A fully dynamic graph algorithm for recognizing proper interval graphs

Louis Ibarra[*]

March 18, 2009

Abstract

We present a fully dynamic graph algorithm to recognize proper interval graphs that runs in $O(\log n)$ worst case time per edge update, where n is the number of vertices in the graph. The algorithm also maintains the connected components and supports connectivity queries in $O(\log n)$ time.

Keywords: chordal graph, clique tree, interval graph, proper interval graph

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 [CLRS01], but fast fully dynamic algorithms are considerably more complicated [HdT01].

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].

[*]College of Computing and Digital Media, DePaul University, 243 S. Wabash Ave., Chicago, IL 60604, U.S.A., email: ibarra@cs.depaul.edu

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 [Iba08]. 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. There is a fully dynamic algorithm for recognizing interval graphs that runs in $O(n \log n)$ time per update [Iba09b]. This algorithm supports updates that yield an interval graph and queries that ask whether a particular update is supported.

Proper interval graphs can also be recognized in $O(m + n)$ time [CKN$^+$95, DHH96]. There is a fully dynamic algorithm for recognizing 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 algorithm supports connectivity queries in $O(\log n)$ time. There is a lower bound of $\Omega(\log n/(\log \log n + \log b))$ amortized time per edge update in the cell probe model of computation with word-size b [HSS01].

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 [Iba09a]. 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 [Iba09a].

1.3 Our results

Our algorithm for recognizing proper interval graphs runs in $O(\log n)$ time per edge update. It also maintains the connected components and supports connectivity queries in $O(\log n)$ time. This matches the running time of the algorithm in [HSS01] for edge updates and connectivity queries, but instead of maintaining a straight enumeration of a proper interval graph as in [HSS01], our algorithm maintains the clique-separator graph of a proper interval graph. The clique-separator graph of a proper interval graph G is a path $\mathcal{P}$ closely related to the clique path P of G. Each vertex v is stored in the leftmost and rightmost nodes of $\mathcal{P}$ that contain v; these nodes specify an interval for v. Each node of $\mathcal{P}$ has a balanced binary tree of the vertices that it stores. When $\mathcal{P}$ is updated, a constant number of trees is moved to the left or right on $\mathcal{P}$, which extends or reduces the intervals for the corresponding vertices. Computing the data structure for a proper interval graph G requires $O(m + n)$ time. Given the data structure, computing an interval representation of G requires $O(n \log n)$ time.

The set of proper interval graphs is a proper subset of the set of interval graphs, which is a proper subset of the set of chordal graphs. Since we will use results for these graph classes, we review them and the clique-separator graph in Section 2. We present the data structure used by the algorithm in Section 3. We present the insert operation in Section 4 and the delete operation in Section 5. We present conclusions in Section 6.

2 Preliminaries

Throughout this paper, let $G = (V, E)$ be a simple, undirected graph and let $n = |V|$ and $m = |E|$. We also write V and E as $V(G)$ and $E(G)$, respectively. 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 $S \subset V$. 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. Figure 1a shows a graph with minimal vertex separators $\{z\}$ and $\{y, z\}$. For $u, v \in V$, S *separates* u and v if S is a uv-separator.

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 is a minimal uv-separator for any $u \in K - S$ and $v \in K' - S$, and furthermore, S is a uv-separator for any $u \in \bar{K} - S$ and $v \in \bar{K}' - S$, where $\bar{K}$ and $\bar{K}'$ are in different subtrees of $T - \{K, K'\}$.*

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. A clique tree can be computed in $O(m+n)$ time [BP93, LPP89, TY84].

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]. Figure 1a shows an interval graph with clique path $(\{x, y, z\}, \{y, z, w\}, \{u, z\}, \{v, z\})$. 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 [Gol04, MM99].

A *proper interval graph* (or *unit interval graph*) is an interval graph with an interval representation where no interval is properly contained in another. Proper interval graphs are a proper subclass of interval graphs. For example, $K_{1,3}$ (the graph consisting of one vertex adjacent to three pairwise nonadjacent vertices) is an interval graph but not a proper interval graph. In fact, an interval graph is a proper interval graph if and only if it has no induced $K_{1,3}$ [Rob69]. $K_{1,3}$ is also called the *claw*. The interval graph in Figure 1a is not a proper interval graph because it has the induced claw $\{u, v, y, z\}$. Proper interval graphs have applications in physical mapping of DNA and in other areas [HSS01, MM99].

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

2.1 The clique-separator 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 $S_2 = \{z\}$.

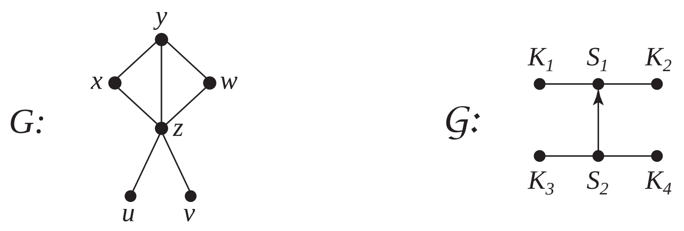


Figure 1: A chordal graph and its clique-separator graph.

The clique-separator graph $\mathcal{G}$ is unique. $\mathcal{G}$ is *connected* if the underlying undirected graph of $\mathcal{G}$ (obtained by replacing every arc with an edge) is connected. $\mathcal{G}$ is connected if and only if G is connected. 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.

A *box* of $\mathcal{G}$ is a connected component of the undirected graph obtained by deleting the arcs of $\mathcal{G}$. 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. Every box is a tree with the clique intersection property [Iba09a]. (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.) For any separator node S with neighbors K and K', $S = K \cap K'$ [Iba09a].

If G is a chordal graph or an interval graph, $\mathcal{G}$ may have many boxes, arcs, and nodes with high degree. If G is a proper interval graph, however, $\mathcal{G}$ has a much simpler structure, as the following theorem shows. If the graph is not connected, the theorem can be applied to each of its connected components.

Theorem 2 ([Iba09a]) *Let G be a connected chordal graph with clique-separator graph $\mathcal{G}$. Then G is a proper interval graph if and only if $\mathcal{G}$ has exactly one box and this box is a path $\mathcal{P}$ such that there do not exist vertices $u \in K_1 \cap K_3$ and $v \in K_2 - (K_1 \cup K_3)$ where (K_1, K_2, K_3) is a subsequence of $\mathcal{P}$.*

Corollary 3 ([Iba09a]) *Let G be a connected chordal graph. Then G is a proper interval graph if and only if G has a unique clique tree and this tree is a path P such that there do not exist vertices $u \in K_1 \cap K_3$ and $v \in K_2 - (K_1 \cup K_3)$ where (K_1, K_2, K_3) is a subsequence of P.*

2.2 Observations

We will use the following observations in the correctness proofs. A *cut edge* is an edge whose removal increases the number of connected components of the graph. For a chordal graph G, we can readily prove that $\{v_1, v_2\}$ is a cut edge of G if and only if $\{v_1, v_2\}$ is a maximal clique of G. The forward direction is easy and the backwards direction uses the fact that in a chordal graph, the shortest cycle (if there is one) containing a given edge is a triangle.

For a connected proper interval graph G, its clique-separator graph $\mathcal{P}$ is unique by definition and its clique path P is unique by Corollary 3. Then by deleting the separator nodes from $\mathcal{P}$, we obtain P, and by inserting separator node $K \cap K'$ between every two consecutive clique nodes K and K' of P, we obtain $\mathcal{P}$. Thus, the clique nodes of $\mathcal{P}$ are the maximal cliques of P in the same order and the separator nodes of $\mathcal{P}$ are the minimal vertex separators corresponding to the edges of P in the same order. Then by Theorem 1-2, if separator node S is between clique nodes $\bar{K}$ and $\bar{K}'$ on $\mathcal{P}$, or equivalently, the $\bar{K}$–$\bar{K}'$ subpath of $\mathcal{P}$ contains S, then S separates any $u \in \bar{K} - S$ and any $v \in \bar{K}' - S$.

3 The data structure for the algorithm

The algorithm maintains the clique-separator graph $\mathcal{G}$ for the proper interval graph G. Each connected component of $\mathcal{G}$ is a path $\mathcal{P}$. Intuitively, each vertex v is stored in the leftmost and rightmost nodes of $\mathcal{P}$ that contain v; by the clique intersection property, every node between these two nodes contains v. Each node N of $\mathcal{P}$ has two balanced binary trees of the vertices that it stores: one tree stores the vertices whose leftmost node is N and the other tree stores the vertices whose rightmost node is N. If v is contained in exactly one node N, then v is stored in both trees of N. When G is updated, $\mathcal{P}$ is updated by adding or deleting a constant number of nodes on $\mathcal{P}$ and moving a constant number of trees to the left or right on $\mathcal{P}$.

A vertex is *simplicial* if the set of its neighbors is a clique. A vertex is simplicial if and only if it is contained in exactly one clique node [BP93]. A vertex is *complex* if it is contained in one or more separator nodes. By the clique intersection property, every vertex is either simplicial or complex.

3.1 The data structure

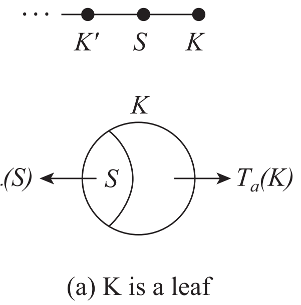
Let G be a proper interval graph with clique-separator graph $\mathcal{G}$. We will store sequences of vertices of G and sequences of nodes of $\mathcal{G}$ in balanced binary trees. In a balanced binary tree such as a B-tree or red-black tree [CLRS01], the height is $O(\log n)$ and an insert, delete, split, or join operation runs in $O(\log n)$ time, where n is the number of leaves in the tree. We will not use the find operation.

Since G has n vertices and $\mathcal{G}$ has at most n clique nodes and $n-1$ separator nodes, every balanced binary tree will have at most $2n-1$ leaves and $O(\log n)$ height.

Each connected component of $\mathcal{G}$ is a path $\mathcal{P}$ and its nodes are stored in a balanced binary tree $T_{\mathcal{P}}$, which gives a left to right ordering of the nodes of $\mathcal{P}$. For each complex vertex v, $leftmost(v)$ and $rightmost(v)$ are the leftmost and rightmost separator nodes of $\mathcal{P}$ that contain v, respectively. Each clique node K has two balanced binary trees: $T_a(K)$ and $T_b(K)$ are identical trees and each contains the simplicial vertices in K. Each separator node S has two balanced binary trees: $T_r(S)$ contains every complex vertex u with $rightmost(u) = S$ and $T_l(S)$ contains every complex vertex u with $leftmost(u) = S$. Thus, every vertex v is contained in exactly two trees: if v is simplicial, then v is in $T_a(K)$ and $T_b(K)$ for some K, and if v is complex, then v is in $T_l(S)$ and $T_r(S')$ for some S and S', where $S = S'$ exactly when v is contained in only one separator node.

The vertices in $T_r(S)$ are sorted by the positions on $\mathcal{P}$ of their $leftmost$ nodes and the vertices in $T_l(S)$ are sorted by the positions on $\mathcal{P}$ of their $rightmost$ nodes. For example, suppose the separator nodes of $\mathcal{P}$ in left to right order are S_1, S_2, S, S_3, S_4. Then $T_r(S)$ contains the vertices whose $leftmost$ node is S_1, followed by the vertices whose $leftmost$ node is S_2, followed by the vertices whose $leftmost$ node is S, and $T_l(S)$ contains the vertices whose $rightmost$ node is S, followed by the vertices whose $rightmost$ node is S_3, followed by the vertices whose $rightmost$ node is S_4. A *block* B is a maximal contiguous sequence of vertices in a T_l or T_r tree such that every vertex in B has the same $rightmost$ or $leftmost$ value, respectively. The first vertex in every block is marked. Each internal node of a T_l or T_r tree is labeled if it is the ancestor of a marked vertex.

Every clique node K has a value $count(K)$, which is the number of vertices v such that the $leftmost(v)$–$rightmost(v)$ subpath of $\mathcal{P}$ contains K, or equivalently, K is between $leftmost(v)$ and $rightmost(v)$ on $\mathcal{P}$. Figure 2 shows the set of vertices of K when K is a leaf or an internal node of $\mathcal{P}$. The figure indicates the trees that contain the simplicial vertices and complex vertices of K. (If K is a leaf, then $count(K) = 0$. If K is an internal node with neighbors S and S', then $count(K) = |S \cap S'|$. By Theorem 2, if $count(K) > 0$, then K contains no simplicial vertices.)



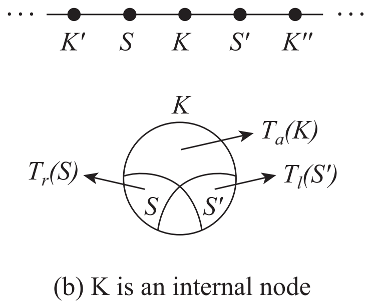


Figure 2: A clique node K on path $\mathcal{P}$.

Figure 6a shows a proper interval graph G and its clique-separator graph $\mathcal{P}$, where $S_1 = \{u, v\}$, $S_2 = \{w, v\}$, and $S_3 = \{x\}$. The first row under $\mathcal{P}$ shows the vertices in the T_a and T_l trees; the second row under $\mathcal{P}$ shows the vertices in the T_b and T_r trees. The vertices are listed in the order in which they appear in each tree, with commas between the blocks. We have $count(K_2) = 1$ and the other $count$ values are 0.

For each connected component $\mathcal{P}$ of $\mathcal{G}$, the algorithm maintains the data structure for $\mathcal{P}$ and the data structure for its reversal $\mathcal{P}^R$. We will present the algorithm for updating $\mathcal{P}$ and omit the algorithm for $\mathcal{P}^R$ since it is symmetric. (Similarly, the algorithm in [HSS01] maintains the straight enumeration and its reversal for each connected component of G.) We will need $\mathcal{P}$ and $\mathcal{P}^R$ when we insert an edge between vertices in different connected components of G.

For nodes N and N', $N <_{\mathcal{P}} N'$ denotes that N is to the left of N' on $\mathcal{P}$. The correctness proofs will use the following lemma, which follows from Theorem 2.

Lemma 4 *Let $S <_{\mathcal{P}} S'$. For any u in $T_r(S)$ and u' in $T_r(S')$, $leftmost(u) \leq_{\mathcal{P}} leftmost(u')$. For any v in $T_l(S)$ and v' in $T_l(S')$, $rightmost(v) \leq_{\mathcal{P}} rightmost(v')$.*

3.2 Basic computations with the data structure

For any vertex v, we decide whether v is simplicial or complex, and we find $leftmost(v)$ and $rightmost(v)$ if v is complex, by following parent pointers from v in the two trees containing v. For any complex vertices u and v, we decide whether $leftmost(u) <_{\mathcal{P}} leftmost(v)$ by finding $leftmost(u)$ and $leftmost(v)$ and then following parent pointers from $leftmost(u)$ and $leftmost(v)$ in $T_{\mathcal{P}}$. We decide whether $rightmost(u) <_{\mathcal{P}} rightmost(v)$ similarly. For any vertices x and y, we decide whether x and y are connected in G by following parent pointers from x and y in the trees containing x and y and then following parent pointers in the $T_{\mathcal{P}}$ trees. Each of these computations runs in $O(\log n)$ time.

For a vertex u in a T_l or T_r tree, let B be the block containing u. We can find the first or last vertex in B by following parent pointers from u until we find a labeled internal node and then following child pointers. We can swap u with the first or last vertex in B and adjust the marks in the appropriate way. We can use the split operation to obtain a tree containing the vertices v of $T_l(S)$ with $rightmost(v) = S$ and a tree containing the vertices v of $T_l(S)$ with $rightmost(v) \neq S$. We denote these trees by $self(T_l(S))$ and $nonself(T_l(S))$, respectively. Similarly, we can process $T_r(S)$ to obtain the trees $self(T_r(S))$ and $nonself(T_r(S))$. Note that $self(T_l(S))$ and $self(T_r(S))$ each contain the vertices v with $leftmost(v) = rightmost(v) = S$. Each of these computations runs in $O(\log n)$ time.

We determine the number of vertices contained in a node in $O(1)$ time, as follows. Let $|T|$ denote the number of vertices stored in tree T. If K is a leaf clique node with left neighbor S (Figure 2a), then $|K| = |T_a(K)| + |T_r(S)|$ and $|S| = |T_r(S)|$. If K is an internal clique node with left neighbor S and right neighbor S' (Figure 2b), then $|K| = |T_a(K)|+|T_r(S)|+|T_l(S')|+count(K)$, $|S| = |T_r(S)| + count(K)$, and $|S'| = |T_l(S')| + count(K)$. These equations are straightforward to prove. For example, suppose K has left neighbor S and right neighbor S'. Then every vertex $v \in K$ is a simplicial vertex in K (counted in $|T_a(K)|$), or a complex vertex in $S - S'$ (counted in $|T_r(S)|$), or a complex vertex in $S' - S$ (counted in $|T_l(S')|$), or a complex vertex in $S \cap S'$ (counted in $count(K)$). (The vertices in $T_l(S)$ or $T_r(S')$ are in $S \cap S'$.)

When an operation inserts a vertex v into a balanced binary tree T, it tests whether v is in the same block as the vertex before or after v in T and adjusts the marks in the appropriate way. The operations use the following subroutines to move a vertex v in a balanced binary tree T.

$MoveLeft(v, T)$:

In T, move v into the first position in the block containing v and mark the vertex after v. If v is in the same block as the vertex before v, then unmark v.

$MoveRight(v, T)$:

In T, move v into the last position in the block containing v and mark v. If v is in the same block as the vertex w after v, then unmark w.

When we refer to both $T_a(K)$ and $T_b(K)$, or perform the same operation on both $T_a(K)$ and $T_b(K)$, for brevity we will write $T_{a+b}(K)$. Similarly, we will write $T_{l+r}(S)$. We denote the balanced binary tree operations as follows: $\leftarrow$ is assignment, $\cup$ is join, $-$ is deletion, and $\emptyset$ is an empty tree.

In a join $T \cup T'$, the vertices in T are placed before the vertices in T'. In a deletion $T - v$, the marks are adjusted in the appropriate way if v is a marked vertex.

3.3 Computing the data structure and an interval representation

There are several algorithms that decide whether G is a chordal graph, and if it is, compute a clique tree T of G [BP93, Gol04]. These algorithms produce a list of the vertices in each maximal clique of T and run in $O(m+n)$ time. If we know G is a proper interval graph, then by Corollary 3, any of these algorithms computes the clique path of each connected component of G. We can verify that G is a proper interval graph in $O(m+n)$ time [CKN$^+$95, DHH96] and then compute the clique path of each connected component of G.

Given the clique path P of a connected proper interval graph G and the lists of the vertices in each maximal clique of P, we find the first and last maximal cliques that contain each vertex and thereby identify the simplicial vertices and the complex vertices of G. We compute the clique-separator graph $\mathcal{P}$ of G by inserting a separator node between every two consecutive clique nodes of P. For every clique node K, we make two lists of the simplicial vertices in K. For every separator node S, we make a list L_l of the vertices u with $leftmost(u) = S$ and a list L_r of the vertices v with $rightmost(v) = S$. In one pass through $\mathcal{P}$, we sort the vertices in the L_l lists by their *rightmost* node, sort the vertices in the L_r lists by their *leftmost* node, and compute the *count* values. Each of these steps requires $O(m+n)$ time. Then we build the T_a, T_b, T_l, and T_r trees in $O(m+n)$ time. Lastly, we build the $T_{\mathcal{P}}$ tree in $O(n)$ time since there are at most $2n-1$ nodes in $\mathcal{P}$. Thus, we construct the data structure in $O(m+n)$ time.

Given the data structure for a proper interval graph G, we compute an interval representation of G as follows. Let $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$ be the connected components of the clique-separator graph $\mathcal{G}$ of G. Let $|\mathcal{P}_i|$ be the number of nodes in $\mathcal{P}_i$. Rank the nodes of $\mathcal{P}_1$ from 1 to $|\mathcal{P}_1|$, the nodes of $\mathcal{P}_2$ from $|\mathcal{P}_1|+1$ to $|\mathcal{P}_1|+|\mathcal{P}_2|$, and so on. Let $rank(N)$ be the rank of node N. Every simplicial vertex in clique node K has the interval $[rank(K) - 1/2, rank(K) + 1/2]$. Every complex vertex v has the interval $[rank(leftmost(v)) - 1, rank(rightmost(v)) + 1]$. Then two vertices are adjacent in G if and only if their intervals intersect. Since computing *leftmost* and *rightmost* requires $O(\log n)$ time for each vertex, computing the interval representation requires $O(n \log n)$ time.

4 The insert operation

Let G be a chordal graph (not necessarily a proper interval graph). Throughout this section, $\{v_1, v_2\} \notin E$. We use $G + \{v_1, v_2\}$ to denote $(V, E \cup \{\{v_1, v_2\}\})$.

Theorem 5 ([Iba08]) *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 $\{K_1, K_2\} \in E(T)$.*

Suppose $v_1 \in K_1$ and $v_2 \in K_2$ and $\{K_1, K_2\} \in E(T)$. Let $S = K_1 \cap K_2$. Then $v_1 \in K_1 - S$ and $v_2 \in K_2 - S$ (Figure 3). Then $K = S \cup \{v_1, v_2\}$, which is not a maximal clique in G, is a maximal clique in $G + \{v_1, v_2\}$. Now K_1 and K_2 may or may not be maximal cliques in $G + \{v_1, v_2\}$; each K_i is a maximal clique in $G + \{v_1, v_2\}$ if and only if $K_i \supset S \cup \{v_i\}$, or equivalently, $|K_i - S| > 1$. Furthermore, K_1 and K_2 are the only maximal cliques of G that can be destroyed by inserting $\{v_1, v_2\}$ [Iba08]. For example, consider the chordal graph G in Figure 1. Inserting edge $\{u, x\}$ yields a chordal graph because there is a clique path of G where the maximal cliques $\{u, z\}$ and $\{x, y, z\}$ are adjacent. Moreover, $\{u, x, z\}$ and $\{x, y, z\}$ are maximal cliques in $G + \{v_1, v_2\}$, but $\{u, z\}$ is not a maximal clique in $G + \{v_1, v_2\}$.

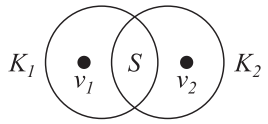


Figure 3: Inserting edge $\{v_1, v_2\}$.

Let G be a proper interval graph with clique-separator graph $\mathcal{G}$. Let $\{v_1, v_2\}$ be the edge to be inserted into G. Since $\{v_1, v_2\} \notin E$, no clique node or separator node contains both v_1 and v_2. We have two main cases, depending on whether or not v_1 and v_2 are in the same connected component of G.

Suppose v_1 and v_2 are in different connected components of G. Then there is always a clique tree T of G such that $v_1 \in K_1$ and $v_2 \in K_2$ and $\{K_1, K_2\} \in E(T)$, where $K_1 \cap K_2 = \emptyset$. Then $G + \{v_1, v_2\}$ is a chordal graph but possibly not a proper interval graph. Let v_1 and v_2 be vertices in connected components G_1 and G_2 of G with clique-separator graphs $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. We will later show that $G+\{v_1, v_2\}$ is a proper interval graph if and only if v_1 and v_2 are simplicial vertices of some clique nodes K_1 and K_2 that are endpoints of $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. If K_1 and K_2 do not exist, the operation rejects the insertion. Otherwise, the operation merges $\mathcal{P}_1$ and $\mathcal{P}_2$ by linking them with path (S_1, K, S_2) where $S_1 = \{v_1\}$ and $K = \{v_1, v_2\}$ and $S_2 = \{v_2\}$. But if v_i is the only vertex in G_i, then $K_i = \{v_i\}$ is not a maximal clique of $G + \{v_1, v_2\}$ and so the operation deletes K_i and S_i. (We present the operation this way for brevity. The operation can be readily rewritten so that it creates only the nodes that are not deleted.)

Suppose v_1 and v_2 are in the same connected component of G. Let $\mathcal{P}$ be the clique-separator graph of this connected component. By Theorem 5, $G + \{v_1, v_2\}$ is chordal if and only if $\mathcal{P}$ has a separator node S with left neighbor K_1 and right neighbor K_2 (implying $S = K_1 \cap K_2$) such that $v_1 \in K_1 - S$ and $v_2 \in K_2 - S$. If $G + \{v_1, v_2\}$ is not a proper interval graph, the operation rejects the insertion. Otherwise, the operation replaces S with path (S_1, K, S_2) where $S_1 = S \cup \{v_1\}$ and $K = S \cup \{v_1, v_2\}$ and $S_2 = S \cup \{v_2\}$. But if K_i is not a maximal clique of $G+\{v_1, v_2\}$, the operation deletes K_i and does not add S_i.

When the operation creates a node, its two trees are initially empty and its *count* value (if it is a clique node) is initially 0.

$Insert(v_1, v_2)$:

v_1 and v_2 are in different connected components of G:

Let v_1 and v_2 be vertices in connected components G_1 and G_2 of G with clique-separator graphs $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. If v_1 and v_2 are simplicial vertices of some clique nodes K_1 and K_2 that are endpoints of $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively, then continue, and otherwise, reject. Assume $\mathcal{P}_1$ has right endpoint K_1 and $\mathcal{P}_2$ has left endpoint K_2 (Figure 4a). We will merge $\mathcal{P}_1 + \mathcal{P}_2$ and merge $\mathcal{P}_2^R + \mathcal{P}_1^R$; we describe only the former since the latter is symmetric. The other cases, e.g. $\mathcal{P}_1$ has left endpoint K_1 and $\mathcal{P}_2$ has left endpoint K_2, are similar.

Add path (S_1, K, S_2) and edges $\{K_1, S_1\}$ and $\{S_2, K_2\}$, where $S_1 = \{v_1\}$ and $K = \{v_1, v_2\}$ and $S_2 = \{v_2\}$ (Figure 4b). If v_1 is not the only vertex in G_1, move v_1 from $T_{a+b}(K_1)$ into $T_{l+r}(S_1)$, and otherwise, move v_1 from $T_{a+b}(K_1)$ into $T_{a+b}(K)$ and delete K_1 and S_1 (Figure 4c). If v_2 is not the only vertex in G_2, move v_2 from $T_{a+b}(K_2)$ into $T_{l+r}(S_2)$, and otherwise, move v_2 from $T_{a+b}(K_2)$ into $T_{a+b}(K)$ and delete K_2 and S_2.

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

Let v_1 and v_2 be vertices in a connected component of G with clique-separator

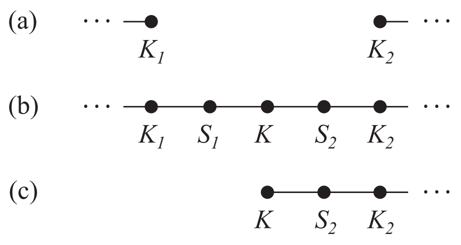


Figure 4: v_1 and v_2 are in different connected components of G.

graph $\mathcal{P}$. If there is a separator node S with left neighbor K_1 and right neighbor K_2 such that $v_1 \in K_1$ and $v_2 \in K_2$ (Figure 5a), then continue, and otherwise, reject. Note v_1 is in $T_{a+b}(K_1)$ or $T_r(S'_1)$, where S'_1 is the left neighbor of K_1 (if it exists), and v_2 is in $T_{a+b}(K_2)$ or $T_l(S'_2)$, where S'_2 is the right neighbor of K_2 (if it exists). If v_1 is in $T_r(S'_1)$ and either $|T_{a+b}(K_1)| > 0$ or $leftmost(v_1) <_{\mathcal{P}} leftmost(u)$ for the last vertex u in $T_r(S'_1)$, then reject. If the symmetric condition holds with v_2, then reject. Let $K = S \cup \{v_1, v_2\}$ and $S_1 = S \cup \{v_1\}$ and $S_2 = S \cup \{v_2\}$.

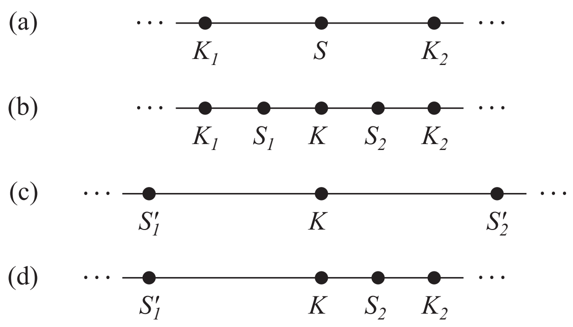


Figure 5: v_1 and v_2 are in the same connected component of G.

If $|K_1 - S| > 1$ and $|K_2 - S| > 1$, then do the following.

1. Replace node S with path (S_1, K, S_2) (Figure 5b). Do $T_l(S_1) \leftarrow T_l(S)$ and $T_r(S_2) \leftarrow T_r(S)$. Do $count(K) \leftarrow count(K_1) + |T_l(S_1)|$.
2. If v_1 is in $T_{a+b}(K_1)$, move v_1 to $T_{l+r}(S_1)$ (make it the first vertex in $T_l(S_1)$). If v_1 is in $T_r(S'_1)$, move v_1 to $T_r(S_1)$, increment $count(K_1)$, and call $MoveRight(v_1, T_l(leftmost(v_1)))$.
3. If v_2 is in $T_{a+b}(K_2)$, move v_2 to $T_{l+r}(S_2)$ (make it the last vertex in $T_r(S_2)$). If v_2 is in $T_l(S'_2)$, move v_2 to $T_l(S_2)$, increment $count(K_2)$, and call $MoveLeft(v_2, T_r(rightmost(v_2)))$.

If $|K_1 - S| = 1$ and $|K_2 - S| = 1$, then do the following.

1. Replace path (K_1, S, K_2) with node K (Figure 5c). Do $T_{a+b}(K) \leftarrow self(T_{l+r}(S))$ and $count(K) \leftarrow count(K_1) - |nonself(T_r(S))|$.
2. If K_1 was an endpoint of $\mathcal{P}$, add v_1 to $T_{a+b}(K)$. Otherwise, do $T_r(S'_1) \leftarrow T_r(S'_1) \cup nonself(T_r(S))$ and call $Check(x, T_l(leftmost(x)))$ for the first vertex x in the old $nonself(T_r(S))$.
3. If K_2 was an endpoint of $\mathcal{P}$, add v_2 to $T_{a+b}(K)$. Otherwise, do $T_l(S'_2) \leftarrow nonself(T_l(S)) \cup T_l(S'_2)$ and call $Check(x, T_r(rightmost(x)))$ for the first vertex x in the old $T_l(S'_2)$.

If $|K_1 - S| = 1$ and $|K_2 - S| > 1$, then do the following. Rename K_1 to K and rename S to S_2 (Figure 5d). If v_2 is in $T_{a+b}(K_2)$, move v_2 to $T_{l+r}(S_2)$

(make it the first vertex in $T_l(S_2)$ and the last vertex in $T_r(S_2)$). If v_2 is in $T_l(S_2')$, move v_2 to $T_l(S_2)$ (make it the last vertex), increment $count(K_2)$, and call $MoveLeft(v_2, T_r(rightmost(v_2)))$.

If $|K_1 - S| > 1$ and $|K_2 - S| = 1$, then this is symmetric to the previous case.

$Check(x, T)$:

Let w be the rightmost marked vertex in T such that $w \leq x$ in the vertex order of T. If the vertex before w is in the same block as w, then unmark w.

End Insert

We show how the operation inserts three edges individually into the graph G in Figure 6a. (a) In $Insert(v, y)$, S_3 has neighbors K_3 and K_4 where $v \in K_3$ and $y \in K_4$; v is in $T_r(S_2)$ and y is in $T_{a+b}(K_4)$. Since $leftmost(v) <_{\mathcal{P}} leftmost(w)$ and w is the last vertex in $T_r(S_2)$, the operation rejects. Note $G + \{v, y\}$ is not a proper interval graph since it has the claw $\{v, t, w, y\}$. (b) In $Insert(w, z)$, S_3 has neighbors K_3 and K_4 where $w \in K_3$ and $z \in K_4$; w is in $T_r(S_2)$ and z is in $T_{a+b}(K_4)$. Since $|K_3 - S_3| > 1$ and $|K_4 - S_3| > 1$, the operation replaces S_3 with path (S', K, S''), where $S' = \{w, x\}$ and $K = \{w, x, z\}$ and $S'' = \{x, z\}$ (Figure 6b), and sets $count(K) = count(K_3) + |T_l(S')| = 0 + 1 = 1$. The operation moves w from $T_r(S_2)$ into $T_r(S')$, sets $count(K_3) = 1$, calls $MoveRight(w, T_l(leftmost(w)))$, and moves z from $T_{a+b}(K_4)$ into $T_{l+r}(S'')$. (c) In $Insert(u, x)$, S_2 has neighbors K_2 and K_3 where $u \in K_2$ and $x \in K_3$; u is in $T_r(S_1)$ and x is in $T_l(S_3)$. Since $|K_2 - S_2| = 1$ and $|K_3 - S_2| = 1$, the operation replaces path (K_2, S_2, K_3) with node $K = \{u, v, w, x\}$ (Figure 6c) and sets $count(K) = count(K_2) - |nonself(T_r(S_2))| = 1 - 1 = 0$. Since K_2 was not an endpoint of $\mathcal{P}$, the operation does the join with $nonself(T_r(S_2)) = \{v\}$ and calls $Check(v, T_l(leftmost(v)))$. Since K_3 was not an endpoint of $\mathcal{P}$ and $nonself(T_l(S_2)) = \emptyset$, the operation is done.

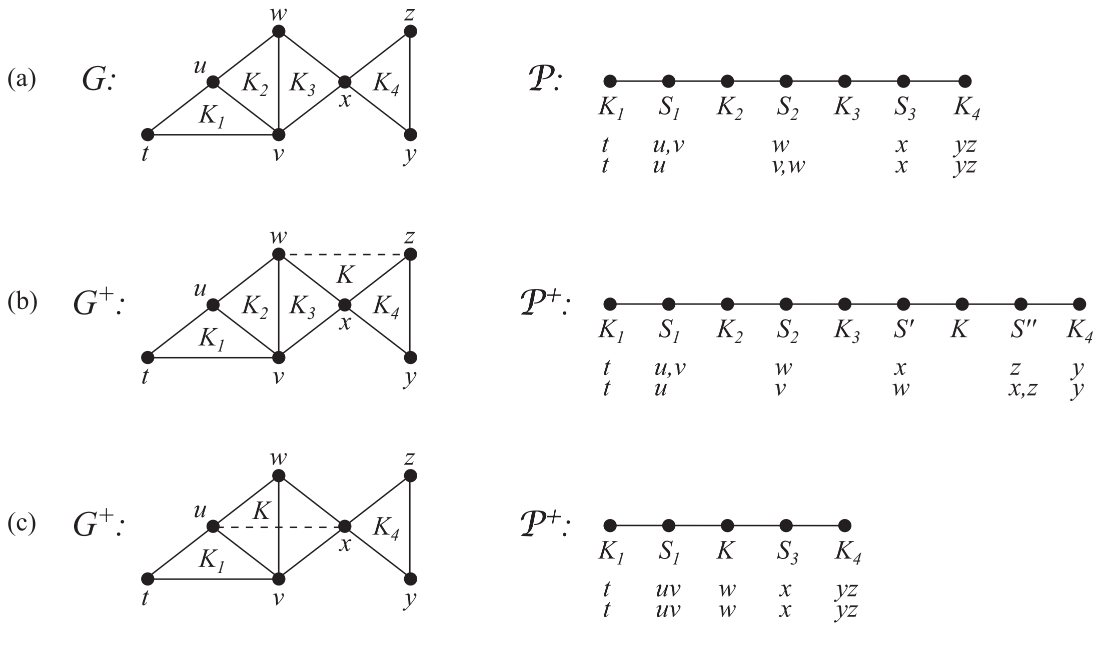


Figure 6: Inserting edges into a proper interval graph G.

For a more complicated example, we show how the operation inserts an edge into the graph G in Figure 7a. We have $S_1 = \{u, v, w\}$, $S_2 = \{w, x\}$, $count(K_2) = 1$, and the other $count$ values are 0. In $Insert(v, y)$, S_2 has neighbors K_2 and K_3 where $v \in K_2$ and $y \in K_3$; v is in $T_r(S_1)$ and

y is in $T_{a+b}(K_3)$. Since $|K_2 - S_2| > 1$ and $|K_3 - S_2| > 1$, the operation replaces S_2 with path (S', K, S''), where $S' = \{v, w, x\}$ and $K = \{v, w, x, y\}$ and $S'' = \{w, x, y\}$, does $T_l(S') \leftarrow T_l(S_2)$ and $T_r(S'') \leftarrow T_r(S_2)$, and sets $count(K) = count(K_2) + |T_l(S')| = 1 + 1 = 2$. The operation moves v from $T_r(S_1)$ to $T_r(S')$, sets $count(K_2) = 2$, calls $MoveRight(v, T_l(leftmost(v)))$, and moves y from $T_{a+b}(K_3)$ to $T_{l+r}(S'')$, making y the last vertex in $T_r(S'')$.

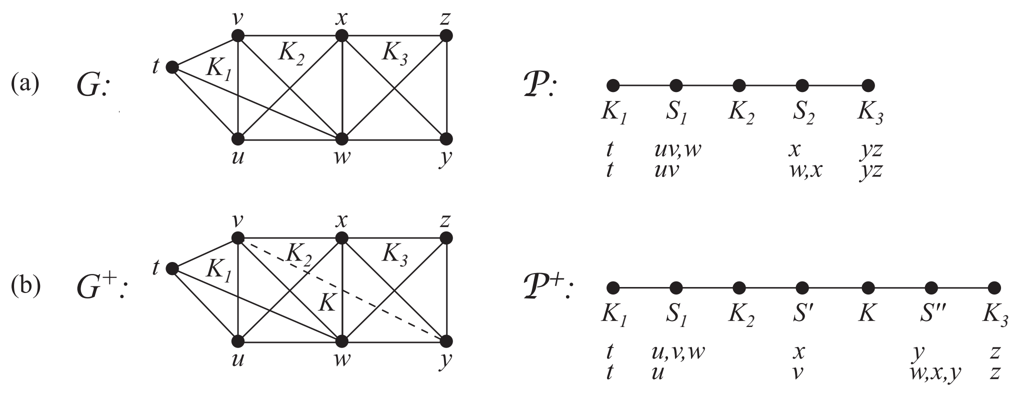


Figure 7: Inserting edge $\{v, y\}$ into a proper interval graph G.

The *Insert* operation uses a constant number of the computations discussed in Subsection 3.2 so it runs in $O(\log n)$ time, where n is the number of vertices of G. In the correctness proofs, we will implicitly use the fact that if a separator node S is between clique nodes $\bar{K}$ and $\bar{K}'$ on $\mathcal{P}$, then S separates any $u \in \bar{K} - S$ and any $v \in \bar{K}' - S$, which implies that u and v are not adjacent.

Theorem 6 *Let G be a proper interval graph with clique-separator graph $\mathcal{G}$. Let v_1 and v_2 be vertices in different connected components of G. If $G^+ = G + \{v_1, v_2\}$ is a proper interval graph, the Insert operation computes its clique-separator graph $\mathcal{G}^+$, and otherwise, the operation rejects.*

Proof Let v_1 and v_2 be vertices in connected components G_1 and G_2 of G with clique-separator graphs $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. Suppose the operation rejects because v_1 is not a simplicial vertex; the v_2 case is symmetric. Then $v_1 \in S$ for at least one separator node S of $\mathcal{P}_1$. Then S has neighbors K_1 and K_2 in $\mathcal{P}_1$. Let $w_1 \in K_1 - S$ and $w_2 \in K_2 - S$ be any vertices. Then v_1 is adjacent to nonadjacent vertices w_1 and w_2 in G. Then $\{v_1, v_2, w_1, w_2\}$ is an induced claw in G^+. Thus, G^+ is not a proper interval graph.

Suppose the operation rejects because v_1 is a simplicial vertex of a clique node K' that is not an endpoint of $\mathcal{P}_1$; the v_2 case is symmetric. Then $\mathcal{P}_1$ has a subpath (K_1, S_1, K', S_2, K_2). Since S_1 and S_2 are neighbors of K, $S_1 \not\subset S_2$ and $S_2 \not\subset S_1$. Then there exist vertices $u_1 \in S_1 - S_2$ and $u_2 \in S_2 - S_1$ (Figure 4a). Let $w_1 \in K_1 - S_1$ and $w_2 \in K_2 - S_2$ be any vertices. Then v_1, u_1, u_2 are pairwise adjacent and v_2, w_1, w_2 are pairwise nonadjacent in G^+. Then $\{v_1, u_1, u_2, v_2, w_1, w_2\}$ is an induced net in G^+ (Figure 8b). Since the net is not an interval graph [MM99], G^+ is not an interval graph. (If H is an interval graph, then every induced subgraph of H is an interval graph.)

Suppose the operation does not reject. Then v_1 and v_2 are simplicial vertices of some clique nodes K_1 and K_2 that are endpoints of $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. Assume $\mathcal{P}_1$ has right endpoint K_1 and $\mathcal{P}_2$ has left endpoint K_2 (Figure 4a); the other cases are similar. We will show that the operation correctly computes $\mathcal{G}^+$. Recall the relationship between the clique path of G and the clique-separator graph of G (see Subsection 2.2). Since $K = \{v_1, v_2\}$ is a cut edge in G^+, K is a maximal clique in G^+. Let P_1 and P_2 be the clique paths of G_1 and G_2. Suppose v_1 and v_2 are not the only vertices in G_1 and G_2. Then K_1 and K_2 are also maximal cliques in G^+. Link P_1 and

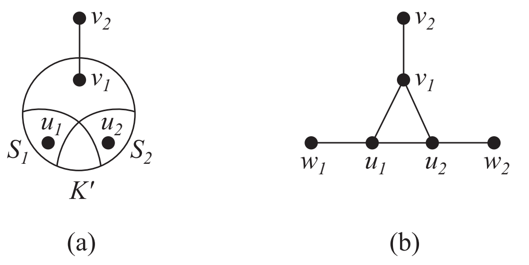


Figure 8: (a) Clique node K' and (b) the net.

P_2 by adding K and edges $\{K_1, K\}$ and $\{K, K_2\}$. Then the new path P^+ is a clique path of G^+, which implies G^+ is an interval graph. Since v_1 is a simplicial vertex of G, any two neighbors of v_1 in G are adjacent. Then G^+ has no induced claw and thus G^+ is a proper interval graph. Since $S_1 = \{v_1\} = K_1 \cap K$ and $S_2 = \{v_2\} = K \cap K_2$ are minimal vertex separators of G^+, the operation correctly computes $\mathcal{G}^+$ (Figure 4b). Now v_1 is a simplicial vertex of K_1 in G and a complex vertex in G^+. Since v_1 is contained in S_1 and no other separator node of $\mathcal{G}^+$, the operation moves v_1 from $T_{a+b}(K_1)$ into $T_{l+r}(S_1)$. The v_2 argument is symmetric.

If v_1 is the only vertex in G_1, then $K_1 = \{v_1\}$ is not a maximal clique in G^+ and $S_1 = \{v_1\}$ is not a separator in G^+. Therefore, the operation deletes K_1 and S_1 and moves v_1 from $T_{a+b}(K_1)$ into $T_{a+b}(K)$ because v_1 is a simplicial vertex of K in G^+ (Figure 4c). If v_2 is the only vertex in G_2, the operation does the symmetric action. If v_1 and v_2 are the only vertices in G_1 and G_2, respectively, then K is a connected component of G^+ and K is an isolated clique node of $\mathcal{G}^+$. Thus, in every case the operation correctly computes $\mathcal{G}^+$ and correctly updates the balanced binary trees. $\square$

Theorem 7 *Let G be a proper interval graph with clique-separator graph $\mathcal{G}$. Let v_1 and v_2 be vertices in the same connected component of G. If $G^+ = G + \{v_1, v_2\}$ is a proper interval graph, the Insert operation computes its clique-separator graph $\mathcal{G}^+$, and otherwise, the operation rejects.*

Proof Let v_1 and v_2 be vertices in a connected component of G with clique-separator graph $\mathcal{P}$. Suppose S does not exist. Then G^+ is not a chordal graph by Theorem 5 and the operation rejects. Suppose S exists (Figure 5a). Then G^+ is a chordal graph. We will show that the operation rejects if and only if G^+ has an induced claw.

Suppose the operation rejects. Assume v_1 is in $T_r(S'_1)$; the v_2 case is symmetric. Suppose $|T_{a+b}(K_1)| > 0$. Let v'_1 be a simplicial vertex of K_1. Then $v'_1 \notin S$ and thus v'_1 is not adjacent to v_2. Let $v''_1 \in K'_1 - S'_1$ where K'_1 is the left neighbor of S'_1. Then v''_1 is adjacent to neither v'_1 nor v_2. Now v_1 is adjacent to v''_1 and v'_1 (since $v_1 \in S'_1$) and adjacent to v_2 in G^+. Therefore, $\{v_1, v''_1, v'_1, v_2\}$ is an induced claw in G^+. Suppose $leftmost(v_1) <_{\mathcal{P}} leftmost(u)$ for the last vertex u in $T_r(S'_1)$. Let $\hat{K}$ be the left neighbor of $\hat{S} = leftmost(v_1)$. Let $\hat{v} \in \hat{K} - \hat{S}$. Then $\hat{v}$ is adjacent to neither u nor v_2. Moreover, u is not adjacent to v_2. Now v_1 is adjacent to $\hat{v}, u$, and v_2 in G^+. Therefore, $\{v_1, \hat{v}, u, v_2\}$ is an induced claw in G^+.

Conversely, suppose G^+ has an induced claw. Since G has no induced claw, v_1 and v_2 must be adjacent vertices in the claw in G^+. Assume v_1 is the degree-3 vertex in the claw; the v_2 case is symmetric. Then $\{v_1, v_2, u, u'\}$ is the induced claw for some u and u'. Since v_1 has nonadjacent neighbors u and u', v_1 is not in $T_{a+b}(K_1)$. Then v_1 is in $T_r(S'_1)$. Let $\hat{K}$ be the left neighbor of $\hat{S} = leftmost(v_1)$. Note $S'_1 = rightmost(v_1)$. Since u and u' are adjacent to v_1 in G, u and u' are contained in nodes to the left of S on $\mathcal{P}$. (Since u and u' are not adjacent to v_2, neither is contained in S.) Since G satisfies Theorem 2, u is not a simplicial vertex of any clique node between $\hat{S}$ and S'_1 on $\mathcal{P}$, although u may be a simplicial vertex of $\hat{K}$ or K_1. Similarly, u is not a complex vertex such that $\hat{S} <_{\mathcal{P}} \mathit{leftmost}(u)$ and $\mathit{rightmost}(u) <_{\mathcal{P}} S'_1$. The same is true for u'. Since u and u' are adjacent to v_1, it follows that $u \in \hat{K}$ and $u' \in K_1$, or vice versa. Suppose the former holds. If u' is a simplicial vertex of K_1, then the operation rejects. Otherwise, u' is a complex vertex in $T_r(S'_1)$. Then since u' is not adjacent to u, $\mathit{leftmost}(v_1) <_{\mathcal{P}} \mathit{leftmost}(u')$. Then $\mathit{leftmost}(v_1) <_{\mathcal{P}} \mathit{leftmost}(w)$ for the last vertex w in $T_r(S'_1)$. Therefore, the operation rejects.

Assuming the operation does not reject, we next show that it correctly computes $\mathcal{G}^+$. Recall the relationship between the clique path of G and the clique-separator graph of G (see Subsection 2.2). If $|K_1 - S| > 1$ and $|K_2 - S| > 1$, then K_1 and K_2 are maximal cliques of G^+. Modify P by inserting K between K_1 and K_2. Then the modified path P^+ is a clique path of G^+. Then G^+ is an interval graph, and since G^+ has no induced claw, G^+ is a proper interval graph. Then by inserting the separator node $K' \cap K''$ between every two consecutive clique nodes K' and K'' on P^+, we obtain $\mathcal{P}^+$. Since $S_1 = K_1 \cap K$ and $S_2 = K \cap K_2$, the operation correctly computes $\mathcal{P}^+$ (Figure 5b). If $|K_1 - S| = 1$ and $|K_2 - S| = 1$, then K_1 and K_2 are not maximal cliques of G^+. Modify P by replacing K_1 and K_2 with K. Then the modified path P^+ is a clique path of G^+ and as above, we can readily show that the operation correctly computes $\mathcal{P}^+$ (Figure 5c). If $|K_1 - S| = 1$ and $|K_2 - S| > 1$, or vice versa, similar arguments show that the operation correctly computes $\mathcal{P}^+$.

We next show that in every case the operation correctly updates the balanced binary trees. Suppose $|K_1 - S| > 1$ and $|K_2 - S| > 1$ (Figure 5b). Recall that $S_1 = S \cup \{v_1\}$ and $K = S \cup \{v_1, v_2\}$ and $S_2 = S \cup \{v_2\}$. Then for every vertex u in $T_l(S)$, $\mathit{leftmost}(u) = S$ in $\mathcal{P}$ and $\mathit{leftmost}(u) = S_1$ in $\mathcal{P}^+$. Therefore, step 1 does $T_l(S_1) \leftarrow T_l(S)$, and likewise, $T_r(S_2) \leftarrow T_r(S)$. Since K has no simplicial vertices, $T_{a+b}(K) = \emptyset$. Then after step 1, the vertices are stored correctly except for v_1 and v_2, which we consider shortly.

The value $\mathit{count}(K)$ is the number of vertices v such that $\mathit{leftmost}(v) <_{\mathcal{P}^+} K <_{\mathcal{P}^+} \mathit{rightmost}(v)$. This equals the sum of (1) the number of vertices in $T_l(S_1)$ plus the number of vertices in $T_r(S_2)$ minus the number of vertices in both $T_l(S_1)$ and $T_r(S_2)$, and (2) the number of vertices u such that $\mathit{leftmost}(u) <_{\mathcal{P}} K_1$ and $K_2 <_{\mathcal{P}} \mathit{rightmost}(u)$. Let $\mathit{self}(S)$ be the number of vertices v with $\mathit{leftmost}(v) = \mathit{rightmost}(v) = S$ in $\mathcal{P}$. Then (1) equals $|T_l(S_1)| + |T_r(S_2)| - \mathit{self}(S)$ because every vertex in both $T_l(S)$ and $T_r(S)$ in $\mathcal{P}$ becomes a vertex in both $T_l(S_1)$ and $T_r(S_2)$ in $\mathcal{P}^+$. And (2) equals $\mathit{count}(K_1) - (|T_r(S)| - \mathit{self}(S))$ because $|T_r(S)| - \mathit{self}(S)$ is the number of vertices w such that $\mathit{leftmost}(w) <_{\mathcal{P}} K_1$ and $\mathit{rightmost}(w) = S$. Now $|T_r(S)|$ in $\mathcal{P}$ equals $|T_r(S_2)|$ in $\mathcal{P}^+$. Then $\mathit{count}(K) = |T_l(S_1)| + |T_r(S_2)| - \mathit{self}(S) + \mathit{count}(K_1) - (|T_r(S_2)| - \mathit{self}(S)) = \mathit{count}(K_1) + |T_l(S_1)|$. (By symmetry, this equals $\mathit{count}(K_2) + |T_r(S_2)|$.)

If v_1 is in $T_{a+b}(K_1)$, then step 2 moves v_1 to $T_{l+r}(S_1)$ because it is a complex vertex with $\mathit{leftmost}(v_1) = \mathit{rightmost}(v_1) = S_1$ in $\mathcal{P}^+$. If v_1 is in $T_r(S'_1)$, then step 2 moves v_1 to $T_r(S_1)$ because it is a complex vertex with $\mathit{rightmost}(v_1) = S_1$ in $\mathcal{P}^+$; since $\mathit{rightmost}(v_1)$ changed, we may need to move v_1 in $T_l(\mathit{leftmost}(v_1))$ to maintain the sorted order. This is done by the *MoveRight* call. Since $\mathit{rightmost}(v_1)$ changed from S'_1 to S_1, v_1 is moved to the immediate right of its former block in $T_l(\mathit{leftmost}(v_1))$. Moreover, since v_1 is the only vertex with $\mathit{rightmost}(v_1) = S_1$, v_1 forms a new block and v_1 is marked. Therefore, *MoveRight* correctly updates $T_l(\mathit{leftmost}(v_1))$. The argument for v_2 is symmetric. Thus, the balanced binary trees are updated correctly in this case. In the remaining cases, we will use the following claim.

Claim Suppose $|K_i - S| = 1$ for $i = 1$ or $i = 2$. Then v_i is a simplicial vertex of K_i if and only if K_i is an endpoint of $\mathcal{P}$.

Proof Suppose $|K_1 - S| = 1$; the other case is symmetric. Then $K_1 = S \cup \{v_1\}$. ($\Leftarrow$) Suppose K_1 is an endpoint of $\mathcal{P}$, i.e., K_1 has no left neighbor. Since $v_1 \in K_1 - S$, it follows by the clique intersection property that v_1 is simplicial. ($\Rightarrow$) Suppose v_1 is a simplicial vertex of K_1. Then K_1 is the only clique node that contains v_1. It follows that if K_1 has a left neighbor S', then $S' \subset S$, which implies that $\mathcal{G}$ has at least one arc. But by Theorem 2, each connected component of $\mathcal{G}$ has exactly one box and thus $\mathcal{G}$ has no arcs, a contradiction. Thus, K_1 is an endpoint of $\mathcal{P}$. $\square$

Suppose $|K_1 - S| = 1$ and $|K_2 - S| = 1$ (Figure 5c). Then $K_1 = S \cup \{v_1\}$ and $K_2 = S \cup \{v_2\}$. Then every vertex v with $\mathit{leftmost}(v) = \mathit{rightmost}(v) = S$ in $\mathcal{P}$ is a simplicial vertex of K in $\mathcal{P}^+$, so the operation does $T_{a+b}(K) \leftarrow \mathit{self}(T_{l+r}(S))$. The value $\mathit{count}(K)$ equals the number of vertices u such that $\mathit{leftmost}(u) <_{\mathcal{P}} K_1$ and $K_2 <_{\mathcal{P}} \mathit{rightmost}(u)$. This equals $\mathit{count}(K_1) - (|T_r(S)| - \mathit{self}(S)) = \mathit{count}(K_1) - |\mathit{nonself}(T_r(S))|$. If K_1 was an endpoint of $\mathcal{P}$, then by the claim v_1 was a simplicial vertex of K_1 and v_1 becomes a simplicial vertex of K, so the operation adds it to $T_{a+b}(K)$. Otherwise, v_1 was a complex vertex in $T_r(S'_1)$ and so v_1 does not need to be moved. Now for every vertex u in $\mathit{nonself}(T_r(S))$, $\mathit{rightmost}(u) = S$ in $\mathcal{P}$ and $\mathit{rightmost}(u) = S'_1$ in $\mathcal{P}^+$, so the operation does $T_r(S'_1) \leftarrow T_r(S'_1) \cup \mathit{nonself}(T_r(S))$. By Lemma 4, the new $T_r(S'_1)$ is sorted properly. Now for every vertex v in the old $\mathit{nonself}(T_r(S))$, we may need to update $T_l(\mathit{leftmost}(v))$.

Let x be the first vertex in the old $\mathit{nonself}(T_r(S))$. Since $\mathit{rightmost}(x)$ changed from S to S'_1, we may need to update $T_l(\mathit{leftmost}(x))$. This is done by the *Check* call. Let w be the rightmost marked vertex in $T_l(\mathit{leftmost}(x))$ such that $w \leq x$. Then w was the first vertex in the old block of $T_l(\mathit{leftmost}(x))$ that contained x and so w was in the old $\mathit{nonself}(T_r(S))$. Then $\mathit{rightmost}(w)$ changed from S to S'_1 and now $\mathit{rightmost}(w) = S'_1$. Then we should unmark w if the vertex w' before w has $\mathit{rightmost}(w') = S'_1$, i.e., w' and w are in the same block. (If w' is in the same block as w, then w' was in the old $T_r(S'_1)$.) No other changes are needed in $T_l(\mathit{leftmost}(x))$, so *Check* correctly updates $T_l(\mathit{leftmost}(x))$.

Let y be any vertex in the old $\mathit{nonself}(T_r(S))$ such that $\mathit{leftmost}(x) \neq \mathit{leftmost}(y)$. By the choice of x, $\mathit{leftmost}(x) <_{\mathcal{P}} \mathit{leftmost}(y)$ (Figure 9). Note $\mathit{rightmost}(y)$ changed from S to S'_1. Suppose there is a vertex w in $T_l(\mathit{leftmost}(y))$ such that $\mathit{rightmost}(w) = S'_1$ before the operation. Then $\mathit{leftmost}(x) <_{\mathcal{P}} \mathit{leftmost}(y) = \mathit{leftmost}(w)$ and $\mathit{rightmost}(w) <_{\mathcal{P}} \mathit{rightmost}(x)$ before the operation, which contradicts Lemma 4. Therefore, w does not exist. Then after $\mathit{rightmost}(y)$ changed from S to S'_1, we do not need to update $T_l(\mathit{leftmost}(y))$. We conclude that the operation correctly processes K_1. The argument for K_2 is symmetric except that in the *Check* call, x is the first vertex in the old $T_l(S'_2)$, w is the rightmost marked vertex in $T_r(\mathit{rightmost}(w))$ such that $w \leq x$, and if the vertex w' before w is in the same block as w, then w' was in the old $\mathit{nonself}(T_l(S))$. It follows that in each of the four cases for K_1 and K_2 (K_1 was an endpoint and K_2 was an endpoint, etc.), the balanced binary trees are updated correctly.

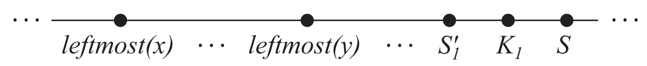


Figure 9: Path $\mathcal{P}$.

Suppose $|K_1 - S| = 1$ and $|K_2 - S| > 1$ (Figure 5d); the opposite case is symmetric. Then $K_1 = S \cup \{v_1\}$. Since $K = K_1 \cup \{v_2\}$ and $S_2 = S \cup \{v_2\}$, the algorithm renames K_1 to K and S to S_2 and then moves v_2. Since v_2 is a complex vertex with $\mathit{leftmost}(v_2) = S_2$ in $\mathcal{P}^+$, the operation moves v_2 to $T_l(S_2)$. Consider v_1. If K_1 was an endpoint of $\mathcal{P}$, then v_1 was a simplicial vertex of K_1, and otherwise, v_1 was a complex vertex in $T_r(S'_1)$. In either case, v_1 does not need to be moved. Moreover, $\mathit{count}(K)$ does not need to be changed because it equals the old $\mathit{count}(K_1)$. Thus, the balanced binary trees are updated correctly in this case, and hence, in every case. $\square$

5 The delete operation

Let G be a chordal graph (not necessarily a proper interval graph). Throughout this section, $\{v_1, v_2\} \in E$. We use $G - \{v_1, v_2\}$ to denote $(V, E - \{\{v_1, v_2\}\})$.

Theorem 8 ([Iba08]) *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 of G that contains both v_1 and v_2. Then K is not a maximal clique in $G - \{v_1, v_2\}$, and $K - \{v_1\}$ and $K - \{v_2\}$, which are not maximal cliques in G, may be maximal cliques in $G - \{v_1, v_2\}$. Furthermore, $K - \{v_1\}$ and $K - \{v_2\}$ are the only maximal cliques that can be created by deleting $\{v_1, v_2\}$ [Iba08]. For example, consider the chordal graph G in Figure 1. Deleting edge $\{y, z\}$ yields a non-chordal graph because two maximal cliques of G contain both y and z. Deleting edge $\{x, y\}$ yields a chordal graph because $\{x, y, z\}$ is the unique maximal clique of G that contains both x and y. Moreover, $\{x, z\}$ but not $\{y, z\}$ is a maximal clique in $G - \{x, y\}$.

Let G be a proper interval graph with clique-separator graph $\mathcal{G}$. Let $\{v_1, v_2\}$ be the edge to be deleted from G. Since $\{v_1, v_2\} \in E$, at least one clique node contains both v_1 and v_2. The operation decides whether v_1 and v_2 are simplicial or complex vertices and then determines which of a number of cases applies. We omit the symmetric cases. If $G - \{v_1, v_2\}$ is not a proper interval graph, then the operation rejects the deletion, and otherwise, the operation computes the clique-separator graph of $G - \{v_1, v_2\}$. Keep in mind that $\{v_1, v_2\}$ is a cut edge of G if and only if $\{v_1, v_2\}$ is a maximal clique of G (see Subsection 2.2).

Delete(v_1, v_2):

Let v_1 and v_2 be vertices in a connected component of G with clique-separator graph $\mathcal{P}$. If $\{v_1, v_2\}$ is a cut edge of G, then $\mathcal{P}$ splits into $\mathcal{P}_1$ and $\mathcal{P}_2$, and the reversal $\mathcal{P}^R$ of $\mathcal{P}$ splits into $\mathcal{P}_2^R$ and $\mathcal{P}_1^R$.

v_1 **is simplicial and** v_2 **is simplicial:** Then v_1 and v_2 are in $T_{a+b}(K)$ for some K. We have three cases. If K has at least one neighbor in $\mathcal{P}$, then reject. If K has no neighbors in $\mathcal{P}$ and $|K| > 2$, then replace K with path $(K_1, \bar{S}, K_2)$ and do $T_{a+b}(K_1) \leftarrow \{v_1\}$ and $T_{a+b}(K_2) \leftarrow \{v_2\}$ and $T_{l+r}(\bar{S}) \leftarrow T_{a+b}(K) - \{v_1, v_2\}$. If K has no neighbors in $\mathcal{P}$ and $|K| = 2$, then $\{v_1, v_2\}$ is a cut edge; replace K with clique nodes K_1 and K_2 and do $T_{a+b}(K_1) \leftarrow \{v_1\}$ and $T_{a+b}(K_2) \leftarrow \{v_2\}$.

v_1 **is complex and** v_2 **is simplicial:** Then v_1 is in $T_l(\hat{S})$ and $T_r(S)$ for some $\hat{S}$ and S and v_2 is in $T_{a+b}(K)$ for some K. If K has left neighbor S or right neighbor $\hat{S}$, and K is a leaf of $\mathcal{P}$, then continue, and otherwise, reject. Assume K has left neighbor S.

If $|T_a(K)| > 1$, then do Case 1a.

If $|T_a(K)| = 1$ and $|K| > 2$, then do Case 1b.

If $|T_a(K)| = 1$ and $|K| = 2$, then $\{v_1, v_2\}$ is a cut edge. Insert v_1 into $T_{a+b}(K')$, where K' is the left neighbor of S, and delete S from $\mathcal{P}$.

v_1 **is complex and** v_2 **is complex:** Then v_1 is in $T_l(\hat{S})$ and $T_r(S)$ for some $\hat{S}$ and S and v_2 is in $T_l(S')$ and $T_r(S'')$ for some S' and S''. If there is a clique node K with left neighbor S and right neighbor S', or left neighbor S'' and right neighbor $\hat{S}$, then continue, and otherwise, reject. Assume K has left neighbor S and right neighbor S'.

If $|T_a(K)| > 0$, then do Case 2a.

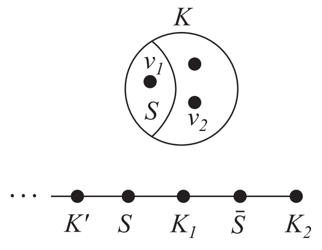
If $|T_a(K)| = 0$ and $|K| > 2$, then do Case 2b.

If $|T_a(K)| = 0$ and $|K| = 2$, then $\{v_1, v_2\}$ is a cut edge. Insert v_1 into $T_{a+b}(K')$ and insert v_2 into $T_{a+b}(K'')$, where K' is the left neighbor of S and K'' is the right neighbor of S', and delete S, K, and S' from $\mathcal{P}$.

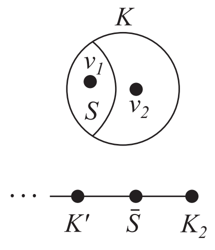
We have considered all the cases where $\{v_1, v_2\}$ is a cut edge, so henceforth we assume $\{v_1, v_2\}$ is not a cut edge. Then $|K| > 2$. Let $K_1 = K - \{v_2\}$, $K_2 = K - \{v_1\}$, and $\bar{S} = K - \{v_1, v_2\}$.

Case 1: K is a leaf with left neighbor S

Figure 2a shows $\mathcal{P}$ in Case 1. Figure 10 shows $\mathcal{P}^-$ in each subcase of Case 1.



(a) Case 1a



(b) Case 1b

Figure 10: K is a leaf with left neighbor S.

Case 1a: v_1 is in $T_r(S)$, v_2 is in $T_{a+b}(K)$, and $|T_a(K)| > 1$ (Figure 10a). If u is the first vertex in $T_r(S)$ and $leftmost(u) <_{\mathcal{P}} leftmost(v_1)$, then reject. Replace K with path $(K_1, \bar{S}, K_2)$. Intuitively, the complex vertices with *rightmost* node S become complex vertices with *rightmost* node $\bar{S}$ and the simplicial vertices in K become complex vertices with *leftmost* and *rightmost* node $\bar{S}$.

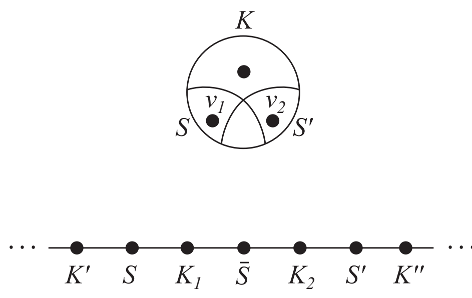
1. $T_{a+b}(K_1) \leftarrow \emptyset$ and $count(K_1) \leftarrow |T_r(S)| - 1$.
   $T_{a+b}(K_2) \leftarrow \{v_2\}$ and $count(K_2) \leftarrow 0$.
2. $T_r(\bar{S}) \leftarrow T_r(S) \cup (T_a(K) - v_2)$.
   $T_l(\bar{S}) \leftarrow (T_b(K) - v_2)$.
3. Move v_1 from $T_r(\bar{S})$ to $T_r(S)$. Call $MoveLeft(v_1, T_l(leftmost(v_1)))$.

Case 1b: v_1 is in $T_r(S)$, v_2 is in $T_{a+b}(K)$, and $|T_a(K)| = 1$ (Figure 10b). If u is the first vertex in $T_r(S)$ and $leftmost(u) <_{\mathcal{P}} leftmost(v_1)$, then reject. Rename S to $\bar{S}$ and rename K to K_2. Intuitively, S becomes $\bar{S}$ and K becomes K_2 when v_1 is removed. Let K' be the left neighbor of S and let S^l be the left neighbor of K'. If $leftmost(v_1) = \bar{S}$, move v_1 from $T_{l+r}(\bar{S})$ to $T_{a+b}(K')$. If $leftmost(v_1) \neq \bar{S}$, move v_1 from $T_r(\bar{S})$ to $T_r(S^l)$ (make it the last vertex in $T_r(S^l)$), decrement $count(K')$, and call $MoveLeft(v_1, T_l(leftmost(v_1)))$.

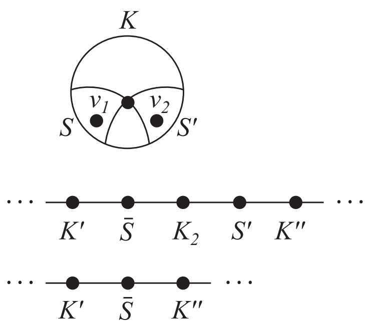
Case 2: K is an internal node with left neighbor S and right neighbor S'

Figure 2b shows $\mathcal{P}$ in Case 2. Figure 11 shows $\mathcal{P}^-$ in each subcase of Case 2.

Case 2a: v_1 is in $T_r(S)$, v_2 is in $T_l(S')$, and $|T_a(K)| > 0$ (Figure 11a). If u is the first vertex in $T_r(S)$ and $leftmost(u) <_{\mathcal{P}} leftmost(v_1)$, or if u is the last vertex in $T_l(S')$ and $rightmost(v_2) <_{\mathcal{P}} rightmost(u)$, then reject. Replace K with path $(K_1, \bar{S}, K_2)$. Intuitively, the complex vertices with *rightmost* node S or *leftmost* node S' become complex vertices with *rightmost* node $\bar{S}$ or *leftmost* node $\bar{S}$, respectively, and the simplicial vertices in K become complex vertices with *leftmost* and *rightmost* node $\bar{S}$.



(a) Case 2a



(b) Case 2b

Figure 11: K is an internal node with left neighbor S and right neighbor S'.

1. $T_{a+b}(K_1) \leftarrow \emptyset$ and $count(K_1) \leftarrow count(K) + |T_r(S)| - 1$.
   $T_{a+b}(K_2) \leftarrow \emptyset$ and $count(K_2) \leftarrow count(K) + |T_l(S')| - 1$.
2. $T_r(\bar{S}) \leftarrow T_r(S) \cup T_a(K)$.
   $T_l(\bar{S}) \leftarrow T_b(K) \cup T_l(S')$.
3. Move v_1 from $T_r(\bar{S})$ to $T_r(S)$. Call $MoveLeft(v_1, T_l(leftmost(v_1)))$.
   Move v_2 from $T_l(\bar{S})$ to $T_l(S')$. Call $MoveRight(v_2, T_r(rightmost(v_2)))$.

Case 2b: v_1 is in $T_r(S)$, v_2 is in $T_l(S')$, and $|T_a(K)| = 0$ (Figure 11b). If u is the first vertex in $T_r(S)$ and $leftmost(u) <_{\mathcal{P}} leftmost(v_1)$, or if u is the last vertex in $T_l(S')$ and $rightmost(v_2) <_{\mathcal{P}} rightmost(u)$, then reject. To shorten the presentation, we will create several nodes and then, in certain cases, delete some of them. The operation can be readily rewritten so that it creates only the nodes that are not deleted. Do Case 2a. Note $T_a(K) = \emptyset$ and $T_b(K) = \emptyset$. If $|S' - S| > 1$ and $|S - S'| > 1$, then stop. Let K' be the left neighbor of S and let S^l be the left neighbor of K' (if it exists).

If $|S' - S| = 1$, then do the following. Intuitively, v_1 is removed from S, which is then deleted. If $leftmost(v_1) = S$, move v_1 from $T_{l+r}(S)$ to $T_{a+b}(K')$. If $leftmost(v_1) \neq S$, move v_1 from $T_r(S)$ to $T_r(S^l)$ (make it the last vertex in $T_r(S^l)$), decrement $count(K')$, and call $MoveLeft(v_1, T_l(leftmost(v_1)))$. Delete S and K_1 and add edge $\{K', \bar{S}\}$. Do $T_l(\bar{S}) \leftarrow T_l(S)$. If $|S - S'| = 1$, then do the symmetric action with S', K_2, and v_2.

In Figure 11b, the top path shows the case $|S' - S| = 1$ and $|S - S'| > 1$, and the bottom path shows the case $|S' - S| = 1$ and $|S - S'| = 1$.

End Delete

We show how the operation deletes four edges individually from the graph G in Figure 12a. (a) In $Delete(y, z)$, y and z are simplicial vertices of K_4. Since K_4 has a neighbor in $\mathcal{P}$, the operation rejects. Note $G - \{y, z\}$ is not a proper interval graph since it has the claw $\{x, y, z, w\}$. (b) In $Delete(x, z)$, x is complex and z is simplicial; x is in $T_r(S_3)$ and z is in $T_{a+b}(K_4)$, where K_4 has left neighbor S_3. Since $|T_a(K_4)| > 1$, the operation does Case 1a. It replaces K_4 with the path $(K_1', \bar{S}, K_2')$, where $K_1' = \{x, y\}, \bar{S} = \{y\}$, and $K_2' = \{y, z\}$ (Figure 12b), and sets $count(K_1') = 1 - 1 = 0$ and $count(K_2') = 0$ (c) In $Delete(v, x)$, v is complex and x is complex; v is in $T_r(S_2)$ and x is in $T_l(S_3)$, where K_3 has neighbors S_2 and S_3. Since $|T_a(K_3)| = 0$ and $|K_3| > 2$, the operation does Case 2b. Since $|S_2 - S_3| > 1$ and $|S_3 - S_2| = 1$, $K_1' = \{w, v\}$ is not a maximal clique and

$K_2' = \{w, x\}$ is a maximal clique (Figure 12c). The operation moves v from $T_r(S_2)$ to $T_r(S_1)$, sets $count(K_2) = 0$, and calls $MoveLeft(v, T_l(leftmost(v)))$. (d) In $Delete(w, x)$, w is complex and x is complex; w is in $T_r(S_2)$ and x is in $T_l(S_3)$, where K_3 has neighbors S_2 and S_3. The operation again does Case 2b, but since v is the first vertex in $T_r(S_2)$ and $leftmost(v) <_{\mathcal{P}} leftmost(w)$, it rejects. Note $G - \{w, x\}$ is not a proper interval graph since it has the claw $\{v, t, w, x\}$.

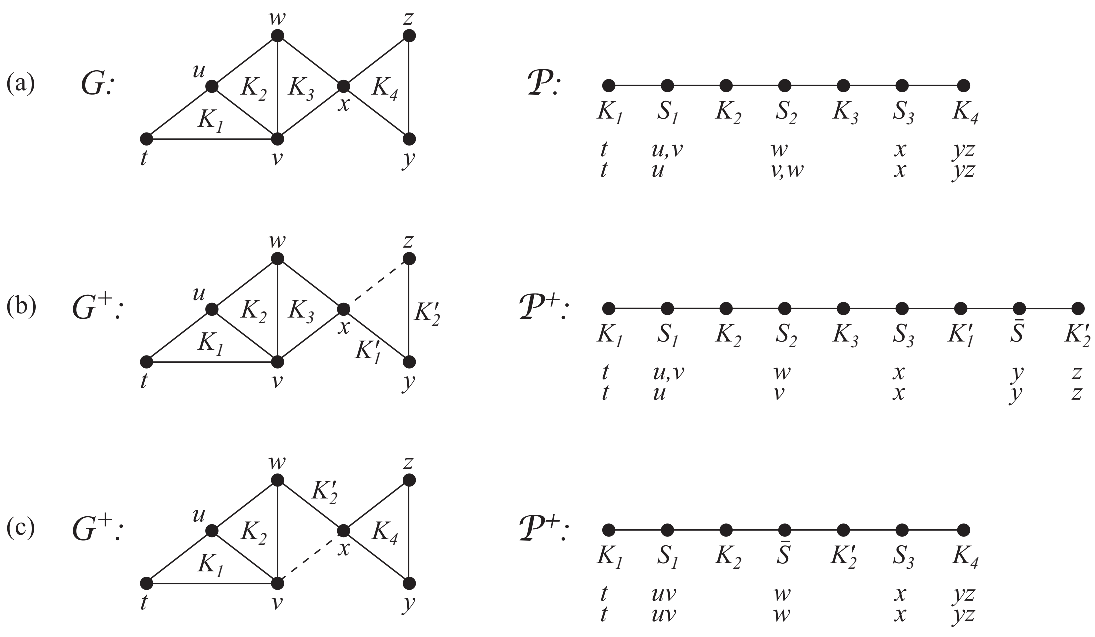


Figure 12: Deleting edges from a proper interval graph G.

As additional examples, we can delete the edges that were inserted in the examples for the *Insert* operation. Deleting $\{w, z\}$ from G^+ in Figure 6b, we have Case 2b with $|S' - S| = 1$ and $|S - S'| = 1$. Deleting $\{u, x\}$ from G^+ in Figure 6c, we have Case 2a. In both deletions, we obtain the clique-separator graph in Figure 6a.

The *Delete* operation uses a constant number of the computations discussed in Subsection 3.2 so it runs in $O(\log n)$ time, where n is the number of vertices of G. In the correctness proof, we will implicitly use the fact that if a separator node S is between clique nodes $\bar{K}$ and $\bar{K}'$ on $\mathcal{P}$, then S separates any $u \in \bar{K} - S$ and any $v \in \bar{K}' - S$, which implies that u and v are not adjacent.

Theorem 9 *Let G be a proper interval graph with clique-separator graph $\mathcal{G}$. If $G^- = G - \{v_1, v_2\}$ is a proper interval graph, the Delete operation computes its clique-separator graph $\mathcal{G}^-$, and otherwise, the operation rejects.*

Proof We begin by showing that in four cases, G^- is not a proper interval graph because it is not an interval graph or it has an induced claw [Rob69]. In the first case, K is a leaf of $\mathcal{P}$ with left neighbor S. In the remaining cases, K is an internal node of $\mathcal{P}$ with left neighbor S and right neighbor S'; since S and S' are neighbors of K, $S \not\subset S'$ and $S' \not\subset S$. Let K' be the left neighbor of S and (if S' exists) let K'' be the right neighbor of S'.

Case 3a: $v_1 \notin S$ and $v_2 \notin S$ (Figure 13a). Let $u \in S$ and $v_3 \in K' - S$ be any vertices. Then $\{u, v_1, v_2, v_3\}$ is an induced claw in G^-.

Case 3b: $v_1 \notin S \cup S'$ and $v_2 \notin S \cup S'$ (Figure 13b). This is the same as the previous case.

Case 3c: $v_1 \in S \cap S'$ and $v_2 \notin S \cup S'$ (Figure 13c). Let $u \in S - S'$ and let $u' \in S' - S$. Then $\{u, u', v_1\}$ and $\{u, u', v_2\}$ are triangles. Let $w' \in K' - S$ and let $w'' \in K'' - S$. Since v_2 is adjacent to neither w' nor w'', $\{u, u', v_1, v_2, w', w''\}$ is an induced Hajós-graph in G^-. Since the Hajós-graph is not an interval graph [MM99], G^- is not an interval graph.

Case 3d: $v_1 \in S - S'$ and $v_2 \notin S \cup S'$ (Figure 13d). Let $u \in S' - S$ and $v_3 \in K'' - S'$ be any vertices. Then $\{u, v_1, v_2, v_3\}$ is an induced claw in G^-.

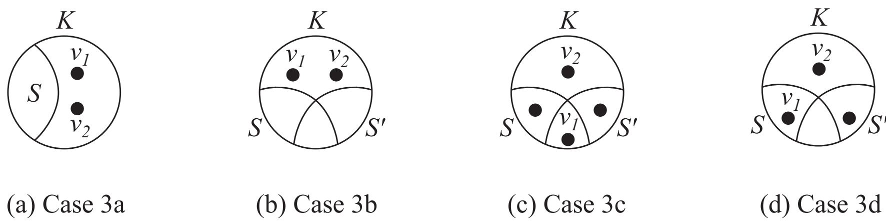


Figure 13: The rejection cases.

Next we consider the first part of the operation and show that it either correctly identifies the correct case or rejects because G^- is not a proper interval graph.

v_1 **is simplicial and** v_2 **is simplicial:** If v_1 and v_2 are simplicial vertices of different clique nodes, then no clique node contains both v_1 and v_2, a contradiction. Then v_1 and v_2 are in $T_{a+b}(K)$ for some K. In the first case, we have Case 3a or Case 3b so the operation rejects. In the second case, $\{v_1, v_2\}$ is not a cut edge and K is a connected component of $\mathcal{G}$. Then $K_1 = K - \{v_2\}$ and $K_2 = K - \{v_1\}$ are maximal cliques of G^- and their intersection is $\bar{S} = K - \{v_1, v_2\}$. Then K is replaced by clique nodes K_1 and K_2, which are adjacent to separator node $\bar{S}$. In the third case, $\{v_1, v_2\}$ is a cut edge and K is a connected component of $\mathcal{G}$. Then K is replaced by clique nodes K_1 and K_2 that contain only v_1 and v_2, respectively.

v_1 **is complex and** v_2 **is simplicial:** Suppose K is between $\hat{S}$ and S on $\mathcal{P}$. Then v_1 is contained in both neighbors of K and we have Case 3c, so the operation rejects. Suppose K is not between $\hat{S}$ and S on $\mathcal{P}$. Since K is the only clique node that contains v_2, K must also contain v_1. Then by the clique intersection property, K has left neighbor S or right neighbor $\hat{S}$. Assume K has left neighbor S. If K is an internal node, then we have Case 3d and the operation rejects. If K is a leaf, then we have Case 1a or Case 1b or $\{v_1, v_2\}$ is a cut edge.

Suppose $\{v_1, v_2\}$ is a cut edge. Then $K = \{v_1, v_2\}$ and $S = \{v_1\}$ (Figure 14). If $\hat{S} \neq S$, then v_1 is contained in at least two separator nodes of $\mathcal{G}$, which implies $\{v_1\}$ is properly contained in at least one separator node of $\mathcal{G}$. Then $\mathcal{G}$ has an arc, contradicting Theorem 2. Thus, $\hat{S} = S$. Then S is the only separator node of $\mathcal{G}$ that contains v_1. Then v_1 is a simplicial vertex of K' in $\mathcal{G}^-$ and $count(K') = 0$. Therefore, the operation inserts v_1 into $T_{a+b}(K')$ and deletes S from $\mathcal{G}$. Then K is an isolated node of $\mathcal{G}^-$ and v_2 is the only vertex in $T_{a+b}(K)$, which is correct since v_2 is an isolated vertex in G^-. Thus, the operation correctly computes $\mathcal{G}^-$. (Note that if $\hat{S} \neq S$, then $\{v_1, v_2\}$ cannot be a cut edge.)

v_1 **is complex and** v_2 **is complex:** If the $\hat{S}$–S and S'–S'' subpaths of $\mathcal{P}$ are not vertex-disjoint, then there is at least one separator node that contains both v_1 and v_2, which means there are at least two clique nodes that contain both v_1 and v_2. Then G^- is not chordal by Theorem 8. In this case, the operation rejects because it doesn't find K. Suppose the $\hat{S}$–S and S'–S'' subpaths of $\mathcal{P}$ are vertex-disjoint. There exists a clique node K that contains both v_1 and v_2,

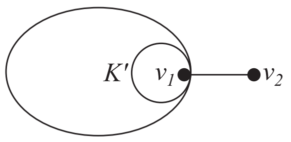


Figure 14: $\{v_1, v_2\}$ is a cut edge of G and K is a leaf of $\mathcal{G}$.

and by the clique intersection property, K is a neighbor of S and S', or a neighbor of S'' and $\hat{S}$. Assume K has left neighbor S and right neighbor S'. Then the operation finds K and we have Case 2a or Case 2b or $\{v_1, v_2\}$ is a cut edge.

Suppose $\{v_1, v_2\}$ is a cut edge. Then $K = \{v_1, v_2\}$, $S = \{v_1\}$, and $S' = \{v_2\}$ (Figure 15). If $\hat{S} \neq S$ or $S' \neq S''$, then an argument similar to the one above shows that $\{v_1\}$ or $\{v_2\}$ is a separator node properly contained in at least one separator node of $\mathcal{G}$, respectively. Then $\mathcal{G}$ has an arc, contradicting Theorem 2. Thus, $\hat{S} = S$ and $S' = S''$. Then S is the only separator node of $\mathcal{G}$ that contains v_1 and S' is the only separator node of $\mathcal{G}$ that contains v_2. Then v_1 and v_2 are simplicial vertices of K' and K'' in $\mathcal{G}^-$, respectively, and $count(K') = count(K'') = 0$. Therefore, the operation inserts v_1 and v_2 into $T_{a+b}(K')$ and $T_{a+b}(K'')$, respectively. The operation deletes S, K, and S' from $\mathcal{G}$, so K' and K'' are leaves in $\mathcal{G}^-$. Thus, the operation correctly computes $\mathcal{G}^-$. (Note that if $\hat{S} \neq S$ or $S' \neq S''$, then $\{v_1, v_2\}$ cannot be a cut edge.)

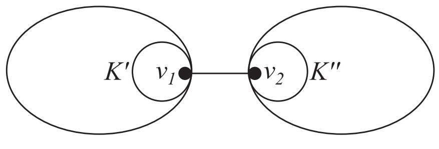


Figure 15: $\{v_1, v_2\}$ is a cut edge of G and K is an internal node of $\mathcal{G}$.

Next we consider the second part of the operation. In each case, we show that if G^- is a proper interval graph, then the operation correctly computes $\mathcal{G}^-$, and otherwise, the operation rejects. Recall that $|K| > 2$ and $K_1 = K - \{v_2\}$, $K_2 = K - \{v_1\}$, and $\bar{S} = K - \{v_1, v_2\}$.

Case 1: K is a leaf with left neighbor S

Let K' be the left neighbor of S (Figure 2a).

Case 1a: v_1 is in $T_r(S)$, v_2 is in $T_{a+b}(K)$, and $|T_a(K)| > 1$ (Figure 10a). Suppose the operation rejects. Then u is the first vertex in $T_r(S)$ and $leftmost(u) <_{\mathcal{P}} leftmost(v_1)$. Let $\hat{K}$ be the left neighbor of $\hat{S} = leftmost(u)$. Let $v_3 \in \hat{K} - \hat{S}$. Then v_3 is adjacent to u but adjacent to neither v_1 nor v_2. Then $\{u, v_1, v_2, v_3\}$ is an induced claw in G^-. Conversely, suppose G^- has an induced claw. Since G has no induced claw, v_1 and v_2 must be two of the three pairwise nonadjacent vertices in the claw. Let $\{u, v_1, v_2, v_3\}$ be the induced claw for some u and v_3 such that u is adjacent to v_1, v_2, and v_3. Since u is adjacent to $v_3 \notin K$, u is not a simplicial vertex of K. Since u is adjacent to v_2, $u \in S$ and so u is in $T_r(S)$. Now since v_3 is not adjacent to v_1, v_3 is contained only in nodes to the left of $leftmost(v_1)$ on $\mathcal{P}$. Then since u is adjacent to v_3, $leftmost(u) <_{\mathcal{P}} leftmost(v_1)$. Then by the sorted order of $T_r(S)$, the operation rejects.

Assuming the operation does not reject, we next show that it correctly computes $\mathcal{G}^-$. Recall the relationship between the clique path of G and the clique-separator graph of G (see Subsection 2.2). Since K_1 and K_2 each contain at least one simplicial vertex of G, K_1 and

K_2 are maximal cliques in G^-. Modify P by replacing K with path (K_1, K_2). Then the modified path P^- is a clique path of G^-, which implies that G^- is an interval graph. Since G^- has no induced claw, G^- is a proper interval graph. Then by inserting the separator node $K'' \cap K'''$ between every two consecutive clique nodes K'' and K''' on P^-, we obtain $\mathcal{G}^-$. Since $S = K' \cap K_1$ and $\bar{S} = K_1 \cap K_2$, it follows that the operation correctly computes $\mathcal{G}^-$.

Next we show that the operation correctly updates the balanced binary trees. We consider each step in turn. 1. Since v_2 is the only simplicial vertex of K that is a simplicial vertex in G^-, we have $T_{a+b}(K_1) \leftarrow \emptyset$ and $T_{a+b}(K_2) \leftarrow \{v_2\}$. Since $|T_r(S)|-1$ vertices change their *rightmost* node from S to $\bar{S}$ (see below), we have $count(K_1) \leftarrow |T_r(S)| - 1$. 2. Since $\bar{S} = K - \{v_1, v_2\}$, the following holds. For every $v \neq v_1$ in $T_r(S)$, $rightmost(v)$ changes from S to $\bar{S}$. For every $v \neq v_2$ in $T_a(K)$, v becomes a complex vertex with $leftmost(v) = rightmost(v) = \bar{S}$. Thus, we have $T_r(\bar{S}) \leftarrow T_r(S) \cup (T_a(K) - v_2)$ and $T_l(\bar{S}) \leftarrow T_b(K) - v_2$. 3. Since $rightmost(v_1) = S$, v_1 is moved from $T_r(\bar{S})$ to $T_r(S)$. Every vertex v with $leftmost(v) = S$ still has $leftmost(v) = S$, so $T_l(S)$ is unchanged. Lastly, we consider the sorted order of the T_l trees. Since every vertex in $T_r(S)$ was moved to $T_r(\bar{S})$ and then v_1 was moved to $T_r(S)$, we need to update the sorted order of only $T_l(leftmost(v_1))$. This is done by the *MoveLeft* call, which moves v_1 to the immediate left of its former block in $T_l(leftmost(v_1))$ and marks v_1.

Case 1b: v_1 is in $T_r(S)$, v_2 is in $T_{a+b}(K)$, and $|T_a(K)| = 1$ (Figure 10b). By the same argument as the previous case, the operation rejects if and only if G^- has an induced claw. Assuming the operation does not reject, we will show that it correctly computes $\mathcal{G}^-$. First, K_1 is not a maximal clique of G^- because $K_1 = S \subset K'$. Second, K_2 is a maximal clique of G^- because v_2 is a simplicial vertex of G. Modify the clique path P of G by replacing K with K_2. Then the modified path is a clique path of G^-. Then as in the previous case, G^- is a proper interval graph. Since $\bar{S} = K' \cap K_2$, it follows that the operation correctly computes $\mathcal{G}^-$.

Next we show that the operation correctly updates the balanced binary trees. Since v_2 was the only simplicial vertex in K, v_2 is the only simplicial vertex in K_2 and no changes to $T_{a+b}(K_2)$ are necessary. We have two cases for v_1. If $leftmost(v_1) = \bar{S}$, then v_1 is moved from $T_{l+r}(\bar{S})$ to $T_{a+b}(K')$ because v_1 becomes a simplicial vertex of K'. If $leftmost(v_1) \neq \bar{S}$, then v_1 is moved from $T_r(\bar{S})$ to $T_r(S^l)$ because $rightmost(v_1)$ becomes S^l; v_1 is made the last vertex in $T_r(S^l)$ and by Lemma 4, the new $T_r(S^l)$ is sorted properly. The *MoveLeft* call moves v_1 to the immediate left of its former block in $T_l(leftmost(v_1))$ and unmarks v_1 if it is in the same block as the vertex before v_1 in $T_l(leftmost(v_1))$.

Case 2: K is an internal node with left neighbor S and right neighbor S'

Let K' be the left neighbor of S and let K'' be the right neighbor of S' (Figure 2b).

Case 2a: v_1 is in $T_r(S)$, v_2 is in $T_l(S')$, and $|T_a(K)| > 0$ (Figure 11a). Suppose the operation rejects. Then an argument similar to the one in Case 1a shows that G^- has an induced claw. Conversely, suppose G^- has an induced claw. Since G has no induced claw, v_1 and v_2 must be two of the three pairwise nonadjacent vertices in the claw. Let $\{u, v_1, v_2, v_3\}$ be the induced claw for some u and v_3 such that u is adjacent to v_1, v_2, and v_3. Since u is adjacent to $v_3 \notin K$, u is not a simplicial vertex of K. Since v_3 is not adjacent to v_1 and v_2, it follows that v_3 is contained only in nodes (1) to the left of $leftmost(v_1)$ on $\mathcal{P}$, or (2) to the right of $rightmost(v_2)$ on $\mathcal{P}$. Assume (1) holds. Then since u is adjacent to v_3, $leftmost(u) <_{\mathcal{P}} leftmost(v_1)$. If $u \in S'$, then $rightmost(v_1) <_{\mathcal{P}} rightmost(u)$, which contradicts Lemma 4. Therefore, $u \notin S'$. Since u is adjacent to v_2, $u \in S$ and thus u is in

$T_r(S)$. Then by the sorted order of $T_r(S)$, the operation rejects. Assume (2) holds. Then the symmetric argument shows that the operation rejects.

Assuming the operation does not reject, we next show that it correctly computes $\mathcal{G}^-$. Since K_1 and K_2 each contain at least one simplicial vertex of G, K_1 and K_2 are maximal cliques in G^-. Let P be the clique path of G. Modify P by replacing K with path (K_1, K_2). Then the modified path is a clique path of G^-, which implies that G^- is an interval graph. Since G^- has no induced claw, G^- is a proper interval graph. Since $S = K' \cap K_1$, $\bar{S} = K_1 \cap K_2$, and $S' = K_2 \cap K''$, it follows that the operation correctly computes $\mathcal{G}^-$.

Next we show that the operation correctly updates the balanced binary trees. We consider each step in turn. 1. Since $K_1 = \bar{S} \cup \{v_1\}$, K_1 contains no simplicial vertices of G^-. Thus, we have $T_{a+b}(K_1) \leftarrow \emptyset$. Since $|T_r(S)| - 1$ vertices change their *rightmost* node from S to $\bar{S}$ (see below), we have $count(K_1) \leftarrow count(K) + |T_r(S)| - 1$. The K_2 argument is symmetric. 2. Since $\bar{S} = K - \{v_1, v_2\}$, the following holds. For every $v \neq v_1$ in $T_r(S)$, $rightmost(v)$ changes from S to $\bar{S}$. For every v in $T_a(K)$, v becomes a complex vertex with $leftmost(v) = rightmost(v) = \bar{S}$. For every $v \neq v_2$ in $T_l(S')$, $leftmost(v)$ changes from S' to $\bar{S}$. Thus, we have $T_r(\bar{S}) \leftarrow T_r(S) \cup T_a(K)$ and $T_l(\bar{S}) \leftarrow T_b(K) \cup T_l(S')$. 3. Since $rightmost(v_1) = S$, v_1 is moved from $T_r(\bar{S})$ to $T_r(S)$. Then the $MoveLeft$ call moves v_1 to the immediate left of its former block in $T_l(leftmost(v_1))$ and marks v_1. The v_2 argument is symmetric.

Case 2b: v_1 is in $T_r(S)$, v_2 is in $T_l(S')$, and $|T_a(K)| = 0$ (Figure 11b). By the same argument as in Case 2a, the operation rejects if and only if G^- has an induced claw. Assuming the operation does not reject, we will show that it correctly computes $\mathcal{G}^-$. If $|S' - S| = 1$, then K_1 is not a maximal clique in G^- because $K_1 = S \subset K'$. If $|S' - S| > 1$, then there exists $v_2' \in S' - S$ such that $v_2' \neq v_2$ and then K_1 is a maximal clique in G^- because K is the only maximal clique in G that contains both v_1 and v_2'. Thus, $|S' - S| > 1$ if and only if K_1 is a maximal clique in G^-. Similarly, $|S - S'| > 1$ if and only if K_2 is a maximal clique in G^-.

Suppose $|S' - S| > 1$ and $|S - S'| > 1$. Then K_1 and K_2 are maximal cliques of G^-. Then the same argument as in Case 2a shows that the operation correctly computes $\mathcal{G}^-$ and correctly updates the balanced binary trees.

Suppose $|S' - S| = 1$ and $|S - S'| > 1$. Then K_2, but not K_1, is a maximal clique in G^-. Then modifying the clique path of G by replacing K with K_2 yields the clique path of G^-. Since G^- has no induced claw, G^- is a proper interval graph. Now $\bar{S} = K' \cap K_2$ because $\bar{S} \cup \{v_1\} = S \subset K'$ and $\bar{S} \cup \{v_2\} = K_2$. Therefore, the operation correctly computes $\mathcal{G}^-$ (see the top path in Figure 11b). Consider the balanced binary trees. After Case 2a is done, $T_r(S) = \{v_1\}$, $T_r(\bar{S}) = T_r(S) - v_1$, $T_l(\bar{S}) = T_l(S') - v_2 = \emptyset$ (since $|S' - S| = 1$), and $T_l(S') = \{v_2\}$. If $leftmost(v_1) = S$, then v_1 is moved from $T_{l+r}(S)$ to $T_{a+b}(K')$ because v_1 becomes a simplicial vertex of K'. If $leftmost(v_1) \neq S$, then v_1 is moved from $T_r(S)$ to $T_r(S^l)$ because $rightmost(v_1)$ becomes S^l; as before the new $T_r(S^l)$ is sorted properly. Then at the end of Case 2b, $T_r(\bar{S}) = T_r(S) - v_1$, $T_l(\bar{S}) = T_l(S) - v_1$ (v_1 may not have been in $T_l(S)$), and $T_{a+b}(K_2) = \emptyset$. Now $|S' - S| = 1$ implies $S - \{v_1\} = \bar{S}$. Since v_1 is in $T_r(S^l)$ or $T_{a+b}(K')$, and v_2 is in $T_l(S')$, it follows that the operation correctly updates the balanced binary trees. If $|S' - S| > 1$ and $|S - S'| = 1$, the argument is symmetric.

Suppose $|S' - S| = 1$ and $|S - S'| = 1$. Then neither K_1 nor K_2 are maximal cliques in G^-. Then modifying the clique path of G by replacing the subpath (K', K, K'') with (K', K'') yields a clique path of G^-. Since G^- has no induced claw, G^- is a proper interval graph. Since $\bar{S} = K' \cap K''$, it follows that the operation correctly computes $\mathcal{G}^-$ (see the bottom path in Figure 11b). Consider the balanced binary trees. After Case 2a is done, $T_r(S) = \{v_1\}$, $T_r(\bar{S}) = T_r(S) - v_1 = \emptyset$, $T_l(\bar{S}) = T_l(S') - v_2 = \emptyset$, and $T_l(S') = \{v_2\}$. An argument similar to the one above shows that at the end of Case 2b, $T_r(\bar{S}) = T_r(S) - v_1$ and $T_l(\bar{S}) = T_l(S) - v_1$. Therefore, the operation correctly updates the balanced binary trees. □

6 Conclusions

We have presented a fully dynamic graph algorithm for proper interval graphs. Its running time matches the running time of the algorithm in [HSS01] and it maintains the proper interval graph's clique-separator graph instead of its straight enumeration as in [HSS01]. One advantage of the clique-separator graph is that we can compute a Hamiltonian cycle of a proper interval graph from its clique-separator graph; this algorithm has the same $O(m + n)$ running time as previous algorithms but it is simpler and shorter [Iba09c].

It would be interesting to extend the algorithm and data structure to handle interval graphs. The clique path of an interval graph is not unique, however, so it would probably be necessary to use a more complicated data structure such as the PQ-tree [BL76] or the train tree [Iba09b]. The algorithm in [Iba09b] maintains the clique-separator graph and the train tree of an interval graph in $O(n \log n)$ per edge insertion or deletion. It may be possible to speed up that algorithm by maintaining the intervals of the vertices as done by the algorithm in this paper.

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.

[BP93] J. R. S. Blair and B. Peyton. An introduction to chordal graphs and clique trees. In A. George, J. R. Gilbert, and J. W. H. Liu, editors, *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.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd edition.* MIT Press and McGraw-Hill Book Co., Cambridge, MA and New York, NY, 2001.

[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.

[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 K. H. 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.

[Iba08] L. Ibarra. Fully dynamic algorithms for chordal graphs and split graphs. *ACM Transactions on Algorithms*, 4(4):40:1–20, 2008.

[Iba09a] L. Ibarra. The clique-separator graph for chordal graphs. 2009. Accepted by Discrete Applied Mathematics.

[Iba09b] L. Ibarra. A fully dynamic graph algorithm for recognizing interval graphs. 2009. Accepted by Algorithmica.

[Iba09c] L. Ibarra. A simple algorithm to find Hamiltonian cycles in proper interval graphs. 2009. Accepted by Information Processing Letters.

[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.

[LPP89] J. G. Lewis, B. W. Peyton, and A. Pothen. A fast algorithm for reordering sparse matrices for parallel factorization. *SIAM J. Computing*, 10(6):1146–1173, 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.

[Rob69] F. S. Roberts. Indifference graphs. In F. Harary, editor, *Proof Techniques in Graph Theory*, pages 139–146. Academic Press, New York, 1969.

[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.

[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.