Perfect Graphs

Chinh T. Hoang
Wilfrid Laurier University

R. Sritharan
University of Dayton

Follow this and additional works at: http://ecommons.udayton.edu/cps_fac_pub
Part of the Graphics and Human Computer Interfaces Commons, and the Other Computer Sciences Commons

eCommons Citation
Hoang, Chinh T. and Sritharan, R., "Perfect Graphs" (2015). Computer Science Faculty Publications. 87.
http://ecommons.udayton.edu/cps_fac_pub/87
# Perfect Graphs

Chinh T. Hoàng*

R. Sritharan†

## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>28.1</td>
<td>Introduction</td>
<td>708</td>
</tr>
<tr>
<td>28.2</td>
<td>Notation</td>
<td>710</td>
</tr>
<tr>
<td>28.3</td>
<td>Chordal Graphs</td>
<td>710</td>
</tr>
<tr>
<td>28.3.1</td>
<td>Characterization</td>
<td>710</td>
</tr>
<tr>
<td>28.3.2</td>
<td>Recognition</td>
<td>712</td>
</tr>
<tr>
<td>28.3.3</td>
<td>Optimization</td>
<td>715</td>
</tr>
<tr>
<td>28.4</td>
<td>Comparability Graphs</td>
<td>715</td>
</tr>
<tr>
<td>28.4.1</td>
<td>Characterization</td>
<td>715</td>
</tr>
<tr>
<td>28.4.2</td>
<td>Recognition</td>
<td>718</td>
</tr>
<tr>
<td>28.4.2.1</td>
<td>Transitive Orientation Using Modular Decomposition</td>
<td>720</td>
</tr>
<tr>
<td>28.4.2.2</td>
<td>Modular Decomposition</td>
<td>720</td>
</tr>
<tr>
<td>28.4.2.3</td>
<td>From the Modular Decomposition Tree to Transitive Orientation</td>
<td>721</td>
</tr>
<tr>
<td>28.4.2.4</td>
<td>How Quickly Can Comparability Graphs Be Recognized?</td>
<td>722</td>
</tr>
<tr>
<td>28.4.3</td>
<td>Optimization</td>
<td>725</td>
</tr>
<tr>
<td>28.5</td>
<td>Interval Graphs</td>
<td>726</td>
</tr>
<tr>
<td>28.5.1</td>
<td>Characterization</td>
<td>727</td>
</tr>
<tr>
<td>28.5.2</td>
<td>Recognition</td>
<td>728</td>
</tr>
<tr>
<td>28.5.3</td>
<td>Optimization</td>
<td>728</td>
</tr>
<tr>
<td>28.6</td>
<td>Weakly Chordal Graphs</td>
<td>729</td>
</tr>
<tr>
<td>28.6.1</td>
<td>Characterization</td>
<td>729</td>
</tr>
<tr>
<td>28.6.2</td>
<td>Recognition</td>
<td>731</td>
</tr>
<tr>
<td>28.6.3</td>
<td>Optimization</td>
<td>732</td>
</tr>
<tr>
<td>28.6.4</td>
<td>Remarks</td>
<td>733</td>
</tr>
<tr>
<td>28.7</td>
<td>Perfectly Orderable Graphs</td>
<td>733</td>
</tr>
<tr>
<td>28.7.1</td>
<td>Characterization</td>
<td>735</td>
</tr>
<tr>
<td>28.7.2</td>
<td>Recognition</td>
<td>735</td>
</tr>
<tr>
<td>28.7.3</td>
<td>Optimization</td>
<td>738</td>
</tr>
<tr>
<td>28.8</td>
<td>Perfectly Contractile Graphs</td>
<td>741</td>
</tr>
<tr>
<td>28.9</td>
<td>Recognition of Perfect Graphs</td>
<td>742</td>
</tr>
<tr>
<td>28.10</td>
<td>x-Bounded Graphs</td>
<td>744</td>
</tr>
</tbody>
</table>

* Acknowledges support from NSERC of Canada.
† Acknowledges support from the National Security Agency, Fort Meade, Maryland.
28.1 INTRODUCTION

This chapter is a survey on perfect graphs with an algorithmic flavor. Our emphasis is on important classes of perfect graphs for which there are fast and efficient recognition and optimization algorithms. The classes of graphs we discuss in this chapter are chordal, comparability, interval, perfectly orderable, weakly chordal, perfectly contractile, and \( \chi \)-bound graphs. For each of these classes, when appropriate we discuss the complexity of the recognition algorithm and algorithms for finding a minimum coloring, and a largest clique in the graph and its complement.

In the late 1950s, Berge [1] started his investigation of graphs \( G \) with the following properties: (i) \( \alpha(G) = \theta(G) \), that is the number of vertices in a largest stable set is equal to the smallest number of cliques that cover \( V(G) \) and (ii) \( \omega(G) = \chi(G) \), that is the number of vertices in a largest clique is equal to the smallest number of colors needed to color \( G \). At about the same time, Shannon [2] in his study of the zero-error capacity of communication channels asked: (iii) what are the minimal graphs that do not satisfy (i) ?, and (iv) what is the zero-error capacity of the chordless cycle on five vertices? In today's language, the graphs \( G \) all of whose induced subgraphs satisfy (ii) are called perfect.

In 1959, it was proved [3] that chordal graphs (graphs such that every cycle of length at least four has a chord) satisfy (i), that is complements of chordal graphs are perfect. In 1960, it was proved [1] that chordal graphs are perfect. These two results led Berge to propose two conjectures which after many years of work by the graph theory community were proved to hold.

**Theorem 28.1** (Perfect graph theorem) If a graph is perfect, then so is its complement.

**Theorem 28.2** (Strong perfect graph theorem) A graph is perfect if and only if it does not contain an odd chordless cycle with at least five vertices, or the complement of such a cycle.

Perfect graphs are prototypes of min-max characterizations in combinatorics and graph theory. The theory of perfect graphs can be used to prove well known theorems such as the Dilworth's theorem on partially ordered sets [4], or the König's theorem on edge coloring of bipartite graphs [5]. On the other hand, algorithmic considerations of perfect graphs have given rise to techniques such as clique cutset decomposition, and modular decomposition. Question (iv) was answered completely in [6]; in the process of doing so, the so-called Lovász's theta function \( \Theta \) were introduced. Theta function satisfies \( \omega(G) \leq \Theta(G) \leq \chi(G) \) for any graph \( G \). Thus, a perfect graph \( G \) has \( \omega(G) = \Theta(G) = \chi(G) \). Subsequently, [7] gave a polynomial time algorithm based on the ellipsoid method to compute \( \Theta(G) \) for any graph \( G \). As a consequence, a largest clique and an optimal coloring of a perfect graph can be found in polynomial time. Furthermore, the algorithm of [7] is robust in the sense of [8]: given the input graph \( G \), it finds a largest clique and an optimal coloring, or says correctly that \( G \) is not perfect; [7] is also the first important paper in the now popular field of semidefinite programming (see [9]).

This paper is a survey on perfect graphs with an algorithmic flavor. Even though there are now polynomial time algorithms for recognizing a perfect graph and for finding an optimal coloring—and a largest clique—of such a graph, they are not considered fast or efficient. Our emphasis is on important classes of perfect graphs for which there are fast and efficient recognition and optimization algorithms. The purpose of this survey is to discuss these classes
of graphs, named below, together with the complexity of the recognition problem and the optimization problems. The reader is referred to [10–12] for background on perfect graphs.

Chordal graphs form a class of graphs among the most studied in graph theory. Besides being the impetus for the birth of perfect graphs, chordal graphs have been studied in contexts such as matrix computation and database design. Chordal graphs have given rise to well known search methods such as lexicographic breadth-first search and maximum cardinality search. We discuss chordal graphs in Section 28.3.

Comparability graphs (the graphs of partially order sets) are also among the earliest known classes of perfect graphs. The well-known Dilworth’s theorem—stating that in a partially ordered set, the number of elements in a largest anti-chain is equal to the smallest number of chains that cover the set—is equivalent to the statement that complements of comparability graphs are perfect. Early results of [13] and [14] imply polynomial time algorithms for comparability graph recognition. But despite much research, there is still no linear-time algorithm for the recognition problem. It turns out that recognizing comparability graphs is equivalent to testing for a triangle in a graph, via an $O(n^2)$ time reduction. We discuss comparability graphs in Section 28.4.

Interval graphs are the intersection graphs of intervals on a line. Besides having obvious application in scheduling, interval graphs have interesting structural properties. For example, interval graphs are precisely the chordal graphs whose complements are comparability graphs. We discuss interval graphs in Section 28.5.

Weakly chordal graphs are graphs without chordless cycles with at least five vertices and their complements. This class of graphs generalizes chordal graphs in a natural way. For weakly chordal graphs, there are efficient, but not linear time, algorithms for the recognition and optimization problems. We discuss weakly chordal graphs in Section 28.6.

An order on the vertices of a graph is perfect if the greedy (sequential) coloring algorithm delivers an optimal coloring on the graph and on its induced subgraphs. A graph is perfectly orderable if it admits a perfect order. Chordal graphs and comparability graphs admit perfect orders. Complements of chordal graphs are also perfectly orderable. Recognizing perfectly orderable graphs is NP-complete; however, there are many interesting classes of perfectly orderable graphs with polynomial time recognition algorithms. We discuss perfectly orderable graphs in Section 28.7.

An even-pair is a set of two nonadjacent vertices such that all chordless paths between them have an even number of edges. If a graph $G$ has an even-pair, then by contracting this even-pair we obtain a graph $G'$ satisfying $\omega(G) = \omega(G')$ and $\chi(G) = \chi(G')$. Furthermore, if $G$ is perfect, then so is $G'$. Perfectly contractile graphs are those graphs $G$ such that, starting with any induced subgraph of $G$ by repeatedly contracting even-pairs we obtain a clique. Weakly chordal graphs and perfectly orderable graphs are perfectly contractile. We discuss perfectly contractile graphs in Section 28.8.

Recently, a polynomial time algorithm for recognizing perfect graphs was given in [15]. We give a sketch of this algorithm in Section 28.9.

A graph $G$ is $\chi$-bound if there is a function $f$ such that $\chi(G) \leq f(\omega(G))$. Perfect graphs are $\chi$-bound. Identifying sufficient conditions for a graph to be $\chi$-bound is an interesting problem. It is proved in [16] that a graph is $\chi$-bound if it does not contain an even chordless cycle. One many ask a similar question for odd cycles [17]: Is it true that a graph is $\chi$-bound if it does not contain an odd chordless cycle with at least five vertices? In Section 28.10, we discuss this question and related conjectures.

We give the definitions used in this chapter in Section 28.2.
28.2 NOTATION

For graph $G = (V, E)$ and $x \in V$, $N_G(x)$ is the neighborhood of $x$ in $G$; we omit the subscript $G$ when the context is clear. Let $d(x)$ denote $|N(x)|$. For $S \subseteq V$, $G[S]$ denotes the subgraph of $G$ induced by $S$, and $G - S$ denotes $G[V - S]$; for $x \in V$, we use $G - x$ for $G - \{x\}$.

$\omega(G)$ is the number of vertices in a largest clique in $G$. $\alpha(G)$ is the number of vertices in a largest stable set in $G$. $\chi(G)$ is the chromatic number of $G$. $\Theta(G)$ is the smallest number of cliques that cover the vertices of $G$. A clique is maximal if it is not a proper subset of another clique. For $A, B \subseteq V$ such that $G[A]$ and $G[B]$ are connected, $S \subseteq V$ is a separator for $A$ and $B$ provided $A$ and $B$ belong to different components of $G - S$. Further, $S$ is a minimal separator for $A$ and $B$ if no proper subset of $S$ is also a separator for $A$ and $B$. We will also call a set $C$ of vertices a cutset if $C$ is a separator for some sets $A, B \subseteq V$; $C$ is a minimal cutset if no proper subset of $C$ is a cutset.

We use $n$ to refer to $|V|$ and $m$ to refer to $|E|$.

In a bipartite graph $G = (X, Y, E)$, $X$ and $Y$ are the parts of the partition of the vertex-set and $E$ is the set of edges. A matching is a set of pairwise non-incident edges.

A set $C$ of $V$ is anti-connected if $C$ spans a connected subgraph in the complement $\overline{G}$ of $G$. For a set $X \subseteq V$, a vertex $v$ is $X$-complete if $v$ is adjacent to every vertex of $X$. An edge is $X$-complete if both its endpoints are $X$-complete. A vertex $v$ is $X$-null if $v$ has no neighbor in $X$.

$C_k$ denotes the chordless cycle with $k$ vertices. A hole is the $C_k$ with $k \geq 4$. An anti-hole is the complement of a hole. $P_k$ denotes the chordless path with $k$ vertices. $K_t$ denotes the clique on $t$ vertices. The $K_3$ is sometimes called a triangle. The complement of a $C_4$ is denoted by $2K_2$. The claw is the tree on four vertices with a vertex of degree 3.

For problems $A$ and $B$, $A \leq B$ via an $f(m, n)$ time reduction means that an instance of problem $A$ can be reduced to an instance of problem $B$ using an algorithm with the worst case complexity of $f(m, n)$, $A \equiv B$ via $f(m, n)$ time reductions means that we have $A \leq B$ as well as $B \leq A$ via $f(m, n)$ time reductions.

Let $O(n^\alpha)$ be the complexity of the current best algorithm to multiply two $n \times n$ matrices. It is currently known that $\alpha < 2.376$ [18].

28.3 CHORDAL GRAPHS

**Definition 28.1** A graph is chordal (or, triangulated) if it does not contain a chordless cycle with at least four vertices.

Chordal graphs can be used to model various combinatorial structures. For example, they are the intersection graphs of subtrees of a tree as we will see later. See [19] for applications of chordal graphs to sparse matrix computations. Chordal graphs are among the earliest known classes of perfect graphs [3,20,21]. We will now discuss the combinatorial structures of chordal graphs.

**28.3.1 Characterization**

**Definition 28.2** A vertex is simplicial if its neighborhood is a clique.

**Theorem 28.3** [21] A graph $G$ is chordal if and only if each of its induced subgraphs is a clique or contains two nonadjacent simplicial vertices.

To prove Theorem 28.3, we need the following two lemmas.

**Lemma 28.1** Any minimal cutset of a chordal graph $G$ is a clique.
Proof. Suppose $C$ is a minimal cutset of $G$ and $A_1, A_2$ are two distinct components of $G - C$. Further, suppose for $x \in C$ and $y \in C$, $xy \notin E(G)$. As $C$ is a minimal cutset of $G$, each of $x, y$ has a neighbor in $A_i, i = 1, 2$. Let $P_i$, $i = 1, 2$, be a shortest path connecting $x$ and $y$ in $G[A_i \cup C]$ such that all the internal vertices of $P_i$ lie in $A_i$. Then, $G[V(P_1) \cup V(P_2)]$ is a hole, a contradiction. \[\square\]

**Lemma 28.2** Let $G$ be a graph with a clique cutset $C$. Consider the induced subgraphs $G_1, G_2$ with $G = G_1 \cup G_2$ and $G_1 \cap G_2 = C$. Then, $G$ is chordal if and only if $G_1, G_2$ are both chordal.

Proof. If $G$ is chordal, then as $G_1$ and $G_2$ are induced subgraphs of a chordal graph, they themselves are chordal; this proves the only if part. For the if part, suppose each of $G_1, G_2$ is chordal, but $G$ has a hole $L$. Then, $L$ must involve a vertex from each of $G_1 - C, G_2 - C$. Therefore, $C$ contains a pair of nonadjacent vertices from $L$, contradicting $C$ being a clique. \[\square\]

**Proof of Theorem 28.3.** The if part is easy: If $G$ is a graph and $x$ is a simplicial vertex of $G$, then $G$ is chordal if and only if $G - x$ is. Now, we prove the only if part by induction on the number of vertices. Let $G$ be a chordal graph. We may assume $G$ is connected, for otherwise by the induction hypothesis, each component of $G$ is a clique or contains two nonadjacent simplicial vertices, and so $G$ contains two nonadjacent simplicial vertices. Let $C$ be a minimal cutset of $G$. By Lemma 28.1, $C$ is a clique. Thus, $G$ has two induced subgraphs $G_1, G_2$ with $G = G_1 \cup G_2$ and $G_1 \cap G_2 = C$. By the induction hypothesis, each $G_i$ has a simplicial vertex $v_i \in G_i - C$ (since $C$ is a clique, it cannot contain two nonadjacent simplicial vertices). The vertices $v_1, v_2$ remain simplicial vertices of $G$, and they are nonadjacent. \[\square\]

**Definition 28.3** For a graph $G$ and an ordering $v_1v_2 \cdots v_n$ of its vertices, let $G_i$ denote $G[{v_1, \cdots, v_n}]$. An ordering $\sigma = v_1v_2 \cdots v_n$ of vertices of $G$ is a perfect elimination scheme (p.e.s.) for $G$ if each $v_i$ is simplicial in $G_i$.

**Theorem 28.4** \cite{21,22} $G$ is chordal if and only if $G$ admits a perfect elimination scheme.

Proof. For any vertex $v$ in a chordal graph $G$, $G - v$ is also chordal; this together with Theorem 28.3 prove the only if part. Since no hole has a simplicial vertex, the if part follows. \[\square\]

**Corollary 28.1** A chordal graph $G$ has at most $n$ maximal cliques whose sizes sum up to at most $m$.

Proof. By induction on the number of vertices of $G$. Let $x$ be a simplicial vertex of $G$. Then, $\{x\} \cup N(x)$ is the only maximal clique of $G$ containing $x$. By the induction hypothesis, $G - x$ has at most $n - 1$ maximal cliques whose sizes sum up to at most $m - d(x)$. Then, the result follows. \[\square\]

**Definition 28.4** Let $F$ be a family of nonempty sets. The intersection graph of $F$ is the graph obtained by identifying each set of $F$ with a vertex, and joining two vertices by an edge if and only if the two corresponding sets have a nonempty intersection.

**Theorem 28.5** \cite{23,24} A graph is chordal if and only if it is the intersection graph of subtrees of a tree.
Proof. By induction on the number of vertices. We prove the if part first. Let \( G = (V, E) \) be a graph that is the intersection graph of a set \( S \) of subtrees of a tree \( T \), that is, every vertex \( v \) of \( V \) is a subtree \( T_v \) of \( T \), and two vertices \( v, u \in V \) are adjacent if and only if \( T_v \) and \( T_u \) intersect. We may assume \( G \) is connected, for otherwise, we are done by the induction hypothesis. By Lemma 28.2, and the induction hypothesis, we only need prove \( G \) is a clique, or contains a clique cutset. We may assume \( G \) is not a clique, and let \( u, v \) be two nonadjacent vertices of \( G \). Then, \( T_u \cap T_v = \emptyset \). Let \( P = x_1, \ldots, x_n \) be the path in \( T \) with \( x_1 \in T_u, x_n \in T_v \) such that all interior vertices of \( P \) are not in \( T_u \cup T_v \). Since \( T \) is a tree, \( P \) is unique; furthermore, all paths with one endpoint in \( T_u \) and the other endpoint in \( T_v \) must contain all vertices of \( P \). Thus, \( x_1x_2 \) is a cut-edge of \( T \). Let \( S' \) be the set of all subtrees of \( S \) that contains the edge \( x_1x_2 \). Then in \( G \), the set \( C \) of vertices that corresponds to the subtrees of \( S' \) forms a clique. We claim \( C \) is a cutset of \( G \). In \( G \), consider a path from \( u \) to \( v \); let the vertices of this path be \( u = t_1, t_2, \ldots, v = t_k \). Some subtree \( T_{t_i} \) must contain the edge \( x_1x_2 \) (because it is the cut-edge of \( T \)). Thus, the vertex that corresponds to \( T_{t_i} \) is in \( C \). We have established the if part.

Now, we prove the only if part. Let \( G = (V, E) \) be a chordal graph. We will prove that there is a tree \( T \) and a family \( S \) of subtrees of \( T \) such that (i) the vertices of \( T \) are the maximal cliques of \( G \), and (ii) for each \( v \in V \), the set of maximal cliques of \( G \) containing \( v \) induces a subtree of \( T \). The proof is by induction on the number of vertices. Suppose that \( G \) is disconnected. Then, the induction hypothesis implies for each component \( C_i \) of \( G \), there is a tree \( T_i \) satisfying (i) and (ii). Construct the tree \( T \) from the trees \( T_i \) by adding a new root vertex \( r \) and joining \( r \) to the root of each \( T_i \). It is easy to see that \( T \) satisfies (i) and (ii). So, \( G \) is connected. We may assume \( G \) is not a clique, for otherwise we are easily done. Consider a simplicial vertex \( v \) of \( G \). As \( v \) is simplicial in \( G \), it is not a cut vertex of \( G \) and therefore, \( G - v \) is connected. By the induction hypothesis, the graph \( G - v \) is the intersection graph of a set \( B \) of subtrees of a tree \( T_B \) satisfying (i) and (ii). Let \( K \) be a maximal clique of \( G - v \) containing \( N_G(v) \) and let \( t_k \) be the vertex of \( T_B \) that corresponds to \( K \). If \( K = N_G(v) \), then we simply add \( v \) to \( t_k \) to get the tree \( T \) from \( T_B \). Otherwise, let \( K' = N_G(v) \cup \{v\} \). Let \( T \) be the tree obtained from \( T_B \) by adding a new vertex \( t_{K'} \) and the edge \( t_kt_{K'} \). Let \( T_K \) be the subtree formed by the single vertex \( t_{K'} \). We construct \( S \) as follows. Add \( T_K \) to \( S \); for each tree \( T_u \in B \), if \( T_u \) corresponds to a vertex in \( N_G(v) \), then add the tree \( T_u \cup \{t_{K'}\} \); otherwise, add \( T_u \) to \( S \). It is seen that (i) and (ii) hold for \( T \) and \( S \).

### 28.3.2 Recognition

Given \( G \), an approach to testing whether \( G \) is chordal is: first generate an ordering \( \sigma \) of vertices of \( G \) that is guaranteed to be a perfect elimination scheme for \( G \) when \( G \) is chordal; then, verify whether \( \sigma \) is indeed a perfect elimination scheme for \( G \). The first linear-time algorithm to generate a perfect elimination scheme of a chordal graph is given in [25]; it uses the lexicographic breadth-first search (LexBFS). We present the maximum cardinality search algorithm for the same purpose.

The maximum cardinality search algorithm (MCS), introduced in [26], is used to construct an ordering of vertices of a given graph; the ordering is constructed incrementally right to left (if \( a \) comes before \( b \) in the order, then we consider \( a \) to be to the left of \( b \) ). An arbitrary vertex is chosen to be the last in the ordering. In each remaining step, from the vertices still not chosen (unlabeled vertices), one with the most neighbors among the already chosen vertices (labeled vertices) is picked with the ties broken arbitrarily.
Algorithm 28.1 MCS

```
input: graph $G$
output: ordering $\sigma = v_1v_2 \cdots v_n$ of vertices of $G$

$v_n \leftarrow$ an arbitrary vertex of $G$;
for $i \leftarrow n - 1$ downto 1 do
    $v_i \leftarrow$ unlabeled vertex adjacent to the most in $\{v_{i+1}, \ldots, v_n\}$;
end for
```

Theorem 28.6 [26] Algorithm MCS can be implemented to run in $O(m + n)$ time.

Proof. We keep the array $set[0] \cdots set[n - 1]$ where $set[j]$ is a doubly linked list of all the unlabeled vertices that are adjacent to exactly $j$ labeled vertices. Thus, initially every vertex belongs to $set[0]$. For each vertex, we maintain the array index of the set it belongs to as well as a pointer to the node containing it in the $set[i]$ lists. Finally, we maintain $last$, the largest index such that $set[last]$ is nonempty. In the $i^{th}$ iteration of the algorithm, a vertex in $set[last]$ is taken to be $v_i$ and $v_i$ is deleted from $set[last]$. For every unlabeled neighbor $w$ of $v_i$, if $w$ belongs to $set[i]$, then we move $w$ from $set[i]$ to $set[i + 1]$. As each set is implemented as a doubly linked list, a single addition or deletion can be done in constant time, and hence all of the above operations can be done in $O(d(v_i))$ time. Finally, in order to update the value of $last$, we increment $last$ once and then we repeatedly decrement the value of $last$ until $set[last]$ is nonempty. As $last$ is incremented at most $n$ times and its value is never less than $-1$, the overall time spent manipulating $last$ is $O(n)$ and we have the claimed complexity. $lacksquare$

Definition 28.5 For vertices $x, y$ of graph $G$ and an ordering $\sigma$ of vertices of $G$, $x <_\sigma y$ denotes that $x$ precedes $y$ in $\sigma$.

Lemma 28.3 [26] Let $\sigma$ be the output of algorithm MCS on chordal graph $G$. Then, $G$ does not have a chordless path $P = (x = u_0)u_1 \cdots u_{k-1}(u_k = y)$ with $k \geq 2$ such that $u_i <_\sigma x$, $1 \leq i \leq k - 1$, and $x <_\sigma y$.

Proof. Suppose such a path existed; from all such chordless paths, pick $P$ so that the position of $x$ in $\sigma$ is as much to the right as possible. Given the logic of the algorithm MCS, as $u_{k-1} <_\sigma x <_\sigma y$, $u_{k-1}y \in E(G)$, and $xy \notin E(G)$, there must exist a vertex $z$ such that $x <_\sigma z$, $xz \in E(G)$, and $u_{k-1}z \notin E(G)$. Let $j$ be the largest index less than $k-1$ such that $u_jz \in E(G)$; such a $j$ exists as $xz \in E(G)$. Let $P'$ be the path $zu_j \cdots u_{k-1}y$. As $G$ is chordal and $P'$ has at least four vertices, $zy \notin E(G)$. Now, whether $x <_\sigma z <_\sigma y$ holds or $x <_\sigma y <_\sigma z$ holds, existence of the chordless path $P'$ violates the choice of $P$, a contradiction. $lacksquare$

Theorem 28.7 [26] If $G$ is chordal, then the output $\sigma = v_1v_2 \cdots v_n$ produced by the algorithm MCS is a perfect elimination scheme for $G$.

Proof. Suppose not, and let $i$ be the smallest such that $v_i$ is not simplicial in $G_i$. Then, there exist $v_j$ and $v_k$ such that $v_i <_\sigma v_j <_\sigma v_k$, $v_iv_j \in E(G)$, $v_jv_k \in E(G)$, and $v_jv_k \notin E(G)$. Then, the chordless path $P = v_jv_1v_k$ contradicts Lemma 28.3. $lacksquare$
Algorithm 28.2 chordal-recognition

**input:** graph $G$

**output:** yes when $G$ is chordal and no otherwise

Run algorithm MCS on $G$ to get $\sigma = v_1v_2\cdots v_n$;

if $\sigma$ is a perfect elimination scheme for $G$ then

output yes

else

output no

end if

Next, we discuss how to verify in linear time [25] whether $\sigma = v_1v_2\cdots v_n$ is a perfect elimination scheme for $G$. The key idea in [25] is that part of the work involved in checking whether $v_i$ is simplicial in $G_i$ can be handed over to an appropriate vertex $v_j$ such that $v_i <_{\sigma} v_j$. In particular, let $v_j$ be the smallest neighbor of $v_i$ such that $v_i <_{\sigma} v_j$. Let $L(v_i) = \{v_k | v_j <_{\sigma} v_k$ and $v_i v_k \in E(G)\}$. In other words, $L(v_i)$ is the set of those neighbors of $v_i$ that follow $v_j$ in $\sigma$.

If $v_j$ is simplicial in $G_j$ and $v_j$ is adjacent to every vertex in $L(v_i)$, then $v_i$ is simplicial in $G_i$. On the other hand, if either $v_j$ is not simplicial in $G_j$ or $v_j$ is not adjacent to some vertex in $L(v_i)$ (making $v_i$ not simplicial in $G_i$), then $\sigma$ is not a perfect elimination scheme for $G$. Further, part of the work involved in checking whether $v_j$ is simplicial in $G_j$ can likewise be deferred to a later vertex.

In the following, the list $bba(v_k)$ is the list of vertices that $v_k$ better be adjacent to; it is the concatenation of the $L(v_i)$ lists handed over to $v_k$ by the $v_i$'s preceding it in $\sigma$.

Algorithm 28.3 pes-verification

**input:** graph $G$ and ordering $\sigma = v_1v_2\cdots v_n$ of vertices of $G$

**output:** yes when $\sigma$ is a p.e.s. for $G$ and no otherwise

for $i \leftarrow 1$ to $n$ do

Initialize $bba(v_i)$ to an empty list;

end for

for $i \leftarrow 1$ to $n - 1$ do

if $v_i$ is not adjacent to some vertex in $bba(v_i)$ then

output no;

stop

end if

Let $v_j$ be the smallest neighbor of $v_i$ such that $v_i <_{\sigma} v_j$;

$L(v_i) \leftarrow \{v_k | v_j <_{\sigma} v_k$ and $v_i v_k \in E(G)\}$;

Append $L(v_i)$ to $bba(v_j)$

end for

output yes

Theorem 28.8 [25] Algorithm pes-verification can be implemented to run in $O(m+n)$ time.

Proof. Assume that the array $v[1] \cdots v[n]$ stores $\sigma$. In order to check whether $v_i$ is adjacent to every vertex in $bba(v_i)$: use a boolean array $flag[i] \cdots flag[n]$ that is initialized in the first step of the entire algorithm. Now, mark the neighbors of $v_i$ in the array $flag$. Then, traverse the list $bba(v_i)$ and check for each member of $bba(v_i)$ whether the corresponding
entry in flag is marked. Finally, unmark the neighbors of \( v_i \) in flag. Thus, this operation takes \( O(|\text{bba}(v_i)| + d(v_i)) \) time. As a vertex \( v_k \) hands over an \( L(v_k) \) list at most once, the total size of all \( \text{bba} \) lists is \( O(m+n) \) and the overall time spent on this operation is \( O(m+n) \).

The rest of the operations can easily be implemented in \( O(m+n) \) time.

### 28.3.3 Optimization

For a chordal graph, a largest clique and an optimal coloring can be found in linear time using the combined results in [25,27]. Even the weighted versions of these problems can be solved efficiently. This will be discussed in the context of the more general class of perfectly orderable graphs in Section 28.7.

The known optimization algorithms for chordal graphs use the clique cutset property. For a general graph, there are polynomial time algorithms [28,29] to find a clique cutset if one exists in the graph. [28,30] discuss optimization algorithm for classes of graphs, more general than chordal, using the clique cutset decomposition.

### 28.4 COMPARABILITY GRAPHS

**Definition 28.6** A graph \( G = (V, E) \) is a comparability graph if there is a partially ordered set \((P, \prec)\) such that \( V = P \) and two vertices of \( G \) are adjacent if and only if the corresponding elements of \( P \) are comparable in the relation \( \prec \).

**Definition 28.7** An orientation of a graph is transitive if whenever \( a \to b, b \to c \) are arcs, \( a \to c \) is an arc.

An ordered graph \((G, \prec)\) corresponds to an orientation in a natural way: for vertices \( a, b \), we orient \( a \to b \) if \( a \prec b \). Now, we can redefine the notion of a comparability graph as follows.

**Definition 28.8** A graph is a comparability graph if it admits an orientation that is both acyclic and transitive.

#### 28.4.1 Characterization

Several theorems on comparability graphs have become folklore. We start with a classical theorem of [13] that as we will see later implies a polynomial time algorithm to recognize a comparability graph.

**Theorem 28.9** [13] If a graph admits a transitive orientation, then it admits an acyclic and transitive orientation.

**Definition 28.9** A subset \( M \) of vertices of a graph \( G = (V, E) \) is a module if any vertex outside of \( M \) is either adjacent to every vertex in \( M \) or adjacent to no vertex in \( M \). Trivially, \( \{x\} \) for any \( x \in V \), and \( V \) are modules. Module \( M \) is nontrivial if \( |M| \geq 2 \) and \( M \subset V \).

To prove Theorem 28.9, we need the following.

**Theorem 28.10** [13] If a graph admits a cyclic transitive orientation, then it contains a nontrivial module.

**Proof.** Let \( G \) be a graph and let \( G' \) be transitive orientation of \( G \) containing a directed cycle \( C \). We may assume \( C \) is a shortest cycle and thus chordless. Since \( G' \) is transitive, \( C \) has length three. We may assume \( G \) has at least four vertices, for otherwise the theorem is trivially true. Let the vertices of \( C \) be \( a, b, c \) in the cyclic order, with \( a \to b, b \to c, c \to a \). A vertex \( x \)
outside \( C \) cannot have exactly one neighbor in \( C \), for otherwise \( x \) and some two vertices in \( C \) violate the transitivity of \( \overrightarrow{G} \). There must be a vertex \( v \) adjacent to exactly two vertices of \( C \), for otherwise \( C \) is a nontrivial module of \( G \). We may assume \( v \) is adjacent to \( b, c \). Let \( X \) be the set of vertices that are adjacent to \( b, c \) such that \( X \) is anti-connected, \( a, v \in X, \) and \( X \) is maximal with respect to this property. Since \( X \) is anti-connected, and \( a \to b, c \to a \), it follows that every \( x \in X \) has \( x \to b, c \to x \). We may assume \( X \) is not a module of \( G \), for otherwise we are done. Thus, there is a vertex \( u \notin X \) such that \( A = N(u) \cap X \) and \( B = X - A \) are not empty. As \( X \) is anti-connected, there are vertices \( x \in A, x' \in B \) with \( xx' \notin E(G) \). Vertex \( u \) must be adjacent to \( b, c \), for otherwise \( \{u, x, b, c\} \) violate the transitivity of \( \overrightarrow{G} \). The maximality of \( X \) means \( u \) cannot be adjacent to both \( b \) and \( c \). We may assume \( ub \in E(G), uc \notin E(G) \). Now, \( \{u, b, x'\} \) or \( \{u, b, c\} \) violates the transitivity of \( \overrightarrow{C} \).

**Lemma 28.4** Let \( G \) be a graph with a nontrivial module \( X \) and \( x \) be a vertex in \( X \). Let \( G_1 \) be the subgraph of \( G \) induced by \((V(G) - X) \cup \{x\}\), let \( G_2 \) be the subgraph of \( G \) induced by \( X \). Then \( G \) is a comparability graph if and only if both \( G_1 \) and \( G_2 \) are.

**Proof.** We obviously need only to prove the if part. Assume both \( G_1 \) and \( G_2 \) admit acyclic transitive orientations \( \overrightarrow{G}_1 \) and \( \overrightarrow{G}_2 \). An acyclic transitive orientation \( \overrightarrow{G} \) of \( G \) can be constructed as follows. Consider adjacent vertices \( a, b \) of \( G \). If \( a \to b \) is an arc in \( G_1 \) or \( G_2 \), then let \( a \to b \) be an arc of \( \overrightarrow{G} \). Otherwise, we may assume \( a \in G_1 - x, b \in X - x \). If \( a \to x \) is an arc of \( G_1 \), then let \( a \to b \) be an arc of \( \overrightarrow{G} \), else let \( b \to a \) be an arc of \( \overrightarrow{G} \). It is easy to verify that \( \overrightarrow{G} \) is an acyclic transitive orientation.

**Lemma 28.4** implies the following.

**Corollary 28.2** A minimally noncomparability graph cannot contain a nontrivial module.

**Proof of Theorem 28.9.** We prove by contradiction. Let \( G \) be a graph such that every transitive orientation of \( G \) is cyclic. Therefore, \( G \) is not a comparability graph, and so \( G \) contains an induced subgraph \( H \) that is minimally noncomparability. Therefore, every transitive orientation of \( H \) is cyclic. By Theorem 28.10, \( H \) contains a proper module, contradicting Corollary 28.2.

**Definition 28.10** Let \( G = (V, E) \) be a graph. The corresponding knotting graph is given by \( K[G] = (V_K, E_K) \) where \( V_K \) and \( E_K \) are defined as follows. For each vertex \( v \) of \( G \) there are copies \( v_1, v_2, \ldots, v_i \) in \( V_K \), where \( i \) is the number of components of \( \overrightarrow{G}[N(v)] \). For each edge \( vw \) of \( E \), there is an edge \( v_iw_j \) in \( E_K \), where \( v \) is contained in the \( j \)th component of \( \overrightarrow{G}[N(w)] \) and \( w \) is contained in the \( i \)th component of \( \overrightarrow{G}[N(v)] \).

An illustration of the knotting relation is shown in Figure 28.1. It is easy to see that if \( G \) is a comparability graph, then its knotting graph \( K(G) \) is bipartite. The converse is also true.

![Figure 28.1](image-url) Graph and its knotting graph.
Theorem 28.11 [14] A graph is a comparability graph if and only if its knotting graph is bipartite.

A characterization of comparability graphs by forbidden induced subgraphs is given in [14] (see [31] for an English translation of [14]).

Definition 28.11 A sequence $\sigma = \{y_1W_1y_2 \cdots y_{2n+1}W_{2n+1}y_1\}$ is an asteroid, more exactly a $(2n+1)$-asteroid, if the $y_i$ are pairwise distinct vertices, each $W_i$ is a path with endpoints $y_i, y_{i+1}$, and $y_i$ has no neighbor in $W_{i+n}$ (subscripts are taken modulo $2n+1$).

Theorem 28.12 [14] A graph $G$ is a comparability graph if and only if its complement $\overline{G}$ contains no asteroid.

By characterizing all minimal asteroids, a list of all minimal non-comparability graphs can be found.

Theorem 28.13 [14] A graph $G$ is a comparability graph if and only if $G$ does not contain as induced subgraphs any of the four graphs shown in Figure 28.2 or the complements of the 14 graphs shown in Figure 28.3.

![Figure 28.2](image_url) Four graphs with non-bipartite knotting graphs.

![Figure 28.3](image_url) Fourteen graphs containing a 3-asteroid.
The reader may verify that the graphs in Figure 28.2 have nonbipartite knotting graphs, and the graphs in Figure 28.3 contain a 3-asteroid.

**Definition 28.12** Given a partial order \((P, \prec)\), a chain is a set of pairwise comparable elements, an anti-chain is a set of pairwise incomparable elements.

A proof of the following well-known theorem is presented later.

**Theorem 28.14** [4] In a partially ordered set \((P, \prec)\), the size of a largest anti-chain is equal to the smallest number of chains needed to cover all elements of \(P\).

Let \((P, \prec)\) be a partial order, and let \(\overrightarrow{G}\) be the transitive orientation of the comparability graph \(G\) of \((P, \prec)\). Because of transitivity, a directed path of \(\overrightarrow{G}\) induces a clique. Thus, a chain of \(P\) corresponds to a clique of \(G\). And, an anti-chain of \(P\) corresponds to a stable set of \(G\). Thus, Theorem 28.14 is equivalent to the statement that the complements of comparability graphs are perfect.

### 28.4.2 Recognition

Consider the problem of determining whether a given graph \(G\) is a comparability graph. Equivalently, the problem asks if \(G\) can be oriented so that the resulting directed graph is acyclic and transitive. First, we consider an algorithm for the problem with the complexity of \(O(mn)\). Then, we discuss a more efficient algorithm.

Suppose \(G\) is a comparability graph, \(xy\) is an edge of \(G\), and some transitive orientation \(\overrightarrow{G}\) of \(G\) contains \(x \to y\). Then, reversing the direction of every arc in \(\overrightarrow{G}\) also yields a transitive orientation of \(G\). Therefore, if we were to test whether \(G\) admits a transitive orientation, it is enough to pick an arbitrary edge \(xy\) of \(G\) and determine whether there exists a transitive orientation of \(G\) that contains \(x \to y\).

Suppose \(xyz\) is a \(P_3\) of \(G\). If a transitive orientation of \(G\) contains \(x \to y\), then it must contain \(z \to y\) also; in this situation, we say that \(x \to y\) forces \(z \to y\). Now, the forced choice of \(z \to y\) might in turn force the orientation of some other edges. The **implication class** of \(x \to y\) consists of all the arcs that are forced, in one or more steps, by the initial choice of \(x \to y\). Clearly, for some edge \(uv\), if the implication class of \(x \to y\) contains \(u \to v\) as well as \(v \to u\), then \(G\) cannot be a comparability graph. Conversely, it can be shown [11] that if the implication class of \(x \to y\) does not contain both \(u \to v\) and \(v \to u\) for any edge \(uv\), then all the edges oriented thus far can be deleted from \(G\), and the process can be repeated on the remaining graph until it has no edges left.

**Theorem 28.15** [11] Algorithm comparability-recognition-1 is correct and it can be implemented to run in \(O(mn)\) time.

Algorithm comparability-recognition-1 produces an acyclic transitive orientation when the input graph is a comparability graph. Since the proof of its correctness is involved, we will not give it here. In this context, we note Theorem 28.9 already implies a simple polynomial time algorithm for recognizing comparability graphs: a graph \(G\) is a comparability graph if and only if for each edge \(xy\), the implication class of \(x \to y\) does not contain both \(u \to v\) and \(v \to u\) for some vertices \(u, v\). Since the number of \(P_3\) of a graph is \(O(nm)\) (each edge can be extended to at most \(n P_3\)), it is not difficult to see that all implication classes of \(G\) can be enumerated in \(O(nm)\) time, and so this simple algorithm runs in \(O(nm)\) time.
Algorithm 28.4 comparability-recognition-1

input: graph $G$
output: yes when $G$ is a comparability graph and no otherwise

$i = 1;$
while $G$ has edges left do
    Pick edge $xy$ and orient it $x \rightarrow y$;
    Enumerate the implication class $D_i$ of $x \rightarrow y$;
    if some $u \rightarrow v$ and $v \rightarrow u$ are in $D_i$ then
        output no;
        stop
    end if
    Let $E_i$ be the set of underlying edges of members of $D_i$;
    $G = G - E_i$;
    $i = i + 1$
end while
output yes

Suppose we had an algorithm that can transitively orient a given comparability graph. Then, we can combine that with an algorithm to verify whether a given orientation of a graph is acyclic and transitive to obtain an algorithm to recognize comparability graphs. This is the basis for the algorithm comparability-recognition-2.

Algorithm 28.5 comparability-recognition-2

input: graph $G$
output: yes when $G$ is a comparability graph and no otherwise

Run on $G$ an algorithm for transitively orienting a comparability graph to obtain the directed graph $H$;
if $H$ is acyclic and transitive then
    output yes
else
    output no
end if

First, we consider the second step of the algorithm comparability-recognition-2, where it is verified whether a given directed graph $H$ is acyclic and transitive. The acyclicity of $H$ can be verified in linear time using standard search algorithms. Having done that, by considering each $P_3$ of $H$, one can easily verify in $O(nm)$ time whether $H$ is transitive. A faster algorithm can be derived using multiplication of Boolean matrices. The following is folklore.

Theorem 28.16 It can be verified in $O(n^2)$ time whether a given directed acyclic graph $G$ is transitive.

Proof. Let $A$ be the adjacency matrix of $G$. Set each entry on the main diagonal of $A$ to 1. Then, $G$ is transitive if and only if $A = A^2$, where $A^2$ is computed via multiplication of Boolean matrices.

In contrast to the verification step, a given comparability graph can be transitively oriented in linear time [32]. Next, we discuss the ideas behind the algorithm.
28.4.2.1 Transitive Orientation Using Modular Decomposition

The overall idea of the algorithm is to first decompose the given comparability graph using a technique called modular decomposition, store the result of the decomposition using a unique tree structure, and then orient the edges of the graph via a post order traversal of the decomposition tree. We note that modular decomposition of graphs in general has many other applications.

Suppose $M$ is a nontrivial module in graph $G = (V, E)$. Then, $G$ can be decomposed into $G_1 = G[V - M \cup \{x\}]$ and $G_2 = G[M]$, where $x$ is any vertex in $M$. By Lemma 28.4, $G$ is a comparability graph if and only if $G_1$ and $G_2$ are. Therefore, the notion of modules is directly relevant to the problems of recognizing comparability graphs and finding a transitive orientation of a comparability graph. Lemma 28.4 shows when $G$ is a comparability graph, it is easy to construct a transitive orientation of $G$ from transitive orientations of $G_1$ and $G_2$. Therefore, when $G$ is a comparability graph that has a nontrivial module, one can find a transitive orientation of $G$ by recursively solving the problem on $G_1$ and $G_2$; thus, the problem essentially reduces to finding a transitive orientation of a comparability graph that has no nontrivial modules. In this case, the problem is solved using the fact [14] that such a graph admits a unique transitive orientation (i.e., the transitive orientation and its reversal are the only possible ones). The notion of modular decomposition of a graph, described next, is a systematic procedure to decompose a graph into modules and record the result as a unique tree structure.

28.4.2.2 Modular Decomposition

The graph is decomposed recursively into subsets of vertices each of which is a module of the graph. The procedure stops when every subset has a single vertex. The result is represented as a tree.

**Definition 28.13** A module which induces a disconnected subgraph in the graph is a parallel module. A module which induces a disconnected subgraph in the complement of the graph is a series module. A module which induces a connected subgraph in the graph as well as in the complement of the graph is a neighborhood module.

If the current set $Q$ of vertices induces a disconnected subgraph, $Q$ is decomposed into its components. A node labeled $P$ (for parallel) is introduced, each component of $Q$ is decomposed recursively, and the roots of the resulting subtrees are made children of the $P$ node. If the complement of the subgraph induced by current set $Q$ is disconnected, $Q$ is decomposed into the components of the complement. A node labeled $S$ (for series) is introduced, each component of the complement of $Q$ is decomposed recursively, and the roots of the resulting subtrees are made children of the $S$ node. Finally, if the subgraph induced by the current set $Q$ of vertices and its complement are connected, then $Q$ is decomposed into its maximal proper submodules (a proper submodule $M$ of $Q$ is maximal if there does not exist module $M'$ of $Q$ such that $M \subset M' \subset Q$); it is known [14] that in this case, each vertex of $Q$ belongs to a unique maximal proper submodule of $Q$. A node labeled $N$ (for neighborhood) is introduced, each maximal proper submodule of $Q$ is decomposed recursively, and the roots of the resulting subtrees are made children of the $N$ node. A graph and its modular decomposition tree are shown in Figure 28.4.

**Theorem 28.17** [32] The modular decomposition tree of a graph is unique and it can be constructed in $O(m + n)$ time.
28.4.2.3 From the Modular Decomposition Tree to Transitive Orientation

Definition 28.14 Let $M$ be the module corresponding to a node of the modular decomposition tree. The quotient graph of $M$ is the graph obtained as follows: take a representative vertex of the graph from the subtree rooted at each child of $M$ in the decomposition tree, and then construct the subgraph induced by the set of chosen vertices.

We note that the choice of the representative vertex is irrelevant. The reader is referred to Figure 28.5 where the quotient graph of the root node of the decomposition tree in Figure 28.4 is shown. Vertex $v_i$ corresponds to the subtree containing the representative vertex $i$ of the graph.

Let us now consider the problem of transversely orienting a comparability graph, given its modular decomposition tree $T$. We do a post order traversal of $T$. Suppose we are at node $D$ of $T$ and all the subtrees of $D$ have already been processed (and hence any edge of the graph with both endpoints in the same subtree of $D$ is already oriented), our goal is to orient any edge of the graph whose endpoints are in different subtrees of $D$. In order to accomplish this, we construct the quotient graph $H$ of $D$. We then transversely orient $H$. Suppose $x$, $y$ are vertices of the graph that are in different subtrees of $D$ such that $v_i$ corresponds to the subtree of $D$ containing $x$ while $v_j$ corresponds to the subtree containing $y$. We add $x \rightarrow y$ to the transitive orientation of the graph if and only if $v_i \rightarrow v_j$ is in the transitive orientation of $H$.

For example, consider the transitive orientation of the quotient graph shown in Figure 28.5. As it contains $v_1 \rightarrow v_3$, each of $4 \rightarrow 2$, $5 \rightarrow 2$, $4 \rightarrow 3$, and $5 \rightarrow 3$ will be added to the transitive orientation of the graph.

The remaining issues to be addressed are construction of the quotient graphs and finding a transitive orientation of each of the quotient graphs. It is easily seen that the sum of the sizes of all the quotient graphs is $O(m + n)$. However, this does not automatically imply that they can all be constructed efficiently. It is shown in [32] that all the required quotient graphs can be constructed in $O(m + n)$ time. Now, let us consider the problem of transversely orienting a quotient graph. The quotient graph of an $S$ node is a complete graph; in this case, we can take any permutation $R$ of the vertices and orient the edges so that $R$ is a topological sort of the resulting orientation. The quotient graph of a $P$ node has no edges.

Now, let $H$ be the quotient graph of an $N$ node. Clearly, $H$ itself does not have any nontrivial modules. Therefore, as noted earlier, $H$ admits a unique transitive orientation.

![Figure 28.5](image-url) Quotient graph of the module corresponding to the root of the tree in Figure 28.4 and its transitive orientation. $v_1$ represents $\{1\}$, $v_3$ represents $\{2, 3\}$, $v_4$ represents $\{4, 5\}$, and $v_{10}$ represents $\{6, 7, 8, 9, 10\}$.
The idea of vertex partitioning is employed in [32] to transitively orient $H$ in linear time and we explain this next. Suppose we are given a partition of $V(H)$ such that for blocks $X$ and $Y$ of the partition every edge of $H$ with an endpoint in $X$ and another in $Y$ is already oriented in a way consistent with some transitive orientation of $H$ (however, an edge with both endpoints inside a block may not yet be oriented). Now, suppose $u \in X$ is adjacent to some vertices in $Y$ and also is nonadjacent to some vertices in $Y$. Then, we can split $Y$ into $Y_1$ (neighbors of $u$) and $Y_2$ (nonneighbors of $u$) and replace the block $Y$ of the partition with $Y_1$ and $Y_2$. Further, for $v \in Y_1$ and $w \in Y_2$ such that $v$ and $w$ are adjacent, as $uw$ is a $P_3$ and the edge $uv$ is already oriented, orientation of the edge $vw$ is forced. In other words, we can now orient every edge of $H$ with an endpoint each in $Y_1$ and $Y_2$. As a result, we would have more blocks in the partition satisfying the property that any edge with endpoints in two different blocks of the partition is already oriented (and any edge with both endpoints in the same block may not yet be oriented). Observe that if a block $Y$ had more than one vertex, then there must be a vertex in a block different from $Y$ that splits $Y$; for otherwise, $Y$ will be a nontrivial module in $H$. Therefore, as $H$ contains no nontrivial modules, the process will terminate with each block containing exactly one vertex and all the edges in $H$ will be oriented. The only remaining issue is finding the initial partition. It is shown in [32] that a source vertex $s$ of a transitive orientation of $H$ can be found in linear time again, using a version of vertex partitioning. Once $s$ is found, we can start with $X = \{s\}$ and $Y = V(H) - X$ as the blocks of the initial partition, with any edge incident on $s$ oriented away from $s$.

**Theorem 28.18** [32] A transitive orientation of a comparability graph can be found in $O(m + n)$ time.

**Corollary 28.3** Comparability graphs can be recognized in $O(n^2)$ time.

### 28.4.2.4 How Quickly Can Comparability Graphs Be Recognized?

Next, we consider the feasibility of recognizing comparability graphs in better time than $O(n^2)$.

**Definition 28.15** A dag is a directed acyclic graph.

An h2dag $G = (X, Y, Z, E)$ is a dag (of height two) in which $\{X, Y, Z\}$ is a partition of the set of vertices of $G$, $E$ is the set of arcs of $G$, each of $X$, $Y$, $Z$ is a stable set, arcs between $X$ and $Y$ are oriented $X$ to $Y$, arcs between $Y$ and $Z$ are oriented $Y$ to $Z$, and arcs between $X$ and $Z$ are oriented $X$ to $Z$. Further, $X = \{x_i \mid 1 \leq i \leq |X|\}$, $Y = \{y_i \mid 1 \leq i \leq |Y|\}$, and $Z = \{z_i \mid 1 \leq i \leq |Z|\}$.

In a tripartite graph $G = (X, Y, Z, E)$, $\{X, Y, Z\}$ is a partition of the set of vertices of $G$, $E$ is set of edges of $G$, and each of $X$, $Y$, $Z$ is a stable set.

Consider the following problems:

**Problem—Comparability**

**Instance:** Graph $G$.

**Question:** Is $G$ a comparability graph?

**Problem—Transitivity**

**Instance:** dag $G$.

**Question:** Is $G$ transitively oriented?
Problem-Transitivity

Instance: $h2dag G$.

Question: Is $G$ transitively oriented?

Problem-Triangle

Instance: Graph $G$.

Question: Does $G$ contain a triangle?

Problem-tripartiteTriangle

Instance: Tripartite graph $G$.

Question: Does $G$ contain a triangle?

Lemma 28.5 [32] Problem-Comparability \(\leq\) Problem-Transitivity via an $O(m + n)$ time reduction.

Proof. Follows from Theorem 28.18.

Lemma 28.6 [33] Problem-Transitivity \(\leq\) Problem-Comparability via an $O(m + n)$ time reduction.

Proof. Let $G = (V, E)$ be the given dag with $|E| \geq 1$. Construct graph $H$ as follows: let $X = \{x_i \mid i \in V\}$, $Y = \{y_i \mid i \in V\}$, and $Z = \{z_i \mid i \in V\}$. Then, $V(H) = \{t\} \cup X \cup Y \cup Z \cup \{s\}$ and $E(H) = \{tx_i \mid x_i \in X\} \cup \{z_is \mid z_i \in Z\} \cup \{x_iz_j \mid i \rightarrow j \in E\} \cup \{y_iz_j \mid i \rightarrow j \in E\} \cup \{x_iz_j \mid i \rightarrow j \in E\}$.

In other words, $H$ has two special vertices $t$ and $s$ and a copy in each of $X$, $Y$, and $Z$ for every vertex $i \in V$. Corresponding to every arc $i \rightarrow j$ in $G$, $H$ has three edges. Finally, $t$ is adjacent to every vertex in $X$ and $s$ is adjacent to every vertex in $Z$. Next, we verify that $G$ is transitive if and only if $H$ is a comparability graph.

Suppose $G$ is transitively oriented. Construct an orientation of $H$ as follows: for every $x_i$, add the arc $x_i \rightarrow t$. For every $z_i$, add the arc $s \rightarrow z_i$. If $i \rightarrow j$ is an arc in $G$, then add the arcs $x_i \rightarrow y_j$, $y_i \rightarrow z_j$, and $x_i \rightarrow z_j$. If the resulting orientation had a violation of transitivity, then we must have $x_i \rightarrow y_j \rightarrow z_k$ (as only a vertex in $Y$ can have an incoming as well as an outgoing arc), but no $x_i \rightarrow z_k$. This would then imply that $G$ has $i \rightarrow j \rightarrow k$ but no $i \rightarrow k$, making it not transitive. Thus, the resulting orientation of $H$ is transitive and therefore, $H$ is a comparability graph.

Now, suppose $H$ is a comparability graph and consider a transitive orientation of $H$. As the reversal of a transitive orientation is also a transitive orientation, we can assume that for some $x_i$, we have the arc $x_i \rightarrow t$. This forces the arc $x_j \rightarrow t$, for every $x_j \in X$. This in turn forces every edge between $X$ and $Y$ to be oriented from $X$ to $Y$ and also forces every edge between $X$ and $Z$ to be oriented from $X$ to $Z$. As $|E| \geq 1$, there must be some edge $x_i z_j$ in $H$ and hence the arc $x_i \rightarrow z_j$ must be in the transitive orientation of $H$. This forces the arc $s \rightarrow z_j$, which in turn forces the arc $s \rightarrow z_i$, for every $z_i \in Z$. Finally, as there cannot be a directed path with two arcs from $s$ to a vertex in $Y$, every edge between $Y$ and $Z$ is oriented from $Y$ to $Z$. In order to verify that $G$ must be transitive, suppose $G$ had $i \rightarrow j \rightarrow k$. Then, $H$ has the $P_3 x_i y_j z_k$ and given the discussion above, the transitive orientation of $H$ has $x_i \rightarrow y_j \rightarrow z_k$, and hence has the arc $x_i \rightarrow z_k$ also. Therefore, $H$ has the edge $x_i z_k$, and given the construction of $H$, $G$ has the arc $i \rightarrow k$. 
Corollary 28.4 Problem-Comparability \( \equiv \) Problem-Transitivity via \( O(m + n) \) time reductions.

Lemma 28.7 \[33\] Problem-Transitivity \( \preceq \) Problem-h2Transitivity via an \( O(m + n) \) time reduction.

Proof. Let \( G = (V, E) \) be the given dag. Construct h2dag \( H = (X, Y, Z, F) \) as follows: \( X = \{x_i \mid i \in V\}, Y = \{y_i \mid i \in V\}, Z = \{z_i \mid i \in V\}, \) and \( F = \{x_i \rightarrow y_j \mid i 
rightarrow j \in E\} \cup \{y_i \rightarrow z_j \mid i \rightarrow j \in E\} \cup \{z_i \rightarrow y_j \mid i \rightarrow j \in E\} \). It is seen that \( G \) has violation \( i \rightarrow j \rightarrow k \) of transitivity if and only if \( H \) has violation \( x_i \rightarrow y_j \rightarrow z_k \) of transitivity. Note that we trivially have Problem-h2Transitivity \( \preceq \) Problem-Transitivity.

Lemma 28.8 \[34\] Problem-Triangle \( \preceq \) Problem-tripartite Triangle via an \( O(m + n) \) time reduction.

Proof. Given \( G = (V, E) \) construct the tripartite graph \( H = (X, Y, Z, F) \) as follows: \( X = \{x_i \mid i \in V\}, Y = \{y_i \mid i \in V\}, Z = \{z_i \mid i \in V\}, \) and \( F = \{x_i y_j, x_j y_i \mid i, j \in E\} \cup \{y_i z_j, y_j z_i \mid i, j \in E\} \cup \{x_i z_j, x_j z_i \mid i, j \in E\}. \) As \( H \) is a tripartite graph, any triangle of \( H \) must involve a vertex from each of \( X, Y, \) and \( Z. \) It is then seen that \( \{i, j, k\} \) form a triangle in \( G \) if and only if \( \{x_i, y_j, z_k\} \) form a triangle in \( H. \) Note that we trivially have Problem-tripartite Triangle \( \preceq \) Problem-Triangle.

Lemma 28.9 \[34\] Problem-h2Transitivity \( \preceq \) Problem-tripartite Triangle via an \( O(n^2) \) time reduction.

Proof. Let \( G = (X, Y, Z, E) \) be the given h2dag. Construct tripartite graph \( H = (X, Y, Z, F) \) where \( F = \{x_i y_j \mid x_i \rightarrow y_j \in E\} \cup \{y_i z_j \mid y_i \rightarrow z_j \in E\} \cup \{x_i z_j \mid x_i \rightarrow z_j \notin E\}. \) It is seen that \( x_i \rightarrow y_j \rightarrow z_k \) is a violation of transitivity in \( G \) if and only if \( \{x_i, y_j, z_k\} \) form a triangle in \( H. \)

Corollary 28.5 Problem-tripartite Triangle \( \equiv \) Problem-h2Transitivity via \( O(n^2) \) time reductions.

Thus, we have the following theorem.

Theorem 28.19 Problem-Comparability \( \equiv \) Problem-Transitivity \( \equiv \) Problem-Triangle via \( O(n^2) \) time reductions.

We note that the current best algorithm to test for a triangle in a graph with \( \Omega(n^2) \) edges runs in \( O(n^3) \) time.
28.4.3 Optimization

In this section, we consider the problems of finding a largest clique, a minimum coloring, a largest stable set, and a minimum clique cover of a comparability graph.

**Theorem 28.20** A largest clique and a minimum coloring of a comparability graph $G$ can be computed in $O(m + n)$ time.

**Proof.** Let $\overrightarrow{G}$ be a transitive orientation of $G$; from Theorem 28.18, $\overrightarrow{G}$ can be computed in $O(m + n)$ time. Observe that a directed path of $\overrightarrow{G}$ corresponds to a clique of $G$ and vice versa.

For a vertex $v$ of $\overrightarrow{G}$, let $\text{height}(v) = 0$ if there is no arc in $\overrightarrow{G}$ leaving $v$; otherwise, $\text{height}(v) = 1 + \max\{\text{height}(w) \mid v \rightarrow w \text{ is an arc in } \overrightarrow{G}\}$. Now, $\text{height}(v)$ can be computed in $O(m + n)$ time for all the vertices in $\overrightarrow{G}$ as follows: compute a topological sort $R$ of $\overrightarrow{G}$, then process the vertices of $\overrightarrow{G}$ by scanning $R$ once from right to left (from largest to smallest), and compute $\text{height}(v)$ when vertex $v$ is processed. During that computation, for every vertex $v$ that has an arc leaving it in $\overrightarrow{G}$, we also record $\text{next}(v) = \text{vertex } w \text{ such that } \text{height}(v) = 1 + \text{height}(w)$.

Then, a longest directed path in $\overrightarrow{G}$, which corresponds to a largest clique of $G$, can be found starting from a vertex $v$ of largest height, following to vertex $\text{next}(v)$, and repeating the process. Further, by assigning color $\ell$ to all the vertices with height $h$, a minimum coloring of $G$ can also be found. That the coloring found is optimal follows from the fact that the number of colors used equals the size of a largest clique of $G$.

Consider the following problems:

**Problem-bipartiteStable**

**Instance:** Bipartite graph $G$ and positive integer $k$.

**Question:** Is there a stable set of size at least $k$ in $G$?

**Problem-bipartiteMatching**

**Instance:** Bipartite graph $G$ and positive integer $k$.

**Question:** Is there a matching of size at least $k$ in $G$?

**Problem-comparabilityStable**

**Instance:** Comparability graph $G$ and positive integer $k$.

**Question:** Is there a stable set of size at least $k$ in $G$?

**Theorem 28.21** [5,35] In a bipartite graph, the size of a largest matching equals the size of a smallest vertex cover.

The proof of the following theorem is adopted from [36].

**Theorem 28.22** [37,38] Let $\overrightarrow{G}$ be a transitive orientation of the comparability graph $G = (V,E)$. Construct bipartite graph $B = (X,Y,F)$ where $X = \{x' \mid x \in V\}$, $Y = \{x'' \mid x \in V\}$, and $F = \{x'y'' \mid x \rightarrow y \text{ is an arc in } \overrightarrow{G}\}$. Suppose $M$ is a largest matching in $B$. Then, $\alpha(G) = \theta(G) = n - |M|$ where $n = |V|$.
Proof [36,38]. For \(x, y \in V\) with \(x'y'' \in M\), refer to \(y\) as successor of \(x\), and to \(x\) as predecessor of \(y\). As \(M\) is a matching, every \(x \in V\) has at most one predecessor and at most one successor.

Every \(u \in V\) defines a unique sequence \(K_u = u_p, \ldots, u_2, u_1, u, u_1, u_2, \ldots, u_s\) where \(u_{i+1}\) is successor of \(u_i\), \(u_1 = u\) has no predecessor, and \(u_s\) has no successor. It then follows from transitivity of \(\overrightarrow{G}\) that whenever \(i < j\), we have the arc \(u_i \rightarrow u_j\) in \(\overrightarrow{G}\). This in turn implies that no two elements of \(K_u\) are the same.

Clearly, every \(x'y'' \in M\) appears as \(u_i' u_i+1''\) for some \(u_i, u_{i+1}\) in a specific sequence \(K_u\). Let the total number of such sequences be \(k\) and the length of the \(i\)th sequence be \(r_i\). Then, \(\sum_{i=1}^{k} r_i = n\) and \(\sum_{i=1}^{k} (r_i - 1) = |M|\). It then follows that \(k = n - |M|\). As each \(K_u\) is a chain in \(\overrightarrow{G}\), and hence corresponds to a clique of \(G\), we have that \(\theta(G) \leq k\).

In order to show that \(\theta(G) \geq k\) also holds, we construct a stable set in \(G\) of size \(k\) based on \(M\). From Theorem 28.21, \(B\) has a vertex cover \(R\) of size \(|M|\). Let \(S = \{x \in V \mid x' \notin R\) and \(x'' \notin R\}\). Note that for \(x, y \in S\), the arc \(x \rightarrow y\) cannot be in \(\overrightarrow{G}\); otherwise, as \(x' \notin R\) and \(x'' \notin R\), \(x'y'' \in F\) is not covered by \(R\). Therefore, \(S\) is a stable set of \(G\) and hence \(\theta(G) \geq |S|\). However, as each \(x \in R\) prevents only one vertex of \(G\) from being a member of \(S\), \(|S| \geq n - |R| = n - |M|\), and therefore \(|S| \geq k\). Thus, we have \(\theta(G) \geq |S| \geq k\) also. Finally, as \(\theta(G) \geq \alpha(G)\) and \(\alpha(G) \geq |S|\) also hold, we have \(k \geq \theta(G) \geq \alpha(G) \geq |S| \geq k\), and we conclude that \(\theta(G) = \alpha(G) = k = n - |M|\).

**Theorem 28.23** Problem-bipartiteStable \(\equiv\) Problem-bipartiteMatching \(\equiv\) Problem-comparabilityStable via \(O(m + n)\) time reductions.

**Proof.** That Problem-bipartiteStable \(\equiv\) Problem-bipartiteMatching follows from Theorem 28.21. As every bipartite graph is a comparability graph, we have Problem-bipartiteStable \(\leq\) Problem-comparabilityStable. That Problem-comparabilityStable \(\leq\) Problem-bipartite-Matching follows from Theorem 28.22.

Given the current best time bounds of \(O(n^{1.5} \sqrt{m/\log n})\) [39] and \(O(n^{2.5}/\log n)\) [40] for computing a largest matching in a bipartite graph, we have the following:

**Corollary 28.6** A largest stable set and a smallest clique cover of a comparability graph can be computed in \(O(\min(n^{1.5} \sqrt{m/\log n}, n^{2.5}/\log n))\) time.

Now, we present a proof of Theorem 28.14.

**Proof of Theorem 28.3** [38]. Construct transitive orientation \(\overrightarrow{G}\) of the comparability graph \(G\) of \((P, \prec)\) by adding arc \(x \rightarrow y\) to \(\overrightarrow{G}\) if and only if \(x \prec y\). As a chain of \((P, \prec)\) corresponds to a clique of \(G\) and an anti-chain of \((P, \prec)\) corresponds to a stable set of \(G\), the proof follows from Theorem 28.22.

### 28.5 INTERVAL GRAPHS

**Definition 28.16** Graph \(G = (V, E)\) is an interval graph if every \(v \in V\) can be mapped to an interval \(I_v\) on the real line such that \(xy \in E\) if and only if \(I_x \cap I_y \neq \emptyset\). When \(G\) is an interval graph, the collection \(\{I_v \mid v \in V\}\) is an interval model for \(G\). For \(v \in V\), \(v_L\) and \(v_R\) denote the left and right endpoints, respectively, of \(I_v\).

It is known that in an interval model for an interval graph, the endpoints can be assumed to be distinct. Thus, the \(2n\) endpoints can be represented by the integers 1 through 2n. Further, for a cost of \(O(n)\) using bin-sort, one can assume the endpoints are given in increasing order.
28.5.1 Characterization

**Theorem 28.24** [41] For a graph \( G = (V, E) \) the following statements are equivalent:

i. \( G \) is an interval graph.

ii. \( G \) is chordal and \( \overline{G} \) is a comparability graph.

iii. There is an ordering \( \mathcal{R} \) of the maximal cliques of \( G \) such that for every \( v \in V \), the maximal cliques containing \( v \) are consecutive in \( \mathcal{R} \).

**Proof.**

(i) \( \Rightarrow \) (ii) Let \( \{I_v \mid v \in V\} \) be an interval model for \( G \). Suppose \( v_1v_2v_3 \cdots v_k \), \( k \geq 4 \) is a chordless cycle in \( G \). For \( 1 \leq i \leq k - 1 \), let \( p_i \) be a point in \( I_{v_i} \cap I_{v_{i+1}} \). Given that \( v_1v_2 \cdots v_{k-1} \) is a chordless path, we can assume \( p_1 < p_2 < \cdots < p_{k-1} \). Then, it is impossible for \( I_{v_i} \) to intersect \( I_{v_{i+1}} \). Therefore, \( G \) is chordal.

For \( x, y \in V \), \( xy \notin E \) if and only if either \( x_R < y_L \) holds or \( y_R < x_L \) holds. For \( xy \notin E \), orient \( x \rightarrow y \) in \( \overline{G} \) if \( x_R < y_L \). It is easily verified that the resulting orientation is acyclic and transitive. Therefore, \( \overline{G} \) is a comparability graph.

(ii) \( \Rightarrow \) (iii) Suppose \( A \) and \( B \) are distinct maximal cliques of \( G \). Then, there must exist \( x \in A \) and \( y \in B \) such that \( xy \notin E \); otherwise, \( A \cup B \) is also a clique of \( G \). Now, consider a transitive orientation of \( G \). For \( w, x \in A \) and \( y, z \in B \) such that \( xy \notin E \) and \( wz \notin E \), if we have \( x \rightarrow y \) in \( \overline{G} \), then we must have \( w \rightarrow z \) in \( \overline{G} \). Suppose not, and we have \( x \rightarrow y \) and \( z \rightarrow w \) in \( \overline{G} \). Clearly, \( w \neq x \) and \( y \neq z \) or else, there is a violation of transitivity in \( \overline{G} \). Further, as \( G \) is chordal, either \( wz \notin E \) or \( xy \notin E \); say, \( wz \notin E \). Then, there is no way to orient the edge \( wz \) in \( \overline{G} \) to avoid a violation of transitivity. Thus, the edges of \( \overline{G} \) that go across \( A \), \( B \) are all oriented either from \( A \) to \( B \), or from \( B \) to \( A \).

Now, for distinct maximal cliques \( A, B, \) and \( C \) of \( G \) and \( w \in A \), \( x, y \in B \), and \( z \in C \), suppose we have \( w \rightarrow x \) and \( y \rightarrow z \) in \( \overline{G} \). Then, we claim \( wz \notin E \) and \( w \rightarrow z \) in \( \overline{G} \). Suppose not. As \( \overline{G} \) is transitively oriented, we can assume \( x \neq y \). Further, \( xz \in E \) and \( y \in E \); otherwise, we have \( x \rightarrow z \) or \( w \rightarrow y \), and the transitivity of \( \overline{G} \) is violated. Now, \( wyxz \) is a chordless cycle in \( G \). So, we have \( wz \notin E \). Now, we must have \( w \rightarrow z \) in \( \overline{G} \) or else, \( z \rightarrow w \rightarrow x \) is a violation of transitivity in \( \overline{G} \).

Now, consider the ordering \( \mathcal{R} \) of the maximal cliques of \( G \) where \( A < B \) in \( \mathcal{R} \) if there exist \( x \in A \) and \( y \in B \) such that we have \( x \rightarrow y \) in \( \overline{G} \); from the above, such a total ordering exists. In order to verify that \( \mathcal{R} \) is the required ordering: for maximal cliques \( A, B, \) and \( C \) with \( A < B < C \) in \( \mathcal{R} \), suppose \( x \in A \), \( x \in C \), but \( x \notin B \). As \( B \) is a maximal clique and \( x \notin B \), there must exist \( y \in B \) such that \( xy \notin E \). As \( A < B \) in \( \mathcal{R} \), we must have \( x \rightarrow y \) in \( \overline{G} \). However, this contradicts \( B < C \) which dictates that we have \( y \rightarrow x \) in \( \overline{G} \).

(iii) \( \Rightarrow \) (i) Consider an ordering \( \mathcal{R} = K_1K_2 \cdots K_p \) of the maximal cliques of \( G \) as stated in the theorem. For \( v \in V \), let \( K_{v_l} \) be the left most maximal clique in \( \mathcal{R} \) that contains \( v \). Similarly, let \( K_{v_R} \) be the right most maximal clique in \( \mathcal{R} \) that contains \( v \). Set \( I_v = [v_L, v_R] \). It is easily verified that \( \{I_v \mid v \in V\} \) is an interval model for \( G \).

**Definition 28.17** A set \( \{x, y, z\} \) of pair-wise nonadjacent vertices of \( G \) is an asteroidal triple if there exists a path between any two of them that does not involve a neighbor of the third.

**Theorem 28.25** [42] \( G \) is an interval graph if and only if \( G \) is chordal and \( G \) does not contain an asteroidal triple.
28.5.2 Recognition

As chordal graphs and complements of comparability graphs can be recognized in polynomial time, a direct consequence of Theorem 28.24 is that interval graphs can be recognized in polynomial time; further, an interval model for an interval graph can also be constructed in polynomial time. The first $O(m + n)$ time algorithm to recognize interval graphs was given in [43] and we describe the ideas employed there next. Given input graph $G = (V, E)$, we first test whether $G$ is chordal (recall that every interval graph is chordal). If $G$ is chordal, then we use the algorithms in [25, 27] to generate all the maximal cliques of $G$; by Corollary 28.1 $G$ has at most $n$ maximal cliques whose sizes sum up to at most $m$. The remaining task is to determine whether an ordering of all the maximal cliques of $G$, as stipulated in Theorem 28.24, exists. In [43] the data structure PQ-tree was used to solve the following problem in $O(m + n)$ time: given a finite set $X$ with $|X| = n$ and a collection $S_1, \ldots, S_k$ of subsets of $X$ with $|S_1| + \cdots + |S_k| = m$, determine if there is an ordering of members of $X$ such that for each $S_i$ the members of $S_i$ occur consecutively in the ordering. In order to use this algorithm for the recognition of interval graphs, we just have to let $X = V$ and let the set of maximal cliques of the chordal graph $G$ to be the collection $S_i$ of subsets.

Subsequently, several linear-time algorithms have been designed to recognize interval graphs; some of these algorithms employ some variation of PQ-trees where as the rest avoid the use of such data structures. In [44], the algorithm from [43] is simplified with the use of modified PQ-trees. An algorithm that relies on modular decomposition of chordal graphs is given in [45]. We remarked in the section on chordal graphs that the algorithm LexBFS [25] can be used to generate a perfect elimination scheme of a chordal graph. An algorithm to recognize interval graphs using LexBFS is given in [46]. The final algorithm that we comment on relies on the following characterization of interval graphs which has been observed by multiple researchers.

**Theorem 28.26** [47–49] $G = (V, E)$ is an interval graph if and only if vertices of $G$ can be ordered $v_1v_2\cdots v_n$ such that for $v_i$, $v_j$, $v_k$ with $i < j < k$, if $v_iv_k \in E$ then $v_jv_k \in E$.

**Proof.** For an interval graph $G$ with an interval model where the endpoints are distinct, an ordering of vertices of $G$ according to the right endpoints of their intervals gives the desired ordering. Conversely, given such an ordering, one can derive an interval model for $G$ by taking the interval for $v_i$ to be $[v_i!, v_i]$ where $v_i!$ is the left most neighbor of $v_i$ in the ordering. □

In [50], a (very complicated) linear-time algorithm is given which employs six passes of LexBFS with various rules for breaking ties when choices have to be made. When the input is an interval graph, the algorithm is guaranteed to produce an ordering satisfying the conditions of Theorem 28.26. In order to test whether a given graph is an interval graph, we run the algorithm in [50] to get an ordering of vertices, and then verify if the ordering satisfies the conditions of Theorem 28.26.

28.5.3 Optimization

As interval graphs are chordal, given the adjacency lists for an interval graph, each of a largest clique, a largest stable set, an optimal vertex coloring, and a smallest vertex cover, as will be discussed in Section 28.7, can be computed in $O(m + n)$ time. However, when the interval model for an interval graph is given as input, it is possible to solve the problems more efficiently. Next, we illustrate this with algorithms for computing a largest clique and an optimal vertex coloring.

We will assume that the $2n$ endpoints in the interval model of the given interval graph $G = (V, E)$ are distinct and they are given in sorted order; recall that the endpoints can be sorted in $O(n)$ time. The algorithms scan the endpoints of the intervals from left to right.
(i.e., from the smallest to the largest). We open an interval when its left endpoint is scanned and we close it when its right endpoint is scanned. Further, an interval itself is open if its left endpoint has been scanned and its right endpoint is yet to be scanned.

First, we consider the problem of computing a largest clique. As a set of pair-wise intersecting intervals must share a common point, the problem reduces to considering each endpoint and computing how many intervals contain that endpoint. In order to do this efficiently, we scan the endpoints from left to right keeping track of the set $K$ of intervals open at any point. The set $K$ can be recorded in a boolean vector of size $n$. For a vertex $v$, when we scan $v_L$, $I_v$ is added to $K$ and it is deleted from $K$ when $v_R$ is scanned. This provides the set up to compute $\omega(G)$ in $O(n)$ time. One can then scan the endpoints again from left to right stopping when $|K| = \omega(G)$. The set $K$ at this point corresponds to a maximum clique of $G$. Thus, a maximum clique of $G$ can be found in $O(n)$ time.

Next, we consider the problem of optimal vertex coloring. We scan the endpoints from left to right and color a vertex $v$ when $v_L$ is scanned. Let $k$, initially set to zero, record the number of colors used at any point. The list $\text{freed-colors}$ contains colors assigned to intervals that have already closed, that is, those whose right endpoints have already been scanned; initially, $\text{freed-colors}$ is empty. For a vertex $v$, when $v_L$ is scanned, if $\text{freed-colors}$ is nonempty, then we remove any color $c$ from $\text{freed-colors}$ and assign it to $v$. If $\text{freed-colors}$ is empty, then we increase $k$ by 1 and assign the color $k$ to $v$ (i.e., $v$ is given a new color). When $v_R$ is scanned, the color assigned to $v$ is added to $\text{freed-colors}$.

It is easily seen that the coloring is proper and that the algorithm can be implemented to run in $O(n)$ time. In order to verify that the coloring is optimal, observe that every time a new color $k$ is assigned to a vertex $v$, as $\text{freed-colors}$ is empty, each of the colors 1 through $k - 1$ has been assigned to an interval that is currently open. Hence each of those $k - 1$ open intervals contains $v_L$ and $v$ belongs to a clique of size $k$ in the graph.

The reader is referred to [51] for a detailed exposition on interplay between representation of graphs and complexity of algorithms.

### 28.6 WEAKLY CHORDAL GRAPHS

A long hole is a chordless cycle with at least five vertices and a long anti-hole is the complement of a long hole.

**Definition 28.18** A graph is weakly chordal (also called weakly triangulated) if it does not contain any long holes or long anti-holes.

It is seen from the definition that the complement of a weakly chordal graph is also weakly chordal. Further, the class of weakly chordal graphs is a proper generalization of the class of chordal graphs.

#### 28.6.1 Characterization

**Definition 28.19** Let $G$ be a graph and $x, y$ be nonadjacent vertices of $G$. $\{x, y\}$ is a two-pair of $G$ if either every induced path between $x$ and $y$ has exactly two edges or $x$ and $y$ belong to different components of $G$. A co-pair of a graph is a two-pair of the complement of the graph.

Weakly chordal graphs were characterized [52] via the presence of two-pairs. As weakly chordal graphs are closed under complementation, the presence of a co-pair also characterizes weakly chordal graphs.

**Theorem 28.27** [52] $G$ is a weakly chordal graph if and only if for every induced subgraph $H$ of $G$, either $H$ induces a stable set or $H$ has a co-pair.
To prove Theorem 28.27, we will need to establish a preliminary result. We first start with a definition.

**Definition 28.20** A handle in a graph $G$ is a proper vertex-subset $H$ with size at least two such that $G[H]$ is connected, some component $J \neq H$ of $G - N(H)$ satisfies $N(J) = N(H)$, and each vertex of $N(H)$ is adjacent to at least an endpoint of each edge of $G[H]$. $J$ is called a co-handle of $H$.

Note that $N(H)$ is a minimal separator of $H$ and $J$.

**Theorem 28.28** [53,54] A graph has a handle if and only if the graph has a $P_3$, and a handle and its co-handle can be found in polynomial time.

When vertex-subset $H$ of $G$ with $|H| \geq 2$ induces a component of $G$, as $N(H) = \emptyset$, $H$ is trivially a handle of $G$; any other component of $G$ can be considered a co-handle of $H$. In this case, it is easily seen that when $G$ is a weakly chordal graph, any co-pair of $G[H]$ is a co-pair of $G$ also. Next, we prove that this holds for any handle $H$ of $G$ when $G$ is a weakly chordal graph.

**Lemma 28.11** [55] Suppose $H$ is a handle of a weakly chordal graph $G$ and $\{x, y\}$ is a co-pair of $G[H]$. Then, $\{x, y\}$ is a co-pair of $G$.

**Proof.** Let $J$ be a co-handle of $H$ in $G$, $I = N(H) = N(J)$, and $R = V(G) - H - I$. Suppose $\{x, y\}$ is a co-pair of $G[H]$ but not a co-pair of $G$.

Then, there exists an induced path $P = x \ldots y$ with at least four vertices in $\overline{G}$. As each vertex in $R$ is adjacent to both $x$ and $y$ in $\overline{G}$, $P$ does not involve any vertex in $R$; therefore, $P$ has at least a vertex from $I$. Now, $P$ cannot have a segment $uvw$ such that $u$ and $w$ are in $H$ but $v$ is in $I$, for otherwise, vertex $v$ of $I$ is not adjacent in $G$ to any endpoint of the edge $uv$ of $G[H]$, contradicting $H$ being a handle of $G$. Thus, at least two consecutive vertices of $P$ are in $I$ and $P$ involves at least an edge of $\overline{G}$ with both endpoints in $I$.

In $\overline{G}$, consider a segment $P' = x_2x_3x_4 \ldots x_r$ of $P$ with $r \geq 4$ such that $x_2$ and $x_r$ are in $H$ but $x_3$ through $x_{r-1}$ are in $I$. Observe that $x_3$ is not adjacent to $x_4$ in $G$. Since $I$ is a minimal separator for $H$ and $J$ in $G$, and $G$ has no long holes, in $G$ every two nonadjacent vertices of $I$ must have a common neighbor in $J$. In particular, $x_3$ and $x_4$ are adjacent in $G$ to some vertex $x_1$ of $J$. Thus in $\overline{G}$, $x_1$ is adjacent to $x_2$, $x_1$ is not adjacent to $x_3$, $x_4$ is not adjacent to $x_3$, and $x_3x_2x_3x_4$ is a $P_4$. Let $x_k$ be the first vertex in $P'$ after $x_4$ such that $x_1$ is adjacent to $x_k$ in $\overline{G}$; such an $x_k$ exists as $x_1$ is adjacent to $x_r$ in $\overline{G}$. Then, $\{x_1, x_2, \ldots, x_k\}$ induces a long hole in $\overline{G}$, contradicting $G$ being weakly chordal.

**Proof of Theorem 28.27.** For one direction, if $G$ is not weakly chordal, then it contains induced subgraph $H$ such that $H$ induces either a long hole or a long anti-hole. It is seen that neither does $H$ induce a stable set nor it contains a co-pair of $H$.

For the other direction, as an induced subgraph of a weakly chordal graph is also weakly chordal, it suffices to prove the theorem for the given weakly chordal graph $G$. Let $G$ be a weakly chordal graph with at least one edge. Let $G = H_0, H_1, \ldots, H_p$, $p \geq 0$, be a sequence of subsets of $V(G)$ such that $H_i$ is a handle of $G[H_{i-1}]$, for $1 \leq i \leq p$, and $G[H_p]$ has no handle. Then, by Theorem 28.28, $G[H_p]$ has no $P_3$, and is a complete multipartite graph. Therefore, every edge of $G[H_p]$ induces a co-pair of $G[H_p]$. Then, by Lemma 28.11, every edge of $G[H_p]$ induces a co-pair of $G[H_{p-1}]$, since $H_p$ is a handle of $G[H_{p-1}]$. Continuing this argument, every edge of $G[H_p]$ induces a co-pair of $G$. The current best recognition and optimization algorithms for weakly chordal graphs exploit the presence of two-pairs and co-pairs.
28.6.2 Recognition

An algorithm to test for the presence of a long hole in a graph is to check whether a $P_3$ of a graph extends into a long hole. As all the $P_3$'s of a graph can be generated in $O(nm)$ time, this can be implemented to run in $O(nm^2)$ time. By running this algorithm on the graph and then on the complement, weakly chordal graphs can be recognized in $O(n^3)$ time. Later, we discuss more efficient algorithms for the same problem.

More generally, whether a $P_k$, $k \geq 2$, of a graph extends into a hole of size at least $k+3$ can be tested in $O(n^2 \alpha)$ time [56], where $O(n^2 \alpha)$ refers to the current best complexity of multiplying two $n \times n$ Boolean matrices, by testing whether an auxiliary directed graph in transitive. The algorithm is as follows: given the $P_k$ $T = v_1 \cdots v_k$ of $G$, first we discard from $G$ all the neighbors of $v_2$ through $v_{k-1}$ that are not on $T$. Now, let $A = N(v_1) - N(v_k) - V(T)$, $B = N(v_k) - N(v_1) - V(T)$, and $D_1, \cdots, D_r$ be the components of $G - (A \cup B \cup V(T))$. Let $M$ be the set formed by adding a vertex $m_i$ corresponding to each $D_i$. Now, construct the directed graph $H$ on the vertex-set $A \cup M \cup B$. For $x \in A$, add the directed edge $x \rightarrow m_i$ provided $x$ is adjacent in $G$ to some vertex in $D_i$. Similarly, for $x \in B$, add the directed edge $m_i \rightarrow x$ provided $x$ is adjacent in $G$ to some vertex in $D_i$. Finally, for $x \in A$ and $y \in B$, add the directed edge $x \rightarrow y$ provided $x$ and $y$ are adjacent in $G$. It can be seen that $G$ has a hole of size at least $k+3$ through $T$ if and only if $H$ is not transitive. As whether a directed acyclic graph is transitive can be tested in $O(n^2 \alpha)$ time (cf. Theorem 28.16) we get the desired result. Thus, as the number of $P_k$'s in a graph is $O(n^k)$, we can check whether a graph has a hole of size at least $t$, $t \geq 5$, in time $O(n^{t-3+\alpha})$.

Using the above mentioned algorithm on the graph and then on the complement of the graph, weakly chordal graphs can be recognized in $O(n^{2+\alpha})$ time which is currently $O(n^{4.376})$ [18]. For the specific case of finding long holes in a graph, an $O(m^2)$ time algorithm is known [57]. By using this on the graph and then on the complement, weakly chordal graphs can be recognized in $O(n^4)$ time. The current best algorithms to recognize weakly chordal graphs run in $O(m^2)$ time [55,58]. However, one of them requires $O(m^2)$ space [58] while the other [55] uses linear amount of space.

Lemma 28.12 [59] Suppose $\{x, y\}$ is a co-pair of graph $G$. Let $H$ be the graph obtained from $G$ by deleting the edge $xy$ but not its endpoints. Then, $G$ is weakly chordal if and only if $H$ is weakly chordal.

Algorithm 28.6 wc-recognition

```
input: graph $G$
output: yes when $G$ is weakly chordal and no otherwise

found ← true;
while found and $G$ has at least one edge do
  if $G$ has co-pair $\{x, y\}$ then
    Delete edge $xy$ from $G$
  else
    found ← false
  end if
end while
if $G$ has no edges then
  output yes
else
  output no
end if
```
Theorem 28.29 [55] Algorithm we-recognition can be implemented to run in \( O(m^2) \) time using \( O(n + m) \) space.

28.6.3 Optimization

Definition 28.21 For a graph \( G \) and a pair \( \{x, y\} \) of nonadjacent vertices in \( G \), the graph \( G/xy \) is obtained from \( G \) by contracting the pair \( \{x, y\} \) as follows: delete vertices \( x \) and \( y \) and introduce vertex \((xy)\) and edges \((xy)u\) for all \( u \) in \( N_G(x) \cup N_G(y) \).

Definition 28.22 Two nonadjacent vertices \( x, y \) in a graph \( G \) form an even-pair if every induced path between them has an even number of edges.

Our interest in even-pairs is motivated by the following two observations.

Lemma 28.13 [60] Let \( G \) be any graph with an even-pair \( \{x, y\} \). Then

i. \( \omega(G/xy) = \omega(G) \);

ii. \( \chi(G/xy) = \chi(G) \).

Proof. We will establish (i) first. Let \( K \) be clique in \( G/xy \). For simplicity, write \( z = (xy) \). If \( z \notin K \), then \( K \) is also a clique in \( G \). Suppose \( z \in K \). Then, either \( x \) or \( y \) must be adjacent in \( G \) to every vertex in \( K - \{z\} \). Otherwise, there exist \( u, v \in K \) such that \( xu \in E(G), xv \notin E(G), yu \notin E(G) \), and \( yv \in E(G) \) so that \( xyzu \) is a \( P_4 \) in \( G \); this contradicts \( \{x, y\} \) being an even-pair of \( G \). Thus, \( G \) also has a clique of size \( |K| \) and \( \omega(G/xy) \leq \omega(G) \). Now suppose \( K \) is a clique in \( G \). Clearly, at most one of \( x \in K, y \in K \) holds. Further, if \( x \in K \) \((y \in K) \), then \( K - \{x\} \cup \{z\} \) \((K - \{y\} \cup \{z\}) \) is a clique in \( G/xy \). Therefore, \( \omega(G) \leq \omega(G/xy) \) also holds and \( \omega(G) = \omega(G/xy) \).

To prove (ii), consider a coloring of \( G/xy \). It gives a coloring of \( G \) by assigning to \( x, y \) the color of \((xy)\). So, we have \( \chi(G/xy) \geq \chi(G) \). Now, we will prove \( \chi(G/xy) \leq \chi(G) \). Consider a coloring of \( G \). If \( x, y \) have the same color, then this color can be assigned to \((xy)\), and we are done. So, assume \( x \) has color 1 and \( y \) has color 2. Let \( B \) be the bipartite graph induced by vertices of colors 1 and 2. \( x \) and \( y \) must belong to different components of \( B \), for otherwise there is an induced odd path in \( B \) between the two vertices, a contradiction to the assumption that \( \{x, y\} \) is an even-pair. Interchange colors 1 and 2 in the component of \( B \) containing \( x \). In the new coloring, \( x \) and \( y \) have the same color, implying as above, that \( \chi(G/xy) \leq \chi(G) \).

The proof of Lemma 28.13 gives a simple algorithm that given a largest clique of \( G/xy \) produces a largest clique of \( G \), and given a coloring of \( G/xy \) with \( k \) colors, produces a coloring of \( G \) with \( k \) colors. If, on subsequent graphs, we can always find an even-pair to contract until we obtain a clique, we could produce a largest clique and an optimal coloring of the original graph. The following lemma shows this is indeed the case for weakly chordal graphs.

Lemma 28.14 [52] Suppose \( G \) is a weakly chordal graph and \( \{x, y\} \) is a two-pair of \( G \). Then, \( G/xy \) is weakly chordal. Further, \( \omega(G) = \omega(G/xy) \) and \( \chi(G) = \chi(G/xy) \).

Proof. We show that if \( G/xy \) is not weakly chordal, then \( G \) is not weakly chordal. Clearly, \( G/xy \) cannot have a long hole or long anti-hole that does not involve \( z = (xy) \). Suppose \( v_1 \cdots v_k \), for \( k \geq 5 \), is a long hole in \( G/xy \). Then, as \( G \) is weakly chordal and given the construction of \( G/xy \), neither \( x \) nor \( y \) is adjacent in \( G \) to each of \( v_2, v_k \). Also, each of \( v_2, v_k \)
is adjacent in $G$ to at least one of $x$, $y$. Without loss of generality, assume that $vx_2 \in E(G)$, $xv_k \notin E(G)$, $yv_k \in E(G)$, and $yv_2 \notin E(G)$. Then, $vx_2 \cdots v_k y$ is chordless path in $G$ with at least five edges, contradicting $\{x, y\}$ being a two-pair of $G$.

Suppose $v_2 \cdots v_k$, is a long anti-hole in $G/xy$ where the ordering of the vertices corresponds to the cyclic ordering of the vertices along the long hole in the complement. As $C_5$ is isomorphic to $C_5^r$, we can assume $k \geq 6$. One of $x$, $y$ must be adjacent in $G$ to each of $v_3$, $v_4$. Otherwise, given the construction of $G/xy$, we can assume $xv_3 \in E(G)$, $xv_4 \notin E(G)$, $yv_4 \notin E(G)$, and $yv_3 \notin E(G)$. Then, $xv_2v_kv_4y$ a chordless path in $G$ with four edges, contradicting $\{x, y\}$ being a two-pair of $G$. Assume $x$ is adjacent in $G$ to each of $v_3$, $v_4$ and let $r$ be the smallest index such that $r \geq 5$ and $xv_r \notin E(G)$; such an $r$ exists as $xv_k \notin E(G)$. Then, $xv_2v_3v_4 \cdots v_r$ is a long anti-hole in $G$, a contradiction. Since two-pairs are even-pairs, the rest of the lemma follows from Lemma 28.13.

Algorithm 28.7 wc-optimization

| input: weakly chordal graph $G$ |
| output: $\chi(G)$ and $\omega(G)$ |

while $G$ is not a complete graph do
  find two-pair $\{x, y\}$ of $G$;
  replace $G$ by $G/xy$
end while

$\chi(G) = |V(G)|$;
$\omega(G) = |V(G)|$;

output $\chi(G)$ and $\omega(G)$

Theorem 28.30 [55] Algorithm wc-optimization can be implemented to run in $O(mn)$ time using $O(m + n)$ space.

For a weakly chordal graph $G$, $\alpha(G)$ and $\theta(G)$ can be computed by running the algorithm wc-optimization on $\overline{G}$.

28.6.4 Remarks

An $O(m^2)$ time algorithm to find a long hole in a given graph is given in [57]. An $O(m^2)$ time algorithm to recognize weakly chordal graphs using $O(m^2)$ space is given in [58]; unlike the algorithm described here, the one in [58] does not use the idea of a two-pair at all. The weighted versions of the clique, coloring, stable set, and clique cover problems can be solved on weakly chordal graphs in $O(n^4)$ time [52,59]. A consequence of algorithm wc-recognition is that graph $G$ is a weakly chordal if and only if an empty graph can be derived from $G$ by repeatedly removing a co-pair. As an interesting contrast, it is proved in [61] that graph $G$ is chordal if and only if $G$ can be derived from an empty graph by repeatedly adding an edge between vertices that form a two-pair. Efficient algorithms for finding a two-pair in a graph are given in [62] and [63]. The fact that weakly chordal graphs are perfect was first established in [64].

28.7 PERFECTLY ORDERABLE GRAPHS

A natural way to color a graph is to impose an order $<$ on its vertices and then scan the vertices in this order, assigning to each vertex $v_i$ the smallest positive integer not assigned
to a neighbor $v_j$ of $v_i$ with $v_j < v_i$. This method, referred to as the greedy algorithm, does not necessarily produce an optimal coloring of the graph (i.e., one using the smallest possible number of colors). However, on a perfectly ordered graph, the algorithm does produce an optimal coloring.

**Definition 28.23** Given an ordered graph $(G, <)$, the ordering $<$ is called perfect if for each induced ordered subgraph $(H, <)$ the greedy algorithm produces an optimal coloring of $H$. The graphs admitting a perfect ordering are called perfectly orderable. An obstruction in an ordered graph is a chordless path with vertices $a$, $b$, $c$, $d$, edges $ab$, $bc$, $cd$ with $a < b$ and $d < c$.

Several well known classes of graphs (in particular, chordal and comparability graphs) are perfectly orderable. It is easy to see that a perfectly ordered graph cannot contain an obstruction. It was shown [65] that this condition is also sufficient.

**Theorem 28.31** [65] A graph is perfectly orderable if and only if it admits an obstruction-free ordering.

We will need the following lemma.

**Lemma 28.15** Let $G$ be a graph and let $C$ be a clique of $G$ such that each $w \in C$ has a neighbor $p(w) \notin C$ such that the set $S$ consisting of the vertices $p(w)$ form a stable set of $G$. If there is an obstruction-free order $<$ such that $p(w) < w$ for all $w \in C$, then some $p(w)$ is $C$-complete.

**Proof.** By induction on the number of vertices in $C$. The induction hypothesis implies that, for each $w \in C$, there is a vertex $f(w) \in C$ such that the vertex $p(f(w))$ is adjacent to all of $C$, except possibly $w$. In fact, we may assume $p(f(w))$ is not adjacent to $w$, for otherwise we are done. Thus, the mapping $f$ is one-to-one and therefore onto, that is $f$ is a bijection. Let $v$ be the smallest vertex in $C$ in the order $<$. There are vertices $a, b$ such that $v = f(b)$ and $b = f(a)$. Now, $p(v), a, b, p(b)$ form an obstruction, a contradiction.

**Proof of Theorem 28.31.** The ‘only if’ part is trivial. We will prove the ‘if’ part by induction on the number of vertices. Let $G$ be a graph with an obstruction-free order $<$. By the induction hypothesis, we only need to prove the greedy algorithm delivers an optimal coloring on $G$. Let $k$ be the number of colors used on $G$. We will prove $G$ contains a clique on $k$ vertices. This obviously shows the coloring produced by the greedy algorithm is optimal. Let $i$ be the smallest integer such that there is a clique $C$ on vertices $v_i, \ldots, v_k$ such that each $v_j$ has color $j$, for $j = i + 1, \ldots, k$. We may assume $i > 0$, for otherwise we are done. Properties of the greedy algorithm imply that each $v_j$ has a neighbor $p(v_j)$ with color $i$ with $p(v_j) < v_j$, for each $v_j \in C$. But Lemma 28.15 implies some $p(v_j)$ is $C$-complete, a contradiction to our choice of $i$.

The proof of Theorem 28.31 shows that perfectly orderable graphs are perfect. In studying perfectly orderable graphs, the following two problems arise naturally: to decide on the complexity of recognizing perfectly orderable graphs and to find a subgraph characterization of perfectly orderable graphs (by subgraph characterization, we mean characterization by minimal forbidden induced subgraphs). The subgraph characterization problem is open but appears to be very difficult. It was proved in [66] that the problem of recognizing perfectly orderable graph is NP-complete. However, many classes of perfectly orderable graphs, together with their polynomial recognition algorithms, have been found. We will discuss some of these classes in this chapter. For a survey on perfectly orderable graphs, see [67].
28.7.1 Characterization

As mentioned before, there is no known characterization by forbidden induced subgraphs of perfectly orderable graphs. We will discuss several subclasses of perfectly orderable graphs that have been much studied.

Definition 28.24 For a $P_4$ with vertices $a, b, c, d$, edges $ab, bc, cd$, the vertices $a, d$ are endpoints, $c, d$ are midpoints of the $P_4$. A vertex is soft if it is not a midpoint or an endpoint of a $P_4$. A graph $G$ is brittle if each of its induced subgraphs contains a soft vertex.

Observation 28.1 Brittle graphs are perfectly orderable.

**Proof.** By induction on the number of vertices. Let $G$ be a brittle graph with a soft vertex $v$. Let $v_1 < v_2 < ... < v_n$ be a perfect order of $G - v$. If $v$ is not the endpoint of a $P_4$, then $v < v_1 < v_2 < ... < v_n$ is a perfect order of $G$. If $v$ is not a midpoint of a $P_4$, then $v_1 < v_2 < ... < v_n < v$ is a perfect order of $G$.

Corollary 28.7 Chordal graphs, their complements, and comparability graphs are perfectly orderable.

**Proof.** Observe that a simplicial vertex is soft and that a soft vertex of a graph remains soft in the complement. Thus, chordal graphs are brittle; by Observation 28.1, they and their complements are perfectly orderable. Since a transitive orientation of a graph contains no obstruction, comparability graphs are perfectly orderable.

28.7.2 Recognition

It is proved in [66] that the problem of recognizing perfectly orderable graphs is NP-complete. We have seen that chordal graphs and their complements are perfectly orderable. Since weakly chordal graphs are a generalization of these two classes, it is of interest to investigate the complexities of recognizing weakly chordal perfectly orderable graphs. In [68], it is shown that this problem is NP-complete by modifying the argument of [66]. Since [68] is an unpublished technical report, we will reproduce the proof here.

**Theorem 28.32** It is NP-complete to determine if a weakly chordal graph is perfectly orderable.

**Proof.** We will reduce the 3SAT problem to our problem. Given a 3SAT formula $E$ with clauses $C_0, C_1, ..., C_{m-1}$ and variables $v_0, v_1, ..., v_{n-1}$ where each clause $C_i$ contains literals $c_{i0}, c_{i1}, c_{i2}$, we construct a weakly chordal graph $G(E)$ such that $E$ is satisfiable if and only if $G(E)$ is perfectly orderable.

For each clause $C_j = (c_{j0}, c_{j1}, c_{j2})$, we define the clause graph $G(C_j)$ as in shown in Figure 28.6. For each variable $v_i$, we define the variable graph $G(v_i)$ as shown in Figure 28.7. In the graph $G(v_i)$, the chordless path between $A_i$ and $B_i$ has $2m$ vertices $v(i,j,1)$ for $j = 0, 1, 2, ..., 2m - 1$.

Next, we obtain the graph $G'(v_i)$ (see Figure 28.8) from $G(v_i)$ by

- If $C_j$ contains $v_i$, adding vertices $v(i, 2j, 2), v(i, 2j, 3)$ and edges $v(i, 2j, 1)v(i, 2j, 2), v(i, 2j, 1)v(i, 2j, 3)$.
- If $C_j$ contains $\overline{v_i}$, adding vertices $v(i, 2j + 1, 2), v(i, 2j + 1, 3)$ and edges $v(i, 2j + 1, 1)v(i, 2j + 1, 2), v(i, 2j + 1, 2)v(i, 2j + 1, 3)$. 

Perfect Graphs □ 735
The graph $G(E)$ is obtained by

i. Taking $m$ disjoint $G(C_j), 0 \leq j \leq m - 1$;

ii. Taking $n$ disjoint $G'(v_i), 0 \leq i \leq n - 1$;

iii. For $k = 1, 2, 3$,

identifying $v(i, 2j, k)$ with $c(j, l, k)$ if $c_{jl} = v_i$;

identifying $v(i, 2j + 1, k)$ with $c(j, l, k)$ if $c_{jl} = \overline{v_i}$;

for each $c(j, l, 0), 0 \leq j \leq m - 1, l = 0, 1, 2$, adding the edge $xc(j, l, 0)$ for all vertices $x$ not in $G(C_j)$.

A vertex is of type $k$ if it is of the form $c(j, l, k)$ for some $j$ and some $l$. We denote by $V_k$ the set of vertices of type $k, 0 \leq k \leq 3$. Our construction is similar to [66], except that $G(v_i)$ is a chordless cycle in [66]. Figure 28.9 shows the interaction between a clause graph and a variable graph; for clarity we do not show all edges coming out of the vertices of type 0.

**Remark 28.1** A vertex $c(j, l, 0)$ (of type 0) is nonadjacent to exactly four vertices of $G(E)$: they are $c(j, l, k), 1 \leq k \leq 3$ and $c(j, l + 1 \mod 3, 2)$. 

![Figure 28.6 Clause graph $G(C_j)$.

![Figure 28.7 Graph $G(v_i)$.

![Figure 28.8 Graph $G'(v_i)$.](image)
It is a routine but tedious matter to prove that $G(E)$ is weakly chordal. For detail, see [68].

For the rest of the proof, we will show that $G(E)$ is perfectly orderable if and only if $E$ is satisfiable. It will be more convenient to work with orientations instead of orders. For an ordered graph, we may construct an oriented graph on the same vertex set as follows: If $ab$ is an edge and $a < b$, then we add the arc $a \rightarrow b$. Thus, an obstruction is a $P_4$ with vertices $a, b, c, d$ and arcs $a \rightarrow b, b \rightarrow c, d \rightarrow c$ (see Figure 28.10). An orientation $\overrightarrow{G}$ of a graph $G$ is perfect if it is acyclic and does not contain an induced obstruction. It is a routine matter to verify the following observation.

**Observation 28.2** The graph $G(v_j)$ admits a perfect orientation, but any perfect orientation of $G(v_j)$ is alternating on the path from $A_j$ to $B_j$.

From now on, the argument of [66] carries through, for the sake of completeness we will complete the proof.

**Claim 28.1** If $G(E)$ admits a perfect orientation, then $E$ is satisfiable.

**Proof.** For each $i, 0 \leq i \leq n-1$, if the vertex $v(i, 0, 1)$ is a source in $G(v_i)$, then the variable $v_i$ is assigned value true; otherwise, it is assigned value false. Note that, by Observation 28.2, $v(i, 0, 1)$ being a source (resp., sink) in $G(v_i)$ implies all $v(i, 2j, 1)$ are sources (resp., sink) in $G(v_i)$.

Consider the graph $G(C_j)$ with $C_j = (c_{j0}, c_{j1}, c_{j2})$. If all three vertices $c(j, l, 1), 0 \leq l \leq 2$, are sinks in the three corresponding graphs $G(v_i)$ where $c_j = v_i$, or $c_j = \overline{v}_i$, then we have $c(j, l, 2) \rightarrow c(j, l, 3)$, and thus $c(j, l, 0) \rightarrow c(j, l+1 \mod 3, 0)$ for $0 \leq l \leq 2$; but then $\overrightarrow{G}$ is not acyclic, a contradiction. Thus, some $c(j, l, 1)$ is a source in $G(v_i)$ with $c_j = v_i$ or $c_j = \overline{v}_i$. If $c_j = v_i$, then $c(j, l, 1) = v(i, 2j, 1)$ implying $v(i, 0, 1)$ is a source in $G(v_i)$, and thus $v_i$ is true. Similarly, if $c_j = \overline{v}_i$, then $v(i, 0, 1)$ is a sink, and thus $v_i$ is false. In both cases, $C_j$ is satisfied.

**Claim 28.2** If $E$ is satisfiable, then $G(E)$ admits a perfect orientation.

**Proof.** Suppose there is a truth assignment of the variables $v_0, v_1, \ldots, v_{n-1}$ that satisfies $E$. For each variable graph $G(v_i)$, we assign a perfect orientation such that $v(0, 0, 1)$ is a source if and only if $v_i$ is true. Such orientation exists by Observation 28.2.

Consider a clause graph $G(C_j)$ with $C_j = (c_{j0}, c_{j1}, c_{j2})$. Suppose $c_{j0}$ is the $i$th variable, that is $c_{j0} = v_i$ or $\overline{v}_i$. Then $c(j, , 1) = v(i, 2j, 1)$ or $v(i, 2j + 1, 1)$. If $c_j$ is a source in $G(v_i)$, then direct $c(j, l, 3) \rightarrow c(j, l, 2)$: otherwise, direct $c(j, l, 2) \rightarrow c(j, l, 3)$, and $c(j, l-1 \mod 3, 0) \rightarrow c(j, l, 0)$. Since $C_j$ contains a true literal, some $c(j, l, 0)$ is a source,
and it follows that $V_0$ contains no directed cycle. Extend the partial orientation of $V_0$ into an acyclic orientation.

Now, for each edge $ab$, we direct $a \rightarrow b$ if $a \in V_0, b \notin V_0$; or if $a \in V_1, b \in V_2$. Every edge of $G$ has been directed. Call the resulting directed graph $\overrightarrow{G}$. It is easy to see that $\overrightarrow{G}$ is acyclic.

Suppose $\overrightarrow{G}$ contains an obstruction $P$ with vertices $a, b, c, d$ and arcs $a \rightarrow b, b \rightarrow c, c \rightarrow d$. Because $V_0$ is a clique, $P$ contains at most two vertices of type 0.

If $P$ contains no vertex of type 0, then $P$ must lie entirely in some $G'(v_i)$ because $V_0$ is a cutset of $G(E)$. But, clearly the orientation of every $G'(v_i)$ is perfect, a contradiction. Suppose $P$ contains one vertex of type 0. The arcs $a \rightarrow b, d \rightarrow c$ imply $b, c \notin V_0$ by our construction. So, we may assume that $a \in V_0$ (for the rest of the proof, we will not argue on the direction of the arc $b \rightarrow c$). This means $a = c(j, l, 0)$ for some $j$ and $l$. Since $cd$ is an edge, we have $\{c, d\} \subseteq \{c(j, l, k) | 1 \leq k \leq 3\}$. Therefore, $b \in G(v_i)$ for some $i$ such that $v_i$ or $\overrightarrow{v}_i$ is a literal of the clause $C_j$. Thus, $b$ is the vertex next to $c(j, l, 1) = v(i, r, 1)$ ($r = 2j$, or $r = 2j + 1$) on the path from $A_i$ to $B_i$ of $G(v_i)$. It follows that $c = c(j, l, 1), d = c(j, l, 2)$. But our construction implies $c(j, l, 1) \rightarrow c(j, l, 2)$, a contradiction.

Now, we may assume that $P$ contains two vertices of type 0. Since $V_0$ is a clique, one of the two middle vertices of $P$ must be of type 0. We may assume $b \in V_0$. Since $a \rightarrow b$, $a$ must be in $V_0$. From Remark 28.1, $P$ is the $P_4$ (i) $c(j, l, 0) c(j, l - 1 \text{ mod } 3, 0) c(j, l, 1) c(j, l, 2)$, or (ii) $c(j, l, 0) c(j, l - 1 \text{ mod } 3, 0) c(j, l, 3) c(j, l, 2)$. In case (i), our construction implies $c(j, l, 1) \rightarrow c(j, l, 2)$, a contradiction. In case (ii), the arc $c(j, l, 2) \rightarrow c(j, l, 3)$ implies $c(j, l, 1)$ is a sink in $G'(v_i)$ (for some appropriate $i$), and our construction implies $c(j, l - 1 \text{ mod } 3, 0) \rightarrow c(j, l, 0)$, a contradiction.

28.7.3 Optimization

In this section, we consider the problems of finding a largest clique, a minimum coloring, a largest stable set, and a minimum clique-cover of perfectly ordered graphs. We note that these four problems (even in their weighted versions) for perfect graphs have been solved in [7]. This algorithm does not exploit the combinatorial structure of a perfect graph, instead it uses deep properties of the ellipsoid method. Thus, it is of interest to optimize the graphs discussed in this chapter by using combinatorial structures.

Theorem 28.33 [69] Given a graph $G$ and a perfect order on $G$, one can find in $O(n + m)$ time a minimum coloring and a largest clique of $G$.

Proof. Let the vertices of $G$ be $v_1, \ldots, v_n$ and the perfect order be $v_1 < \ldots < v_n$. We will show that the greedy coloring algorithm can be implemented in linear time on $G$. Vertices are colored in the order given by $\prec$. Suppose we are about to process vertex $v_j$. We find the smallest integer $t$ such that no neighbor $x$ of $v_j$ has color $t$, and assign color $t$ to $v_j$. The index $t$ can be computed by traversing the adjacency list of $v_j$ and computing the number $a_i$ of neighbors of $v_j$ with color $i$; $t$ is the smallest index such that $a_t = 0$ (we may assume all the $a_i$ are initially set to 0). At most $d(v_j)$ number $a_i$ are modified in computing $t$. After $v_j$ is colored, we reset these $a_i$ to 0. So, the cost of coloring $v_j$ is $O(d(v_j))$. Thus, we can color $G$ in time $O(n + m)$.

From the proof of Theorem 28.31, we can extract a largest clique of $G$ in linear time. Let $k$ be the number of colors used by the greedy algorithm. We will show how to find a clique $C$ with $k$ vertices. Start with a vertex $x$ of color $k$, put $x$ in $C$. We go backward in $\prec$ to enlarge $C$. Suppose $C$ contains vertices $w_1, w_{i+1}, \ldots, w_k$ with $i > 1$ and $w_j$ having color $j$, $j = i, \ldots, k$. Let $S_{i-1}$ be the set of vertices of color $i - 1$. The proof of Theorem 28.31 implies there is a vertex $s \in S_{i-1}$ that is $C$-complete and so can be added to $C$. Such vertex can be found by scanning the adjacency list of every vertex $x$ in $S_{i-1}$ and computing the number of
The adjacency list of each vertex of $G$ is scanned at most once, so the algorithm runs in linear time.

**Theorem 28.34** [69] Given a graph $G$ and a perfect order on its complement $\overline{G}$, one can find in $O(n + m)$ time a largest stable set and a minimum clique cover of $G$.

**Proof.** Let the vertices of $G$ be $v_1, \ldots, v_n$ and the perfect order on the complement of $G$ be $v_1 < \ldots < v_n$. To stay within the linear-time bound, we will obviously not construct $\overline{G}$. We process the vertices in this order and produce a coloring of $\overline{G}$. Let the variable $b_i$ count the number of vertices of color $i$. Suppose we are processing vertex $v_j$. Then $v_j$ can be colored $i$ if in $\overline{G}$, $v_j$ is not adjacent to any vertex of color $i$, that is, in $G$, $v_j$ is adjacent to $b_i$ vertices of color $i$. This condition can be tested by scanning the adjacency list of $v_j$. If such color $i$ exist, then we would choose the smallest such $i$ for $v_j$; otherwise, we color $v_j$ with a new color. The cost of coloring $v_j$ is $O(d(v_j))$, so we can color $G$ in $O(n + m)$ time. This coloring is a partition of $G$ with a minimum number of cliques.

Now we show how to find a largest stable set of $G$. Let $k$ be the number of colors used on $\overline{G}$ by the greedy algorithm. We will show how to find a stable set $S$ of $G$ with $k$ vertices. Start with a vertex $x$ of color $k$, put $x$ in $S$. We go backward in $<$ to enlarge $S$. Suppose $S$ contains vertices $w_i, w_{i+1}, \ldots, w_k$ with $i > 1$ and $w_j$ having color $j$, $j = i, \ldots, k$. Let $S_{i-1}$ be the set of vertices of color $i - 1$. The proof of Theorem 28.31 implies there is a vertex $s \in S_{i-1}$ that is $S$-null and so can be added to $S$. Such vertex $s$ can be found by scanning the adjacency list of every vertex $s$ in $S_{i-1}$. The adjacency list of each vertex of $G$ is scanned at most once, so the algorithm runs in linear time.

Several classes $C$ of perfectly ordeable graphs have the property that if $G$ is in $C$ then not only that $G$ is perfectly orderable, but its complement $\overline{G}$ also is (for example, brittle graphs, and therefore chordal graphs). Theorem 28.34 is useful for optimizing these graphs.

**Corollary 28.8** [27] There is a linear-time algorithm for finding a largest clique, a minimum coloring, a largest stable set, and a minimum clique cover for a chordal graph.

**Proof.** Let $G$ be a chordal graph with a perfect elimination scheme $<$. Then $<$ is a perfect order on $\overline{G}$, and the reverse of $<$ is a perfect order on $G$. The result follows from Theorems 28.33 and 28.34.

A linear-time algorithm to recognize a co-chordal graph (complement of a chordal graph) and to construct a perfect order of such a graph is given in [70]. Thus, we have the following corollary.

**Corollary 28.9** [70] There is a linear-time algorithm for finding a largest clique, a minimum coloring, a largest stable set, and a minimum clique cover for a co-chordal graph.

Actually, for a perfectly ordered graph, there are algorithms to solve more general optimization problems. Consider the following.

**Minimum weighted coloring.** Given a weighted graph $G$ such that each vertex $x$ has a weight $w(x)$ which is a positive integer. Find stable sets $S_1, S_2, \ldots, S_k$ and integers $I(S_1), \ldots, I(S_k)$ such that for each vertex $x$ we have $w(x) \leq \sum_{x \in S} I(S_i)$ and that the sum of the numbers $I(S_i)$ is minimized. This sum is called the weighted chromatic number and denoted by $\chi_w(G)$.

**Maximum weighted clique.** Given a weighted graph $G$ such that each vertex $x$ has a weight $w(x)$ which is a positive integer. Find a clique $C$ such that $\Sigma_{x \in C} w(x)$ is maximized. This sum is called the weighted clique number and denoted by $\omega_w(G)$. 
Definition 28.25 A stable set of a graph $G$ is strong if it meets all maximal cliques of $G$. (Here, as usual, Maximal is meant with respect to set-inclusion, and not size. In particular, a maximal clique may not be a largest clique.) A graph is strongly perfect if each of its induced subgraphs contains a strong stable set.

Theorem 28.35 [65] Perfectly orderable graphs are strongly perfect. And if a perfect order on $G$ is given, then a strong stable set of $G$ can be found in linear time.

Proof. By induction on the number of vertices. We only need to prove that a graph $G$ with a perfect order $<$ contains a strong stable set. Let $S$ be the set of vertices colored with color 1 by the greedy algorithm. Assume that $S$ is not a strong stable set, for otherwise we are done. So, consider a maximal clique $C$ such that no vertex in $C$ has color 1. Properties of the greedy algorithm implies each vertex $v \in C$ has a neighbor $p(v)$ of color 1 with $p(v) < v$. But then Lemma 28.15 implies some $p(v)$ is $C$-complete, a contradiction. The fact that $S$ can be found in linear time follows from Theorem 28.33.

Theorem 28.36 [71] If there is a polynomial time algorithm $A$ to find a strong stable set of a strongly perfect graph then there is a polynomial time algorithm $B$ to find a minimum weighted coloring and maximum weighted clique of a strongly perfect graph. If algorithm $A$ runs in time $O(f(n))$ then algorithm $B$ runs in time $O(nf(n))$. Moreover if algorithm $A$ is strongly polynomial then so is algorithm $B$.

Proof. For a perfect graph $G$, it is known that $\chi_w(G) = \omega_w(G)$. Let $G$ be a strongly perfect (and therefore, perfect) graph with a weight function $w$ on its vertices. We will show the problem on $G$ can be transformed to the problem on a smaller graph $G'$ with an $O(f(n))$ time reduction. Suppose we can find a strong stable set $S$ of $G$ in $O(f(n))$ time. Let $T$ be a vertex in $S$ with the smallest weight among all vertices of $S$. Define a new weight function $w'(v) = w(v) - w(T)$ for each $v \in S$, and $w'(v) = w(v)$ for each $v \in G - S$. Let $X = \{v | w'(v) = 0\}$. Since $X \subseteq S$, $X$ is not empty. Consider the graph $G' = G - T$. Since every maximal clique of $G$ meets $S$, we have $\omega_w(G) = \omega_w(G') + w(T)$, and the same holds for $\chi_w(G) = \chi_w(G') + w(T)$. Suppose $S_1, \ldots, S_k$ is a minimum weighted coloring of $G'$ with weights $I(S_i)$ for $i = 1, \ldots, k$, and $I(S) = w(T)$. Similarly, if $C'$ is a maximum weighted clique of $G'$, then a maximum weighted clique of $G$ can be found as follows. If $C' \cap (S - X) \neq \emptyset$, then $C = C' \cup \{T\}$; otherwise, $C = C' \cup \{y\}$ where $y$ is a vertex in $X$ that is $(C')$-complete, $y$ exists because $S$ is a strong stable set (note that for $C$, we use the original weight function $w$).

We may recursively apply the above reduction until we get a trivial graph in at most $n$ steps. Since the complexity of our procedure does not depend on the size of the minimum $w(v)$, the reduction is strongly polynomial.

Theorems 28.35 and 28.36 implies the following.

Corollary 28.10 Given a graph $G$ and a perfect order on $G$, maximum weighted clique and minimum weighted coloring can be solved in $O(nm)$ time.

For comparability and chordal graphs, these two problems can be solved even faster.

Theorem 28.37 [71] If $G$ is a comparability graph or a chordal graph, then maximum weighted clique and minimum weighted coloring can be solved in $O(n^2)$ time.

Space-efficient algorithms for maximum weighted clique and minimum weighted coloring of co-chordal graphs are given in [70]. Theorem 28.36 shows that the problem of finding
strong stable set of a strongly perfect graph is of some consequence. However, no polynomial algorithm for solving this problem is known. Finding a strong stable set of an arbitrary graph is NP-hard [71].

28.8 PERFECTLY CONTRACTILE GRAPHS

Recall the definition of an even-pair in Section 28.6. Even-pairs play a central role in the study of perfect graphs, as illustrated by the following two results.

Lemma 28.16 [60] Let $G$ be a perfect graph with an even-pair $\{x, y\}$. Then $G/xy$ is perfect.

Lemma 28.17 [72] No minimal imperfect graph contains an even-pair.

From the above, it is of interest to know which perfect graphs contain even-pairs.

**Definition 28.26** A graph $G$ is *even-contractile* if there is a sequence $G_0 = G, G_1, \ldots, G_k$ such that $G_k$ is a clique, and for $i \leq k - 1$, $G_{i+1}$ is obtained from $G_i$ by a contraction of some even-pair of $G_i$.

An even-contractile graph $G$ has $\chi(G) = \omega(G)$ by Lemma 28.13. But this class seems to be difficult to characterize: perhaps because the class is not hereditary. Now, consider the following definition from [73].

**Definition 28.27** A graph is perfectly contractile if each of its induced subgraphs is even-contractile.

By Lemma 28.13, perfectly contractile graphs are perfect. Most classes of graphs discussed in this chapter are perfectly contractile. Lemma 28.14 implies the following.

**Theorem 28.38** [52] Weakly chordal graphs are perfectly contractile.

A graph is called a *Meyniel* graph if each of its odd cycle with at least five vertices has two chords. Perfection of Meyniel graphs was established in [74]. Note that chordal graphs are Meyniel graphs.

**Theorem 28.39** [75] Meyniel graphs are perfectly contractile.

**Theorem 28.40** [76] Perfectly orderable graphs perfectly contractile.

**Definition 28.28** A prism is a graph that consists of two vertex-disjoint triangles (cliques of size three) and three vertex-disjoint paths, each of length at least one and having an endpoint in each triangle, with no other edge than those in the two triangles and in the three paths. A prism is odd if all three paths are odd.

The following beautiful and challenging conjecture was proposed in [77].

**Conjecture 28.1** [77] A graph is perfectly contractile if and only if it contains no odd hole, no anti-hole, and no odd prism.

**Definition 28.29** A graph is an *Artemis* graph if it contains no odd hole, no anti-hole, and no prism.

Validity of Conjecture 28.1 was partially established by the following remarkable result.

**Theorem 28.41** [78] Artemis graphs are perfectly contractile.

An $O(n^2m)$ time algorithm to color an Artemis graph is given in [79]. Note that weakly chordal graphs and perfectly orderable graphs are Artemis graphs. An $O(n^3)$ time algorithm for recognizing an Artemis graph is given in [80].
28.9 RECOGNITION OF PERFECT GRAPHS

In this section, we give a sketch of a polynomial time algorithm to recognize a perfect graph. By the strong perfect graph theorem, the problem is equivalent to determining if a graph is Berge (graphs with no odd holes and no odd anti-holes). A polynomial time algorithm to solve this problem is given in [15]. The algorithm can be divided into three phases. In the first phase, given a graph $G$, the algorithm looks for one of five configurations. Each of these five configurations can be detected in time $O(n^9)$ or faster. If $G$ contains one of these, then $G$ is not Berge; otherwise, every shortest odd hole of $G$ has a special property called amenable. Given an odd hole $C$ of length at least seven, a set $X$ of vertices is a near-cleaner if it contains all vertices that have two neighbors of distance at least three in $C$ and $X \cap C$ is a subset of the vertex set of some path of length three of $C$. Amenable odd holes are those odd holes such that all near-cleaners have some special adjacency property (definitions not given here will be given later). If the first phase does not produce an odd hole or odd antihole, the second phase will generate $O(n^5)$ sets that are guaranteed to contain all near-cleaners of some amenable odd hole if one exists. Finally, the third phase provides an $O(n^4)$ algorithm that given a graph and a near-cleaner for a shortest odd hole finds an odd hole. Now, we describe the algorithm in more detail.

Definition 28.30 A pyramid is an induced subgraph formed the union of a triangle \( \{b_1, b_2, b_3\} \), a fourth vertex $a$, and three induced paths $P_1, P_2, P_3$, satisfying:

- For $i = 1, 2, 3$, the endpoints of $P_i$ are $a, b_i$.
- For $i \leq i < j \leq 3$, $a$ is the only vertex in both $P_i, P_j$, and $b_i b_j$ is the only edge between $P_i - a$ and $P_j - a$.
- $a$ is adjacent to at most one of $b_1, b_2, b_3$.

Definition 28.31 A jewel is the graph formed by a cycle with vertices $v_1, v_2, \ldots, v_5$ and edges $v_i v_{i+1}$ (with the subscript taken modulo 5) and an induced path $P$ such that $v_1 v_3, v_2 v_4, v_1 v_4$ are nonedges, $v_1, v_4$ are the endpoints of $P$, and there is no edges between $\{v_2, v_3, v_5\}$ and the interior vertices of $P$.

Definition 28.32 A configuration of type $T_1$ is the hole on five vertices.

Definition 28.33 A configuration of type $T_2$ is a sequence $v_1, v_2, v_3, v_4, P, X$ such that

- $v_1, v_2, v_3, v_4$ induce a $P_4$ with endpoints $v_1, v_4$.
- $X$ is an anticomponent of the set of all $\{v_1, v_2, v_4\}$-complete vertices.
- $P$ is an induced path in $G \setminus (X \cup \{v_2, v_3\})$ between $v_1, v_4$, and no interior vertex of $P$ is $X$-complete or adjacent to $v_2$ or adjacent to $v_3$.

Definition 28.34 A configuration of type $T_3$ is a sequence $v_1, \ldots, v_6, P, X$ such that

- $v_1, \ldots, v_6$ are distinct vertices
- $v_1 v_2, v_2 v_3, v_3 v_4, v_4 v_5, v_5 v_6$ are edges, and $v_1 v_3, v_2 v_4, v_1 v_5, v_2 v_5, v_1 v_6, v_2 v_6, v_4 v_5$ are nonedges
- $X$ is an anticomponent of the set of all $\{v_1, v_2, v_3\}$-complete vertices, and $v_3, v_4$ are not $X$-complete.
• P is an induced path of \( G \setminus (X \cup \{v_1, v_2, v_3, v_4\}) \) between \( v_5, v_6 \), and no interior vertex of \( P \) is \( X \)-complete or adjacent to \( v_1 \) or adjacent to \( v_2 \).

• If \( v_5v_6 \) is an edge, then \( v_6 \) is not \( X \)-complete.

In [15], it is shown that a pyramid can be detected in \( O(n^9) \) time, a jewel in \( O(n^6) \) time, a configuration of type \( T_1 \) in \( O(n^5) \) time (obviously), a configuration of type \( T_2 \) or \( T_3 \) in \( O(n^5) \) time.

**Theorem 28.42** [15] If \( G \) or \( \overline{G} \) contains a pyramid, a jewel, or a configuration of type \( T_1, T_2, \) or \( T_3 \), then \( G \) is not Berge.

Given a hole \( C \) of length at least seven, a vertex \( x \) is \( C \)-major if \( x \) has two neighbors in \( C \) whose distance in \( C \) is at least three. A hole \( C \) of \( G \) is amenable if (i) \( C \) is a shortest odd hole of length at least seven of \( G \), and (ii) for every anticonnected set \( X \) of \( C \)-major vertices, there is an \( X \)-complete edge in \( C \).

**Theorem 28.43** [15] If \( G \) contains no pyramid, and no configuration of type \( T_1, T_2, \) or \( T_3 \), and both \( G, \overline{G} \) contains no jewel, then every shortest odd hole of \( G \) is amenable.

Recall that a set \( X \) of vertices is a near-cleaner for an odd hole \( C \) of length at least seven if it contains all \( C \)-major vertices, and \( X \cap C \) is a subset of the vertex set of some path of length three of \( C \).

**Theorem 28.44** [15] There is an \( O(n^5) \) algorithm which given a graph \( G \) outputs \( O(n^5) \) subsets of \( V(G) \) such that if \( C \) is an amenable odd hole of \( G \), then one of the subsets is a near-cleaner for \( C \).

**Theorem 28.45** [15] There is an \( O(n^4) \) algorithm which given a graph \( G \) containing no pyramid or jewel, and a subset \( X \) of \( V(G) \) outputs an odd hole, or determines that there is no shortest odd hole \( C \) of \( G \) such that \( X \) is a near-cleaner for \( C \).

The steps needed to recognize a perfect graph are described in Algorithm 28.8. There are two bottlenecks to making the algorithm run faster than \( O(n^9) \) time.

---

**Algorithm 28.8** perfect graph recognition

**input:** graph \( G \\
**output:** a determination that \( G \) is Berge or not

1. Determine if \( G \) or \( \overline{G} \) contains a pyramid, or a jewel, or a configuration of type \( T_1, T_2, \) or \( T_3 \). If it does, output \( G \) is not Berge, and stop
2. Produce \( O(n^5) \) subsets \( X \) of \( V(G) \) using Theorem 28.44. These subsets contain all near-cleaners of some odd hole of \( G \), if such an odd hole exists
3. For each subset \( X \) of (2), run the algorithm of Theorem 28.45. If an odd hole is produced, output \( G \) is not Berge, and stop
4. Run (2) and (3) with \( G \) replaced by \( \overline{G} \)
5. Output \( G \) is Berge

---

The first one is that as of present, there is no algorithm to detect a pyramid in time faster than \( O(n^9) \). The second involves the near-cleaners. It is not known if given a near-cleaner, one can find an odd hole in time faster than \( O(n^4) \). It is also not known if a graph can have fewer than \( O(n^5) \) near-cleaners.
28.10 \( \chi \)-Bounded Graphs

**Definition 28.35** A graph \( G \) is \( \chi \)-bounded if there is a function \( f \) such that \( \chi(G) \leq f(\omega(G)) \).

We have seen that perfect graphs are \( \chi \)-bounded. One may wonder about sufficient conditions on the holes of a graph for it to be \( \chi \)-bounded. Some interesting conditions have been found.

**Theorem 28.46** [16] If a graph \( G \) is even hole-free, then \( G \) contains a vertex whose neighborhood can be partitioned into two cliques. In particular, \( G \) satisfies \( \chi(G) \leq 2\omega(G) - 1 \).

References [81-83] give two different polynomial time algorithms (of high complexity) for finding an even hole in a graph.

It is reasonable to expect that graphs without odd holes have bounded chromatic number. Before discussing this matter, we will need a definition.

**Definition 28.36** A \( k \)-division of a graph \( G \) with at least one edge is a partition of \( V(G) \) into \( k \) sets \( V_1, \ldots, V_k \) such that no \( V_i \) contains a clique with \( \omega(G) \) vertices. A graph is \( k \)-divisible if each induced subgraph of \( G \) with at least one edge admits a \( k \)-division.

It is easy to see the following.

**Lemma 28.18** A \( k \)-divisible graph \( G \) has \( \chi(G) \leq k^{\omega(G)} - 1 \).

Consider the following conjectures.

**Conjecture 28.2** [84] A graph is 2-divisible if and only if it is odd hole-free.

The above conjecture implies that an odd hole-free graph \( G \) has \( \chi(G) \leq 2^{\omega(G)} - 1 \), and thus is \( \chi \)-bounded. The conjecture is known to hold for claw-free graphs [84], 2\( K_2 \)-free graphs [17], and \( K_4 \)-free graphs [85]. The problem of recognizing odd hole-free graphs is open.

We now mention a number of conjectures related to \( \chi \)-bounded graphs and forbidden subgraphs.

**Conjecture 28.3** [84] Let \( F \) be any forest on \( k \) vertices. Then any graph \( G \) that does not contain \( F \) as induced subgraph is \( k \)-divisible.

It is not known if Conjecture 28.3 holds for claw-free graphs.

**Definition 28.37** Let \( G \) be a graph with at least one hole. The hole number \( h(G) \) of \( G \) is the length of the longest hole in \( G \).

**Conjecture 28.4** [84] Let \( G \) be a graph with at least one hole. Then \( G \) is \( (h(G) - 2) \)-divisible.

The following special case of Conjecture 28.4 is still open.

**Conjecture 28.5** [84] If \( G \) is a triangle-free graph with at least one hole, then \( \chi(G) \leq h(G) - 2 \).
References


