GetPreds(S, u)$ returns the maximal elements of the set of predecessors of S that do not contain vertex $u \in S$, and $Remove(S)$ deletes S and adjusts the edges and arcs incident on S.

$Delete(v_1, v_2)$:

Find a clique node K that contains both v_1 and v_2 and reject if K is not unique. Let $N_1 = K - \{v_2\}$, $N_2 = K - \{v_1\}$, and $\bar{S} = K - \{v_1, v_2\}$. If $|K| > 2$, then do Case 1 or Case 2 or Case 3. If $|K| = 2$, then $\{v_1, v_2\}$ is a cut edge; split G into the connected components G_1

and G_2 that contain v_1 and v_2, respectively, and do the following for $i = 1$ and $i = 2$. If v_i is the only vertex in G_i, then add clique node N_i. Otherwise, if $\mathit{components}_{\mathcal{G}}(N_i) = 2$, then call $\mathit{Remove}(N_i)$.

At the end of the operation, delete K and its incident edges.

Case 1: K is a leaf with neighbor S

Case 1a: $v_1 \notin S$, $v_2 \notin S$, and $|K - S| = 2$ (Figure 21a). Then $S = \bar{S}$. Add clique nodes N_1, N_2 and edges $\{S, N_1\}, \{S, N_2\}$.

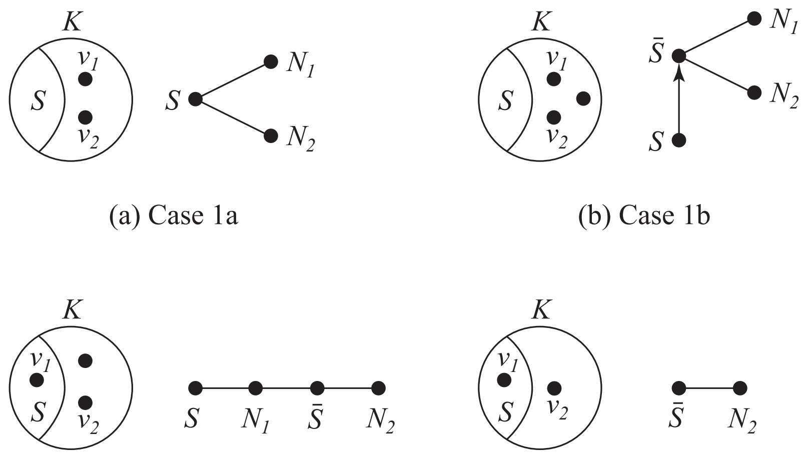
Case 1b: $v_1 \notin S$, $v_2 \notin S$, and $|K - S| > 2$ (Figure 21b). Then $S \subset \bar{S}$. Add clique nodes N_1, N_2, separator node $\bar{S}$, edges $\{\bar{S}, N_1\}, \{\bar{S}, N_2\}$, and arc $(S, \bar{S})$.

Case 1c: $v_1 \in S$, $v_2 \notin S$, and $|K - S| > 1$ (Figure 21c). Then $\bar{S} \not\subset S$ and $S \not\subset \bar{S}$. Add clique nodes N_1, N_2, separator node $\bar{S}$, and edges $\{S, N_1\}, \{N_1, \bar{S}\}, \{\bar{S}, N_2\}$. For each $S' \in \mathit{GetPreds}(S, v_1)$, add arc $(S', \bar{S})$.

Case 1d: $v_1 \in S$, $v_2 \notin S$, and $|K - S| = 1$ (Figure 21d). Then $S = N_1 = \bar{S} \cup \{v_1\}$. Then $\bar{S}$ is a node of $\mathcal{G}$ if and only if $(\bar{S}, S)$ is an arc of $\mathcal{G}$. Do the appropriate case.

Subcase 1: $\bar{S}$ is a node of $\mathcal{G}$. Add clique node N_2 and edge $\{\bar{S}, N_2\}$. If $\mathit{components}_{\mathcal{G}}(S) = 2$, then call $\mathit{Remove}(S)$.

Subcase 2: $\bar{S}$ is not a node of $\mathcal{G}$. Add clique node N_2, separator node $\bar{S}$, and edge $\{\bar{S}, N_2\}$. For each $S' \in \mathit{GetPreds}(S, v_1)$, add arc $(S', \bar{S})$ and delete (S', S) if (S', S) is an arc of $\mathcal{G}$. Let T be the train tree of $\mathcal{G}$. If S is not an offspring of any Q-node of T, then add arc $(\bar{S}, S)$. Otherwise, S is an offspring of a Q-node x. Use binary search to mark the leftmost and rightmost offspring of x that contain $\bar{S}$, and then mark every intervening offspring. For each marked separator node S' that does not have a marked predecessor, add arc $(\bar{S}, S')$. If $\mathit{components}_{\mathcal{G}}(S) = 2$, then call $\mathit{Remove}(S)$.



(a) Case 1a

(b) Case 1b

(c) Case 1c

(d) Case 1d

Figure 21: K is a leaf with neighbor S.

Case 2: K is an internal node with neighbors S and S'

Case 2a: $v_1 \notin S \cup S'$ and $v_2 \notin S \cup S'$ (Figure 22a). Add clique nodes N_1, N_2, separator node $\bar{S}$, edges $\{N_1, \bar{S}\}, \{\bar{S}, N_2\}$, and arcs $(S, \bar{S}), (S', \bar{S})$.

Case 2b: $v_1 \in S \cap S'$ and $v_2 \notin S \cup S'$ (Figure 22b). Add clique nodes N_1, N_2, separator node $\bar{S}$, and edges $\{S, N_1\}, \{S', N_1\}, \{N_1, \bar{S}\}, \{\bar{S}, N_2\}$.

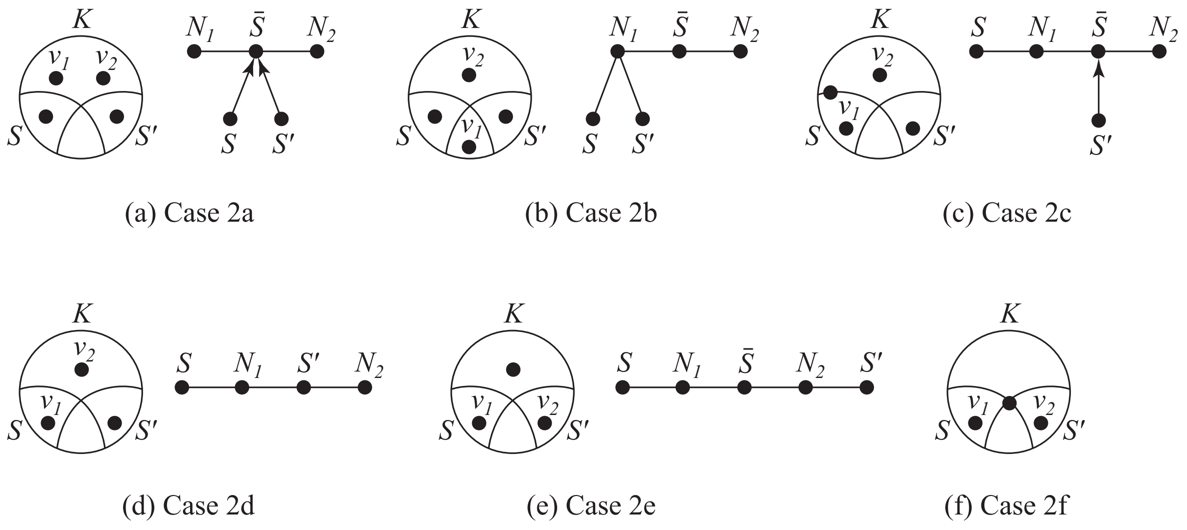
Case 2c: $v_1 \in S - S'$, $v_2 \notin S \cup S'$, and $|K - S'| > 2$ (Figure 22c). Add clique nodes N_1, N_2, separator node $\bar{S}$, edges $\{S, N_1\}, \{N_1, \bar{S}\}, \{\bar{S}, N_2\}$, and arc $(S', \bar{S})$. For each $S'' \in \mathit{GetPreds}(S, v_1)$, if S'' is not a predecessor of S', then add arc $(S'', \bar{S})$.

Case 2d: $v_1 \in S - S'$, $v_2 \notin S \cup S'$, and $|K - S'| = 2$ (Figure 22d). Add clique nodes N_1, N_2 and edges $\{S, N_1\}, \{N_1, S'\}, \{S', N_2\}$.

Case 2e: $v_1 \in S - S'$, $v_2 \in S' - S$, and $|K - (S \cup S')| > 0$ (Figure 22e). Add clique nodes N_1, N_2, separator node $\bar{S}$, and edges $\{S, N_1\}, \{N_1, \bar{S}\}, \{\bar{S}, N_2\}, \{N_2, S'\}$. Compute $\mathcal{S} = \mathit{GetPreds}(S, v_1)$ and $\mathcal{S}' = \mathit{GetPreds}(S', v_2)$. For each $S'' \in \mathcal{S} \cup \mathcal{S}'$, add arc $(S'', \bar{S})$.

Case 2f: $v_1 \in S - S'$, $v_2 \in S' - S$, and $|K - (S \cup S')| = 0$ (Figure 22f). Theorem 22 will prove the following. If $|S' - S| = 1$, then it follows that $\bar{S}$ is a node of $\mathcal{G}$ if and only if $(\bar{S}, S)$ is an arc of $\mathcal{G}$. If $|S - S'| = 1$, then it follows that $\bar{S}$ is a node of $\mathcal{G}$ if and only if $(\bar{S}, S')$ is an arc of $\mathcal{G}$. If $|S' - S| > 1$ and $|S - S'| > 1$, then $\bar{S}$ is not a node of $\mathcal{G}$.

If $\bar{S}$ is not a node of $\mathcal{G}$, then add separator node $\bar{S}$, and if $\bar{S} \subset S$ and/or $\bar{S} \subset S'$, then add arc $(\bar{S}, S)$ and/or $(\bar{S}, S')$, respectively. If $|S' - S| > 1$, then add clique node N_1 and edges $\{S, N_1\}, \{N_1, \bar{S}\}$, and if $|S' - S| = 1$ and $\mathit{components}_{\mathcal{G}}(S) = 2$, then call $\mathit{Remove}(S)$. Do the symmetric action with $|S - S'|$.



(a) Case 2a (b) Case 2b (c) Case 2c (d) Case 2d (e) Case 2e (f) Case 2f

Figure 22: K is an internal node with neighbors S and S'.

Case 3: K **has no neighbors**

Add clique nodes N_1, N_2, separator node $\bar{S}$, and edges $\{N_1, \bar{S}\}, \{\bar{S}, N_2\}$.

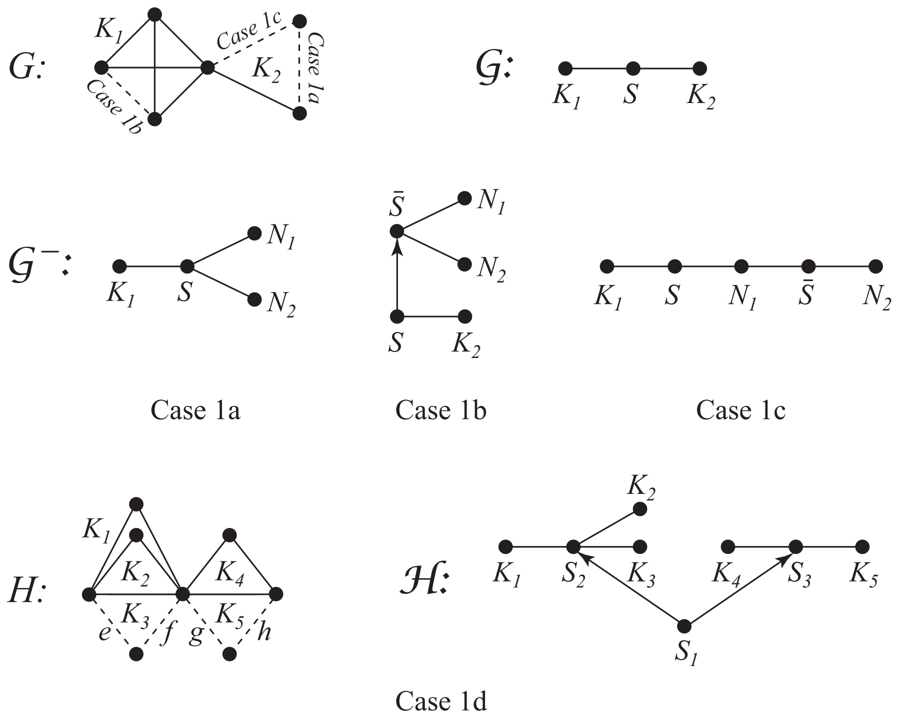


Figure 23: Examples of Case 1.

GetPreds(S, v) :

Compute the set of maximal elements of $\{S' \mid S' \rightsquigarrow S \text{ and } v \notin S\}$ by examining the predecessors of S in reverse topological order. Return this set.

Remove(S) :

Delete S and its incident edges and arcs. For each old immediate predecessor S^p of S, do the following. Mark the successors of S^p. For each old immediate successor S^s of S that is unmarked, add arc (S^p, S^s). For any old neighbor $K' \neq K$ of S that is unmarked, add edge $\{S^p, K'\}$. Unmark the successors of S^p.

End Delete

Figure 23 shows an interval graph G, its clique-separator graph $\mathcal{G}$, and the new clique-separator graph when the edges labeled Case 1a, Case 1b, and Case 1c are individually deleted from G. Figure 23 also shows an interval graph H, its clique-separator graph $\mathcal{H}$, and four examples of Case 1d. If edge e or edge h is deleted, then $\bar{S} = S_1$ is a node of $\mathcal{G}$, but in the other cases $\bar{S}$ is not a node of $\mathcal{G}$. If edge e or edge f is deleted, then $S = S_2$ is a node of $\mathcal{G}^-$ but in the other cases $S = S_3$ is not a node of $\mathcal{G}^-$. Figure 24 shows examples of Case 2.

Theorem 22 *If $G^- = G - \{v_1, v_2\}$ is a chordal graph, the Delete operation computes its clique-separator graph $\mathcal{G}^-$, and otherwise, the operation rejects.*

Proof We can readily prove that $K = \{v_1, v_2\}$ if and only if $\{v_1, v_2\}$ is a cut edge of G. The backwards direction is easy and the forward direction uses the fact that in a chordal graph, the shortest cycle (if there is one) containing a given edge is a triangle. If $K = \{v_1, v_2\}$, then

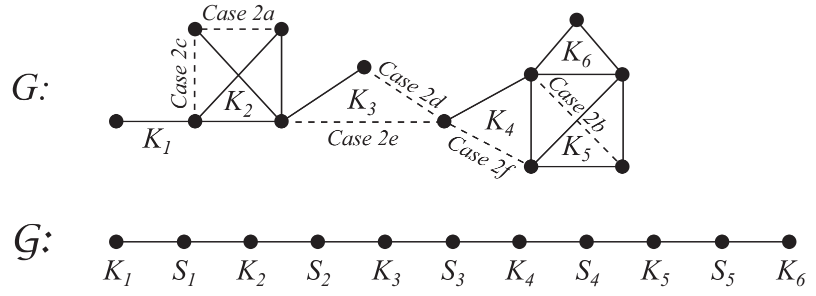


Figure 24: Examples of Case 2.

G splits into the connected components G_1 and G_2. Then if v_i is the only vertex in G_i, then $N_i = \{v_i\}$ is the only maximal clique of G_i. Otherwise, v_i is a minimal vertex separator of G and so N_i is a neighbor of K. By Theorem 21, the changes are correct. If $K \neq \{v_1, v_2\}$, then we have Case 1 or Case 2 or Case 3.

Case 1: K is a leaf with neighbor S

By Lemma 2, the vertices in $K - S$ are not contained in any other node of $\mathcal{G}$. In Cases 1a–1c, $N_1 \neq S$ and $N_2 \neq S$. In Cases 1b–1c, $\bar{S}$ contains a vertex in $K - S$ and so $\bar{S}$ is not a node of $\mathcal{G}$. Then by Theorem 21, the nodes added in Cases 1a–1c are correct. From the containment relations among the new nodes, it follows that the edges and arcs added in Cases 1a–1c are correct. Moreover, the edges and arcs incident on S in $\mathcal{G}$ are still in $\mathcal{G}^-$, except for $\{S, K\}$. The following comments discuss the edges and arcs incident on $\bar{S}$.

Case 1a: This case does not add $\bar{S}$.

Case 1b: Since $\bar{S}$ contains a vertex in $K - S$, there are no other edges or arcs from $\bar{S}$. There are no other arcs to $\bar{S}$ because for any separator node S' of $\mathcal{G}$, if $S' \subset \bar{S}$, then $S' \subseteq S$.

Case 1c: As before, there are no other edges or arcs from $\bar{S}$. For any separator node S' of $\mathcal{G}$, if $S' \subset \bar{S}$, then $S' \subseteq S - \{v_1\}$. Then the immediate predecessors of $\bar{S}$ are the maximal elements of $\{S' \mid S' \rightsquigarrow S \text{ and } v_1 \notin S'\}$, which are the elements returned by $GetPreds(S, v_1)$.

Case 1d: Since $\bar{S}$ and N_2 are nodes of $\mathcal{G}^-$, $\{\bar{S}, N_2\}$ is an edge of $\mathcal{G}^-$. Since $S = N_1$, if $components_{\mathcal{G}}(S) = 2$, then S is not a node of $\mathcal{G}^-$ and the operation calls $Remove(S)$, which deletes S. In Subcase 1, $\bar{S}$ is a separator node of $\mathcal{G}$ and so any other edges and arcs incident on $\bar{S}$ already exist and we are done. In Subcase 2, $\bar{S}$ is not a separator node of $\mathcal{G}$ and so the operation adds $\bar{S}$. The immediate predecessors of $\bar{S}$ are the nodes returned by $GetPreds(S, v_1)$, so the arcs to $\bar{S}$ are computed correctly. The immediate successors of $\bar{S}$ are the minimal elements of the set of nodes of $\mathcal{G}$ containing $\bar{S}$, or equivalently, the minimal elements of $leaves(T)$ containing $\bar{S}$, where T is the train tree of $\mathcal{G}$. (Notice S is a minimal element of $leaves(T)$ containing $\bar{S}$, but S may not be a node of $\mathcal{G}^-$.)

If S is an offspring of some Q-node x of T, then by Lemma 7-2, every minimal element of $leaves(T)$ containing $\bar{S}$ is also an offspring of x. Otherwise, S is not an offspring of any Q-node of T. Then S is a child of a P-node y such that y is the root of T or y has a P-node parent z. In the first case, S is contained in every node of $leaves(T)$ and so S is the only minimal element of $leaves(T)$ containing $\bar{S}$. In the second case, S is contained in every node of $leaves(y)$ (Figure 5 shows a similar case). It follows that if $S' \neq S$ is a minimal element of $leaves(T)$ containing $\bar{S}$, then $S' \in leaves(T) - leaves(y)$. Then S_z divides S and S' by

Property 1. Then $\bar{S} \subset S$ and $\bar{S} \subset S'$ imply that $\bar{S} \subset S_z \subset S$, a contradiction because $\bar{S} \cup \{v_1\} = S$. Thus, S is the only minimal element of $leaves(T)$ containing $\bar{S}$.

Consider the arcs from $\bar{S}$. If S is not an offspring of any Q-node of T, then S is the only minimal element of $leaves(T)$ containing $\bar{S}$ and so arc $(\bar{S}, S)$ is added. Otherwise, S is an offspring of some Q-node x and every minimal element of $leaves(T)$ containing $\bar{S}$ is an offspring of x. By the clique intersection property, every marked offspring contains $\bar{S}$. Suppose some marked separator node S' is not a minimal element of $leaves(T)$ containing $\bar{S}$. Then $S'' \to S'$ for some element S'' of $leaves(T)$ containing $\bar{S}$. By Lemma 8, $S'' \in leaves(x)$, or else $\bar{S} \subset S'' \subset S$, a contradiction. By Lemma 8 again, S'' is an offspring of x, or else $S'' \not\to S'$, a contradiction. Then S'' is marked and arc $(\bar{S}, S')$ is not added. It follows that the arcs added are from $\bar{S}$ to the minimal elements of $leaves(T)$ containing $\bar{S}$. The arc $(\bar{S}, S)$ is also added. If S is not a node of $\mathcal{G}^-$, then S and $(\bar{S}, S)$ are deleted by *Remove*(S).

Consider the neighbors of $\bar{S}$. If $\bar{S}$ has a neighbor $K' \neq N_2$ in $\mathcal{G}^-$, then K' is a clique node of $\mathcal{G}$. Then $\bar{S} \subseteq K' \cap K$ and by Lemma 2, $\bar{S} \subseteq S'$ for some neighbor S' of K' in $\mathcal{G}$. This is a contradiction unless $S' = S$ and S is deleted. Therefore, *Remove*(S) deletes S and adds the edge $\{\bar{S}, K'\}$. Thus, the operation correctly computes the edges and arcs incident on $\bar{S}$.

Case 2: K is an internal node with neighbors S and S'

Since S and S' are neighbors of K, S and S' are incomparable and so $|S - S'| > 0$ and $|S' - S| > 0$. Then in Cases 2a–2e, $S \neq N_1 \neq S'$ and $S \neq N_2 \neq S'$. Moreover, Lemma 2 implies that $\bar{S}$ is not a node of $\mathcal{G}$ except in Case 2d where $\bar{S} = S$. Then by Theorem 21, the nodes added in Cases 2a–2e are correct. From the containment relations among the new nodes, it follows that the edges and arcs added in Cases 2a–2e are correct. Moreover, the edges and arcs incident on S and S' in $\mathcal{G}$ are still in $\mathcal{G}^-$, except for $\{S, K\}$ and $\{S', K\}$. The following comments discuss the edges and arcs incident on $\bar{S}$.

Case 2a: By Lemma 2, there are no other arcs to $\bar{S}$ and no arcs from $\bar{S}$.

Case 2b: Since clique node N_1 has three neighbors, the train tree algorithm will reject.

Case 2c: By Lemma 2, there are no arcs from $\bar{S}$. Moreover, for any separator node S'' of $\mathcal{G}$, if $S'' \subset \bar{S}$, then $S'' \subseteq S'$ or $S'' \subseteq S - \{v_1\}$. Then the immediate predecessors of $\bar{S}$ are S' and the nodes in *GetPreds*(S, v_1) that are not predecessors of S'.

Case 2d: This case does not add $\bar{S}$.

Case 2e: By Lemma 2, there are no arcs from $\bar{S}$ and the immediate predecessors of $\bar{S}$ are the nodes in $\mathcal{S} \cup \mathcal{S}'$.

Case 2f: If $|S' - S| = 1$, then $S - \{v_1\} = \bar{S}$, which implies that $\bar{S}$ is a node of $\mathcal{G}$ if and only if $(\bar{S}, S)$ is an arc of $\mathcal{G}$. If $|S - S'| = 1$, the argument is symmetric. If $|S' - S| > 1$ and $|S - S'| > 1$, then $\bar{S} \not\subset S$ and $\bar{S} \not\subset S'$, and by Lemma 2, $\bar{S}$ is not a node of $\mathcal{G}$. If $\bar{S}$ is not a node of $\mathcal{G}$, then the operation adds $\bar{S}$ and any arcs from $\bar{S}$ to S or S'; the nodes S or S' may be deleted next. If $|S' - S| > 1$, then S and N_1 are nodes of $\mathcal{G}^-$ and $\{S, N_1\}, \{N_1, \bar{S}\}$ are edges of $\mathcal{G}^-$. If $|S' - S| = 1$, then $N_1 = S$ and the operation uses the condition in Theorem 21 to decide whether to call *Remove*(S). The argument for $|S - S'|$ is symmetric. $\square$

Theorem 23 *The Delete operation runs in $O(n \log n)$ time, where n is the number of vertices of G.*

Proof By Theorem 6, every separator node has at most two immediate predecessors, every clique node has at most two neighbors, and the size of $\mathcal{G}$ is $O(n)$. Each node has a characteristic vector of the vertices it contains. Then $\mathit{GetPreds}(S, v)$ and $\mathit{Remove}(S)$ run in $O(n)$ time. Then every case requires $O(n)$ time except for the binary search in Case 1d, which requires $O(n \log n)$ time. $\square$

8 Conclusions

We have presented a dynamic graph algorithm for recognizing interval graphs that maintains both the clique-separator graph and the train tree. It may be possible to improve the running time by maintaining only the train tree. This would probably require augmenting the train tree data structure with additional information about the vertices contained in the nodes.

When the interval graph is a connected proper interval graph, its clique-separator graph contains exactly one box, which is a path, and no arcs [Iba06]. Then the dynamic graph algorithm in this paper can be simplified considerably: the train tree is unnecessary and several cases in the insert and delete operations may be omitted since they produce a clique-separator graph that violates the given properties. The resulting algorithm [Iba08b] is simpler than the algorithm in [HSS01] and it has the same running time of $O(\log n)$ time per edge update. There may be other subclasses of interval graphs for which one can simplify the dynamic graph algorithm or improve its running time.

Sparsification is a general technique that has been used to develop dynamic graph algorithms from static graph algorithms [EGIN97]. For a variety of graph problems, sparsification converts a static algorithm that runs in $T(n, m)$ time into a dynamic algorithm that runs in $T(n, O(n))$ time. This conversion requires that the graph problem have a *sparse certificate* [EGIN97]. Since interval graphs can be recognized in $O(m + n)$ time, if sparse certificates for interval graphs could be found efficiently, then this would probably yield a dynamic graph algorithm for interval graphs that runs in $O(n)$ time per update. It seems difficult, however, to find sparse certificates for interval graphs or for chordal graphs.

References

[BL76] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Computer and System Sciences*, 13:335–379, 1976.

[BLS99] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey.* Society for Industrial and Applied Mathematics, Philadelphia, 1999. Monograph.

[BP93] J. R. S. Blair and B. Peyton. An introduction to chordal graphs and clique trees. In *Graph Theory and Sparse Matrix Computation*, pages 1–29. Springer-Verlag, New York, 1993. IMA Vol. 56.

[CKN$^+$95] D. G. Corneil, H. Kim, S. Nataranjan, S. Olariu, and A. P. Sprague. Simple linear time recognition of unit interval graphs. *Information Processing Letters*, 55:99–104, 1995.

[COS98] D. G. Corneil, S. Olariu, and L. Stewart. The ultimate interval graph recognition algorithm? In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 175–180, 1998.

[DHH96] X. Deng, P. Hell, and J. Huang. Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. *SIAM J. Computing*, 25(2):390–403, 1996.

[EGI98] D. Eppstein, Z. Galil, and G. F. Italiano. Dynamic graph algorithms. In M. J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 8. CRC Press, Boca Raton, FL, 1998.

[EGIN97] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification - A technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

[FG65] D. Fulkerson and O. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

[FK99] J. Feigenbaum and S. Kannan. Dynamic graph algorithms. In Rosen, editor, *Handbook of Discrete and Combinatorial Mathematics*, chapter 17. CRC Press, Boca Raton, FL, 1999.

[Gol04] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57).* Elsevier B.V., Amsterdam, 2004.

[HdT01] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

[HL89] C. Ho and R. C. T. Lee. Counting clique trees and computing perfect elimination schemes in parallel. *Information Processing Letters*, 31:61–68, 1989.

[HM99] W. L. Hsu and T. H. Ma. Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs. *SIAM J. Computing*, 28(3):1004–1020, 1999.

[HSS01] P. Hell, R. Shamir, and R. Sharan. A fully dynamic algorithm for recognizing and representing proper interval graphs. *SIAM J. Computing*, 31:289–305, 2001.

[Hsu96] W. L. Hsu. On-line recognition of interval graphs. In *Proceedings of Combinatorics and Computer Science, LNCS 1120*, pages 27–38, 1996.

[Iba06] L. Ibarra. The clique-separator graph for chordal graphs. Submitted. Available at http://facweb.cs.depaul.edu/ibarra/research.htm, 2006.

[Iba08a] L. Ibarra. Fully dynamic algorithms for chordal graphs and split graphs. *ACM Transactions on Algorithms*, 4(4):40:1–20, 2008. Available at http://facweb.cs.depaul.edu/ibarra/research.htm.

[Iba08b] L. Ibarra. A simple fully dynamic graph algorithm for recognizing proper interval graphs. Submitted. Available at facweb.cs.depaul.edu/ibarra/research.htm, 2008.

[Joh85] D. S. Johnson. The NP-completeness column: an ongoing guide. *J. Algorithms*, 6:434–451, 1985.

[KM89] N. Korte and R. H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Computing*, 18(1):68–81, 1989.

[Lun90] M. Lundquist. *Zero patterns, chordal graphs, and matrix completions*. PhD thesis, Dept. of Mathematical Sciences, Clemson University, 1990.

[MM99] T. A. McKee and F. R. McMorris. *Topics in Intersection Graph Theory*. Society for Industrial and Applied Mathematics, Philadelphia, 1999. Monograph.

[RTL76] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Computing*, 5(2):266–283, 1976.

[Tar75] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.

[TY84] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Computing*, 13(3):566–579, 1984.