Treewidth, partial k-trees, and chordal graphs

Delpensum INF 334 - Institutt for informatikk

Pinar Heggernes

September 26, 2006

Many graph problems that are NP-hard on general graphs, have polynomial time solutions if the input graph has bounded treewidth or if it belongs to a restricted graph class. In this document, we review some of the techniques for coping with NP-hardness of graph problems. In particular, we explain graph parameters treewidth and pathwidth, and give examples of polynomial time algorithms for graphs of small treewidth and pathwidth. Furthermore, we study the class of chordal graphs and several subclasses of chordal graphs, since many hard problems are polynomially solvable on these classes. In addition, some important and useful graph theoretical notions are mentioned and explained.

1 Subgraphs and separators

In this section we give some basic definitions concerning graphs and some graph problems.

Given a graph G, if its vertex and edge sets are not given names, then $V(G)$ denotes the set of vertices of G, and $E(G)$ denotes the set of edges of G. Usually, we use $n = |V(G)|$ and $m = |E(G)|$. We denote the neighbors of a vertex v by $N(v)$. Given a graph $G = (V, E)$, let $K_n = (V, E_V)$ denote the complete graph on vertex set V; thus every vertex in V is adjacent to every other vertex of V. The *complement* of G is the following graph: $\bar{G} = (V, E_V - E)$.

Let us now define subgraphs and induced subgraphs. For a given graph $G = (V, E)$, any graph that has a subset of V as its vertex set and a subset of E as its edge set, is a *subgraph* of G. We need to distinguish between subgraphs and induced subgraphs. For example, removing a *vertex* (and all its incident edges) from G results in an *induced subgraph.* However, just removing an *edge* between two vertices of G results in a *subgraph* which is not necessarily induced.

Definition 1.1 *Given a graph $G = (V, E)$ and a subset $U \subseteq V$, the* subgraph of G induced *by U is the graph $G[U] = (U, D)$, where $(u, v) \in D$ if and only if $u, v \in U$ and $(u, v) \in E$.*

A *clique* is a set of vertices that induce a complete subgraph of G, and a *maximal clique* is a clique which is not a subset of any other clique. The size of the largest, or *maximum*, clique of G is denoted by $\omega(G)$, and it is called the *clique number* of G. A *clique cover* of size c is a partition of the vertices of G into c disjoint cliques. (Note that an edge is a clique consisting of two vertices.) $k(G)$ is the size of a smallest possible clique cover of G.

An *independent set* is a set of vertices no two of which are adjacent. $\alpha(G)$ is the number of vertices in an independent set of maximum cardinality. A *c-coloring* is a partition of the vertices into c independent sets. The vertices belonging to the same independent set are "colored" with the same color, and adjacent vertices receive different colors. We say that G is c-colorable. $\chi(G)$ is the smallest possible number c for which there exists a c-coloring of G, and it is called the *chromatic number* of G.

Observe that $\omega(G) \leq \chi(G)$ and $\alpha(G) \leq k(G)$, since every vertex of a maximum clique (maximum independent set) must be contained in a different partition segment in any minimum coloring (minimum clique cover). Note also the following equalities: $\omega(G) = \alpha(\bar{G})$ and $\chi(G) = k(\bar{G})$. Deciding $\omega(G), \alpha(G), \chi(G)$, and $k(G)$ are NP-hard problems [13].

We end this section with the definition of a minimal separator, which is central in coming sections. Given a graph $G = (V, E)$, a set of vertices $S \subset V$ is a *separator* if the subgraph of G induced by $V - S$ is disconnected. The set S is a uv-separator if u and v are in different connected components of $G[V - S]$. A uv-separator S is minimal if no subset of S separates u and v.

Definition 1.2 *S is a* minimal separator *of G if there exist two vertices u and v in G such that S is a minimal uv-separator.*

2 Treewidth, pathwidth and partial k-trees

Treewidth is a parameter that gives a measure of how "tree-like" or "close to being a tree" a graph is. The smaller the treewidth, the more tree-like the graph is. As many NP-hard graph problems have simple and polynomial (even linear) time solutions on trees, it is often the case that these problems can also be solved in polynomial time for graphs that have constant treewidth, using dynamic programming techniques. In order to define treewidth, we need first to define a tree decomposition of a graph.

Definition 2.1 *A* tree decomposition *of a graph $G = (V, E)$ is a pair*

$$(\{X_i \mid i \in I\}\,,\ \ T = (I, M))$$

where $\{X_i \mid i \in I\}$ is a collection of subsets of V (also called bags*), and T is a tree, such that:*

- $\bigcup_{i \in I} X_i = V$
- $(u, v) \in E \Rightarrow \exists i \in I$ *with* $u, v \in X_i$
- *For all vertices $v \in V$, $\{i \in I \mid v \in X_i\}$ induces a connected subtree of T.*

The last condition of Definition 2.1 can be replaced by the following equivalent condition:

- $i, k, j \in I$ and j is on the path from i to k in $T \Rightarrow X_i \cap X_k \subseteq X_j$.

Lemma 2.2 [7] *Let* $(\{X_i \mid i \in I\},\ \ T = (I, M))$ *be a tree decomposition of* $G = (V, E)$*, and let* $K \subseteq V$ *be a clique in* G*. Then there exists an* $i \in I$ *with* $K \subseteq X_i$*.*

Exercise 2.3 *Prove Lemma 2.2.*

The *width* of a decomposition $(\{X_i \mid i \in I\},\ \ T = (I, M))$ is $\max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G, $tw(G)$, is the minimum width over all tree decompositions of G.

Corollary 2.4 (of Lemma 2.2) *For every graph* G*,* $tw(G) \geq \omega(G) - 1$*.*

A *path decomposition* is a tree decomposition $(\{X_i \mid i \in I\},\ \ T = (I, M))$ such that T is a path. The *pathwidth* of a graph G, $pw(G)$, is the minimum width over all path decompositions of G. Since every path decomposition is also a tree decomposition, pathwidth $\geq$ treewidth for all graphs.

Example 2.5 *Let* G *be the graph shown in Figure 1 a).*

Let $I = \{1, 2, 3, 4\}, X_1 = \{a, b, f\}, X_2 = \{b, d, f\}, X_3 = \{b, c, d\}, X_4 = \{d, e, f\}$*. Let* T *be the tree shown in Figure 1 b).* $(\{X_i \mid i \in I\}, T)$ *is a tree decomposition of* G*.*

Let $J = \{1, 2, 3\}, Y_1 = \{a, b, f\}, Y_2 = \{b, c, e, f\}, Y_3 = \{c, d, e\}$*. Let* P *be the path shown in Figure 1 c).* $(\{Y_j \mid j \in J\}, P)$ *is a path decomposition of* G*.*

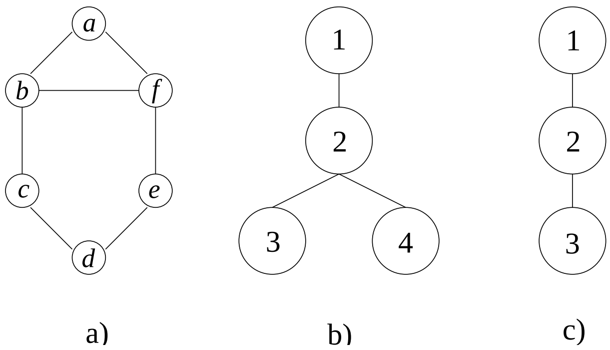

Figure 1: The graph of Example 2.5 has treewidth 2 and pathwidth 2 (why?).

A tree decomposition of width equal to the treewidth is called an *optimal* tree decomposition. A path decomposition of width equal to the pathwidth is called an *optimal* path decomposition. Unfortunately, computing treewidth and pathwidth are NP-hard problems.

Theorem 2.6 [1] *The following problems are NP-complete:*

- *Given a graph $G = (V, E)$ and an integer $k < |V|$, is the treewidth of $G \leq k$?*
- *Given a graph $G = (V, E)$ and an integer $k < |V|$, is the pathwidth of $G \leq k$?*

However, the k-treewidth problem can be solved in linear time [6]: Given a graph G, is the treewidth of G at most k? In this case, k is a constant and not a part of the input (the hidden constant in the running time of the algorithm is exponential in k). If the answer is yes, the algorithm of Bodlaender constructs also an optimal tree decomposition in linear time. Thus for graphs that have treewidth bounded by a constant, their treewidth and a corresponding optimal tree decomposition can be constructed in linear time. This is important, since we need an optimal tree decomposition when designing polynomial time algorithms for graphs of bounded treewidth.

3 Dynamic programming based on a tree decomposition

Several problems that are NP-complete on general graphs have polynomial time algorithms for graphs that have treewidth bounded by a constant. We will look at one such example, maximum independent set [5]. In this problem, given a graph G, we want to compute $\alpha(G)$.

As we have seen in Chapter 10 of [19], any non-redundant tree decomposition of a graph with n vertices has at most n tree nodes. Given any non-redundant tree decomposition of G with decomposition tree T, a *binary* decomposition tree T', with the same width as T, can be constructed in polynomial time, such that T' has a polynomial number of tree nodes. Of course this new tree decomposition might be redundant, but the number of tree nodes stay polynomial in n. Consequently, when we have a graph of treewidth at most k, we can first compute its treewidth and an optimal tree decomposition in linear time, and then we can turn this tree decomposition into a binary tree decomposition with the same width in polynomial time. Therefore, we can always assume that we have an optimal binary tree decomposition.

Suppose that we have a binary tree decomposition $(\{X_i \mid i \in I\}, T = (I, M))$ of input graph G, with root r and width $k = tw(G)$. For each $i \in I$, let $Y_i = \{v \in X_j \mid j = i \text{ or } j \text{ is a descendant of } i\}$.

Note that if $v \in Y_i$, and $v \in X_j$ for some node $j \in I$ that is not a descendant of i in T, then $v \in X_i$. Similarly, if $v \in Y_i$ and v is adjacent to a vertex $w \in X_j$ in G with j not a descendant of i in T, then $v \in X_i$ or $w \in X_i$ (or both). As a consequence, when we have an independent set W of $G[Y_i]$, and we want to extend this to an independent set of G, then we only need to examine the vertices of X_i rather than whole Y_i. The only important part is which vertices of X_i belong to W, and we do not need to consider which vertices of $Y_i - X_i$ belong to W. Of the latter, only the *number* of the vertices in W is important.

For $i \in I$ and $Z \subseteq X_i$, let $s_i(Z)$ be the maximum size of an independent set W in $G[Y_i]$ under the constraint $W \cap X_i = Z$. Let $s_i(Z) = -\infty$ if no such independent set exists.

The algorithm to solve the independent set problem on G basically consists of computing the tables s_i for each node $i \in I$ (each table s_i has an entry $s_i(Z)$ for each subset Z of X_i). This is done in a bottom-up manner in the tree T: each table s_i is computed after the tables of the children of node i are computed. For a leaf node i in T, the following formula can be used to compute all $2^{|X_i|}$ values in the table s_i.

$$s_i(Z) = \begin{cases} |Z| & \text{if } Z \text{ is an independent set} \\ -\infty & \text{if } Z \text{ is not an independent set} \end{cases}$$

For an internal node i with children t and q, and for each subset Z of X_i, the entry $s_i(Z)$ equals to:

$$\max_{Z \cap X_t = Z' \cap X_i \text{ and } Z \cap X_q = Z'' \cap X_i} \{s_t(Z') + s_q(Z'') + |Z \cap (X_i - X_t - X_q)| - |Z \cap X_t \cap X_q|\}$$

when Z is an independent set, and $s_i(Z) = -\infty$ when Z is not an independent set.

The idea behind the formula for internal nodes is to take the maximum over all sets $Z' \subseteq X_t$ that agree with Z in which vertices of $X_i \cap X_t$ belong to the independent set, and similarly for $Z'' \subseteq X_q$. Vertices in $Z \cap (X_i - X_t - X_q)$ are not counted yet, so their number should be added, while vertices in $Z \cap X_t \cap X_q$ are counted twice, hence their number should be subtracted once.

For each node $i \in I$ the table s_i is computed in a bottom-up order until the table s_r is computed. The maximum size of an independent set in G is equal to $\max_{Z \subseteq X_r} s_r(Z)$. This describes an algorithm that solves the independent set problem on G in $O(2^{3k}n)$ time, since $|X_i| \leq k+1$ for all $i \in I$. It is also possible to construct the independent set itself using standard dynamic programming techniques.

In this example each table entry gives information about an equivalence class of partial solutions. The number of such equivalence classes is bounded by some constant when the treewidth is bounded by a constant. Tables can be computed using only the tables of the children nodes. This is thus an example of dynamic programming, where solutions for subproblems are stored in tables and used when needed for the solutions of larger subproblems.

4 Partial k-trees

Partial k-trees are equivalent to graphs of treewidth at most k. Thus any polynomial time algorithm for graphs of bounded treewidth is a polynomial time algorithm for partial k-trees. Partial k-trees are useful as they might be seen as a tool to gain more insight in graphs of bounded treewidth.

Definition 4.1 *The class of k-*trees *is defined recursively as follows:*

- *The complete graph on k vertices i a k-tree.*
- *A k-tree G with $n+1$ vertices ($n \geq k$) can be constructed from a k-tree H with n vertices by adding a vertex adjacent to exactly k vertices, namely all vertices of a k-clique of H.*

The following theorem gives several alternative characterizations of k-trees.

Theorem 4.2 [23] *Let $G = (V, E)$ be a graph. The following statements are equivalent:*

- *G is a k-tree.*
- *G is connected, G has a k-clique, but no $(k+2)$-cliques, and every minimal separator of G is a k-clique.*
- *G is connected, $|E| = k|V| - \frac{1}{2}k(k+1)$, and every minimal separator of G is a k-clique.*
- *G has a k-clique, but not a $(k+2)$-clique, and every minimal separator of G is a clique, and for all distinct non-adjacent pairs of vertices $x, y \in V$, there exist exactly k vertex disjoint paths from x to y.*

Definition 4.3 *A* partial k-tree *is a graph that contains all the vertices and a subset of the edges of a k-tree.*

Theorem 4.4 [25] *G is a partial k-tree if and only if G has treewidth at most k.*

Proof. $\Rightarrow$: In this direction, we want to show that every partial k-tree has treewidth at most k. We will actually show a stronger result, namely that k-trees have treewidth at most k. Since every partial k-tree is a subgraph of a k-tree, and the treewidth of a graph is at least as large as the treewidth of any of its subgraphs, the result in this direction will follow. Let $G = (V, E)$ be a k-tree with $|V| > k+1$. There is a vertex $v \in V$ such that $G[V - \{v\}]$ is a k-tree, and the neighbors of v induce a clique K of size k in G. Using induction, we can assume that $G[V - \{v\}]$ has treewidth at most k with a corresponding tree decomposition $(\{X_i \mid i \in I\}, T = (I, M))$. (The base case of induction is when $G[V - \{v\}]$ is a complete graph on k vertices; such a graph has clearly treewidth $k - 1$.) By Lemma 2.2, there is an $i' \in I$ with $X_{i'}$ containing all neighbors of v in G, with $K \subseteq X_{i'}$. Let $J = I \cup \{j\}$, where $j \notin I$, and let $X_j = K \cup \{v\}$. Now, $(\{X_i \mid i \in J\}, T = (J, M \cup \{i', j\}))$ is a tree decomposition of G with width k.

$\Leftarrow$: In this direction, we want to show that every graph with treewidth at most k is a partial k-tree. Let $G = (V, E)$ be a graph with $|V| > k + 1$, and let $(\{X_i \mid i \in I\}, T = (I, M))$ be a tree decomposition of G of width at most k. We will prove, with induction on $|V|$, that there is a k-tree $H = (V, E')$ such that for every $i \in I$, $G[X_i]$ is a subgraph of a clique of H with $k + 1$ vertices.

If $|V| = k + 1$ then we are done. Otherwise, take a leaf node $l \in I$ and let $j \in I$ be the only neighbor of $l \in T$. If $X_l \subseteq X_j$, then we can remove l from T and continue with the remaining tree decomposition. Let v be the only vertex in $X_l - X_j$ (otherwise X_l can be divided into several tree nodes such that the resulting new leaf node satisfies this requirement). Now it is important to note the following things: Since v does not appear in any X_i for $i \neq l$, all neighbors of v must appear in X_l. Since $|X_l| \leq k + 1$, v has at most k neighbors. Also, because v is the only vertex in $X_l - X_j$, all neighbors of v must also appear in X_j. Suppose that the induction hypothesis on $G[V - \{v\}]$ with tree decomposition $(\{X_i\} \mid i \in I, i \neq l\}, T[I - \{l\}])$ results in a k-tree H'. Note in particular that $G[X_j]$ is a subgraph of a clique of size at most $k+1$ in H'. Remember that v has at most k neighbors in G and they all belong to X_j. Therefore, the neighbors of v induce a subgraph of a k-clique C of H' in G. Now, we can add v with edges to all vertices of C to H' (which will make a $(k + 1)$-clique), and get the desired k-tree H. □

5 Chordal graphs

In addition to the fact that several NP-complete problems have polynomial time solutions for chordal graphs, chordal graphs are a large and important class of graphs with applications in several areas, like the solution of sparse symmetric systems of linear equations [22], data-base management systems [24], knowledge-based systems [11], and computer vision [9].

Definition 5.1 *A graph is* chordal *if every cycle of length* > 3 *has a chord.*

A *chord* is an edge joining two nonconsecutive vertices of a cycle. Equivalent to Definition 5.1, a chordal graph does not contain an induced cycle of length > 3. All induced subgraphs of a chordal graph are also chordal (why?).

Theorem 5.2 [10] *A graph G is chordal if and only if every minimal separator of G is a clique.*

Proof. $\Rightarrow$: Let $G = (V, E)$ be chordal and let S be a minimal separator of G. Let x and y be any two vertices in S. We will show that (x, y) must be an edge of G. Let a and b be the vertices for which S is a minimal ab-separator, and let A and B be the connected components of $G[V - S]$ containing respectively a and b. There must exist a path between x and y through vertices belonging to A. Let p_1 be a shortest such path. Let analogously p_2 be a shortest path between x and y through vertices of B. Paths p_1 and p_2 joined together make a cycle of length at least 4. Since G is chordal, this cycle must have a chord. Since no edges exist between vertices of A and vertices of B, the edge (x, y) must be present and a chord of the mentioned cycle.

$\Leftarrow$: Let G be a graph where each minimal separator is a clique. Assume that G is not chordal, and let $w, x, y, z_1, ..., z_k, w$ be a chordless cycle of length at least 4 in G ($k \geq 1$). Any minimal wy-separator of G must contain x and at least one z_i for $1 \leq i \leq k$. Since all minimal separators are cliques, the edge (x, z_i) must belong to G contradicting that the mentioned cycle is chordless. □

Corollary 5.3 *k-trees are chordal.*

Definition 5.4 *A vertex is called* simplicial *if its adjacency set induces a clique.*

Lemma 5.5 [10] *A chordal graph is either complete or has at least two nonadjacent simplicial vertices.*

Proof. Let G be a chordal graph which is not complete. The proof is by induction on the number of vertices n. The base case is when $n = 2$ and G has two isolated vertices that are both simplicial. Let $n > 2$ and assume that the lemma holds for all such graphs with fewer than n vertices. Let a and b be two nonadjacent vertices of G and let S be a minimal ab-separator. The induced subgraph $G[V - S]$ has at least two connected components. Let $G[A]$ be the connected component containing a and $G[B]$ the connected component containing b. Now $G[A \cup S]$ is a chordal graph with fewer than n vertices and thus is either complete (every vertex of A is simplicial) or has at least two nonadjacent simplicial vertices one of which must belong to A since S is a clique. Since A has no neighbors outside of $A \cup S$, all simplicial vertices of $G[A \cup S]$ that belong to A are also simplicial vertices of G. Thus G has at least one simplicial vertex that belongs to A. With the same argument, G has another simplicial vertex that belongs to B, and the proof is complete. □

This lemma provides a necessary condition for recognizing chordal graphs. Consider the following algorithm [12]. Repeat the following step until no simplicial vertices are left: Find a simplicial vertex and remove it from the graph. If the graph is chordal, there will be a simplicial vertex at each step, by the above lemma. Therefore, if the remaining graph is not empty at the end of this process, then we can conclude that the input graph is not chordal. If no vertices remain at the end of the process, then the order in which the vertices are removed is called a *perfect elimination order*. Chordal graphs are exactly the class of graphs having perfect elimination orders, as will be proved in the following theorem. Consequently, if the remaining graph is empty, then we can conclude that the input graph is chordal.

Theorem 5.6 [12] *A graph is chordal if and only if it has a perfect elimination ordering.*

Proof. $\Rightarrow$: We have already explained, in the discussion above, how a perfect elimination ordering of a chordal graph can be found. More formally, assume that G is a chordal graph with n vertices, and assume that the theorem is true for chordal graphs with fewer vertices. Let v be a simplicial vertex of G. Now, the graph $G[V - \{v\}]$ has a perfect elimination ordering β. Let β be an ordering that orders v first and the rest of the vertices in G in the same order as β. Clearly, β is a perfect elimination ordering of G.

$\Leftarrow$: Let G be a graph and let $v_1, v_2, ..., v_n$ be a perfect elimination ordering of the vertices of G. Assume that G has a chordless cycle c of length greater than 3. Let v_i be the vertex on c whose label i is smaller than any other vertex on c. Since the given ordering is a perfect elimination ordering, the higher numbered neighbors of v_i induce a clique in G. But then c must have at least one chord, namely the edge joining the two neighbors of v_i in c. □

Any graph G can be turned into a chordal graph by adding edges, and the resulting chordal graph is called *triangulation* of G. The above idea can be used to compute a triangulation of input graph G: Choose any vertex x to start with, and add the necessary edges so that the neighbors of x become a clique. Remove x from the modified graph, and continue this process until all vertices are processed. In the end, all the added edges of each step are added to the original graph G and this results in a *filled* graph, which is a triangulation of G. This algorithm is popularly referred to as *the elimination game*. Let $\beta = v_1, v_2, ..., v_n$ be the order in which the elimination game processes the vertices of a given graph G and produces the filled graph G^+_β. To see that G^+_β is indeed chordal, observe that β is a perfect elimination order of G^+_β. How many edges G^+_β has depends on the ordering β. Triangulations can also be computed in other ways [17], for example by making every minimal separator into a clique by adding edges. Nevertheless, computing a triangulation with the minimum number of edges is equivalent to computing an ordering β that results in a G^+_β with the minimum number of edges. Unfortunately, this is an NP-hard problem [26].

5.1 Chordal graphs as intersection graphs

Definition 5.7 *Let F be a family of nonempty sets. The* intersection graph *of F is obtained by representing each set in F by a vertex, and connecting two vertices by an edge if and only if their corresponding sets intersect.*

Definition 5.8 *A family $\{T_i \mid i \in I\}$ of subsets of a set T is said to satisfy the* Helly property *if the following condition is satisfied:*

$$J \subseteq I \text{ and } T_i \cap T_j \neq \emptyset \;\forall i, j \in J \Rightarrow \bigcap_{j \in J} T_j \neq \emptyset$$

Lemma 5.9 *A family of subtrees of a tree satisfies the Helly property.*

Proof. Let T be a tree and let $\{T_i \mid i \in I\}$ be a set of subtrees of T. Suppose $T_i \cap T_j \neq \emptyset$ for all $i, j \in J$. Consider three vertices a, b, and c on T. Let S be a set of indices s such that T_s contains at least two of these three vertices, and let P_1, P_2, and P_3 be paths in T connecting a with b, b with c, and a with c, respectively. Since T is a tree, it follows that $P_1 \cap P_2 \cap P_3 \neq \emptyset$. But each $T_s, s \in S$, contains one of these paths P_i. Therefore,

$$\bigcap_{s \in S} T_s \supseteq P_1 \cap P_2 \cap P_3 \neq \emptyset. \tag{1}$$

Assume now by induction that

$$T_i \cap T_j \neq \emptyset \ \forall i, j \in J \Rightarrow \bigcap_{j \in J} T_j \neq \emptyset$$

for all index sets J of size $\leq k$. This is certainly true for $k = 2$. Consider then a family of subtrees $\{T_1, ..., T_{k+1}\}$. By the induction hypothesis there exist vertices a, b, c on T such that

$$a \in \bigcap_{j=1}^{k} T_j \ , \ b \in \bigcap_{j=2}^{k+1} T_j \ , \ c \in T_1 \cap T_{k+1}.$$

Moreover, every T_j contains at least two of the vertices a, b, c. Hence, by (1), $\bigcap_{j=1}^{k+1} T_j \neq \emptyset$. $\square$

Theorem 5.10 [14] *Let $G = (V, E)$ be an undirected graph, and let $\mathcal{K}$ be the set of maximal cliques of G, with $\mathcal{K}_v$ the set of all maximal cliques that contain vertex v of G. The following statements are equivalent:*

(i) *G is chordal.*

(ii) *G is the intersection graph of a family of subtrees of a tree.*

(iii) *There exists a tree $T = (\mathcal{K}, \mathcal{E})$ whose vertex set is the set of maximal cliques of G such that each of the induced subgraphs $T[\mathcal{K}_v]$ is connected.*

Proof. $(iii) \Rightarrow (ii)$: Assume that there exists a tree $T = (\mathcal{K}, \mathcal{E})$ that satisfies (iii) and let $u, v \in V$. Now $(u, v) \in E \Leftrightarrow u, v \in A$ for some clique $A \in \mathcal{K} \Leftrightarrow \mathcal{K}_u \cap \mathcal{K}_v \neq \emptyset \Leftrightarrow T[\mathcal{K}_u] \cap T[\mathcal{K}_v] \neq \emptyset$. Thus G is the intersection graph of the family of subtrees $\{T[\mathcal{K}_v] \mid v \in V\}$ of T.

$(ii) \Rightarrow (i)$: Let $\{T_v \mid v \in V\}$ be a family of subtrees of a tree T such that $(u, v) \in E \Leftrightarrow T_u \cap T_v \neq \emptyset$. Suppose that G contains a chordless cycle $v_0, v_1, ..., v_{k-1}, v_0$ with $k > 3$ corresponding to the sequence of subtrees $T_0, T_1, ..., T_{k-1}, T_0$ of the tree T; that is $T_i \cap T_j \neq \emptyset \Leftrightarrow i$ and j differ by at most 1 modulo k. All arithmetic will be done in modulo k.

Choose a vertex a_i from $T_i \cap T_{i+1}$ for $i = 0, ..., k-1$. Let b_i be the last common vertex on the unique simple paths from a_i to a_{i-1} and a_i to a_{i+1}. These paths lie in T_i and T_{i+1} respectively, so that b_i also lies in $T_i \cap T_{i+1}$. Let P_{i+1} be the simple path connecting b_i and b_{i+1}. Clearly $P_i \subseteq T_i$, so $P_i \cap P_j = \emptyset$ for i and j differing by more than 1 mod k. Moreover, $P_i \cap P_{i+1} = \{b_i\}$ for $i = 0, ..., k-1$. Thus, $\bigcup_i P_i$ is a simple cycle in T contradicting the definition of a tree.

$(i) \Rightarrow (iii)$: We use induction on the size of G. Assume that the implication is true for all graphs having fewer vertices than G. If G is complete then T is a single vertex and the result is trivial. If G is disconnected then by induction there exists a corresponding tree satisfying (iii) for each connected component of G. These trees can combined to a connected tree satisfying (iii) by adding edges between the trees.

Let us assume that G is connected and not complete. Choose a simplicial vertex a of G, and let $A = \{a\} \cup N(a)$. Clearly A is a maximal clique of G. Let $U = \{u \in A \mid N(u) \subset A\}$, and $Y = A - U$. The sets U, Y, and $V - A$ are nonempty since G is connected and not complete. Consider the induced subgraph $G' = G[V - U]$, which is chordal and has fewer vertices than G. By induction, let T' be a tree whose vertex set K' is the set of maximal cliques of G' such that for each vertex $v \in V - U$, the set $K'_v = \{X \in K' \mid v \in X\}$ induced a connected subtree of T'.

Now, either $K = K' + \{A\} - \{Y\}$ or $K = K' + \{A\}$ depending on whether or not Y is a maximal clique of G'. Let B be any maximal clique of G' containing Y. 1. If $B = Y$, then we obtain T from T' by renaming $B \to A$. 2. If $B \neq Y$, then we obtain T from T' by connecting a new vertex A to B. In either case (1 or 2), $K_u = \{A\}$ for all $u \in U$ and $K_v = K'_v$ for all $v \in V - A$, each of which induces a subtree of T. We need only worry about the sets $\{K_y \mid y \in Y\}$. In case 1, $K_y = K'_y + \{A\} - \{B\}$, which induces the same subtree as K'_y since only names were changed. In case 2, $K_y = K'_y + \{A\}$, which clearly induces a subtree.

Thus we have constructed the required tree T and the proof of the theorem is complete. □

Example 5.11 *Let G be the graph shown in Figure 2 a). Let T_1 be the tree given in b) and T_2 the tree shown in c) of the same figure. We will show that both trees have families of subtrees which G is an intersection graph of.*

Let F_1 be the family of subtrees of T_1 induced by the following subsets: $\{1\}, \{1,2,3\}, \{3\}, \{3,2,4\}, \{4\}, \{1,2,4\}$. Then G is the intersection graph of F_1, where vertices $a, b, c, ..., f$ of G correspond to the subsets in the given order.

Let F_2 be the family of subtrees of T_2 induced by the following subsets: $\{t,u\}$, $\{u,v,w\}$, $\{w,x\}$, $\{w,v,y\}$,$\{y,z\}$, $\{y,v,u\}$. Again, G is the intersection graph of F_2, where the vertices of G correspond to the subsets in the presented order.

5.2 Clique trees

A tree with the property described in Theorem 5.10 (*iii*) is in fact called a *clique tree* of G. Recall the definition of a tree decomposition. If G is a chordal graph, then any clique tree of G is also a tree decomposition of G. However, the opposite is not necessarily true. The proof of Theorem 5.10 described how to construct a clique tree. In this section, we will concentrate on constructing the clique tree in practice. We will first see how to find all the maximal cliques of a chordal graph efficiently.

The problem of finding all the maximal cliques of a general graph is NP-hard. However, for chordal graphs, the number of maximal cliques is at most $|V|$, and these can be found in linear time by a modification of Maximum Cardinality Search (MCS) [24]. The original MCS algorithm proceeds as follows: Start with an arbitrary vertex v and give v the label $|V|$. Next, select a vertex w with highest number of already labeled neighbors, and give w the label $|V| - 1$.

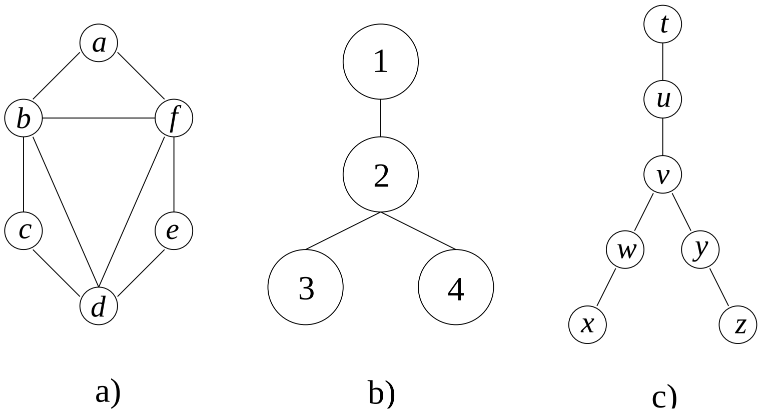

Figure 2: The graphs of Example 5.11.

Continue this process until all the vertices are labeled. The resulting labeling is actually a perfect elimination ordering of G when G is a chordal graph [24]. The MCS algorithm can be implemented to run in $O(n+m)$ time for a graph with n vertices and m edges.

The modified MCS algorithm finds all the maximal cliques of a chordal graph [4]. It proceeds as follows: Start with an arbitrary vertex v, give v the label $|V|$, and start a new maximal clique that contains v. Let $k=0$. At each next step, select a vertex w with highest number of already labeled neighbors and let this number be k'; give w the label $|V|-1$. If $k'>k$ then this means that we are still within the same maximal clique as in the previous step; thus place w in the current clique. Otherwise, start a new clique which w and all its already numbered neighbors belong to. In either case, set $k'=k$ at the end of each step. With a few additions to this algorithm, it can actually construct the clique tree, too.

A general graph can have many minimal separators. However, for connected chordal graphs the number of minimal separators is at most $|V|-1$. The clique tree is a useful structure to express the information on maximal cliques and minimal separators of a chordal graph.

Definition 5.12 *The* clique graph *of a chordal graph G is a graph $\mathcal{G}$ whose vertices are the cliques of G and two vertices are connected by an edge if their corresponding cliques intersect in G.*

Let us define an edge (C_i, C_j) in a clique graph to be equivalent to the set of vertices in $C_i \cap C_j$ for the cliques C_i and C_j in G. It is important to note that both the vertices and the edges represent cliques that belong to G. The vertices represent the maximal cliques, where the edges represent the intersection between maximal cliques. For the next theorem, let the *weight* of an edge of $\mathcal{G}$ be the number of vertices in the intersection it represents.

Theorem 5.13 [3] *A* clique tree *of a chordal graph G is a maximum weight spanning tree of the clique graph of G.*

Example 5.14 *Let G be the graph given in Figure 3 a). The clique graph $\mathcal{G}$ of G is given in Figure 3 b), and a clique tree, which for this example is unique, is given in Figure 3 c). The numbers on the edges of the clique graph are the weights of the edges.*

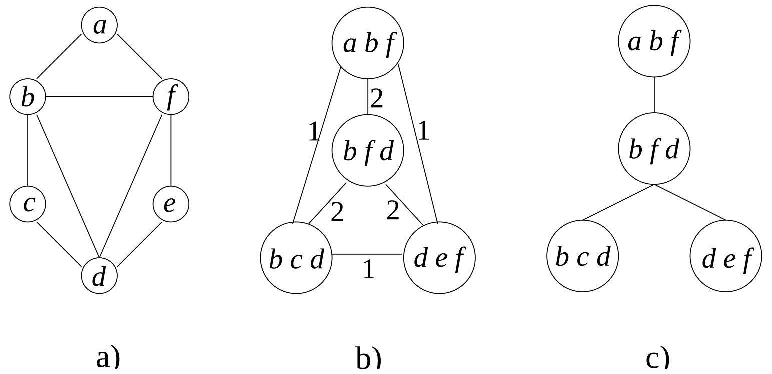

Figure 3: The graphs of Example 5.14.

Note that the clique graph of G is unique, whereas G may have several different clique trees.

Theorem 5.15 [18] *Given a chordal graph G and any clique tree T of G, a set of vertices S is a minimal separator of G if and only if $S = C_i \cap C_j$ for an edge (C_i, C_j) in T.*

Thus a clique tree is a convenient tool to represent all the maximal cliques and the minimal separators of a chordal graph. It follows that the number of minimal separators of a chordal graph $G = (V, E)$ is at most $|V| - 1$.

The following connections can be proved using the previous lemmas and theorems.

Lemma 5.16 *Given any graph G and a tree decomposition $(\{X_i \mid i \in I\}, \ T = (I, M))$ of G, let H be the graph obtained by adding edges to G so that each X_i becomes a clique. Then H is chordal.*

Lemma 5.17 *Let G be any graph, and let H be a triangulation of G with the smallest possible treewidth. Then $tw(G) = tw(H)$.*

Exercise 5.18 *Prove the two lemmas above.*

From the two lemmas above, we can readily conclude that any clique tree of a chordal graph G is a tree decomposition of G of minimum width. Thus for a chordal graph G, the treewidth is one less than the size of the largest clique in G, and hence can be found in linear time. Note also that, as a consequence, when G is not chordal, finding a triangulation of minimum treewidth is equivalent to finding a triangulation of minimum largest clique size (i.e., the largest clique is as small as possible). Finding such a triangulation is an NP-hard problem.

6 Interval graphs

In the previous sections we mentioned that chordal graphs are intersection graphs of subtrees of a tree. Interval graphs are an important subclass of chordal graphs, and they are the intersection graphs of subpaths of a path. This will be clear from the following definition:

Definition 6.1 *A graph $G = (V, E)$ is an* interval graph *if there is a mapping I of the vertices of G into sets of consecutive integers such that for each pair of vertices $v, w \in V$ the following is true: $(v, w) \in E \Leftrightarrow I(v) \cap I(w) \neq \emptyset$.*

Theorem 6.2 [15] *G is an interval graph if and only if G has a clique tree that is a simple path.*

Example 6.3 *Let G be the graph shown in Figure 4 a). A clique tree T of G is given in b). To see that G is an interval graph, we can use the following mapping I: $I(a) = \{1\}$, $I(b) = \{1, 2, 3\}$, $I(c) = \{3, 4\}$, $I(d) = \{4\}$, $I(e) = \{2, 3, 4\}$, $I(f) = \{1, 2\}$.*

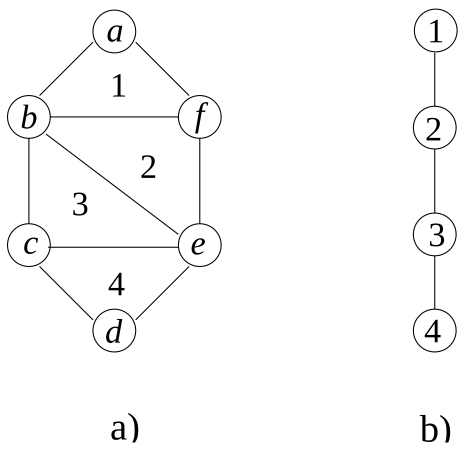

Figure 4: The graphs of Example 6.3.

Observe that the given clique tree is also a tree decomposition and a path decomposition of G. Since interval graphs have clique trees that are paths, these will also correspond to path decompositions. Because of the connection between tree decompositions and clique trees for chordal graphs, we can see that pathwidth equals to treewidth for interval graphs.

For a general graph G, the pathwidth is one less than the minimum size of the largest clique in an interval completion of G. Finding an interval completion where the size of the largest clique is as small as possible is an NP-hard problem. Even for chordal graphs it is an open question whether or not such an interval completion can be found in polynomial time.

Interval graphs have at least as many applications as chordal graphs. Many scheduling problems can be modeled as interval graphs, since the intervals often represent time intervals. Here is an example from [16]: Let $c_1, c_2, ..., c_n$ be chemical compounds that must be refrigerated under closely monitored conditions. If compound c_i must be kept at a *constant* temperature between t_i and t'_i degrees, how many refrigerators are needed to store all the compounds? To model this problem, let G be an interval graph with vertices $c_1, c_2, ..., c_n$, and connect two vertices whenever the temperature intervals of their corresponding compounds intersect. By the Helly property, for each clique K in G, there will be at least one temperature t such that t is within the interval of each vertex of K. A refrigerator that is set at temperature t will thus be able to store all compounds whose vertices belong to K. Thus the number of refrigerators can be minimized by finding a minimum clique cover of G.

7 NP-hard problems that are solvable in polynomial time for chordal and interval graphs

In Section 5, we already saw how to compute $\omega(G)$ in linear time when G is a chordal graph. Actually, when G is chordal, also $\chi(G), \alpha(G)$, and $k(G)$ can be computed in linear time.

Lemma 7.1 *If G is a chordal graph, then $\omega(G) = \chi(G)$ and $\alpha(G) = k(G)$.*

The above lemma can be proved using a clique tree of G. Let us, for example, consider a maximum independent set of G. Let T be a clique tree of G, and let Z be a bag which is a leaf in T. Observe that Z must contain at least one simplicial vertex v, and that G has a maximum independent set S that contains v. The idea behind this is very similar to the idea of maximum independent sets for trees. At most one vertex of Z can belong to S. If any other vertex of Z belongs to S, we can swap this vertex with v. If no other vertex of Z belongs to S then v can definitely be placed in S, which contradicts S being of maximum size. So, we can place a simplicial vertex in S, delete the whole neighborhood of this vertex from G, and continue in this manner. This process will also give us a minimum clique cover of G. For the chromatic number, the algorithm for listing all maximal cliques of G can be modified to give $\chi(G)$.

Exercise 7.2 *Prove Lemma 7.1.*

In fact, chordal graphs belong to a larger class of graphs called *perfect graphs*.

Definition 7.3 *A graph $G = (V, E)$ is* perfect *if it satisfies the following properties:*

1. $\omega(G[X]) = \chi(G[X])$ *for all* $X \subseteq V$
2. $\alpha(G[X]) = k(G[X])$ *for all* $X \subseteq V$.

Obviously, any graph G satisfies 1. if and only if its complement graph $\bar{G}$ satisfies 2. In fact, it can be shown that these two requirements are equivalent. Thus a graph is perfect if and only if its complement is perfect.

Theorem 7.4 [21] (The perfect graph theorem) *For a graph $G = (V, E)$, the following are equivalent:*

- $\omega(G[X]) = \chi(G[X])$ *for all* $X \subseteq V$
- $\alpha(G[X]) = k(G[X])$ *for all* $X \subseteq V$.
- $\omega(G[X]) \cdot \alpha(G[X]) \geq |X|$ *for all* $X \subseteq V$.

Theorem 7.5 [16] *Chordal graphs are perfect.*

In 1960 Claude Berge conjectured that a graph is perfect if and only if neither the graph nor its complement contains a chordless cycle of odd length at least 5. Recently, this conjecture was proved by Paul Seymour and one of his students, Maria Chudnovsky, based upon their previous work with Neil Robertson and Robin Thomas. The proof is 150 pages long.

Theorem 7.6 [8] (The strong perfect graph theorem) *A graph G is perfect if and only if neither G nor $\bar{G}$ contains a chordless cycle of odd length ≥ 5.*

We have seen examples of NP-hard problems that become solvable in linear time for chordal, and thus, interval graphs. Since interval graphs are a more restricted class than chordal graphs, there are naturally problems for which polynomial time algorithms are developed for interval graphs, but not for chordal graphs. Two such examples are *vertex ranking* and *bandwidth*.

The vertex ranking (also called *minimum elimination tree height*) problem asks to find a labeling of the vertices of the input graph G with a minimum number of different labels in such a way that every path between a pair of vertices with the same label should contain at least one vertex with a higher label. This problem is solvable in polynomial time for interval graphs [2], but it is not known whether a polynomial time algorithm exists for chordal graphs.

The bandwidth problem asks to find an ordering of the vertices of G from 1 to n, such that the largest difference between the endpoints of an edge is minimized. This problem is also solvable in polynomial time for interval graphs [20], whereas it is NP-hard for chordal graphs and even for trees with maximum degree 3 [13].

References

[1] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

[2] B. Aspvall and P. Heggernes. Finding minimum height elimination trees for interval graphs in polynomial time. *BIT*, 34:484–509, 1994.

[3] P. A. Bernstein and N. Goodman. Power of natural semijoins. *SIAM J. Comput.*, 10(4):751–771, 1981.

[4] J. R. S. Blair and B. W. Peyton. An introduction to chordal graphs and clique trees. In J. A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Sparse Matrix Computations: Graph Theory Issues and Algorithms*, pages 1–30. Springer Verlag, 1993. IMA Volumes in Mathematics and its Applications, Vol. 56.

[5] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[6] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[7] H. L. Bodlaender and R. H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Disc. Meth.*, 6:238–255, 1993.

[8] M. Chudnovsky, N. Roberston, P. Seymour, and R. Thomas. The strong perfect graph theorem. *Annals of Mathematics*. To appear.

[9] F. R. K. Chung and D. Mumford. Chordal completions of planar graphs. *J. Comb. Theory*, 31:96–106, 1994.

[10] G.A. Dirac. On rigid circuit graphs. *Anh. Math. Sem. Univ. Hamburg*, 25:71–76, 1961.

[11] R. E. England, J. R. S. Blair, and M. G. Thomason. Independent computations in a probablistic knowledge-based system. Technical Report CS-90-128, Department of Computer Science, The University of Tennessee, Knoxville, Tennessee, 1991.

[12] D.R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Math.*, 15:835–855, 1965.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Co., 1978.

[14] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Combin. Theory Ser. B*, 16:47–56, 1974.

[15] P. C. Gilmore and A. J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, 16:539–548, 1964.

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

[17] P. Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*. To appear.

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

[19] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley, 2005.

[20] D. J. Kleitman and R. V. Vohra. Computing the bandwidth of interval graphs. *SIAM J. Disc. Math.*, 3(3):373–375, 1990.

[21] L. Lovasz. A characterization of perfect graphs. *J. Comb. Theory*, 13:95–98, 1972.

[22] D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, New York, 1972.

[23] D. J. Rose. On simple characterizations of k-trees. *Discrete Math.*, 7:317–322, 1974.

[24] 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. Comput.*, 13:566–579, 1984.

[25] J. van Leeuwen. Graph algorithms. In *Handbook of Theoretical Computer Science, A: Algorithms and Complexity Theory*. North Holland, 1990.

[26] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.