COMPLEXITY OF GENERALIZED COLOURINGS OF CHORDAL GRAPHS

by

Juraj Stacho

M.Sc., Comenius University, Slovakia, 2005

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Spring 2008

APPROVAL

Name: Juraj Stacho

Degree: Doctor of Philosophy

Title of thesis: Complexity of Generalized Colourings of Chordal Graphs

Examining Committee: Dr. Joseph G. Peters
Chair

Dr. Pavol Hell, Senior Supervisor

Dr. Ramesh Krishnamurti, Supervisor

Dr. Gábor Tardos, SFU Examiner

Dr. Ross M. McConnell, External Examiner,
Computer Science Department,
Colorado State University

Date Approved: March 25, 2008

Abstract

The generalized graph colouring problem (GCOL) for a fixed integer k, and fixed classes of graphs $\mathcal{P}_1, \ldots, \mathcal{P}_k$ (usually describing some common graph properties), is to decide, for a given graph G, whether the vertex set of G can be partitioned into sets $V_1, \ldots, V_k$ such that, for each i, the induced subgraph of G on V_i belongs to $\mathcal{P}_i$. It can be seen that GCOL generalizes many natural colouring and partitioning problems on graphs.

In this thesis, we focus on generalized colouring problems in chordal graphs. The structure of chordal graphs is known to allow solving many difficult combinatorial problems, such as the graph colouring, maximum clique and others, in polynomial, and in many cases in linear time. Our study of generalized colouring problems focuses on those problems in which the sets $\mathcal{P}_i$ are characterized by a single forbidden induced subgraph. We show, that for $k = 2$, all such problems where the forbidden graphs have at most three vertices are polynomial time solvable in chordal graphs, whereas, it is known that almost all of them are NP-complete in general. On the other hand, we show infinite families of such problems which are NP-complete in chordal graphs. By combining a polynomial algorithm and an NP-completeness proof, we answer a question of Broersma, Fomin, Nešetřil and Woeginger about the complexity of the so-called subcolouring problem on chordal graphs. Additionally, we explain, how some of these results generalize to particular subclasses of chordal graphs, and we show a complete forbidden subgraph characterization for the so-called monopolar partitions of chordal graphs.

Finally, in the last part of the thesis, we focus on a different type of colouring problem – injective colouring. We describe several algorithmic and (in-)approximability results for injective colourings in the class of chordal graphs and its subclasses. In the process, we correct a result of Agnarsson et al. on inapproximability of the chromatic number of the square of a split graph.

Keywords: chordal graphs; graph colouring; subcolouring; injective colouring; forbidden subgraph characterization; polynomial time algorithms

Subject Terms: Graph Theory; Graph Coloring; Graph Algorithms; Perfect Graphs; Homomorphisms (Mathematics); Graph Grammars

Mojim

rodičom

Milanovi a Darine

To

my parents

Milan and Darina

Acknowledgments

The author would like to seize the opportunity to express his deepest gratitude to his thesis advisor Dr. Pavol Hell for his guidance, enthusiasm, his profound insight, and broad knowledge, and most importantly for his perpetual support during author's graduate studies at Simon Fraser University.

The author also wishes to thank his parents for their unconditional support throughout his entire academic studies and wishes to dedicate this thesis to them.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The graph colouring problem is probably one of the oldest problems of graph theory. Here, the task is to label the vertices of a given graph using as few colours as possible in such a way that no adjacent vertices receive identical labels. The origins of this problem can be traced all the way back to the famous problem of four colours, formulated by Francis Guthrie in 1852, which asks whether any map of countries can be coloured using at most four colours in such a way that no two countries sharing a boundary use the same colour. As it turned out later, this is just a special case of the graph colouring problem in planar graphs. Since then, graph colouring and its variants have appeared in many different guises in combinatorics, optimization, discrete geometry, set theory, and many other branches of discrete mathematics. Also, many practical problems such as time tabling, sequencing, and scheduling are related to the problem of graph colouring.

Unfortunately, graph colouring is one of the first known NP-complete problems. This tells us that it is unlikely that an efficient algorithm for graph colouring exists, where, of course, by efficient, we mean running in polynomial time (possibly also randomized). This, however, does not mean that the problem is not efficiently solvable in some special case, and indeed, there are numerous special cases in which the problem has a nice algorithmic solution. These cases usually form classes of graphs with some structural property which allows solving the problem efficiently.

Another line of research on graph colouring focuses on variants and generalizations of this notion. This is, of course, inspired by practical problems for which the language of

graph colourings is not very convenient. Such examples are the problems of edge colouring, subcolouring, circular colouring, graph homomorphism, and many others [47]. In particular, the graph homomorphism problem is noted for having many practical applications [17, 29, 49, 54].

In this thesis, we study a particular common generalization of graph colouring and related graph problems. It is defined as follows.

The generalized graph colouring problem (GCOL) for a fixed integer k, and fixed classes of graphs $\mathcal{P}_1, \ldots, \mathcal{P}_k$, is to decide, given a graph G, whether the vertex set of G can be partitioned into sets $V_1, \ldots, V_k$ such that, for each $i \in \{1, \ldots, k\}$, the induced subgraph of G on V_i belongs to $\mathcal{P}_i$. In some cases, we shall consider the following list version of GCOL.

The generalized list colouring problem (List-GCOL) for a fixed integer k, and fixed classes of graphs $\mathcal{P}_1, \ldots, \mathcal{P}_k$, is to decide, given a graph G with lists $\ell(v) \subseteq \{1, \ldots, k\}$ for all $v \in V(G)$, whether there exists a partition $V_1, \ldots, V_k$ of $V(G)$ with $G[V_i] \in \mathcal{P}_i$ for each $i \in \{1, \ldots, k\}$, which *respects* the lists ℓ, that is, for any vertex $v \in V(G)$ and any $i \in \{1, \ldots, k\}$, if v is in V_i, then $i \in \ell(v)$.

We say that a class $\mathcal{P}$ of graphs is *(induced)* hereditary, if $\mathcal{P}$ is closed under taking (induced) subgraphs, that is, if H is an (induced) subgraph of $G \in \mathcal{P}$, then also $H \in \mathcal{P}$. We say that $\mathcal{P}$ is *(co-) additive*, if $\mathcal{P}$ is closed under taking disjoint unions (respectively joins) of graphs. It can be seen that for any set $\mathcal{P}$ of graphs, the membership problem of $\mathcal{P}$ can be expressed as GCOL simply by putting $k = 1$ and $\mathcal{P}_1 = \mathcal{P}$. Hence, GCOL, in general, can be as difficult as it gets, even undecidable. This is still true even if we insist that $\mathcal{P}$ is additive and hereditary. (Consider the smallest additive hereditary class of graphs containing chordless cycles whose sizes are from an undecidable set of integers.) We therefore focus on those GCOL problems for which the membership problem of the classes $\mathcal{P}_i$ is decidable in polynomial time. Observe that all such problems are clearly in the class NP. In particular, if all $\mathcal{P}_i$ are additive and induced hereditary, the complexity of GCOL is completely resolved by the following result of Farrugia.

Theorem 1.1. *[26] Let $k \geq 2$, and let $\mathcal{P}_1, \ldots, \mathcal{P}_k$ be additive induced-hereditary classes. Then the problem GCOL for $\mathcal{P}_1, \ldots, \mathcal{P}_k$ is NP-hard, unless $k = 2$ and $\mathcal{P}_1 = \mathcal{P}_2 = \mathcal{O}$ (the set of all edgeless graphs), in which case the problem is polynomial time solvable.*

Note that this theorem does not assume polynomial time membership of the classes $\mathcal{P}_i$. Similarly, Farrugia proves the following result.

Theorem 1.2. *[26] Let $k \geq 2$, and let $\mathcal{P}_1, \ldots, \mathcal{P}_k$ be additive or co-additive induced-hereditary classes. Then the problem GCOL for $\mathcal{P}_1, \ldots, \mathcal{P}_k$ is NP-hard, unless $k = 2$ and the membership for both $\mathcal{P}_1$ and $\mathcal{P}_2$ is polynomial time solvable.*

Observe that this theorem does not say anything about the complexity in the case when $k = 2$, $\mathcal{P}_1$ is additive, $\mathcal{P}_2$ is co-additive, and both are polynomially recognizable. In fact, not much is known in this case with the exception of the following result of Feder, Hell, Klein, and Motwani [27], which was also independently shown by Alekseev, Farrugia, and Lozin [3] in a somewhat weaker form.

Theorem 1.3. [3, 27] *Let $\mathcal{P}_1$ be a class of graphs G with clique number $\omega(G) < k$, and $\mathcal{P}_2$ be a class of graphs G with independence number $\alpha(G) < \ell$. Then GCOL for $\mathcal{P}_1, \mathcal{P}_2$ can be solved in time $O(n^{2R(k,\ell)+2}T(n))$, where $R(k,\ell)$ is the Ramsey number of k and ℓ, and $T(n)$ is the time complexity of the membership problem for $\mathcal{P}_1$, respectively $\mathcal{P}_2$.*

We remark that (induced) hereditary classes $\mathcal{P}$ are particularly interesting since they can be characterized by sets of *forbidden (induced) subgraphs* $\mathcal{F}(\mathcal{P})$, that is, graphs such that $G \in \mathcal{P}$, if and only if, G does not contain F as an (induced) subgraph for all $F \in \mathcal{F}(\mathcal{P})$. This follows from the fact the (induced) subgraph order of graphs has no infinite descending chain. We note that, in particular, if the set $\mathcal{F}(\mathcal{P})$ is finite, then the membership problem for $\mathcal{P}$ is polynomially solvable, which is a fact frequently used in the literature.

In the following chapters, we do not consider the GCOL problem in its full generality, but rather focus on a restricted case, where all classes $\mathcal{P}_i$ are induced hereditary and each is characterized by a single forbidden induced subgraph. As follows from the above, almost all such problems are NP-complete in general. However, this is not necessarily the case when the input of the problem is restricted.

This motivates us to study the complexity of GCOL in structured classes of graphs, whose properties may allow us to solve many instances of GCOL efficiently. In this thesis, we focus on the class of chordal graphs and its subclasses. This is a class of graphs with particularly nice structure, and one for which many combinatorial problems are known to be efficiently solvable.

The thesis is structured as follows. In the remaining sections of this chapter, we first introduce chordal graphs and describe their structural properties, and then briefly mention some similarly structured classes of graphs which are related to chordal graphs. After that, we describe three simple, yet important cases of the GCOL problem which are polynomially

solvable in all graphs, but are not all covered by the above theorems. Next, in Chapter 2, we describe some general tools for constructing efficient algorithms for combinatorial problems on graphs, namely dynamic programming techniques, greedy algorithms, graph grammars, and MSO-definable properties. After that, in Chapters 3, 5, and 6, we establish the complexity of selected GCOL problems in chordal graphs, namely the monopolar partition, the polar partition, the subcolouring, and the P_j- and K_j-free P_k-transversal problems, in that order, either by giving a polynomial time algorithm, or by proving NP-completeness of the problem. Additionally, in Chapter 4, we completely describe all forbidden induced subgraphs for the monopolar partition problem in chordal graphs. After that, in Chapter 7, we establish the complexity of several other cases of the GCOL problem in chordal graphs which, in particular, will allow us to deduce the following theorem.

Theorem 1.4. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be induced hereditary classes of graphs, each characterized by a single forbidden induced subgraph having at most three vertices. Then GCOL for $\mathcal{P}_1, \mathcal{P}_2$ is polynomial time solvable in the class of chordal graphs.*

Later, in Chapter 7, we also discuss extensions of these results to strongly chordal graphs and chordal comparability graphs. Finally, in Chapter 8, we describe a different colouring problem on graphs, the injective colouring problem, and solve a number of complexity results for this problem in chordal graphs.

1.1 Definitions

A *graph* G is a pair (V, E) where V is a set of *vertices* (also called a *vertex set*), and E, called a set of *edges* (or an *edge set*), is a binary relation on V, that is, $E \subseteq V \times V$. For a graph G, the set of vertices and the set of edges of G is denoted by $V(G)$ and $E(G)$, respectively.

A graph G is *undirected*, if $E(G)$ is symmetric. In case we want to emphasize that G is not necessarily undirected, we say that G is *directed*, or that G is a *digraph*. A graph G is *loopless*, if $E(G)$ is irreflexive. Note that the definition of graph above does not allow multiple edges between two vertices; in other words, our graphs are what is sometimes referred to as *simple*.

We say that vertices u and v are *adjacent* in G (or that u and v are *neighbours* in G), if $uv \in E(G)$ or $vu \in E(G)$. A walk W in G is a sequence of vertices $W = u_1, u_2, \ldots u_k$ such that for each $1 \leq i < k$, the vertices u_i and u_{i+1} are adjacent. A walk $W = u_1, \ldots, u_k$ in G is *closed*, if the vertices u_1 and u_k are also adjacent. A (closed) walk W in G is a *path*

(respectively a *cycle*), if no vertex of W appears on W more than once. We denote by P_k the path on k vertices with $k-1$ edges, and denote by C_k the cycle on k vertices with k edges. We say that G is *connected*, if, for any two vertices u, v of G, there exists a path that connects them, that is, a path $u = u_1, u_2, \ldots, u_k = v$. Otherwise, we say that G is *disconnected*. We say that a graph is *acyclic*, if it contains no cycle. An undirected acyclic graph is called a *forest*. A connected forest is called a *tree*.

We say that a graph H is a *subgraph* of G and denote it by $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. We say that a subgraph H of G is *induced*, if $E(H) = E(G) \cap V(H) \times V(H)$; in that case we write $H \leq G$. For any subset S of the vertices of G, we denote by $G[S]$ the induced subgraph of G whose vertex set is S. We also say that $G[S]$ is a subgraph of G *induced on* S. For a subset S of vertices (respectively edges), we denote by $G - S$ the subgraph of G that is obtained by removing from G the vertices (edges) of S. (Note that with each removed vertex we must also remove all edges incident to it.) In the case that S consists only of a single element x, we write $G - x$ instead of $G - \{x\}$.

For a connected graph G, a vertex u is a *cutpoint* of G, if the graph $G - u$ is disconnected. An edge $e = uv$ is a *bridge* of G, if the graph $G - e$ is disconnected. A subset S of vertices (edges) of G is a *vertex (edge) separator* of G, if $G - S$ is disconnected.

We say that two graphs G and H are *isomorphic* and denote it by $G \cong H$, if there exists a bijective mapping $f : V(G) \to V(H)$ such that $uv \in E(G)$, if and only if, $f(u)f(v) \in E(H)$ for any $u, v \in V(G)$. The *complement* of a graph G is the graph $\overline{G}$ with vertices $V(G)$ and edges uv where $u \neq v$ and $uv \notin E(G)$. The *union* $G \cup H$ of two graphs G and H is the graph with vertices $V(G) \cup V(H)$ and edges $E(G) \cup E(H)$. The *disjoint union* $G \uplus H$ of G and H is the union of graphs G' and H, where $G \cong G'$ and $V(G') \cap V(H) = \emptyset$. The *join* $G + H$ of two graphs G and H is the complement of $\overline{G} \uplus \overline{H}$.

A graph G is *complete*, if it contains edges between all pairs of distinct vertices. We denote by K_n the complete graph on n vertices. A *clique* of G is a complete subgraph of G, and an *independent set* of G is an induced subgraph of G having no edges. For any graph G, we denote by $\omega(G)$, and $\alpha(G)$, the size of a maximum clique in G, and the size of a maximum independent set in G, respectively.

For a subset S of the vertices of G, we denote by $N(S)$ the *open neighbourhood* of S, that is, the set of vertices of $G - S$ that are adjacent to at least one vertex of S. We denote by $N[S]$ the *closed neighbourhood* of S, that is, $N[S] = N(S) \cup S$. If S contains only one element u, we abbreviate $N(\{u\})$ and $N[\{u\}]$ to $N(u)$ and $N[u]$, respectively. We let $\deg(u) = |N(u)|$

be the *degree* of u, and let $\Delta(G)$ be the maximum degree among the vertices of G.

Throughout the whole thesis, unless indicated otherwise, n denotes the number of vertices of G, and m denotes the number of edges of G.

A *colouring* of a graph G is a mapping $c : V(G) \to \mathbb{N}$ where $\mathbb{N}$ is the set of all natural numbers. A *k-colouring* of G is a mapping $c : V(G) \to \{1, 2, \ldots, k\}$. A colouring c of G is a *proper* colouring, if, for any $u, v \in V(G)$, we have $c(u) \neq c(v)$ whenever $uv \in E(G)$. The *chromatic number* of G, denoted by $\chi(G)$, is the smallest integer k such that G admits a proper k-colouring.

A *dominating set* S in G is a subset of vertices of G such that any vertex of G not in S has at least one neighbour in S. The *domination number* of G, denoted by $\gamma(G)$, is the size of a smallest dominating set in G.

For any two vertices u, v of a connected graph G, we denote $d(u, v)$ the *distance* between u and v, that is, the length of a shortest path between u and v in G. We denote by G^k the *k-th power* of G, that is, the graph obtained from G by making adjacent any two vertices in distance at most k.

A *tree decomposition* of a connected graph G is a pair (T, X) such that T is a tree, X is a mapping which assigns each vertex $u \in V(T)$ a *bag* $X(u)$ of vertices of G, and

(i) for each edge $xy \in E(G)$, there exists $u \in V(T)$ with $x, y \in X(u)$, and

(ii) for each vertex $x \in V(G)$, the vertices $u \in V(T)$ with $x \in X(u)$ induce a connected subgraph of T.

The *width* of a tree decomposition (T, X) is the size of a largest bag of T minus one. The *treewidth* $tw(G)$ of G is the minimum width of a tree decomposition of G.

For a graph H, we say that G is H-free is G contains no induced subgraph isomorphic to H. For a set of graphs $\mathcal{H}$, we say that G is $\mathcal{H}$-free is G contains no induced subgraph isomorphic to some $H \in \mathcal{H}$.

We say that a partition of the vertex set $V(G)$ of a graph G into sets $V_1 \cup V_2$ is *monopolar*, if V_1 induces an independent set (that is, a K_2-free graph), and V_2 induces a P_3-free graph. A partition of the vertex set $V(G)$ of a graph G into sets $V_1 \cup V_2$ is *unipolar*, if V_1 induces a clique (that is, a $\overline{K}_2$-free graph), and V_2 induces a P_3-free graph. A partition of the vertex set $V(G)$ of a graph G into sets $V_1 \cup V_2$ is *polar*, if V_1 induces a $\overline{P}_3$-free graph, and V_2 induces a P_3-free graph. A *k-subcolouring* of a graph G is a partition of the vertices of G into k sets $V_1 \cup V_2 \cup \ldots \cup V_k$, such that each V_i induces a P_3-free graph.

An *H-transversal* of a graph G is a subset S of the vertices of G that intersects each induced copy of H in G, that is, $G-S$ is H-free. In particular, an *F-free H-transversal* of G is an H-transversal S of G which induces an F-free subgraph of G. Any such H-transversal S can be interpreted as a partition of vertices of G into sets $V_1 = S$ and $V_2 = V(G) \setminus S$ such that V_1 is F-free, and V_2 is H-free.

1.2 Chordal graphs

A graph G is *chordal* (*triangulated* or *rigid-circuit* [38]), if it does not contain an induced subgraph isomorphic to C_k for $k \geq 4$. The class of chordal graphs is an interesting and widely studied class of graphs. The graphs in this class possess numerous useful structural properties. Here, we shall mention some of them.

An *elimination ordering* π of a graph G is a numbering of the vertices of G from 1 to n. The *fill-in* F_π caused by the ordering π is the set of edges defined as follows.

$$F_\pi = \left\{ uv \;\middle|\; \begin{array}{l} u \neq v,\ uv \notin E(G) \text{ and there exists a } u\text{-}v \text{ path } P \text{ in } G \\ \text{with } \pi(w) < \min\{\pi(u), \pi(v)\} \text{ for all } u, v \neq w \in P \end{array} \right\}$$

An elimination ordering π is *perfect*, if $F_\pi = \emptyset$. Equivalently, one can say that a total ordering $\prec$ on the vertices of G is a perfect elimination ordering of G, if whenever y and z are neighbours of x such that $x \prec y$ and $x \prec z$, we have that y and z are adjacent. An elimination ordering π is *minimal*, if there is no elimination ordering σ with $F_\sigma \subsetneq F_\pi$. The graph $G_\pi = (V, E \cup F_\pi)$ is the *fill-in graph* for π.

Elimination orderings arise in the study of Gaussian elimination of sparse symmetric matrices [56]. The following properties establish a connection between perfect elimination orderings and chordal graphs. We say that a vertex v in a graph G is *simplicial*, if the neighbourhood of v induces a clique in G.

Proposition 1.5. [19] *Any chordal graph contains a simplicial vertex.*

Proposition 1.6. [57] *Any elimination ordering π is a perfect elimination ordering of G_π.*

Proposition 1.7. [19, 57] *A graph G has a perfect elimination ordering, if and only if, G is chordal.*

Perfect elimination orderings play an important role in many algorithms for chordal graphs. In particular, it can be shown [38] that using a perfect elimination ordering of a

chordal graph G, one can compute in time $O(n+m)$ the chromatic number $\chi(G)$ of G, the size of a maximum clique $\omega(G)$ of G, and also the size of a maximum independent set $\alpha(G)$ of G; all these problems are intractable in general. Moreover, for any chordal graph G, the chromatic number of G and the size of a maximum clique of G are equal. Hence, since the class of chordal graphs is induced hereditary, this shows that chordal graphs are perfect. We discuss more examples of perfect graphs in the next section.

Now, let T be a tree and $\mathcal{T} = \{T_1, \ldots, T_n\}$ be a collection of (connected) subtrees of T. The *vertex intersection graph* of $\mathcal{T}$ is a graph G with vertex set $V(G) = \{v_1, \ldots, v_n\}$ in which two vertices v_i and v_j are adjacent, if and only if, the trees T_i and T_j share a vertex, that is, $V(T_i) \cap V(T_j) \neq \emptyset$. (A similar class of so-called *edge intersection graphs* of paths in a tree has also been studied [39].) We have the following property.

Proposition 1.8. [36] *A graph G is chordal, if and only if, G is the vertex intersection graph of subtrees of some tree.*

A family of sets $\{X_1, \ldots, X_n\}$ is said to satisfy the *Helly property*, if, for each collection $\{X_i\}_{i \in I}$ of pairwise intersecting sets of the family, there exists an element which belongs to each set of the collection, that is, $\bigcap_{i \in I} X_i \neq \emptyset$. The following claim is easy to observe.

Proposition 1.9. *A family of subtrees of a tree satisfies the Helly property.*

This implies the following well-known fact.

Proposition 1.10. [33] *Any chordal graph G on n vertices has at most n maximal cliques.*

Lastly, we have the following property.

Proposition 1.11. [38] *Every minimal vertex separator in a chordal graph induces a clique.*

This property allows one to construct a decomposition of any chordal graph using its minimal separators. This is captured by the following notion.

A *clique-tree* T of a connected chordal graph G is a tree such that *(i)* each vertex $v \in V(T)$ corresponds to a maximal clique C_v of G, and *(ii)* for any vertex $x \in V(G)$, the vertices $v \in V(T)$ with $x \in C_v$ induce a connected subgraph T_x in T.

The following property explains a connection between vertex intersection graphs of subtrees of a tree and clique-trees of chordal graphs.

Proposition 1.12. *Any chordal graph G is the vertex intersection graph of subtrees T_x of a clique-tree T of G.*

Finally, we establish a connection between perfect elimination orderings and clique-trees of chordal graphs.

Let T be a fixed clique-tree of a chordal graph G. Let us consider T rooted at a vertex r chosen arbitrarily, and let $\sqsubset$ be a strict partial order on the vertices of T defined as follows: $u \sqsubset v \iff u$ is a descendant of v. Based on this, one can easily observe that the vertex set of any (connected) subgraph of T has a (unique) maximal element with respect to $\sqsubset$. Hence, for any vertex $x \in V(G)$, let m_x be the unique maximal element of T_x (the connected subgraph of T formed by vertices $v \in V(T)$ with $x \in C_v$) with respect to $\sqsubset$.

Now, let $\prec$ be a strict partial order on the vertices of G defined as follows: $x \prec y \iff m_x \sqsubset m_y$. We have the following property, which characterizes the strict partial order $\prec$.

Proposition 1.13. *Any linear extension of $\prec$ is a perfect elimination ordering of G.*

(We remark that this useful fact appears to not have been previously observed explicitly.)

Proof. First, we prove that $y \not\prec x$ implies $y \in C_{m_x}$, for any two adjacent vertices x, y. Since $xy \in E(G)$, there must exist a clique C_u such that $x, y \in C_u$. Hence, by the definition of m_x and m_y, we have $u \sqsubset m_x$ and $u \sqsubset m_y$. This implies that both m_x and m_y belong to the path from u to the root of T. Hence, either $m_x \sqsubset m_y$, or $m_y \sqsubset m_x$. But, since we assume that $y \not\prec x$, it follows that $u \sqsubset m_x \sqsubset m_y$. Now, since $y \in C_{m_y}$ and $y \in C_u$, it follows that y must belong to each clique on the path from u to m_x, and hence, $y \in C_{m_x}$.

Now, let π be a linear extension of $\prec$, and let x, y, z be vertices such that $\pi(x) < \pi(y) < \pi(z)$, and xy and xz are edges of G. Since $\pi(x) < \pi(y)$, and π is a linear extension of $\prec$, we must have $y \not\prec x$. Similarly, we have $z \not\prec x$. By the above, it follows that $y \in C_{m_x}$ and also $z \in C_{m_x}$. Now, since C_{m_x} is a clique, yz must be an edge. This shows that π is indeed a perfect elimination ordering. □

Now, we introduce a fundamental concept for exploring the structure of a graph. *Lexicographic breadth-first search* is an algorithm that, given a graph G, constructs a special breadth-first search ordering of the vertices of G. In the process of exploring the graph, the algorithm assigns to the vertices labels formed by (ordered) lists of numbers from $\{1, \ldots, n\}$, and using these labels, it decides which vertex to explore next. The algorithm always chooses an unprocessed vertex with lexicographically largest label among the unprocessed vertices (ties are broken arbitrarily), where lexicographic order is just the usual dictionary order, e.g., $9, 7, 6, 1 < 9, 8, 5$ and $6, 4, 3 < 6, 4, 3, 2$. The algorithm as just described is summarized below as Algorithm 1.1.

Algorithm 1.1: Lexicographic Breadth-First search.

Input: A graph G

Output: An elimination ordering π

```
1 set label(v) ← ∅ for all v ∈ V(G)
2 for i ← n downto 1 do
3     pick an unnumbered vertex v with lexicographically largest label
4     π(i) ← v /* the vertex v becomes numbered */
5     for each unnumbered w adjacent to v do
6         append i to label(w)
```

We have the following property of this algorithm.

Proposition 1.14. [38] *A graph G is chordal, if and only if, Algorithm 1.1 on G produces a perfect elimination ordering π of G.*

Now, it follows that one can decide in time $O(n+m)$ whether a given graph G is chordal just by computing an elimination ordering π using Algorithm 1.1 on G, and then testing whether π is a perfect elimination ordering by simply checking whether the neighbours of any vertex v that appear in π after v form a clique; both steps can easily be implemented in time $O(n+m)$ [38].

We remark that Lexicographic breadth-first search algorithm was originally introduced by Rose, Tarjan and Leuker in [57] as a simple linear time algorithm for recognizing chordal graphs (as we just described). It has since found numerous applications outside chordal graphs, for instance, efficient recognition of cographs (P_4-free graphs), P_4-sparse graphs, P_4-reducible graphs, AT-free graphs, interval graphs, unit interval graphs, and their powers [11].

We should mention that there exists yet another efficient algorithm by Tarjan [64] for recognition of chordal graph. *Maximum cardinality search* is an algorithm that numbers vertices of a graph from n to 1 where the next vertex to be numbered is one that is adjacent to the most numbered vertices (ties are broken arbitrarily). This algorithm also produces a perfect elimination ordering given a chordal graph, yet it should be pointed out that the orderings produced by Maximum cardinality search and the orderings produced by Lexicographic breadth-first search are not exactly the same, and moreover there are perfect elimination orderings that neither of these algorithms can produce.

We close by mentioning that recently a unified approach to graph search algorithms generalizing both the above algorithms has been discovered [12].

1.3 Related classes

A graph G is *perfect* if, for every induced subgraph H of G, the chromatic number $\chi(H)$ is equal to the clique number $\omega(H)$. The class of perfect graphs includes such classes of graphs as bipartite graphs, chordal graphs, cographs, comparability graphs, and their complements [38, 61]. It is an interesting and important class of graphs, and has been a focus of attention for more than a half of a century now, which is in part due to the result of Grötschel, Lovász and Schrijver [41] who gave a polynomial time algorithm for all the basic combinatorial problems (χ,α,ω) in perfect graphs. There are many interesting results known about perfect graphs. We mention here only the most notable ones conjectured by Berge [4], namely The Weak Perfect Graph Theorem proved by Lovász [18] and The Strong Perfect Graph Theorem proved by Chudnovsky, Robertson, Seymour and Thomas [10].

Theorem 1.15 (The Weak Perfect Graph Theorem). [18]
A graph is perfect, if and only if, its complement is also perfect.

Theorem 1.16 (The Strong Perfect Graph Theorem). [10]
A graph is perfect, if and only if, it has no induced odd cycle or its complement.

As we already remarked, some problems that are difficult in general like the graph colouring or the maximum clique problem, admit a polynomial time solution in perfect graphs. Unfortunately, the algorithms for these problems are not always combinatorial, and in addition, there are problems that are not tractable even in perfect graphs. It is therefore natural to ask in which restricted classes of perfect graphs these problems have nice combinatorial solutions and perhaps a polynomial solutions for problems intractable in perfect graphs. We already mentioned such a subclass, namely the class of chordal graphs in which many difficult problems admit even linear time algorithms.

In what follows, we briefly describe some other interesting examples of such classes, and summarize complexities of selected combinatorial problems in these classes at the end.

A graph G is *split*, if G can be partitioned into a clique and an independent set with no other restriction on the edges between the two. A graph G is *co-chordal*, if $\overline{G}$ is chordal. It can be seen that any split graph is also chordal, and since the complement of a split graph is also split, any split graph is also co-chordal. In fact, the converse is also true.

Theorem 1.17. [38] *A graph G is split, if and only if, G is both chordal and co-chordal.*

A subset of edges $F \subseteq E(G)$ of an undirected graph G is an *orientation* of G, if $F \cap F^{-1} = \emptyset$ and $F \cup F^{-1} = E(G)$, where F^{-1} is the set of edges vu such that $uv \in F$.

A graph G is a *comparability graph*, if there exists a transitive orientation of G, that is, an orientation F of G with $F^2 \subseteq F$ where $F^2 = \{ac \mid ab, bc \in F \text{ for some } b\}$. A graph G is *co-comparability*, if $\overline{G}$ is comparability. We remark that basic facts about comparability graphs go all the way back to the paper of Gallai [35].

For a permutation π of $\{1, \ldots, n\}$, let $G[\pi]$ be a graph with vertices $V(G[\pi]) = \{1, \ldots, n\}$ and edges $ij \in E(G[\pi]) \iff (i-j)\big(\pi^{-1}(i) - \pi^{-1}(j)\big) < 0$.

A graph G is called a *permutation graph*, if $G \cong G[\pi]$ for some permutation π. It can be shown that any permutation graph is also a comparability graph. Moreover, one can observe that the complement of a permutation graph $G = G[\pi]$ is the graph $G[\pi^R]$ where π^R is the reverse of π, that is, $\pi^R(i) = \pi(n+1-i)$ for all $1 \leq i \leq n$. Hence, $\overline{G}$ is also a permutation graph, and it follows that any permutation graph is also a co-comparability graph. In fact, again the converse (of both these statements) is also true.

Theorem 1.18. [38] *A graph G is permutation, if and only if, G is both comparability and co-comparability.*

A graph G is a *cograph*, if G can be constructed from single vertex graphs using the operations of join and union (or equivalently the operations of complement and union). This construction can be represented by a tree whose leaves are the vertices of G, and inner nodes are labeled either 0 or 1, denoting the operation of union or join respectively. We call this tree a *tree representation* of G. In addition, it can be shown [13] that for any cograph G, there exists a unique minimum size tree that represents the construction of G; such a tree is called the *cotree* of G. We remark that using the cotree of a cograph G, one can solve many graph problems on G efficiently and usually in linear time.

In addition, cographs can be also described in terms of minimal forbidden induced subgraphs. The following theorem is from [13].

Theorem 1.19. [13] *A graph G is a cograph, if and only if, G contains no induced P_4.*

Moreover, the authors in [13] show a multitude of equivalent definitions of cographs that independently appeared in the literature. In particular, it can be observed that cographs form a proper subclass of permutation graphs.

For a family of intervals $\mathcal{I} = \{I_1, \ldots, I_n\}$ on the real line, the intersection graph of $\mathcal{I}$ is the graph with vertices $\{v_1, \ldots, v_n\}$ and edges $v_i v_j \iff I_i \cap I_j \neq \emptyset$.

A graph G is an *interval graph*, if G is the intersection graph of a family of intervals on the real line. It is easy to see that an interval graph cannot contain an induced C_k with $k \geq 4$; thus, each interval graph is also a chordal graph. On the other hand, of any two disjoint intervals, one must lie before the other one, and hence, the complement of an interval graph is a comparability graph. The converse of this is also true.

Theorem 1.20. [38] *A graph G is interval, if and only if, G is both chordal and co-comparability.*

In fact, it is shown in [38], that a graph is an interval graph, if and only if, it is co-comparability and has no induced C_4. Additionally, by a result of Lekkerkerker and Boland [53], we have the following characterization of interval graphs. An *asteroidal triple* of a graph G is a triple of mutually non-adjacent vertices such that for any two vertices of the triple there exists a path in G that avoids the neighbourhood of the third vertex in the triple. We say that a graph G is *AT-free*, if G does not contain any asteroidal triple.

Theorem 1.21. [53] *A graph G is an interval graph, if and only if, G is chordal and contains no asteroidal triple (is AT-free).*

We remark that AT-free graphs form an interesting class of their own. Though not necessarily perfect, they nevertheless possess a structure useful for the design of efficient algorithms [61].

A graph G is a *circular-arc graph*, if G is the intersection graph of arcs of a circle. Circular-arc graphs are not necessarily perfect, e.g., C_5 is circular-arc, but not perfect.

We close this section by summarizing complexities of selected combinatorial graph problems in these graph classes. The results are presented in Table 1.1 and are taken from [61]. Here, n and m, as usual, refer to the number of vertices respectively edges of an input graph,

and n^α is the complexity of matrix multiplication. We remark that all problems with the exception of recognition assume that an appropriate representation of a graph is given, e.g., a set of intervals for an interval graph.

	Recognition	$\chi(G)$	$\alpha(G)$	$\gamma(G)$	HAM
All graphs	-	NPc	NPc	NPc	NPc
Perfect	$O(n^9)$	polynomial	polynomial	NPc	NPc
Chordal	$O(n+m)$	$O(n+m)$	$O(n+m)$	NPc	NPc
Split	$O(n+m)$	$O(n+m)$	$O(n+m)$	NPc	NPc
AT-free	$O(n^\alpha)$	open	$O(n^3)$	$O(n^6)$	open
Interval	$O(n+m)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Circular-arc	$O(n+m)$	NPc	$O(n)$	$O(n)$	$O(n^2 \log n)$
Comparability	$O(n^\alpha)$	$O(n+m)$	$O(n^2 m)$	NPc	NPc
Co-comparability	$O(n^\alpha)$	$O(n^2 m)$	$O(n+m)$	$O(nm^2)$	$O(n^3)$
Permutation	$O(n+m)$	$O(n \log\log n)$	$O(n \log\log n)$	$O(n)$	$O(n+m)$
Cographs	$O(n+m)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Table 1.1: A summary of complexities of selected graph problems.

1.4 Polynomial time cases

In this section, we mention three simple examples of GCOL problems solvable in general graphs in polynomial time, namely, the 2-colouring problem, the split partition problem, and the unipolar partition problem. In all these problems $k = 2$, and the classes $\mathcal{P}_1$ and $\mathcal{P}_2$ are induced hereditary and are characterized by exactly one forbidden induced subgraph, namely, K_2 for $\mathcal{P}_1 = \mathcal{P}_2$ for the 2-colouring problem, K_2 for $\mathcal{P}_1$ and $\overline{K}_2$ for $\mathcal{P}_2$ for the split partition problem, and $\overline{K}_2$ for $\mathcal{P}_1$ and P_3 for $\mathcal{P}_2$ for the unipolar partition problem. We remark that, by complementation, it will follow that the graphwise complements of these problems are also polynomially solvable. Additionally, we will also mention how these algorithms can be extended to solve the list versions of these problems. In all the cases, since $k = 2$, this is equivalent to allowing precoloured vertices of the input graph (that is, preassigned to either the set V_1 or V_2 of the partition we seek).

2-colouring

A polynomial time algorithm for this problem is probably the best known graph algorithm and probably one of the simplest ones. The algorithm performs a depth-first search on a given graph while labelling its vertices with labels 0 and 1. The first vertex is labeled 0 and its neighbours 1. Then each time a vertex v is visited, the neighbours of v are labeled with a label different from the label of v. After all vertices are processed, the graph is 2-colourable (or bipartite), if and only if, the labeling is a proper 2-colouring. The correctness of this algorithm is straightforward. The complexity is also simple to analyze, since each vertex is clearly processed in time $O(|N(v)|)$, and hence, we have $O(n + m)$ total running time. We summarize this in the following theorem.

Theorem 1.22. *There exists an $O(n + m)$ time algorithm for the 2-colouring problem.*

It can be immediately seen that this algorithm also allows precoloured vertices. Now, we discuss the complementary problem, that is, the problem of recognizing co-bipartite graphs (the complements of 2-colourable graphs). As remarked above, we can use the algorithm for 2-colourability to solve this problem by first, computing the complement of a given graph, and then running the above 2-colouring algorithm on the complement. This can take as much as $\Theta(n^2)$ time, since computing the complement takes $\Theta(n^2)$. Fortunately, it can be observed that if a graph is co-bipartite, then it must have at least $\binom{t}{2} + \binom{n-t}{2}$ edges for some t,

which is at least $n^2/4 - n/2$. Hence, if the input graph has less than $n^2/4 - n/2$ edges, we reject, and otherwise, we compute the complement and run the 2-colouring algorithm on the complement. The former can be clearly accomplished in time $O(m)$, whereas in the latter, we have $m \geq n^2/2 + n/2$, and hence, $O(n^2) = O(n + m)$. This gives the following theorem.

Theorem 1.23. *There exists an $O(n+m)$ time algorithm for recognizing co-bipartite graphs.*

Again, this algorithm can be seen to also allow precolouring. Finally, we remark that bipartite graphs and also co-bipartite graphs form induced hereditary classes, where the former is characterized by all odd chordless cycles, and the latter by their complements.

Split partition

The split partition problem is a problem of partitioning a graph into a clique and an independent set. The first linear time algorithm is due to Hammer and Simeone [38]. Their algorithm is based on degree sequences of graphs.

Theorem 1.24. *[38] Let G be a graph with vertices $v_1, \ldots, v_n$ such that $\deg(v_1) \geq \deg(v_2) \geq \ldots \geq \deg(v_n)$, and let $M = \max\{i \mid \deg(v_i) \geq i-1\}$. Then G is a split graph, if and only if,*

$$\sum_{i=1}^{M} \deg(v_i) = M(M-1) + \sum_{i=M+1}^{n} \deg(v_i).$$

Furthermore, if this is the case, then $\omega(G) = M$.

This theorem implies that deciding whether G is split can be done by computing the degrees of the vertices of G, then sorting the vertices according to their degrees, computing the value M, and verifying the above condition. Each of these steps can be easily accomplished in time $O(n + m)$ (note that we can sort the degrees in time and space $O(n)$). Hence, this proves the following theorem.

Theorem 1.25. *There exists an $O(n + m)$ time algorithm for recognizing split graphs.*

On the other hand, it is not clear how to extend this algorithm to allow precoloured vertices. This was resolved much later by Hell et al. in [43]. In fact, they solve a more general problem of recognizing chordal (k, ℓ)*-graphs*, that is, chordal graphs which are partitionable into k independent sets and ℓ cliques. As a special case they obtain a simple greedy algorithm for recognizing split graphs which also allows precolouring.

We now briefly explain their algorithm. The algorithm first tests whether the input graph G is chordal, since any split graph is necessarily also chordal [38]. If yes, it constructs a perfect elimination ordering $\pi = v_1, \ldots, v_n$ of G, and finds a largest index j such that $G[v_1, \ldots, v_{j-1}]$ is an independent set. Then it assigns $v_1, \ldots, v_{j-1}$ to the independent set and v_j to the clique, and processes the remaining vertices v_i of G in the order of π as follows. If v_i is adjacent to v_j, then it assigns v_i to the clique, otherwise, if v_i is not adjacent to any of the vertices assigned so far to the independent set, it assigns v_i to the independent set. If even that is not possible, the algorithm declares G not split. After all vertices are processed, the algorithm returns the clique and the independent set that it constructed.

We now discuss the correctness of this algorithm. First, it can be seen that if the algorithm does not declare G not split, it returns a valid split partition. This follows, since each time a vertex v_i is assigned to the independent set, it is not adjacent to all other vertices of this set, and each time v_i is assigned to the clique, it is adjacent to v_j, which is the first vertex of the clique in π, and hence, v_i is also adjacent to all other vertices of the clique. (Any two vertices adjacent to v_j which appear in π after v_j must be necessarily neighbours, because π is a perfect elimination ordering.)

Now, if the algorithm declares G not split, then we have a vertex v_i which is not adjacent to v_j, but it is adjacent to some vertex v_k of the independent set. Also, since the independent set $v_1, \ldots, v_{j-1}$ is maximal possible, the vertex v_j must also be adjacent to a vertex v_ℓ with $\ell < j$. Now, observe that v_j is not adjacent to v_k. Otherwise, if $v_k v_j$ is an edge of G and $k < j$, then we have edges $v_k v_j$ and $v_k v_i$ with $k < j < i$, and hence, there must be an edge $v_j v_i$, since π is a perfect elimination ordering. Also, if $v_k v_j$ is an edge and $j < k$, then we have that the algorithm, when processing v_k, would put v_k into the clique instead. Similarly, there is no edge between v_ℓ and v_i and no edge between v_ℓ and v_k, since otherwise we would have $v_j v_i$ and $v_j v_k$ in the edges set, respectively. This gives us that the vertices v_i, v_j, v_k, v_ℓ induce a copy of $2K_2$ (two independent edges), which is known to be not split, and hence, G is not split.

Altogether, if G is not split, then either G is not chordal and we have a chordless cycle C_k with $k \geq 4$ in G, or G contains $2K_2$. This proves the well known result that the minimal forbidden induced subgraphs of split graphs are C_4, C_5 and $2K_2$. (Observe that any chordless cycle with six or more vertices already contains an induced $2K_2$.)

Theorem 1.26. *A graph G is split, if and only if, G does not contain an induced C_4, C_5, and $2K_2$.*

Now, it is not difficult to see how the above algorithm can be modified to also allow precoloured vertices. Let S_0 and C_0 denote the precoloured vertices of G, that is, vertices preassigned to the independent set and the clique, respectively. The modified algorithm works almost exactly as the original algorithm, but instead of finding largest j such that $G[v_1, \ldots, v_{j-1}]$ is an independent set, it finds largest j such that $G[v_1, \ldots, v_{j-1}] \cup S_0$ is an independent set. Also, when processing a vertex v_i, it assigns v_i to the clique, if v_i is adjacent to v_j, and $v_i \notin S_0$, and otherwise, it assigns v_i to the independent set, if $v_i \notin C_0$, and v_i is not adjacent to all vertices assigned (so far) to the independent set and also not adjacent to all vertices in S_0. Again, if neither of these steps is possible, the algorithm declares G not split partitionable with respect to the precolouring. The correctness of this (modified) algorithm follows similarly as for the original algorithm.

Theorem 1.27. *There exists an $O(n + m)$ time algorithm to decide, for a graph G with precoloured vertices, whether there exists a split partition of G that extends the precolouring.*

In addition, the analysis of this algorithm gives us all precoloured minimal forbidden induced subgraphs for split graphs. (A more formal treatment of this notion is given in Chapter 4.) These are the graphs depicted in Figure 1.1; the vertices marked with {b} are the vertices preassigned to the clique, the vertices marked with {r} are the vertices preassigned to the independent set, and the dotted lines represent non-edges.

Theorem 1.28. *A graph G with precoloured vertices admits a split partition that extends the precolouring, if and only if, G does not contain any of the configurations in Figure 1.1 as an induced subgraph.*

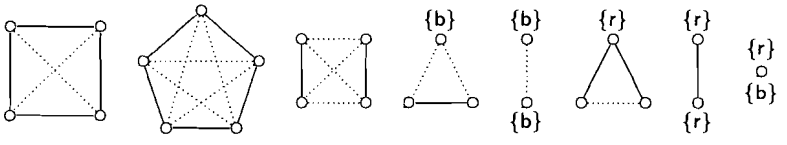

Figure 1.1: All minimal precoloured forbidden induced subgraphs of split graphs.

Finally, since the complement of a split graph is also split, we use the same algorithm for the complementary problem.

Unipolar partition

In this final section, we describe a polynomial time algorithm for the unipolar partition problem in general graphs. Recall, that a partition of the vertex set of G into sets V_1 and V_2 is unipolar, if V_1 induces a clique, and V_2 induces a P_3-free subgraph in G. (Note that a graph is P_3-free, if and only if, it is a disjoint union of cliques.) The first polynomial algorithm for this problem is reportedly due to [67], however, we were unable to obtain the aforementioned paper, and hence, we describe here our solution to this problem.

First, we need to mention a useful property of elimination orderings and their fill-in (see page 7). A *minimal ordering* π of a graph G with fill-in F_π is an elimination ordering of G such that there is no other elimination ordering σ of G with $F_\sigma \subsetneq F_\pi$. It is known that a minimal ordering of a graph can be found in time $O(nm)$ [57].

The following lemma is taken from [66].

Lemma 1.29. [66] *Let π be a minimal ordering of a graph G, and let C be a clique separator of G. Then no edge in F_π joins vertices in different connected components of $G - C$.*

In fact, it is easy to observe that this lemma is true even if C is not a separator.

Proposition 1.30. *Let π be a minimal ordering of G. Then G is unipolar, if and only if, there exists a maximal clique M of G_π and a connected component K of $G - M$ (or $K = \emptyset$) such that*

(i) $G - M$ is a disjoint union of cliques, and

(ii) there exists co-bipartite partition $X \cup Y$ of $G[M \cup K]$ such that X contains all vertices of $M \cup K$ which have neighbours in $G - M - K$.

Proof. First, let G be unipolar, and let $B \cup D$ be a unipolar partition of G, where B induces a clique, and D induces a P_3-free subgraph in G. Since G is a subgraph of G_π, there must exist a maximal clique M of G_π which completely contains B. Since B is a clique of G, by Lemma 1.29 (and the remark below the lemma), there are no edges in G_π that join two connected components of $G - B$. Hence, since M is a clique in G_π which contains B, there exists at most one connected component C of $G - B$ having vertices in M. (In case no such component exists, we let $C = \emptyset$.) Now, we let $K = C \setminus B$ and $T = C \cap M$. Hence, $M = B \cup T$ and $T \subseteq C$. We observe that K is a connected component of $G - M$ (or $K = \emptyset$), and that $G - M$ is a disjoint union of cliques, since $V(G - M) \subseteq D$.

Now, let $S \subseteq M$ be the vertices of $M \cup K$ which have neighbours in $G-M-K$. Since C is a connected component of $G-B$, no vertex of C is has a neighbour in $G-B-C = G-M-K$, and we must have $S \cap T = \emptyset$, and hence, $S \subseteq B$. It follows that there exists a co-bipartitie partition $X \cup Y$ of $G[M \cup K]$ with $X \subseteq S$, since we can take $X = B$ and $Y = C$.

Conversely, let $X \cup Y$ be a co-bipartite partition of $G[M \cup K]$ with $S \subseteq X$. Clearly, both X and Y induce cliques in G, and the vertices of Y are not adjacent to the vertices of $G - M - K$, because $S \cap Y = \emptyset$. Also, by (i), $G - M$ is a disjoint union of cliques. Hence, it follows that $B \cup D$, where $B = X$ and $D = V(G) \setminus X$, is a unipolar partition of G. □

Now, the algorithm for recognizing unipolar graphs follows. Given a graph G, we first compute the connected components of G. Clearly, G is unipolar, if and only if, some connected component of G is unipolar, and all other connected components are cliques. Hence, if G contains two connected components which are not cliques, we reject G. Otherwise, we let H be a connected component of G such that $G - H$ is a disjoint union of cliques. Next, we obtain a minimal ordering π of H, construct the graph H_π, and enumerate all maximal cliques of H_π. For each maximal clique M of H_π, we first test that $H - M$ is a disjoint union of cliques, and then for each connected component K of $H - M$ (and also for $K = \emptyset$), we determine the set S of all vertices of M having neighbours in $H - M - K$. After that, we precolour the vertices of S with 0 and run the co-bipartite recognition algorithm on $G[M \cup K]$. If successful, we obtain cliques X and Y which partition $G[M \cup K]$ such that $S \subseteq X$, and we return the partition $X \cup V(G) \setminus X$. If any of these tests fails, we choose another M or K. If the tests fail for all M and K, we declare G not unipolar.

The correctness of the above algorithm follows directly from Proposition 1.30. We now discuss its complexity. It is known that for any connected graph H, a minimal ordering π of H and the graph H_π can be both obtained in time $O(nm)$ [57]. Also, computing connected components and testing whether a graph is a disjoint union of cliques can be each accomplished in time $O(n + m)$. Observe that H_π is a graph on the vertices of H. Therefore, enumerating the maximal cliques M of H_π takes $O(n^2)$ time. Now, for each clique M, computing the connected components of $H - M$ and testing whether $H - M$ is a disjoint union of cliques, both takes $O(n + m)$ time. Also, computing the set S for some choice of K, and the co-bipartite test (by Theorem 1.23), each take $O(n + m)$ time. Since there can be at most n connected components of $H - M$, and there are at most n cliques M in G_π, the total running time is clearly $O(n^2 m)$. We summarize this in the following theorem.

Theorem 1.31. *There exists an $O(n^2m)$ time algorithm for recognizing unipolar graphs.*

Now, it is not difficult to adapt the above algorithm for precolouring. Let B_0 and D_0 be the vertices of G which are preassigned to the clique part, respectively, to the P_3-free part of the unipolar partition. Any time we choose a maximal clique M, if $G - M$ contains a vertex from B_0, we reject such choice of M. Also, in the co-bipartite test, we in addition precolour by 0 the vertices from B_0 and precolour by 1 the vertices from D_0. It follows that for any partition $B \cup D$ returned by this (modified) algorithm, no vertex of D_0 belongs to B, since the vertices of D_0 are precoloured by 1 in the co-bipartite test, and hence, for the co-bipartite partition $X \cup Y$ we obtain, the vertices of D_0 in M are assigned to Y, and $B = X$. The same argument also gives that no vertex of B_0 in M belongs to D. Finally, no vertex of B_0 in $G - M$ belongs to D, since $B \subseteq M$, and $G - M$ cannot contain a vertex from B_0, because we would reject such M. On the other hand, if $B \cup D$ is a unipolar partition of G such that $B_0 \subseteq B$ and $D_0 \subseteq D$, it can be seen that for a maximal clique M of G_π which contains B, we have that $G - M$ contains no vertex from B_0. Also, by Proposition 1.30, there must exist a connected component K of $G - M$ (or $K = \emptyset$) with $M \cup K = B \cup C$, where C is a connected component of $G - B$ (or $C = \emptyset$). Hence, since $B_0 \subseteq B$, and no vertex of D_0 is in B, because $B \cup D$ is a partition of the vertex set of G, we obtain that $B \cup C$ respects the precolouring in the co-bipartite test, which proves that the co-bipartite test will succeed for this choice of K. Hence, the (modified) algorithm is correct.

Theorem 1.32. *There exists an $O(n^2m)$ algorithm to decide, for a graph G with precoloured vertices, whether there exists a unipolar partition of G that extends the precolouring.*

We close by mentioning that, at the time of writing, a complete minimal forbidden induced subgraph characterization of unipolar graphs is not known. However, by a result of [34], chordal unipolar graphs are precisely those chordal graphs which do not contain $2P_3$ (two copies of P_3 with no edges between the copies) as an induced subgraph.

Chapter 2

Tools

2.1 Matrix Partitions

In this section, we introduce and discuss matrix partition problems, which form a class of interesting graph problems related to the problem of GCOL. In particular, the techniques and algorithms presented here will become useful in later chapters.

Let M be a $k \times k$ symmetric matrix with entries from $\{0, 1, *\}$. An *M-matrix partition* (or just *M-partition*) of a graph G is a partition of the vertices $V(G)$ into sets $V_1, \ldots, V_k$ such that for any $i, j \in \{1 \ldots k\}$ and vertices $x \in V_i$ and $y \in V_j$, if $M_{i,j} = 1$ and $x \neq y$, then we have $xy \in E(G)$, and if $M_{i,j} = 0$, then we have $xy \notin E(G)$. For a fixed $k \times k$ matrix M, the *M-partition problem* is a problem of deciding, given a graph G, whether G admits an M-partition. Similarly, the *list M-partition problem* is a problem of deciding, given a graph G with lists $\ell(v) \subseteq \{1 \ldots k\}$ for all $v \in V(G)$, whether G admits an M-partition $V_1, \ldots, V_k$ such that for all $v \in V(G)$ and $i \in \{1 \ldots k\}$, if $v \in V_i$, then $i \in \ell(v)$.

Matrix partition problems are a natural generalization of colouring problems. They are simpler than GCOL problems, because each part of the partition can only be either a clique, an independent set, or a graph with no restriction. However, using the off-diagonal entries, we may in addition specify connections between the parts of the partition, that is, all edges, no edges, or unrestricted, which makes this problem different from GCOL. On the other hand, a number of special cases of the GCOL problem are expressible as matrix partition problems, which will allow us to use results about matrix partitions to solve these problems efficiently. (This connection is further discussed in Chapter 7.)

We now briefly discuss the complexity of matrix partition problems. It is known, by a

result of [9], that if the size of the matrix M is at most four, and it is not one of the first two matrices in Figure 2.1, then the list M-partition problem is polynomial time solvable or NP-complete. Namely, it is polynomial time solvable, unless M or its complement is the third or the fourth matrix in Figure 2.1, or contains a submatrix corresponding to the 3-colouring problem, or the stable cutset problem, in which cases the problem is NP-complete.

$$\begin{pmatrix} 0 & * & 0 & * \\ * & 0 & * & * \\ 0 & * & * & * \\ * & * & * & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & * & 1 & * \\ * & 1 & * & * \\ 1 & * & * & * \\ * & * & * & 0 \end{pmatrix} \quad \begin{pmatrix} * & * & 0 & * \\ * & * & * & 0 \\ 0 & * & * & * \\ * & 0 & * & * \end{pmatrix} \quad \begin{pmatrix} * & * & 0 & 0 \\ * & 0 & * & 0 \\ 0 & * & 0 & * \\ 0 & 0 & * & * \end{pmatrix}$$

Figure 2.1: The stubborn problem, its complement, and two NP-complete cases.

For larger matrices M, the complexity of the (list) M-partition problem is largely unknown. A notable exception are the problems solvable by the following *sparse-dense partition* algorithm introduced in [27].

Let $\mathcal{S}$ and $\mathcal{D}$ be two induced hereditary classes of graphs, and let c be a constant such that any graph in $\mathcal{S} \cap \mathcal{D}$ has at most c vertices. We can think of $\mathcal{S}$ as a sparse class and of $\mathcal{D}$ as a dense class. A *sparse-dense* partition of a graph G is a partition of $V(G)$ into two sets $V_{\mathcal{S}}, V_{\mathcal{D}}$ such that $G[V_{\mathcal{S}}] \in \mathcal{S}$ and $G[V_{\mathcal{D}}] \in \mathcal{D}$. The following theorem from [27] characterizes all sparse-dense partitions of graphs.

Theorem 2.1. [27] *Any graph G on n vertices has at most n^{2c} sparse-dense partitions. Moreover, all sparse-dense partitions of G can be enumerated in time $O(n^{2c+2}T(n))$, where $T(n)$ is the time needed to recognize an n-vertex graph in $\mathcal{S}$ or an n-vertex graph in $\mathcal{D}$.*

(Note that we have already mentioned a simpler version of this theorem as Theorem 1.3.)

On the other hand, in chordal graphs, large classes of matrices are solvable in polynomial time. The following theorem is proved in [28].

Theorem 2.2. [28] *Let M be a $k \times k$ matrix with entries $\{0, 1, *\}$. If all diagonal entries of M are zero, or all diagonal entries of M are one, then the chordal list M-partition problem can be solved in time $O(nk^3(4k)^k)$, or $O(n^{2k+1})$, respectively.*

In addition, this theorem, when combined with the sparse-dense algorithm, gives a more general result [28], which is described in the theorem below. We say that a square matrix M of size $k+\ell$ is an (A, B, C)-block matrix, if M is of the form $M = \left(\begin{array}{c|c} A & C^T \\ \hline C & B \end{array}\right)$, where A is a

symmetric $k \times k$ matrix with all zeroes on the main diagonal, B is a symmetric $\ell \times \ell$ matrix with all ones on the diagonal, and C is a $k \times \ell$ matrix. It is easy to see that any symmetric matrix M with no $*$ on the main diagonal can be transformed into an (A, B, C)-block matrix M' by simultaneously permuting rows and columns. Moreover, the M'-partition and the M-partition problems are clearly equivalent. A matrix is said to be *crossed*, if each non-$*$ entry in the matrix belongs to a row or a column of non-$*$ entries.

Theorem 2.3. [28] *Suppose that M is an (A, B, C)-block matrix. If C is crossed, then the chordal list M-partition problem can be solved in polynomial time.*

2.2 Dynamic programming

In this section, we introduce a general model for solving combinatorial problems in graphs using dynamic programming on a tree-like structure associated with the graph.

Recall that a tree decomposition of a (connected) graph G is a pair (T, X) where T is a tree and X is a collection of sets $X(u) \subseteq V(G)$, for all $u \in V(T)$ with the properties, (i) for each edge $xy \in E(G)$, there exists $u \in V(T)$ with $x, y \in X(u)$, and (ii) for each $x \in V(G)$, the vertices $u \in V(T)$ with $x \in X(u)$ induce a connected subgraph of T. We remark that we include in our consideration all tree decompositions (not just those of minimum width).

We shall consider any tree decomposition (T, X) rooted as some vertex $r \in V(T)$. For a vertex $v \in V(T)$, we denote by T_v the subtree of T rooted at v. For a subset W of the vertices of $V(T)$, we denote by $\mathcal{C}(W)$ the set $\bigcup_{w \in W} X(w)$. In particular, $G_v = G[\mathcal{C}(T_v)]$.

Tree decompositions play an important role in efficient algorithmic solutions for graph problems. In particular, for the graphs that admit a tree decomposition of a constant width, many difficult graph problems are solvable in polynomial time. (We discuss this case in detail in Section 2.5.) On the other hand, there are classes of graphs with unbounded treewidth, which nevertheless admit special tree decompositions also allowing efficient solutions to difficult graph problems. An example of such class is the class of chordal graphs; each chordal graph has a clique-tree, which is a tree decomposition whose each bag induces a clique. In what follows, we shall investigate some properties of tree decompositions, which make particular graph problems efficiently solvable.

As a first problem, we mention the well-known algorithm for the proper k-colouring problem. Let G be a graph, and let (T, X) be a (rooted) tree decomposition of G. The

algorithm processes the vertices of T in a bottom-up order, and for each vertex v, it computes the set $C(v)$ of all possible proper k-colourings of $G[X(v)]$, which can be extended to a proper k-colouring of G_v. For any leaf v, the set $C(v)$ is just the set of all proper k-colourings of $G[X(v)]$. For any other node v, the set $C(v)$ is the set of proper k-colouring c of $G[X(v)]$ such that, for each child w of v, there exists a k-colouring c_w in $C(w)$ such that c and c_w match on the vertices of $X(v) \cap X(w)$. After all vertices are processed, G has a proper k-colouring, if and only if, the set $C(r)$, for the root r of T, is non-empty. The correctness of this algorithm is easy to see. Now, we discuss its complexity. Let t be the size of a maximum bag $X(u)$ for $u \in V(T)$, and let $N = |V(T)|$. Clearly, there are at most k^t possible colourings of any bag $X(u)$ with k colours, and hence, at most k^t possible proper k-colourings. For each such colouring, we search for matching colourings in the sets $C(w)$ for each child w, which takes $O(t)$ time (after some preprocessing of $C(w)$). Thus, processing a single vertex of T takes $O(t \cdot k^t \cdot \deg(v))$. Altogether, for all vertices of T, we have $\sum_{v \in V(T)} O(t \cdot k^t \cdot \deg(v)) = O(|E(T)| \cdot t \cdot k^t) = O(N \cdot t \cdot k^t)$. Hence, if t is a constant, then the complexity is linear in N. (Note that for $k \geq t$, the graph G is always k-colourable.) Now, it follows that for graphs G, whose treewidth is bounded above by a constant, the k-colouring problem has a linear time solution for any k. This is in contrast to the fact that in general the k-colouring problem in NP-complete.

Now, we modify the above algorithm so it actually computes the chromatic number of G. We do it as follows. Instead of considering only proper k-colourings of each bag $X(v)$, we consider all proper n-colourings, but, in addition, for each such colouring c, we store the minimum number of colours $m(c)$ we need to extend this colouring to G_v. Each time we consider a k-colouring c of $G[X(v)]$ and matching colourings c_w of $G[X(w)]$ for all children w of v, we compute the maximum m of $m(c_w)$ among the children w of v, and the minimum number of colours $m(c)$ for c is just the maximum of k and m. This follows from the fact that, since the colourings are matching, we do not need more colours than what we already use. The chromatic number of G is then $m(c)$ for a colouring c from $C(r)$, where r is the root of T, such that $m(c)$ is smallest possible. The correctness is straightforward. We now discuss the complexity of this algorithm. Again, let t be the size of a maximum bag $X(u)$ for $u \in V(T)$, and $N = |V(T)|$. Observe that for each bag $X(u)$, we never need to use more than $|X(u)| \leq t$ colours when colouring $G[X(u)]$. Hence, using the analysis from the above algorithm, we obtain that the complexity of this algorithm is $O(N \cdot t \cdot t^t)$, which is again linear in N, if t is a constant.

Now, suppose that t is not a constant, in which case, both the above algorithms are inefficient. However, this may not be so if we know something more about the tree decomposition we use. For instance, if the graph $G[X(u)]$, for each $u \in V(T)$, has only a small number of possible different proper colourings, say at most α, then the running time is clearly $O(N \cdot t \cdot \alpha)$. Even better, if we know that each $G[X(u)]$ has at most α possible non-isomorphic colourings, that is, colourings such that no one can be obtained from another by renaming the colours. Again, the running time is $O(N \cdot t \cdot \alpha)$. Things also become simpler, if for each vertex v, and each child w of v, the set $X(v) \cap X(w)$ induces a clique. Then any proper colouring c_w of $G[X(w)]$ will match any proper colouring c of $G[X(v)]$ (up to renaming the colours). Then all we have to do is to test, for each $v \in V(T)$, whether there exists a proper k-colouring of $G[X(v)]$ for the former algorithm, or determine the chromatic number of $G[X(v)]$ for the latter algorithm. Finally, if G is chordal and (T, X) is a clique-tree of G, then we have all of the above, since each bag $X(u)$ of T is a clique, and the running time is $O(N \cdot t)$ for both algorithms, since each clique has only one non-isomorphic proper colouring.

From the above examples, the reader should be able to get sufficiently acquainted with the general idea of dynamic programming on tree decomposition. Therefore, we can now proceed to defining this idea formally. Before going further, we would like to caution the reader that the following definitions are not meant to give a substantially different perspective on known dynamic programming methods, but rather, are an attempt to capture the common properties of these methods on tree decompositions of graphs, to the extent they are used in our algorithms in the later chapters.

Dynamic programming on graphs (Formal definition)

Let $\mathbb{U}$ denote the set of all graphs. An *(induced hereditary) graph problem* Π is a tuple $(\Sigma, \rho, \pi, \sigma, \mu, \eta)$ such that

(i) Σ is a (finite) alphabet,

(ii) $\rho : \mathbb{U} \to 2^{\Sigma^*}$ maps graphs $G \in \mathbb{U}$ to *all* (feasible and infeasible) solutions to Π for G,

(iii) $\mu : \Sigma^* \to \mathbb{R}$ is a *valuation* function which assigns real numbers to the solutions to Π,

(iv) $\pi : \mathbb{U} \to 2^{\Sigma^*}$ maps graphs $G \in \mathbb{U}$ to the *feasible* solutions to Π for G ($\pi(G) \subseteq \rho(G)$),

(v) $\sigma : \mathbb{U} \times \Sigma^* \to \Sigma^*$ is a *subgraph* mapping which for each $G \in \mathbb{U}$, each induced subgraph H of G, and each $s \in \rho(G)$, satisfies

(a) $\sigma(H,s) \in \rho(H)$, and

(b) if $s \in \pi(G)$, then also $\sigma(H,s) \in \pi(H)$, and $\mu(\sigma(H,s)) \leq \mu(s)$;

(vi) $\eta : \Sigma^* \times \Sigma^* \to \Sigma^*$ is an *exchange* mapping which for each $G \in \mathbb{U}$, each induced subgraph H of G, each $s_G \in \rho(G)$, and each $s_H \in \rho(H)$, satisfies

(a) $\eta(s_G, s_H) \in \rho(G)$, and

(b) $\sigma(H, \eta(s_G, s_H)) = s_H$

The *decision problem* for Π is the problem of deciding, given a graph G, whether $\pi(G) \neq \emptyset$. The *search problem* for Π is the problem of finding $s \in \pi(G)$ for a given graph G, if s exists. An *optimization problem* for Π is the problem of finding $s_{opt} \in \pi(G)$ (if it exists), for a given graph G, such that $\mu(s_{opt}) = \min_{s \in \pi(G)} \mu(s)$.

A *dynamic programming scheme* $\mathcal{S}$ for Π is a tuple $(\Gamma, \mathcal{A}, \varphi, \approx)$ such that

(i) Γ is a (finite) alphabet,

(ii) $\approx$ is an equivalence relation on Γ^*,

(iii) $\varphi : \mathbb{U} \times \Sigma^* \to \Gamma^*$ is a *fingerprint* mapping,

(iv) for each $G \in \mathbb{U}$, each rooted tree decomposition (T, X) of G, each $v \in V(T)$ with a parent u, and each $s_G \in \pi(G)$, $s_v \in \pi(G_v)$ the following *exchange condition* is satisfied;

if $\varphi\Big(G\big[X(u) \cap X(v)\big], \sigma(G_v, s_G)\Big) \approx \varphi\Big(G\big[X(u) \cap X(v)\big], s_v\Big)$, then

(a) $\eta(s_G, s_v) \in \pi(G)$, and

(b) if $\mu(s_v) \leq \mu\big(\sigma(G_v, s_G)\big)$, then $\mu\big(\eta(s_G, s_v)\big) \leq \mu(s_G)$,

(v) if, for each vertex v with a parent u, we denote by $F(v)$ the pairs (f, s_v) such that $s_v \in \pi(G_v)$, and $f = \varphi\big(G\big[X(u) \cap X(v)\big], s_v\big)$, then $\mathcal{A}$ is an algorithm which, given

(a) a vertex v of a (rooted) tree decomposition (T, X) of G,

(b) the sets $F(z)$ for all $z \in V(T_v) \setminus \{v\}$, and

(c) $f \in \Gamma^*$ if v is not the root of T,

returns $s_v \in \pi(G_v)$ such that

(a) $\mu(s_v) = \min\{\mu(s) \mid s \in \pi(G_v)\}$, if v is the root of T, or

(b) $\mu(s_v) = \min\{\mu(s) \mid (f, s) \in F(v)\}$, if v is not the root of T,

or announces that such s_v does not exists.

We remark that the above exchange condition justifies the behaviour of the algorithm $\mathcal{A}$, meaning that, for each fingerprint f, we only need to keep one solution s_v which has

the fingerprint f on $G[X(u) \cap X(v)]$, since all other solutions with the same fingerprint are equivalent by the exchange condition. Also, we explain why the (partial) solution $s_v \in \pi(G_v)$ for f can be taken with the smallest value. If there exists an solution $s' \in \pi(G)$ whose fingerprint $f' = \varphi(G[X(u) \cap X(v)], s'_v)$, where $s'_v = \sigma(G_v, s')$, is equal to f, that is, $f \approx f'$, then, by the exchange condition, $s = \eta(s', s_v) \in \pi(G)$, and by the properties of the subgraph function, $s'_v \in \pi(G_v)$. Also, the fingerprint for s on $G[X(u) \cap X(v)]$ is necessarily f, since, by the properties of the exchange function, $\sigma(G_v, s) = \sigma(G_v, \eta(s', s_v)) = s_v$, and hence, $\varphi(G[X(u) \cap X(v)], \sigma(G_v, s)) = \varphi(G[X(u) \cap X(v)], s_v) = f$. Therefore, if s_v has smallest possible value for f, then $\mu(s_v) \leq \mu(s'_v)$, and hence, by the exchange condition, $\mu(s) \leq \mu(s')$. This shows that s is at least as good solution as s', and also proves that any optimal solution for G gives rise to optimal solutions on all subgraphs G_v, which justifies the choice of s_v.

Finally, we remark that the above definitions are tailored to capture the problems which we investigate here, and may be too specific to capture other graph algorithms which use dynamic programming on a tree decomposition of a graph.

Now, we explain how to solve the optimization problem for Π, using a dynamic programming scheme $\mathcal{S}$ for Π; this will also imply algorithms for both the decision and the search problem. First, for a given graph G, we construct a (not necessarily optimal) tree decomposition (T, X) of G. Then, for each leaf v of T, we construct the set $F(v)$ as follows. We enumerate all possible fingerprints f, and then add (f, s_v) into $F(v)$, if the algorithm $\mathcal{A}$ on v and f returns s_v. After that, for each vertex v in T, for which the sets $F(z)$ of all descendants z of v have been computed, we compute the set $F(v)$ by, again, enumerating all fingerprints f and running $\mathcal{A}$ on v, f, and the sets $F(z)$. Finally, for the root of T, we simply run $\mathcal{A}$ on the sets $F(z)$, and the answer we obtain will be the answer for G.

It can be seen from the above description that the complexity of this algorithm is $O(N \times M \times h(W, M))$, where N is the number of vertices in T, M is the number of possible fingerprints, W is the size of a largest set $X(v)$, and $h(W, M)$ is the running time of $\mathcal{A}$.

Next, we illustrate the above definitions on the example from the beginning of this section. Namely, for the graph colouring problem Π, the tuple $(\Sigma, \rho, \pi, \sigma, \mu, \eta)$ is as follows.

(i) the alphabet Σ is $\{0, 1\}$,

(ii) $\rho(G)$ are all possible (not necessarily proper) $|V(G)|$-colourings of G,

(iii) $\mu(c)$ is the number of colours used in the colouring $c \in \rho(G)$,

(iv) $\pi(G)$ are all proper $|V(G)|$-colourings of G,

(v) $\sigma(H, c)$ is the restriction of c onto H, where $c \in \rho(G)$, and

(vi) $\eta(c_G, c_H)$ is the colouring of G obtained from c_G by recolouring the vertices of H by c_H, where $c_G \in \rho(G)$, $c_H \in \rho(H)$, and H is an induced subgraph of G,

The dynamic programming scheme $\mathcal{S}$ for the graph colouring problem Π is as follows.

(i) the alphabet Γ is again $\{0, 1\}$,

(ii) $\varphi(G, c)$ is the colouring f of G obtained by restricting c onto the vertices of G,

(iii) we have $f \approx f'$, if and only if, both f and f' are the same colourings (up to renaming colours), and

(iv) the algorithm $\mathcal{A}$, given v, (possibly) f, and fingerprints $F(z)$ for $z \in V(T_v) \setminus \{v\}$, considers all possible proper colourings c of $G[X(v)]$ (up to renaming colours) which are extensions of f (if f is provided as input), and for each such colouring c, if, for each child w or v, there is a pair (f_w, c_w) in $F(w)$ such that f_w is a restriction of c onto $X(w) \cap X(v)$ (again, up to renaming colours), and $\mu(c_w)$ is smallest possible, then the algorithm combines c and the colourings c_w into a colouring c_v of G_v, and returns c_v, otherwise announces that c_v does not exist.

It is not difficult to verify that the above satisfies the definition of Π and $\mathcal{S}$, and, in particular, it can be seen that the exchange condition is satisfied, since for any vertex v with a parent u, the set $X(u) \cap X(v)$ is a separator of G, and hence, if proper colourings c and c_v of G and G_v, respectively, are the same on $X(u) \cap X(v)$, we can colour G with c and then recolour the vertices of G_v by c_v, and we must obtain a proper colouring c', because there are no edges between different connected components of $G - \big(X(u) \cap X(v)\big)$ and $G_v - \big(X(u) \cap X(v)\big)$ is one (or more) of those components. Also, the number of colours used in c' is never more than the maximum of the number of colours used in c and the number of colours used in c_v, which shows the second part of the exchange condition.

Now, using the scheme $\mathcal{S}$, we already know that we can solve the optimization problem for Π using dynamic programming on a tree decomposition. Let us look at the complexity of this algorithm. Again, let (T, X) be a tree decomposition of a graph G of width $t - 1$, let $N = |V(T)|$, and let α denote the maximum number of non-isomorphic proper colourings of $G[X(u)]$ for $u \in V(T)$. Recall that the fingerprints in the scheme $\mathcal{S}$ are the proper colourings of $G[X(u) \cap X(v)]$ where $uv \in E(T)$. Hence, there are at most α possible different fingerprints, and it can be seen that the running time of the algorithm $\mathcal{A}$ on any

node $v \in V(T)$ altogether for all fingerprints is $O\big(\alpha \cdot t \cdot \deg(v)\big)$. This gives us that the running time of the algorithm for the graph colouring problem is $O\Big(\sum_v \alpha \cdot t \cdot \deg(v)\Big) = O\big(N \cdot t \cdot \alpha\big)$.

We remark that the running times from the beginning of the section are just special cases of this formula. For later use, we now summarize this algorithm in the following theorem.

Theorem 2.4. *Let G be a graph, and (T, X) be a tree decomposition of G of width $t-1$. Then one can compute $\chi(G)$ in time $O(N \cdot t \cdot \alpha)$, where N is the number of nodes in T, and α is the maximum number of non-isomorphic colourings of $G\big[X(v)\big]$ for $v \in V(T)$.*

Finally, the last example of an algorithm which uses dynamic programming on a tree decomposition that we discuss here is the algorithm from Theorem 2.2 for the M-partition problem in chordal graphs for matrices M with all diagonal entries 0.

First, we observe that an M-partition of a graph G for a matrix M, which has all 0 on the main diagonal, is a partition of the vertices of G into k independent sets with possibly (depending on M) all, or no edges between some of the sets. It can be seen that any graph G admitting such M-partition must necessarily be properly k-colourable. This implies, in particular, that the clique number $\omega(G)$ of G is at most k. Moreover, if G is chordal, this gives us that the treewidth of G must be at most $k-1$. The algorithm from [28], which we are going to describe, works, in fact, for any (not necessarily chordal) graph G of bounded treewidth, and any matrix M (not only for 0-diagonal matrices), and also solves the list version of the problem.

We say that a node v of a rooted tree decomposition (T, X) of G is an *introduce node*, if v has at exactly one child u, and $X(u) = X(v) \setminus \{x\}$ for some $x \in V(G)$; v is a *forget node*, if v has exactly one child u, and $X(v) = X(u) \setminus \{x\}$ for some $x \in V(G)$; v is a *start node*, if v has no children and $|X(v)| = 1$, and v is a *join node*, if v has exactly two children u and w, and $X(v) = X(u) = X(w)$. We say that a rooted tree decomposition (T, X) is a *nice*, if each node of T is either an introduce node, a forget node, a start node, or a join node. It can be easily seen that any tree decomposition (T, X) of width t can be transformed into an equivalent nice tree decomposition (T', X') of the same width, where $|V(T')| \leq t \cdot |V(T)|$.

Now, let (T, X) be a nice tree decomposition of G. Recall the definitions of T_v and G_v from before. The algorithm for the list M-partition problem is based on dynamic programming on a nice tree decomposition (T, X) of G. It works as follows. For each vertex $v \in V(T)$, it computes the set $F(v)$ of pairs (Ξ, S), where $S \subseteq \{1 \ldots k\}$, and Ξ is an M-partition of $X(v)$ which can be extended to an M-partition $\Sigma = V_1 \cup \ldots \cup V_k$ of G_v such

that S is the set of those i for which $V_i \setminus X(v)$ is non-empty. The sets $F(v)$ are computed first for the leaves of T and then for other vertices of T in a bottom-up order. Clearly, if the set $F(r)$ for the root r of T is non-empty, then G admits an M-partition, otherwise not.

We now explain how the algorithm computes the sets $F(v)$. If v is a start node then $X(v) = \{x\}$ for some $x \in V(G)$, and the set $F(v)$ consists of all pairs (Ξ, S) where Ξ is a partition $V_1 \cup \ldots \cup V_k$ with $V_i = \{x\}$ and $V_j = \emptyset$ for all $j \neq i$, and $S = \emptyset$.

If v is a forget node, then v has a unique child u with $X(v) = X(u) \setminus \{x\}$ for some $x \in V(G)$, and the set $F(v)$ consists of pairs (Ξ, S) such that $(\Xi', S') \in F(u)$, Ξ is a restriction of $\Xi' = V_1' \cup \ldots \cup V_k'$ onto $X(v)$, and $S = S' \cup \{i\}$ where $x \in V_i'$.

If v is an introduce node, then v has a unique child u with $X(u) = X(v) \setminus \{x\}$ for some $x \in V(G)$, and the set $F(v)$ consists of pairs (Ξ, S) such that $\Xi = V_1 \cup \ldots \cup V_k$ is an M-partition of $X(v)$ with the property that $x \in V_j$ implies $i \notin S$ for each i with $M_{i,j} = 1$, and also $(\Xi', S) \in F(u)$ where Ξ' is a restriction of Ξ onto $X(u)$.

Finally, if v is a join node, then v has exactly two children u, w with $X(u) = X(v) = X(w)$, and the set $F(v)$ consists of pairs (Ξ, S) such that $(\Xi, S') \in F(u)$ and $(\Xi, S'') \in F(w)$ where $S = S' \cup S''$, and there is no $i \in S'$, and $j \in S''$ such that $M_{i,j} = 1$.

Now, the problem Π and the scheme $\mathcal{S}$ corresponding to the above algorithm can be defined as follows. The sets $\rho(G)$ and $\pi(G)$ consists of all colourings of G using k colours, and all M-partitions of G, respectively. The valuation function $\mu(c)$ is 0 for any c, and $\Gamma, \Sigma, \sigma, \eta$ are exactly like for the graph colouring problem. The fingerprint mapping $\sigma(H, c)$ for $c \in \rho(G)$, gives a pair (Ξ, S), where Ξ is the restriction of c onto H, and S is the list of colours that appear in $G - V(H)$. Fingerprints f and f' are the equivalent, $f \approx f'$, if and only if, they are equal, $f = f'$. Finally, the algorithm $\mathcal{A}$ is what is described in the above paragraphs.

The complexity is analyzed similarly as before. Let $t-1$ be the treewidth of G. First, we observe that the number of possible fingerprints (Ξ, S) is at most $2^k k^t$ (at most t vertices in each bag, and exactly k colours). Next, it can be seen that for each node of the tree decomposition and any Ξ, processing fingerprints (Ξ, S) for all S can be done in time $O(k^2 4^k)$; observe that this comes from the complexity of testing the condition for the join nodes. Hence, the complexity of $\mathcal{A}$ for all fingerprints is $O(k^2 4^k k^t)$. Finally, it is known that for any graph G, there exists an optimal tree decomposition of G whose tree has at most n nodes. This tree decomposition can be transformed into a nice tree decomposition of G which has the the same width and has at most $n \cdot t$ nodes. Therefore, it follows that for the whole algorithm the complexity is $O(n \cdot t \cdot k^2 4^k k^t)$.

We close by remarking that, as in the earlier examples, the problems we investigate in Chapters 3, 5 and our solutions to them can be seen to be (formally) expressible as (induced hereditary) graph problems Π with dynamic programming schemes $\mathcal{S}$ which use tree decompositions of particular properties (e.g., the so-called block-cutpoint tree).

2.3 Greedy algorithms

In this section, we describe some examples of problems for which an optimal solution is obtained using greedy choices, rather than using an exhaustive (dynamic) enumeration of all possibilities as in the previous section. In chordal graphs, such algorithms are usually based on perfect elimination orderings. Using the properties of these orderings, the algorithms are allowed to make greedy choices, which always lead to correct solutions.

First, we remark that we have already discussed examples of this type of algorithms in Chapter 1, namely, the algorithm LexBFS, and the algorithm for split graph recognition (see Theorem 1.25). Here, in addition, we mention three other such algorithms. The simplest one is the well-known algorithm for the graph colouring problem in chordal graphs [38]. The algorithm processes vertices of a given graph G in the reverse of a perfect elimination ordering of G, and colours vertices with numbers $\{1 \dots n\}$. Every time the algorithm processes a vertex, it assigns this vertex the smallest possible colour (number) which is not present in the already coloured neighbours of that vertex. This always leads to an optimal proper colouring, since the set of forward neighbours in a perfect elimination ordering forms a clique. Hence, if a vertex v is given a colour i, there must exist a clique of size $i-1$ in the neighbourhood of v, which clearly cannot be coloured with less then $i-1$ colours.

Theorem 2.5. *There exists an $O(n+m)$ algorithm to find the chromatic number and an optimal proper colouring of a chordal graph.*

Next, we discuss the algorithm for computing the maximum size independent set in chordal graphs [38] due to Gavril. This algorithm also processes the vertices of G, but this time, in the order given by a perfect elimination ordering π of G. When a vertex v is processed, the algorithm assigns v to the set S, and removes from G the forward (in π) neighbours F_v of v. After all vertices are processed, S is a maximum independent set of G. This can be seen as follows. First, S is clearly independent, since whenever we process a vertex, we remove all its neighbours from further consideration; hence, $|S| \leq \alpha(G)$. On the other

hand, the sets F_v, $v \in S$, are all cliques, since π is a perfect elimination ordering. Also, they are pairwise disjoint and, clearly, cover each vertex of G. Hence, they form a proper colouring of $\overline{G}$; thus, $|S| \geq \chi(\overline{G})$. But G is perfect, and hence, by Theorem 1.15, also $\overline{G}$ is perfect; thus, $\chi(\overline{G}) = \omega(\overline{G}) = \alpha(G)$. Hence, S is indeed a maximum size independent set of G.

Theorem 2.6. *There exits an $O(n+m)$ time algorithm to find a maximum size independent set and an optimal clique cover of a chordal graph.*

Finally, we explain a more complex algorithm, which also uses perfect elimination orderings and makes greedy choices. It is the algorithm from Theorem 2.2 for the M-partition problem in chordal graphs for matrices M with all diagonal entries 1. Note that an M-partition of a graph G for M with all diagonal entries 1 is a partition of the vertices of G into k cliques with possibly all, or no edges between some of the cliques.

The algorithm starts by computing a perfect elimination ordering π of G, and by assigning to each vertex $v \in V(G)$ a list $\ell(v) = \{1 \dots k\}$. Then, for each clique i, it either decides that the clique is empty and removes i from all lists, or it chooses vertices $x_i, y_i \in V(G)$, and sets $\ell(x_i) = \{i\}$ and $\ell(y_i) = \{i\}$. Then it removes i from each vertex that appears in π before x_i or after y_i, and, for each $j \in \{1 \dots k\}$, it removes j from the list of each vertex $z \in V(G)$ such that zx_i or zy_i is an edge and $M_{i,j} = 0$, or zx_i or zy_i is not an edge and $M_{i,j} = 1$. If the list of some vertex v becomes empty, the algorithm declares G not M-partitionable. Otherwise, for each $v \in V(G)$, it chooses $i \in \ell(v)$, and assigns v into the clique i. The correctness of this algorithm can be easily argued using the properties of perfect elimination orderings (cf. Theorem 7.5). The running time of this algorithm is $O(n^{2k+1}k^2)$, since there are at most $1 + n + \binom{n}{2} \leq n^2$ choices for each clique, and processing the lists, for each choice, takes $O(k^2 n)$ time. In addition, the algorithm can be immediately adapted for solving the list case by replacing the initial lists ℓ with the input lists ℓ_0. Finally, we remark that the algorithm actually computes all possible M-partitions of G; each such partition can be obtained from some lists computed by the algorithm by a greedy choice.

We conclude this section by mentioning other examples of greedy algorithms we discuss in later chapters. Namely, in Chapter 7, we have an algorithm for the P_4-free P_4-transversal problem in chordal comparability graphs, which is based on a modification of LexBFS, and also the algorithm in Theorem 7.5, which is based on the above M-partition algorithm.

2.4 Graph Grammars

Graph grammars originated as a natural generalization of formal language theory to graphs and since found their applications in numerous areas of computer science such as VLSI layout schemes, database design, modeling of concurrent systems, pattern recognition, compiler construction and others. Several different flavours of grammars for graphs have been studied; some are based on replacing vertices, and some based on replacing (hyper)-edges, while others replace whole subgraphs; here, additionally, mechanisms for "gluing" graphs can vary. In all of these models, nodes or edges of graphs are usually labeled, and it is the labels that control the way graphs are transformed by the grammar.

In this section, we discuss one particular model of graph grammars, namely, the so-called hyperedge replacement grammars (HRG). These grammars are especially interesting, since they are based on a context-free graph transformation mechanism, which allows, in most cases, efficient algorithms for recognition and other problems.

Let C be an arbitrary (fixed) set of *labels* and let $type : C \to \mathbb{N}$ be a *typing* function. A *hypergraph* H over C is a tuple $(V_H, E_H, att_H, lab_H, ext_H)$ where V_H is a finite set of nodes, E_H is a finite set of hyperedges, $att : E_H \to V_H^*$ is a mapping assigning a sequence of pairwise distinct *attachment nodes* $att_H(e)$ to each $e \in E_H$, $lab_H : E_H \to C$ is a mapping that *labels* each hyperedge such that $type(lab_H(e)) = |att_H(e)|$, and $ext_H \in V_H^*$ is a sequence of pairwise distinct external nodes. Note that, in this model, hyperedges are ordered subsets of vertices and are labeled according to their cardinality. Also note that the external nodes ext_H, usually, are not needed to be specified. We denote by $\mathcal{H}_C$ the set of hypergraphs over C.

Now, we define the *hyperedge replacement* mechanism. Let $H \in \mathcal{H}_C$ and let $e_1, \ldots, e_k$ be hyperedges of H to be replaced by hypergraphs $H_1, \ldots, H_k \in \mathcal{H}_C$ where $type(e_i) = |ext_{H_i}|$ for $1 \leq i \leq k$. The hypergraph $H[e_1/H_1, \ldots, e_k/H_k]$ is constructed from the disjoint union of H and $H_1, \ldots, H_k$ by identifying the vertices of $att_H(e_i)$ with the vertices of ext_{H_i} in their respective orders, for each $1 \leq i \leq k$, and then removing hyperedges $e_1, \ldots, e_k$. It can be seen that this mechanism is "context-free", because the result of a replacement depends only on the hyperedges that are being replaced, and not on their relationship with the rest of the hypergraph. Also, it does not matter whether the hyperedges are replaced simultaneously or one by one, or in what order they are replaced. That is, we have $H[e_1/H_1, \ldots, e_k/H_k] = H[e_1/H_1] \ldots [e_k/H_k]$ and $H[e_1/H_1][e_2/H_2] = H[e_2/H_2][e_1/H_1]$, and $H[e_1/H_1][e_2/H_2] = H[e_1/H_1[e_2/H_2]]$ for edges e_i in appropriate hypergraphs.

A *hyperedge replacement grammar* HRG is a tuple (N, T, P, S) where $N \subseteq \mathcal{C}$ is a set of *nonterminals*, $T \subseteq \mathcal{C}$ with $T \cap N = \emptyset$ is a set of *terminals*, $P \subseteq N \times \mathcal{H}_{\mathcal{C}}$ is a finite set of *productions* with $type(A) = |ext_R|$ whenever $(A, R) \in P$, and $S \in N$ is the *start symbol*.

A *derivation step* in HRG is the binary relation $\underset{P}{\Longrightarrow}$ on $\mathcal{H}_{\mathcal{C}}$ with $H \underset{P}{\Longrightarrow} H'$ whenever $H' = H[e/R]$ where $lab_H(e) = A$ and $(A, R) \in P$. We denote by $\underset{P}{\Longrightarrow}^*$ the transitive closure of $\underset{P}{\Longrightarrow}$. For $A \in \mathcal{C}$, we denote by $A^\bullet$ the hypergraph with a single hyperedge A attached to $type(A)$ vertices. ($A^\bullet$ is usually called a *handle*.) The *hypergraph language* $L(HRG)$ *generated* by the grammar HRG is $L_S(HRG)$, where for $A \in N$, the set $L_A(HRG)$ consists of all hypergraphs in $\mathcal{H}_T$ derivable from $A^\bullet$ by applying productions of P. That is,

$$L_A(HRG) = \{H \in \mathcal{H}_T \mid A^\bullet \underset{P}{\Longrightarrow}^* H\}.$$

The context-free nature of hyperedge replacement allows one to define derivation trees for hyperedge replacement grammars similar to the ones defined for context-free languages. (For simplicity we deviate, in what follows, from the standard definition given in [58].)

A *derivation tree* $\mathcal{T}$ in $HRG = (N, T, P, S)$ is a tree whose each node is labeled either by a terminal symbol $A \in T$, a nonterminal symbol $A \in N$, or by a production rule $(A, R) \in P$ such that

(i) a node v labeled by $A \in T$ has no children, and we define $res(v) = A^\bullet$,

(ii) a node v labeled by $A \in N$ either has no children and $res(v) = A^\bullet$, or has exactly one child w labeled by a production rule $(A, R) \in P$ and $res(v) = res(w)$, and

(iii) a node v labeled by a production rule $(A, R) \in P$ has a child v_i labeled by $lab_R(e_i)$, for each hyperedge $e_i \in E(R)$, where $1 \leq i \leq k = |E(R)|$, and we define $res(v) = R[e_1/H_1, \ldots, e_k/H_k]$, where $H_i = res(v_i)$.

The graph defined by $\mathcal{T}$ is $res(r)$, where r is the root of $\mathcal{T}$. Also, it can be seen that, for any hypergraph $H \in L_A(HRG)$, there exists a derivation tree $\mathcal{T}$ such that $res(r) = H$, where r is the root of $\mathcal{T}$, and is labeled by A.

A hyperedge replacement grammar $HRG = (N, T, P, S)$ is said to be of *order* k, if $type(A) \leq k$ for all $A \in N$. An example of a grammar of order three generating all partial 3-trees (graph of treewidth at most three) can be seen in Figure 2.2. This grammar uses two nonterminal labels A and S, where S is the starting nonterminal, and one terminal label T. The vertices marked as $\circ$ and $\bullet$ represent nodes, where the latter are the external nodes ext_H (the labels indicate their ordering in ext_H). The vertices in boxes represent

hyperedges, and the edges in the diagram connect each hyperedge e with its attachment nodes $att_H(e)$, where the labels on the edges indicate their ordering in $att_H(e)$.

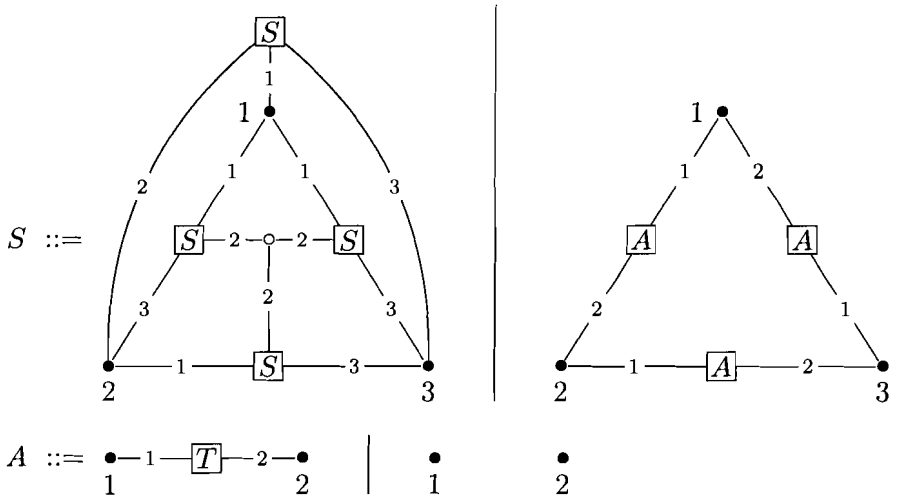

Figure 2.2: An example of a hyperedge replacement grammar generating all partial 3-trees.

We now briefly explain how this grammar generates all partial 3-trees. In the first step of the derivation, if we apply the second rule for S, we obtain a triangle whose edges are labeled with A; subsequent applications of the rules for A either remove some of these edges, or replace them with a terminal edge. On the other hand, if we apply the first rule for S, we obtain a hypergraph with four vertices such that any three of them form a hyperedge labeled with S. Each subsequent application of this rule to a hyperedge e labeled with S, replaces e with four new vertices, where three of them (marked 1, 2, 3) are identified with the vertices incident to e, and such that, again, any three of these four vertices form a hyperedge labeled with S. This precisely mimics the generation of 3-trees, and the role of the hyperedges labeled with S is to maintain all possible triples of vertices forming a triangle in the generated 3-tree. Eventually, each hyperedge labeled with S is replaced with a triangle (using the second rule for S), some of whose edges are removed later and other are turned into terminal edges. Note that in the graph generated this way we can have many parallel edges between any two vertices. The final simple graph representing a partial 3-tree is obtained by removing all but one parallel edge between any two vertices.

Next, we briefly mention some properties of hyperedge replacement grammars. In terms of their generative power, they can generate all context-free string languages as well as some

context-sensitive languages (for appropriately defined string graphs). Also, as a consequence of the hyperedge replacement rule, graphs generated by a hyperedge replacement grammar of order k are always at most k-connected. Hence, since the description of HRG is finite, no HRG can generate graphs of arbitrary large connectivity. Also, for any k, there exists a HRG of order k generating all partial k-trees (see Figure 2.2 for an example), whereas no HRG of order $k-1$ can generate them [58]. It follows that graphs generated by HRG's of different orders form a proper infinite hierarchy.

Now, we discuss the complexity of the membership problem for HRG. It is not difficult to see that the problem is in NP. One has to guess the derivation and test whether it generates the input graph; the derivation can be described by a polynomial (in fact even linear [58]) number of bits, since it is not possible to erase vertices. In general, the membership problem turns out to be NP-hard, and in particular, the following is true.

Theorem 2.7. [50] *There exists a hyperedge replacement grammar of order two that generates an NP-complete graph language of maximum degree two. There exists an hyperedge replacement grammar of order two that generates an NP-complete graph language of connected graphs.*

Note that, in this theorem, the first grammar generates disconnected graphs with unbounded number of connected components, whereas the second grammar generates graphs of unbounded degree. This is in particular important, since if the grammar only generates hypergraphs of bounded degree and of bounded number of connected components (or more precisely, hypergraphs whose k-separability is $O(\log n)$), the membership problem is polynomial time solvable. Note that the k-separability of a hypergraph H is the maximum number of connected components of $H - X$ where $X \subseteq V_H$ and $|X| = k$.

Theorem 2.8. [52, 58] *Let HRG be a hyperedge replacement grammar of order k. If for each n-vertex hypergraph $H \in L(HRG)$, k-separability of H is $O(\log n)$, then the membership problem for HRG is polynomial time solvable. In case k-separability is $O(1)$ for all $H \in L(HRG)$, the membership problem is in $LOGCFL$.*

We conclude by mentioning the following results. A *k-uniform* hypergraph or simply a *k-hypergraph* is a hypergraph whose all hyperedges are of cardinality k. In [21], it was shown that, if an HRG of order k generates only k-hypergraphs, there exists a cubic time algorithm (for any k) recognizing graphs in $L(HRG)$, whereas, if the grammar of order k is allowed to use hyperedges of all cardinalities, it can generate an NP-complete graph language [20].

Theorem 2.9. [21] *Let HRG be a hyperedge replacement grammar of order k. Then there exists a cubic time algorithm to decide, given a k-hypergraph H, whether $H \in L(HRG)$.*

Theorem 2.10. [20] *For $k \geq 3$, there exists a hyperedge replacement grammar of order k that generates an NP-complete set of k-connected hypergraphs.*

2.5 Treewidth and Monadic Second Order Logic

In this final section of this chapter, we discuss a connection between graphs of bounded treewidth and monadic second order logic of graphs.

Let $\mathcal{V}$ be a countable alphabet of *variables* (denoted by x,y,$z\ldots$). Let $\mathcal{R}$ be a (finite) vocabulary of *relational symbols* R and their arities $\rho(R)$. Let $\mathcal{X}$ be a countable alphabet of *relational variables* (denoted by X,Y,$Z\ldots$) and their arities $\rho(X)$.

A *second order formula* (SO) is a (finite) formula that can be constructed from atomic formulas using binary operations $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$ and quantification symbols $\forall x, \exists x, \forall X, \exists X$, where atomic formulas are $x = y$, $R(x_1, \ldots, x_k)$, and $X(x_1, \ldots, x_k)$ for $x, y, x_1, \ldots, x_k \in \mathcal{V}$, $R \in \mathcal{R}$, and $X \in \mathcal{X}$ with $\rho(R) = \rho(X) = k$.

A second order formula φ is called a *monadic second order formula* (MS), if φ contains only relational variables $X \in \mathcal{X}$ of arity one (the so-called *set variables*); the arities of relational symbols $R \in \mathcal{R}$ in φ are unrestricted.

The *(relational) structures* of second order logic and *satisfiability* of a formula in a structure are defined in the usual way. (We omit the formal details.)

The main theorem of this section is the following theorem, which deals with hypergraph replacement grammars and their relation to properties of hypergraphs definable by MS.

Theorem 2.11. [14, 15, 16] *Let L be a MS-definable set of (labeled) hypergraphs (that is, the set of (labeled) hypergraphs satisfying some MS formula φ). Let HRG be any hyperedge replacement grammar.*

i) One can construct a hyperedge replacement grammar HRG′ generating $L \cap L(HRG)$.

ii) For every derivation tree $\mathcal{T}$ of HRG, one can decide in time $O(size(\mathcal{T}))$ whether the graph G in $L(HRG)$ defined by $\mathcal{T}$ is in L.

We remark that the set of all partial k-trees (graphs of treewidth at most k) is definable by a hypergraph replacement grammar [58] (see Figure 2.2 for the case $k = 3$) . Additionally,

it can be observed that for each graph H generated by this grammar, there exists a derivation tree $\mathcal{T}$ whose size is $O(|V(H)|)$. This gives us the following fundamental theorem.

Theorem 2.12. [14, 15, 16] *Any property of graphs of treewidth at most k that is expressible in Monadic Second Order Logic is decidable in linear time.*

We close this section by remarking that many graph problems (including the ones we study in later chapters) are expressible as MS formulas; hence, by this theorem, they are efficiently solvable on graphs of bounded treewidth.

Chapter 3

Polar colourings of Chordal Graphs

3.1 Monopolar Chordal Graphs

Recall, that a partition of the vertex set $V(G)$ of a graph G into sets $V_1 \cup V_2$ is *monopolar*, if V_1 induces an independent set, and V_2 induces a P_3-free graph. We say that a graph G is monopolar, if G admits a monopolar partition. It can be seen that a graph is P_3-free, if and only if, it is a disjoint union of cliques. In the following, for simplicity, we shall always refer to the independent set of a monopolar partition as A, and to the disjoint union of cliques in the partition as D. Hence, we say that a graph is monopolar, if it can be partitioned into an independent set A, and a disjoint union of cliques D.

The problem of monopolar partitions of graphs was previously studied in the literature for some restricted classes of graphs. In particular, in [24], the authors show a polynomial time algorithm for finding a monopolar partition of a cograph, which follows from a finite forbidden induced subgraph characterization of monopolar cographs which they describe. They also show a similar result for unipolar cographs and polar cographs. We remark that their definition of monopolarity slightly differs from ours, namely, our monopolar graphs corresponds to the so-called *stable monopolar* graphs of [24].

In this section, we describe a linear time algorithm for recognizing monopolar chordal graphs. In fact, our algorithm solves the more general case of the list monopolar partition problem in the class of chordal graphs. We observe that the disjoint union of two monopolar graphs is again monopolar. Hence, in what follows, we shall focus on connected graphs.

First, we prove the following interesting property of chordal graphs which will allow us to construct the algorithm for testing monopolarity. Recall that $w(H)$ is the size of a

maximum clique in H, and $\kappa(H)$ is the vertex connectivity of H.

Proposition 3.1. *For any separator S of a chordal graph G, we have $\omega(G[S]) \geq \kappa(G)$.*

Proof. Let S be a separator of G, that is, $G - S$ is a disconnected graph. Since G is chordal, by Proposition 1.11, there must exist a clique $C \subseteq S$, such that $G - C$ is disconnected. Clearly, $|C| \leq \omega(G[S])$. Now, let a and b be vertices from two different connected components of $G - C$. By Menger's Theorem [18], there must exist at least $\kappa(G)$ internally vertex disjoint paths connecting a to b in G, that is, paths no two of which share vertices other than a and b. Since C separates a from b, all these paths must go through C. Moreover, since the paths are internally vertex disjoint, and $a, b \notin C$, there cannot be more than $|C|$ such paths. This implies $\kappa(G) \leq |C| \leq \omega(G[S])$ as required. □

For 2-connected graphs, we have the following simple corollaries.

Proposition 3.2. *In any monopolar partition $A \cup D$ of a 2-connected chordal graph G, the set D induces a clique.*

Proof. Suppose that D induces a disconnected graph. Then, since G is 2-connected, we have $\kappa(G) \geq 2$, and A is a separator in G (separating the connected components of $G[D]$). Hence, by Proposition 3.1, we have $\omega(G[A]) \geq \kappa(G) \geq 2$, but A is an independent set, and hence $\omega(G[A]) = 1$. It follows that $G[D]$ is connected, hence it must be a clique. □

Proposition 3.3. *A 2-connected graph G is both chordal and monopolar if and only if it is a split graph.*

Proof. Clearly, any split graph is both chordal and monopolar. Conversely, let G be a 2-connected monopolar chordal graph, and let $A \cup D$ be a monopolar partition of G. By Proposition 3.2, the set D induces a clique, hence $A \cup D$ is clearly a split partition of G. □

It follows now that, if we have a 2-connected graph, checking its monopolarity amounts to checking whether the graph admits a split partition. This can be accomplished in time $O(n + m)$ (cf. Theorem 1.25) even in the list case, that is, if some vertices are precoloured (preassigned either to the independent set or the clique).

If the graph is not 2-connected, we consider the structure of its blocks. A *block* of a graph G is a maximal induced subgraph of G which cannot be disconnected by the removal

of a single vertex. Hence, a block is either a 2-connected maximal induced subgraph of G, or an induced subgraph isomorphic to K_2. We call the former a *non-trivial* block, and the latter a *trivial* block. Recall that a cutpoint of a graph is a vertex whose removal disconnects the graph. We shall need the following definition.

Let X be a set, and $\mathcal{B}$ be a collection of sets. The *incidence graph* of X and $\mathcal{B}$ is the bipartite graph whose vertices are the elements of X and the sets of $\mathcal{B}$, such that $x \in X$ is adjacent to $B \in \mathcal{B}$, if and only if, $x \in B$.

The *block-cutpoint tree* of a graph G is the incidence graph of the cutpoints and blocks of G. It is an easy observation, that it is in fact a tree [68]. Similarly, we define a modified version of the block-cutpoint tree which we shall need for our algorithm. The *block-vertex tree* $T(G)$ of a graph G is the incidence graph of the vertices, and the blocks of G. Again, it can be seen that it must be a tree. In the subsequent text, we shall refer to the vertices of $T(G)$ as nodes to distinguish them from the vertices of G, in cases where ambiguity may arise.

We shall always consider the tree $T(G)$ rooted at some arbitrarily chosen node *root* which is a vertex of G, that is, not a block of G. We shall say that a trivial block respectively a non-trivial block of G, which is a child of a node v in $T(G)$, is a *trivial child* respectively *non-trivial child* of v. Also, we shall call the children of the children of v, the *grandchildren* of v, and in particular, the children of the trivial children of v will be referred to as the *trivial grandchildren* of v.

Now, we are ready to describe our algorithm for monopolarity. The algorithm takes as the input a chordal graph G with lists $\ell_0(v) \subseteq \{\mathsf{r}, \mathsf{b}\}$, for each $v \in V(G)$, and outputs a monopolar partition $A \cup D$ of G which *respects* the lists ℓ_0 (that is, a partition $A \cup D$ such that $\mathsf{r} \in \ell_0(v)$, for each $v \in A$, and $\mathsf{b} \in \ell_0(v)$, for each $v \in D$) or announces that none exists. The algorithm performs the following steps. First, it constructs the block-vertex tree of G. Then, it performs a bottom-up search on the block-vertex tree while modifying the lists $\ell_0(v)$ of processed vertices by removing from $\ell_0(v)$ those colours which are never used to colour v in any desired monopolar partition of G. After this procedure, the modified lists ℓ^* are used to greedily construct a monopolar partition of G by again processing the vertices of the block-vertex tree of G, but now in a top-down order, and greedily choosing colours for the vertices while propagating the colours to the rest of the graph to ensure a monopolar partition is obtained. We show that, unless we have $\ell^*(v) = \emptyset$, for some $v \in V(G)$, this greedy procedure will always succeed in finding a desired monopolar partition of G.

The algorithm is summarized below as Algorithm 3.1.

Algorithm 3.1: Monopolar graph recognition.

Input: A connected chordal graph G with lists $\ell_0(v) \subseteq \{\mathsf{r}, \mathsf{b}\}$, for $v \in V(G)$.
Output: A monopolar partition of G respecting ℓ_0 (if it exists).

1 Obtain the block-vertex tree T of G, root T at arbitrary $root \in V(G)$.
2 Construct the reduced lists ℓ^* using Algorithm 3.2
3 **if** $\ell^*(v) = \emptyset$ for some v in G **then**
4 **return** "G has no monopolar partition respecting ℓ_0"
5 **else**
6 Extract a monopolar partition $A \cup D$ from ℓ^* using Algorithm 3.4
7 **return** $A \cup D$

Now, we explain how to obtain the reduced lists ℓ^*. Starting with the initial lists $\ell \leftarrow \ell_0$, we explore the block-vertex tree of G in a bottom-up fashion eliminating colours from $\ell(v)$ for the explored vertices v. The elimination is performed by applying a set of rules (explained later). Once *root* is reached, the lists $\ell^* \leftarrow \ell$ are returned. The algorithm is summarized below as Algorithm 3.2.

Algorithm 3.2: Constructing reduced lists.

Input: A chordal graph G with lists $\ell_0(v) \subseteq \{\mathsf{r}, \mathsf{b}\}$, for $v \in V(G)$, and the block-vertex tree T of G.
Output: The reduced lists ℓ^*.

1 Initialize $\ell \leftarrow \ell_0$ and $S \leftarrow \emptyset$ (the set of processed vertices)
2 **while** $S \neq V(G)$ **do**
3 pick a vertex $v \in V(G) \setminus S$ which has all grandchildren in S
4 add v to S, and process its list $\ell(v)$ by performing the following two steps.
5 Apply the Rules 1 - 8.
6 Apply the Block Rule for each non-trivial child of v.
7 $\ell^* \leftarrow \ell$ (the final lists)
8 **return** ℓ^*

Note, that for any vertex v of G chosen by the algorithm in line 4, the role of the rules is

to (possibly) reduce the list $\ell(v)$ by eliminating the colours that we know can no longer be used. The rules are depicted in Figure 3.1, in left to right order. We use ⊠ to denote trivial blocks, □ for non-trivial blocks, and ○ for cutpoints. The description of the rules follows.

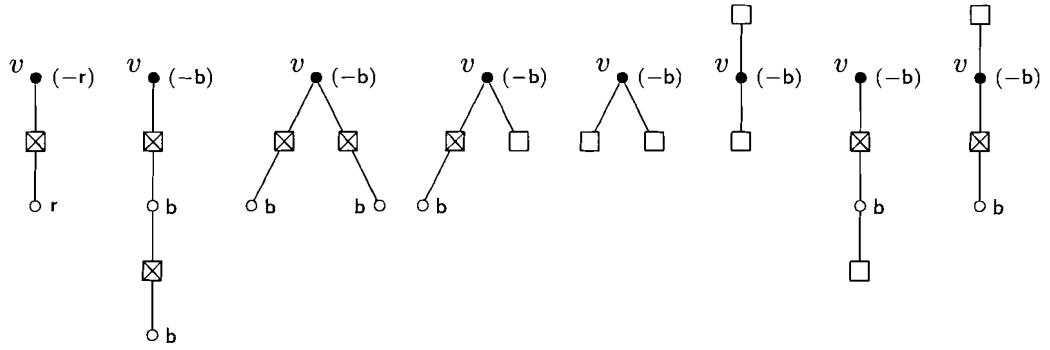

Figure 3.1: Rules 1 - 8.

Algorithm 3.3: The Rules 1 - 8 and the Block Rule.

Rule 1 If some trivial grandchild w of v has $\ell(w) = \{\mathsf{r}\}$, then remove r from $\ell(v)$.

Rule 2 If some trivial grandchild w of v, and some trivial grandchild w' of w, have $\ell(w) = \ell(w') = \{\mathsf{b}\}$, then remove b from $\ell(v)$.

Rule 3 If some trivial grandchildren w, w' of v have $\ell(w) = \ell(w') = \{\mathsf{b}\}$, then remove b from $\ell(v)$.

Rule 4 If some trivial grandchild w of v has $\ell(w) = \{\mathsf{b}\}$, and v has a child that is a non-trivial block, then remove b from $\ell(v)$.

Rule 5 If v has two children that are non-trivial blocks, then remove b from $\ell(v)$.

Rule 6 If both the parent of v and some child of v are non-trivial blocks, then remove b from $\ell(v)$.

Rule 7 If some trivial grandchild w of v has $\ell(w) = \{\mathsf{b}\}$, and w has a child that is a non-trivial block, then remove b from $\ell(v)$.

Rule 8 If some trivial grandchild w of v has $\ell(w) = \{\mathsf{b}\}$, and the parent of v is a non-trivial block, then remove b from $\ell(v)$.

The Block Rule If v has a non-trivial child H, remove from $\ell(v)$ the colour r respectively b, if there exists no split partition $X \cup Y$ of H respecting ℓ (that is, a partition where X is an independent set, Y is a clique, and $\mathsf{r} \in \ell(x)$ and $\mathsf{b} \in \ell(y)$ for each $x \in X$, $y \in Y$) such that $v \in X$ respectively $v \in Y$.

We remark that the rules can be justified by simple observation about monopolar partitions. For example, in the first rule, the colour r is removed from the list of the vertex v, if v has a trivial grandchild w whose list contains only r. In other words, we have that v is adjacent to w, and for any monopolar partition $A \cup D$, the vertex w must belong to the independent set A. Hence, since A is an independent set, v will never belong to A, so we can safely remove r from the list of v. The other rules are justified quite similarly.

Now, note that the rules are easy to implement directly. The Block Rule requires us to test whether a graph with some vertices precoloured r or b admits a split partition; there is a simple linear time algorithm for this (cf. [43] or Theorem 1.25).

We have illustrated an example application of the rules in Figure 3.2. On the left is a chordal graph. It has one vertex with list $\{\mathsf{r}\}$, all other vertices have lists $\{\mathsf{r},\mathsf{b}\}$. In the middle is its block-vertex tree. The unlabeled vertices of the tree all have lists $\{\mathsf{r},\mathsf{b}\}$, and the vertex with list $\{\mathsf{r}\}$ can be seen at the bottom of the tree as a child of a non-trivial block. When the Block Rule is applied to this block, it forces the list of the parent of this block to be $\{\mathsf{b}\}$. Further applications of Rules 7, 1, 5, 1, and 2, in that order, result in what can be seen on the right.

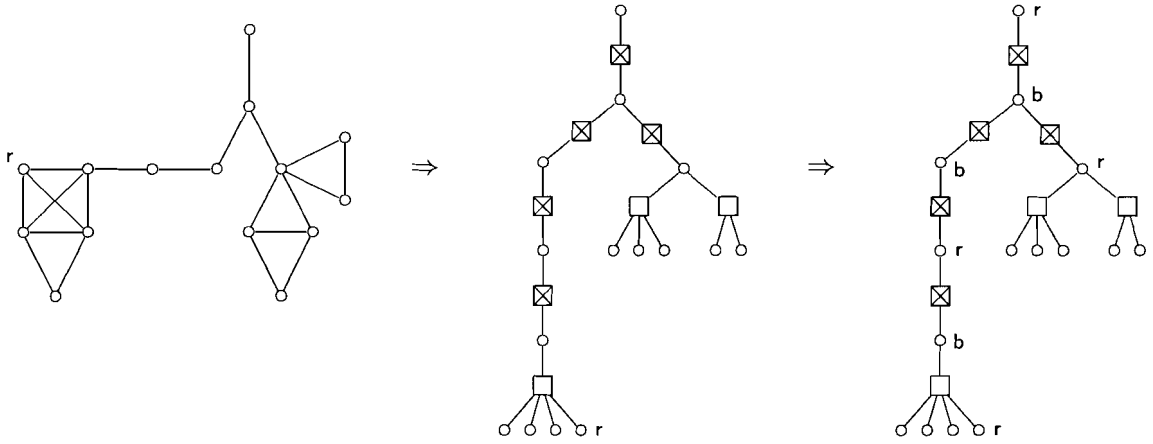

Figure 3.2: An illustration of the monopolarity recognition algorithm.

It remains to explain how to find a monopolar partition respecting ℓ^*, provided that $\ell^*(v) \neq \emptyset$, for all $v \in V(G)$. We use a top-down procedure which reduces the lists $\ell^*(v)$ to single element lists, starting from *root*, and proceeding down in the tree. After all the vertices are processed, a monopolar partition $A \cup D$ of G respecting ℓ^* is constructed. The algorithm is summarized below as Algorithm 3.4.

Algorithm 3.4: Extracting a monopolar partition.

Input: A chordal graph G with reduced lists $\ell^*(v) \neq \emptyset$, for $v \in V(G)$.

Output: A monopolar partition $A \cup D$ respecting ℓ^*.

```
1  Initialize ℓ ← ℓ* and S ← ∅ (processed vertices)
2  if ℓ(root) = {r, b} then
3      set ℓ(root) to either {r} or {b}
4  while S ≠ V(G) do
5      pick v ∈ V(G) \ S that has all ancestor vertices in S, and add v to S
6      if ℓ(v) = {r} then set ℓ(w) ← {b} for all trivial grandchildren w of v
7      if ℓ(v) = {b} then
8          set ℓ(w) ← {r} for all trivial grandchildren w of v which have r ∈ ℓ(w)
9          if there is a trivial grandchild w0 with r ∉ ℓ(w0) then
10             set ℓ(w0) ← {b} and set ℓ(w') ← {r} for all trivial grandchildren w' of w0
11             add w0 to S
12     for each non-trivial child (block) H of v do
13         Obtain a split partition X ∪ Y of H respecting ℓ
14         set ℓ(x) = {r}, for x ∈ X, and ℓ(y) = {b}, for y ∈ Y
15 A ← the vertices v with ℓ(v) = {r}
16 D ← the vertices v with ℓ(v) = {b}
17 return A ∪ D
```

Now, we prove the correctness of all the above algorithms. We start with an easy observation which we will use frequently.

Proposition 3.4. *Every vertex of a 2-connected chordal graph lies in a triangle.*

Proof. Suppose otherwise. Let v be a vertex in a 2-connected chordal graph G whose neighbourhood $N(v)$ is an independent set. Since G is 2-connected, we have $\kappa(G) \geq 2$, and $|N(v)| \geq 2$. Also, there must exist a vertex w which is not adjacent to v, since otherwise v is a cutpoint of G. It follows that $N(v)$ is a cutset of G separating v from w, and by Proposition 3.1, we have $\omega(G[N(v)]) \geq \kappa(G) \geq 2$. But $N(v)$ is an independent set, and hence $\omega(G[N(v)]) = 1$, a contradiction. □

Proposition 3.5. *In any monopolar partition $A \cup D$ of a non-trivial block of a chordal graph, each vertex has a neighbour in D.*

Proof. Let v be a vertex of a non-trivial block H of G. Since H is 2-connected, by Proposition 3.4, the vertex v must have two neighbours u and w in H joined by an edge. Hence, if $A \cup D$ is a monopolar partition of G, then both u and w cannot belong to the independent set A since they are adjacent, and hence, at least one of them must belong to D. □

Next, we show that any monopolar partition of G respecting the input lists ℓ_0 also respects the reduced lists ℓ^*.

Proposition 3.6. *A monopolar partition $A \cup D$ of G respects ℓ_0, if and only if, it respects ℓ^*.*

Proof. Let $A \cup D$ be a monopolar partition of G respecting ℓ^*. It follows directly from Algorithm 3.2 that we have $\ell^*(v) \subseteq \ell_0(v)$, for any $v \in V(G)$. Hence, $A \cup D$ also respects ℓ_0.

Conversely, let $A \cup D$ be a monopolar partition of G respecting ℓ_0, and let $\ell^{(t)}$ denote the lists ℓ during the execution of Algorithm 3.2 on G after applying t rules. We prove the following statement by induction on the number of applications of the rules.

For all t, the partition $A \cup D$ respects $\ell^{(t)}$.

For $t = 0$, clearly, $\ell^{(t)} = \ell_0$ so the claim is true. Hence, suppose that $t \geq 1$, and assume that $A \cup D$ respects $\ell^{(t-1)}$. The lists $\ell^{(t)}$ are obtained from $\ell^{(t-1)}$ by applying a single rule, either one of Rules 1 - 8, or the Block Rule.

(i) If Rule 1 is applied, we must have a vertex v with a trivial grandchild w such that $\ell^{(t-1)}(w) = \{\mathsf{r}\}$, $\ell^{(t)}(v) = \ell^{(t-1)}(v) \setminus \{\mathsf{r}\}$, and $\ell^{(t)}(u) = \ell^{(t-1)}(u)$, for all $u \neq v$. Since $A \cup D$ respects $\ell^{(t)}$, we must have $w \in A$. Now, since v and w are adjacent in G, we must have $v \notin A$, which shows that $A \cup D$ also respects $\ell^{(t)}$.

(ii) If Rule 2 is applied, we have vertices v, w, w' where w is a trivial grandchild of v, and w' is a trivial grandchild of w, such that $\ell^{(t-1)}(w) = \ell^{(t-1)}(w') = \{\mathsf{b}\}$, $\ell^{(t)}(v) = \ell^{(t-1)}(v) \setminus \{\mathsf{b}\}$, and $\ell^{(t)}(u) = \ell^{(t-1)}(u)$, for all $u \neq v$. Since $A \cup D$ respects $\ell^{(t-1)}$, we have $w, w' \in D$, hence $v \notin D$ since otherwise v, w, w' is an induced P_3 in $G[D]$. It follows that $A \cup D$ respects $\ell^{(t)}$.

(iii) If Rule 3 is applied, the proof follows similarly.

(iv) If Rule 4 is applied, we have a vertex v with a trivial grandchild w, and a non-trivial child H, such that $\ell^{(t-1)}(w) = \{\mathsf{b}\}$, $\ell^{(t)}(v) = \ell^{(t-1)}(v) \setminus \{\mathsf{b}\}$, and $\ell^{(t)}(u) = \ell^{(t-1)}(u)$,

for all $u \neq v$. By Proposition 3.5, the vertex v must have a neighbour w' in H such that $w' \in D$. Also, since $A \cup D$ respects $\ell^{(t-1)}$, we have $w \in D$. Again, it follows that $v \notin D$, and hence, $A \cup D$ respects $\ell^{(t)}$.

(v) If Rule 5 is applied, we have a vertex v with non-trivial children H, H', such that $\ell^{(t)}(v) = \ell^{(t-1)}(v) \setminus \{\mathsf{b}\}$, and $\ell^{(t)}(u) = \ell^{(t-1)}(u)$, for all $u \neq v$. By Proposition 3.5, the vertex v must have neighbours w in H, and w' in H' such that $w, w' \in D$. Again, $v \notin D$, and hence $A \cup D$ respects $\ell^{(t)}$.

(vi) If Rule 6 is applied, the proof follows similarly.

(vii) If Rule 7 or 8 is applied, the proof is similar to (iv).

(viii) Finally, if the Block Rule is applied, we have a vertex v with a non-trivial child H, and $\ell^{(t)}(u) = \ell^{(t-1)}(u)$, for all $u \neq v$. Let $X = A \cap V(H)$, and $Y = D \cap V(H)$. Clearly, $X \cup Y$ is a monopolar partition of H. In fact, since H is 2-connected, by Proposition 3.2, $X \cup Y$ is a split partition of H. Now, recall that $v \in V(H)$. Hence, if $v \in X$, then clearly $v \in A$, and since $A \cup D$ respects $\ell^{(t-1)}$, we must have $\mathsf{r} \in \ell^{(t-1)}(v)$. It follows that $\mathsf{r} \in \ell^{(t)}(v)$, since $X \cup Y$ is a split partition of H respecting $\ell^{(t-1)}$ as required by the Block Rule. Similarly, if $v \in Y$, then $v \in D$, which implies $\mathsf{b} \in \ell^{(t-1)}(v)$, and hence $\mathsf{b} \in \ell^{(t)}(v)$ by the same argument. In both cases, we obtain that $A \cup D$ respects $\ell^{(t)}$.

Now, since $\ell^* = \ell^{(t)}$ for some t, the claim follows. □

Next, we prove the following statement about the reduced lists ℓ^*, which will be used to prove the correctness of the extraction procedure in Algorithm 3.4.

Proposition 3.7. *Suppose that $\ell^*(v) \neq \emptyset$, for all $v \in V(G)$. Then, for any $v \in V(G)$,*

if $\mathsf{b} \in \ell^*(v)$, then

- *(i) $\mathsf{r} \in \ell^*(w)$ for any trivial grandchild w of v, except possibly for one trivial grandchild w_0, and if w_0 exists, then*
  - *(a) $\ell^*(w_0) = \{\mathsf{b}\}$,*
  - *(b) $\mathsf{r} \in \ell^*(w')$ for any trivial grandchild w' of w_0,*
  - *(c) both v and w_0 have only trivial blocks as children, and*
  - *(d) if $v \neq$ root, then the parent of v is a trivial block,*
- *(ii) there exists a split partition $X \cup Y$ of H respecting ℓ^* with $v \in Y$, for any non-trivial child H of v,*
- *(iii) v belongs to at most one non-trivial block,*

and if $\mathsf{r} \in \ell^*(v)$, *then* $\begin{cases} \text{(i) } \mathsf{b} \in \ell^*(w) \text{ for any trivial grandchild } w \text{ of } v, \\ \text{(ii) there exists a split partition } X \cup Y \text{ of } H \text{ respecting } \ell^* \text{ with} \\ \quad v \in X, \text{ for any non-trivial child } H \text{ of } v. \end{cases}$

Proof. First, we note that during the execution of Algorithm 3.2, once a vertex v is processed, its list $\ell(v)$ remains the same in all subsequent steps of the algorithm, and hence, it is equal to $\ell^*(v)$. In the remainder of the proof, we shall use this fact frequently.

Suppose that $\mathsf{r} \in \ell^*(v)$ but $\mathsf{b} \notin \ell^*(w)$ for some trivial grandchild w of v. Then $\ell^*(w) = \{\mathsf{r}\}$ since otherwise $\ell^*(w) = \emptyset$. Now, since v is processed after w, it follows, by Rule 1, that we must have $\mathsf{r} \notin \ell^*(v)$, a contradiction.

Similarly, suppose that $\mathsf{b} \in \ell^*(v)$ but $\mathsf{r} \notin \ell^*(w_0)$, for some trivial grandchild w_0 of v. Then $\ell^*(w_0) = \{\mathsf{b}\}$, and, by Rules 4, 7 and 8, both v and w_0 do not have any non-trivial children or parents, since otherwise $\mathsf{b} \notin \ell^*(v)$. Moreover, by Rules 2 and 3, both v and w_0 have no other trivial grandchildren w with lists $\ell^*(w) = \{\mathsf{b}\}$. Hence, $\mathsf{r} \in \ell^*(w)$, for all trivial grandchildren $w \neq w_0$ of v, and also $\mathsf{r} \in \ell^*(w')$, for all trivial grandchildren w' of w_0.

Now, suppose that v belongs to two non-trivial blocks. Then either both blocks are children of v, or one is a parent of v and one a child of v. In the former case, by Rule 5, we have $\mathsf{b} \notin \ell^*(v)$, and in the latter case, by Rule 6, also $\mathsf{b} \notin \ell^*(v)$.

Finally, let H be a non-trivial child of v. If $\mathsf{r} \in \ell^*(v)$, then we clearly must have a split partition $X \cup Y$ of H respecting ℓ^* with $v \in X$, as otherwise r would be removed from the list of v by the Block Rule. Similarly, if $\mathsf{b} \in \ell^*(v)$, it follows that there must exist a split partition $X \cup Y$ of H respecting ℓ^* with $v \in Y$. Note that here we use the fact that v is processed after all the other vertices of H are processed. □

Finally, we show the correctness of Algorithms 3.4 and 3.1, in that order.

Proposition 3.8. *If* $\ell^*(v) \neq \emptyset$, *for all* $v \in V(G)$, *then Algorithm 3.4 correctly outputs a monopolar partition of* G *respecting* ℓ^*.

Proof. Let $S^{(t)}$ and $\ell^{(t)}$ denote the set S and the lists ℓ during the execution of the algorithm after t iterations of the while-loop. Let $G^{(0)} = G\big[\{root\}\big]$, and for $t \geq 1$, let $G^{(t)}$ denote a subgraph of G induced on the vertices of the blocks of G which contain at least one vertex of $S^{(t)}$. Observe that the graph $G^{(t)}$ is precisely the graph on all vertices of G whose list was altered by the algorithm in the first t iterations of the while-loop. Hence, for all $v \notin V\big(G^{(t)}\big)$, we have $\ell^{(t)}(v) = \ell^*(v)$. In what follows, we shall use this fact frequently.

Let $Q^{(t)}$ denote the vertices of G having all ancestor vertices in $S^{(t)}$. Observe that $Q^{(t)} \subseteq V\big(G^{(t)}\big)$. Let $A^{(t)}$ denote the vertices v of $G^{(t)}$ with $\ell^{(t)}(v) = \{\mathsf{r}\}$, and let $D^{(t)}$ denote the vertices v of $G^{(t)}$ with $\ell^{(t)}(v) = \{\mathsf{b}\}$.

We now prove that the following claims are true for all t.

(i) For all $v \in Q^{(t)}$, if $v \neq root$, then either the parent of v is a non-trivial block, or v is a trivial grandchild of w, and $\ell^{(t)}(v) \cap \ell^{(t)}(w) = \emptyset$.

(ii) $A^{(t)} \cup D^{(t)}$ is a monopolar partition of $G^{(t)}$ respecting ℓ^*.

We prove the claims by induction on t. If $t = 0$, then $V\big(G^{(t)}\big) = \{root\}$, $S^{(t)} = \emptyset$, $Q^{(t)} = \{root\}$, and $\ell^{(t)}(root) \subseteq \ell^*(root)$ is either $\{\mathsf{r}\}$ or $\{\mathsf{b}\}$. In both cases, both claims are satisfied. Hence, suppose that $t \geq 1$, and assume the claims hold for $t-1$. Let v be the vertex picked in the t-th iteration of the while-loop. Observe that $v \in Q^{(t-1)}$. Also, observe that, since the lists of the vertices of $G^{(t-1)}$ are not altered in the t-th iteration of the while-loop, we have that $A^{(t-1)} \subseteq A^{(t)}$, and $D^{(t-1)} \subseteq D^{(t)}$.

Now, let $x_1, \ldots, x_r$ and $H_1, \ldots, H_p$ be the trivial grandchildren and the non-trivial children of v, respectively. Let $X_1 \cup Y_1, \ldots, X_p \cup Y_p$ be the split partitions of $H_1, \ldots, H_p$ respecting $\ell^{(t-1)}$ which are used by the algorithm. Note that since $A^{(t-1)} \cup D^{(t-1)}$ is a monopolar partition of $G^{(t-1)}$ respecting ℓ^*, and $v \in G^{(t-1)}$, we have $\ell^{(t-1)}(v) \subseteq \ell^*(v)$. Also, for any $i \in \{1 \ldots p\}$, all vertices w of H_i except v are not in $G^{(t-1)}$, and hence $\ell^{(t-1)}(w) = \ell^*(w)$. Hence, it follows from Proposition 3.7, that the split partitions $X_1 \cup Y_1, \ldots, X_p \cup Y_p$ of $H_1, \ldots, H_p$ must exist, and they all respect ℓ^*.

First, suppose that $\ell^{(t)}(v) = \{\mathsf{r}\}$. Hence, $A^{(t)} = A^{(t-1)} \cup X_1 \cup \ldots \cup X_p$, and $D^{(t)} = D^{(t-1)} \cup \{x_1\} \cup \ldots \cup \{x_r\} \cup Y_1 \cup \ldots \cup Y_p$. Observe that $Q^{(t)}$ contains the vertices $x_1, \ldots, x_r$ as well as all children of $H_1, \ldots, H_p$. It follows immediately that the claim (i) is satisfied. We now show that (ii) is also true. Since $\mathsf{r} \in \ell^{(t)}(v) = \ell^{(t-1)}(v) \subseteq \ell^*(v)$, by Proposition 3.7, we have $\mathsf{b} \in \ell^*(x_i)$, and hence $\ell^{(t)}(x_i) \subseteq \ell^*(x_i)$, for all $i \in \{1 \ldots r\}$. Moreover, since the split partitions of $H_1, \ldots, H_p$ respect ℓ^*, and $A^{(t-1)} \cup D^{(t-1)}$ respects ℓ^*, it follows that $A^{(t)} \cup D^{(t)}$ also respects ℓ^*. It remains to show, that $A^{(t)} \cup D^{(t)}$ is a monopolar partition. Observe that v is clearly separating the vertices of $A^{(t-1)}$ and the vertices $X_1 \cup \ldots \cup X_p$. Also, v is separating the sets X_i, X_j, for $i \neq j$. Since, in addition, both $A^{(t-1)}$ and the sets $X_1, \ldots, X_p$ are independent, we obtain that $A^{(t)}$ must be an independent set. Similarly, we obtain that the set $D^{(t)}$ induces a disjoint union of cliques, since v separates any two of the sets $D^{(t-1)}, \{x_1\}, \ldots, \{x_r\}, Y_1, \ldots, Y_p$, and each of these sets induces a disjoint union of cliques.

Next, suppose that $\ell^{(t)}(v) = \{\mathsf{b}\}$ and the vertex w_0 does not exist. Hence, $A^{(t)} = A^{(t-1)} \cup \{x_1\} \cup \ldots \cup \{x_r\} \cup X_1 \cup \ldots \cup X_p$, and $D^{(t)} = D^{(t-1)} \cup Y_1 \cup \ldots \cup Y_p$. Again, the claim (i) follows immediately, and since $\mathsf{b} \in \ell^{(t)}(v) = \ell^{(t-1)}(v) \subseteq \ell^*(v)$, by Proposition 3.7, we have that $\mathsf{r} \in \ell^*(x_i)$, for all $i \in \{1 \ldots r\}$. Since, in addition, the split partitions of $H_1, \ldots, H_p$ respect ℓ^*, and $A^{(t-1)} \cup D^{(t-1)}$ respects ℓ^*, we obtain that $A^{(t)} \cup D^{(t)}$ respects ℓ^*. Now, since v separates any two of the sets $A^{(t-1)}, \{x_1\}, \ldots, \{x_r\}, X_1, \ldots, X_p$, and each is an independent set, we have that $A^{(t)}$ must be an independent set. Now, we show that $D^{(t)}$ induces a disjoint union of cliques. Since $\mathsf{b} \in \ell^*(v)$, by Proposition 3.7, we must have that v either has no non-trivial children, that is, $p = 0$, or it has exactly one non-trivial child, $p = 1$, and has a trivial parent. If v has no non-trivial children, then $D^{(t)} = D^{(t-1)}$, and the claim follows. On the other hand, if v has a exactly one non-trivial child, and a trivial parent, then v is a grandchild of w, and since $v \in Q^{(t-1)}$, we have that $\ell^{(t-1)}(v) \cap \ell^{(t-1)}(w) = \emptyset$. It follows that $w \in A^{(t-1)}$, and we observe that $v \in Y_1$. Hence, $D^{(t)} = (D^{(t-1)} \setminus \{v\}) \cup Y_1$, and the vertex w clearly separates the set $D^{(t-1)} \setminus \{v\}$ from Y_1. Since each of these sets induces a disjoint union of cliques, the claim again follows.

Finally, suppose that $\ell^{(t)}(v) = \{\mathsf{b}\}$ and the vertex w_0 exists. Without loss of generality, we may assume that $x_r = w_0$. Let $y_1, \ldots, y_s$ be the trivial grandchildren of w_0. Hence, we have $A^{(t)} = A^{(t-1)} \cup \{x_1\} \cup \ldots \{x_{r-1}\} \cup \{y_1\} \cup \ldots \cup \{y_s\}$, and $D^{(t)} = D^{(t-1)} \cup \{w_0\}$. Since $\mathsf{b} \in \ell^{(t)}(v) = \ell^{(t-1)}(v) \subseteq \ell^*(v)$, by Proposition 3.7, we have $\mathsf{r} \in \ell^*(x_i)$, for all $i \in \{1 \ldots r-1\}$, and $\mathsf{r} \in \ell^*(y_j)$, for all $j \in \{1 \ldots s\}$. Now, it follows that $A^{(t)} \cup D^{(t)}$ respects ℓ^*, since also $A^{(t-1)} \cup D^{(t-1)}$ respects ℓ^*. We observe that the vertices v and w_0 clearly separate any two of the sets $A^{(t-1)}, \{x_1\}, \ldots, \{x_{r-1}\}, \{y_1\}, \ldots, \{y_s\}$, and since any of these sets is an independent set, we obtain that $A^{(t)}$ must be an independent set. Now, again by Proposition 3.7, since $\mathsf{b} \in \ell^*(v)$, we have that both v and w_0 have no non-trivial children, and v has a trivial parent. It follows that $A^{(t)} \cup D^{(t)} = V(G^{(t)})$, and that $Q^{(t)} = \{x_1, \ldots, x_{r-1}, y_1, \ldots, y_s\}$; hence, the claim (i) follows. Also, we have that v must be a grandchild of a vertex w, and since $v \in Q^{(t-1)}$, we have $\ell^{(t-1)}(v) \cap \ell^{(t-1)}(w) = \emptyset$, and hence $w \in A^{(t-1)}$. We again observe that w separates the set $D^{(t-1)} \setminus \{v\}$ from $\{v, w_0\}$, and since each induces a disjoint union of cliques, we obtain that also $D^{(t)}$ induced a disjoint union of cliques, and that completes the proof of (i) and (ii).

Now, since the partition $A \cup D$ returned by the algorithm is $A^{(t)} \cup D^{(t)}$ for some t, the claim (ii) implies the correctness of Algorithm 3.4. □

Theorem 3.9. *Algorithm 3.1 is correct.*

Proof. The correctness of Algorithm 3.1 now follows easily. First, suppose that G admits a monopolar partition $A \cup D$ respecting ℓ_0. By Proposition 3.6, we have that $A \cup D$ also respects ℓ^*, and hence, $\ell^*(v) \neq \emptyset$ for all $v \in V(G)$, and the algorithm will answer correctly.

On the other hand, suppose that G does not admit any monopolar partition respecting ℓ_0, and let ℓ^* be the reduced lists computed by the algorithm. If $\ell^*(v) \neq \emptyset$, for all $v \in V(G)$, it follows from Proposition 3.8, that Algorithm 3.4 will produce a monopolar partition $A \cup D$ of G respecting ℓ^*. By Proposition 3.6, the partition $A \cup D$ must also respect ℓ_0. That leads to a contradiction. Hence $\ell^*(v) = \emptyset$ for some vertex $v \in V(G)$, and the algorithm will again answer correctly. □

We now briefly discuss the complexity of Algorithm 3.1. It is known that all blocks of a connected graph can be obtained in time $O(n + m)$ [65]. Hence, constructing the block-vertex tree clearly also takes $O(n+m)$ time. Now, we look at Algorithm 3.2. As mentioned already, there exists an $O(n + m)$ time algorithm for list split partition problem. This is used as a subroutine in Algorithm 3.2, but only once on each block of G, and hence, it can be easily argued that this takes in total $O(n+m)$ time. Next, it can be seen that Rules 1 - 8 can be implemented in time $O(deg(v))$ for any vertex v, if we, in addition, store in each vertex after its processing whether it has a non-trivial child, and whether it has a trivial grandchild w with $\ell(w) = \{\mathsf{b}\}$. This implies that Algorithm 3.2 can be easily implemented in time $O(n + m)$. Similarly, it can be argued that also Algorithm 3.4 can be implemented in time $O(n + m)$. Hence, it follows that the time complexity of Algorithm 3.1 is $O(n + m)$.

Note that here we assume that the graph G is connected. However, it can be seen that, if G is disconnected, we can run the algorithm on each of its connected components, and then combine the results. Clearly, the time complexity is still $O(n + m)$.

So, we can summarize the results of this section in the following theorem.

Theorem 3.10. (Monopolar chordal graphs) *There is an $O(n + m)$ time algorithm to decide, for a chordal graph G with lists ℓ, whether G admits a monopolar partition which respects ℓ, and to find such partition if exists. In particular, the algorithm can be used to decide, whether a chordal graph is monopolar.*

3.2 Polar Chordal Graphs

Recall, that a partition of the vertex set $V(G)$ of a graph G into sets $V_1 \cup V_2$ is *polar*, if V_1 induces a $\overline{P}_3$-free graph, and V_2 induces a P_3-free graph. It can be seen that a P_3-free graph is a disjoint union of cliques, and a $\overline{P}_3$-free graph is a complete multipartite graph. We say that a graph G is polar, if G admits a polar partition.

We remark that, a complete multipartite chordal graph G, that is, the join of independent sets $W_1, \ldots, W_\ell$, cannot have more than one vertex in more than one set W_i. (Otherwise, the vertices $u_1, u_2 \in W_i$, and $v_1, v_2 \in W_j$, $j \neq i$, yield an induced four-cycle u_1, v_1, u_2, v_2 in G.) Hence, a chordal graph G is $\overline{P}_3$-free, if and only if, G is the join of a clique and an independent set. (The clique represents all parts W_i having just one vertex.)

We adopt the following convenient notation. We shall refer to the vertices of V_1 and V_2 as *red* and *blue* vertices, respectively. We shall refer to the red independent set as A, the red clique as B, and the blue cliques of the disjoint union as $C_1, \ldots, C_k$. Hence, we say that a graph is polar, if it can be partitioned into a red independent set A, a red clique B, and blue cliques $C_1, \ldots, C_k$, where A and B are completely adjacent, and there are no edges between any two different cliques C_i, C_j. Using this notation, it can be seen that a polar partition is also a monopolar partition (see Section 3.1), if and only if, it has the red clique B empty, $B = \emptyset$. Similarly, a polar partition is also a unipolar partition (see Section 1.4), if and only if, it has the red independent set A empty, $A = \emptyset$. These correspondences will play an important role later in the algorithm.

In what follows, we describe a polynomial time algorithm for recognizing polar chordal graphs. In fact, our algorithm solves the more general case of the list polar partition problem in the class of chordal graphs. Hence, we say that a graph G with lists $\ell(v)$, for all $v \in V(G)$, is *list polar*, if G admits a polar partition $A, B, C_1, \ldots, C_k$ which *respects* the lists ℓ, that is, for each $v \in A \cup B$, we have $\mathsf{r} \in \ell(v)$, and for each $v \in C_1 \cup \ldots \cup C_k$, we have $\mathsf{b} \in \ell(v)$.

Let G be a chordal graph with lists $\ell(v)$, for all $v \in V(G)$, and let M be a clique of G. We say that M is *good*, if there exists a polar partition $A, B, C_1, \ldots, C_k$ of G which respects ℓ, and $B = M$. Note that, if M is a good clique, then $G - M$ is monopolar. We say that M is *nice*, if there exists polar partition $A, B, C_1, \ldots, C_k$ of G which respects ℓ, and

(M1) $M = B \cup T$ where $T \subseteq C_1$, and

(M2) for each connected component K of $G - M$, if K contains a vertex $a \in A$, then we have $N(a) \cap M \supsetneq N(b) \cap M$ for each vertex b of $K - a$.

Note that in both cases $A \cap M = \emptyset$. We say that a clique M is *almost good*, if it is good or if $M \setminus \{a\}$ is good for some $a \in M$, and that a clique M is *almost nice*, if it is nice or if $M \setminus \{a\}$ is nice for some $a \in M$.

We have the following structural characterization of polar chordal graphs.

Proposition 3.11. (Polarity of chordal graphs) *Let G be a chordal graph with lists $\ell(v) \subseteq \{\mathsf{r}, \mathsf{b}\}$, for all $v \in V(G)$. Then G is list polar, if and only if, at least one of the following conditions holds.*

(i) G admits a monopolar partition which respects lists ℓ.

(ii) G admits a unipolar partition which respects lists ℓ.

(iii) G has two non-adjacent vertices u, v such that $N(u) \cap N(v)$ is an almost good clique.

(iv) G contains a maximal clique that is almost nice.

Proof. First, we observe that, by the definitions, any of the conditions $(i) - (iv)$ implies that G with lists ℓ is list polar. To show the converse, we assume that G is list polar, and the conditions $(i) - (iii)$ are not fulfilled. We show that this implies that (iv) must be true.

Hence, consider a polar partition of G which respects ℓ and has parts $A, B, C_1, \ldots, C_k$. As usual, we assume that the vertices of A, B are coloured red, and the vertices of $C_1, \ldots, C_k$ are blue. Let T be the set of all vertices of $C_1, \ldots, C_k$ which are adjacent to each vertex of B. Since we assume that the condition (ii) is not fulfilled, the set A has to have at least two vertices, and any two vertices u, v of A must have a common blue neighbour z, otherwise $N(u) \cap N(v) = B$ is a good clique, contradicting (iii). We now observe that z must be adjacent to every vertex w in B, else we would have the induced four-cycle u, z, v, w without chords. In other words, $z \in T$, and hence the set T must be non-empty.

Now, we show that any two vertices u, v of T must be adjacent. Suppose otherwise, and let $N = N(u) \cap N(v)$. By the definition of T, we have $N \supseteq B$. We observe that N cannot contain any blue vertex w, since otherwise u, w, v is a blue P_3. Now, it follows that either N or $N \setminus \{a\}$, for some $a \in A$, is equal to B, since A is an independent set. Hence, $N = N(u) \cap N(v)$ is an almost good clique, which contradicts (iii).

It follows that T induces a clique in G, and since it consist only of blue vertices, it must be contained in some blue clique of the polar partition, without loss of generality, the clique C_1. Hence, we have that $M = T \cup B$ is a clique with $T \subseteq C_1$.

Now, we show that each connected component of $G - M$ contains at most one red vertex. Suppose otherwise, and let K be a connected component of $G - M$ that contains two distinct

red vertices; let u and v be two closest such vertices. Note that, since u, v must be in A, they are not adjacent. Now, since K is connected, there exists in K a shortest path P from u to v (of length at least two); it follows from the choice of u, v that $P - \{u, v\}$ contains only blue vertices. Since B is not empty, consider the cycle wPw for any $w \in B$. (Since u, v are in A, they must be adjacent to w). By chordality of G, and the fact that P is a shortest (and hence induced) path, w must be adjacent to all vertices of P. It now follows that each vertex x of $P - \{u, v\}$ must be adjacent to all vertices w of B, and hence x belongs to $T \subseteq M$, contradicting the fact that x is in $G - M$. (Recall that T consists of those blue vertices which are completely adjacent to B.) Thus, every component K of $G - M$ has at most one red vertex.

Now, let K be a connected component of G that contains a red vertex $a \in A$. Since a belongs to A, it must be adjacent to each vertex of B, and hence $N(a) \cap M \supseteq B$. By the previous paragraph, K cannot contain more than one red vertex, and hence $K - a$ is a disjoint union of cliques which consist of blue vertices only. Now, let b be any vertex of $K - a$. Since b is blue, it must belong to some clique C_i.

Suppose first that $i \neq 1$. Since $T \subseteq C_1$, the vertex $b \in C_i$ is not adjacent to any vertex in T. It follows that $N(b) \cap M \subseteq B$. Now, if $N(b) \cap M = B$, then, for any $z \in T$, $N = N(z) \cap N(b)$ is an almost good clique; this is because $N(z) \supseteq B$, and b and z are non-adjacent blue vertices, which implies that either $N = B$ or $N = B \cup \{a\}$. (Note that the only red vertex that b can be adjacent to is a.) This contradicts *(iii)*, and therefore, we must have $N(b) \cap M \subsetneq B$, which implies $N(a) \cap M \supsetneq N(b) \cap M$.

Now, suppose that $i = 1$. Let c be the first vertex after a on a shortest path from a to b in K (possibly $c = b$). Clearly, we must have $c \in C_1 \setminus T$. Now, since C_1 is a clique, we have that $N(c) \cap M \supseteq T$. Since $c \notin T$, there must exist a vertex $w \in B$, which is not adjacent to c. Now, it follows that a must be adjacent to each vertex z in T, since otherwise a, w, z, c is an induced four-cycle. Hence, $N(a) \cap M = M$. Now, since also $b \notin T$, we must have $N(b) \cap B \neq B$, and hence $N(b) \cap M \neq M$, which implies $N(a) \cap M \supsetneq N(b) \cap M$.

It remains to observe that M is either a maximal clique of G, or becomes a maximal clique of G by the addition of a single vertex a of A, in which case $M \cup \{a\}$ is almost nice. This follows from the fact that A is an independent set, and that no blue vertex can be completely adjacent to M, since, by the definition of T, it would already be in $T \subseteq M$. □

Next, we describe how to test whether a clique B is good.

Proposition 3.12 (Good Clique). *Let B be a clique in a chordal graph G with lists $\ell(v) \subseteq \{\mathsf{r}, \mathsf{b}\}$, for all $v \in V(G)$. Remove r from the list $\ell(v)$ of each vertex v in $G - B$ which is not completely adjacent to B, and denote these modified lists ℓ^*. Then B is good, if and only if, $G - B$ admits a monopolar partition which respects the (modified) lists ℓ^*.*

Proof. If there exists a monopolar partition $A \cup D$ of $G - B$ which respects ℓ^*, then $A, B, C_1, \ldots, C_k$, where $C_1, \ldots, C_k$ are the connected components of $G[D]$, is clearly a polar partition of G, since for each vertex $v \in A$, we have $\mathsf{r} \in \ell^*(v)$, and hence, v is completely adjacent to B. Conversely, if B is good, then there must exist a polar partition $A, B, C_1, \ldots, C_k$ of G which respects ℓ. This implies that $\mathsf{r} \in \ell(v)$ for each $v \in A$. Hence, $A \cup D$, where $D = C_1 \cup \ldots \cup C_k$, is a monopolar partition of $G - B$, and it respects ℓ^*, because the vertices of A are completely adjacent to B, and hence, $\mathsf{r} \in \ell^*(v)$ for each $v \in A$. □

Now, it clearly follows from Theorem 3.10 that testing for a good clique can be implemented in $O(n + m)$ time.

In the remainder of this section, we describe how to test whether a maximal clique M is almost nice, assuming we have a chordal graph G with lists $\ell(v) \subseteq \{\mathsf{r}, \mathsf{b}\}$, for all $v \in V(G)$, such that G does not admit a monopolar, or a unipolar partition which respects ℓ, and has no non-adjacent vertices u, v for which $N(u) \cap N(v)$ is an almost good clique.

We proceed as follows. First, for each connected component K of G, we search for a vertex $v_K \in V(K)$ such that $N(v_K) \cap M \supsetneq N(x) \cap M$, for all $x \in V(K)$, $x \neq v_K$. Clearly, there cannot be two such vertices, so, if there is none, we set $v_K = nil$. If this happens, and K is not a clique, then we must reject M. Also, we reject M, if $K - v_K$ is not a disjoint union of cliques. Otherwise, we precolour by blue the vertices of $K - v_K$, and if K is not a clique, we also precolour by red the vertex v_K. Note that at this point, all but possibly a single vertex of each connected component of $G - M$ is precoloured by blue, and the blue precoloured vertices form a disjoint union of cliques. Now, we colour by blue each vertex $v \in V(G)$ with $\mathsf{r} \notin \ell(v)$, and colour by red each vertex $v \in V(G)$ with $\mathsf{b} \notin \ell(v)$. If some vertex receives both colours by the above steps, we reject M.

Otherwise, we proceed to finding the clique C_1. In particular, we want the set $C = C_1 \setminus T$ which belongs to $G - M$. For this, we try all possible choices, that is, either $C = \emptyset$, or $C = V(K)$, where K is a clique connected component of $G - M$, or C is a connected component of $K - v_K$ for some connected component K of $G - M$. For any such choice, we precolour by blue all vertices of C, and precolour by red each vertex of M which is not

adjacent to at least one vertex of C. Then, we propagate the colours of the vertices using the following rules as long as possible.

- **Propagation Rule 1.** If v in $G - M$ is red, then all its non-neighbours in M are blue.
- **Propagation Rule 2.** If u in M is blue, then all its neighbours in $G - (M \cup C)$ are red.

If a vertex receives both red and blue colours by the above, then we declare M not nice. Otherwise, we colour red all uncoloured vertices in M, and colour blue all uncoloured vertices in $G - M$. Then, we set A to consist of all red vertices in $G - M$, set B to consist of all red vertices in M, set $C_1, \ldots, C_k$ to be the connected components of $G - A - B$, and we declare M nice. The details of this algorithm are summarized as Algorithm 3.5. Now we prove its correctness.

Proposition 3.13 (Nice Clique). *Algorithm 3.5 is correct.*

Proof. Let $A, B, C_1, \ldots, C_k$ be the sets computed by Algorithm 3.5 when it declares M nice. Observe that the partition $A, B, C_1, \ldots, C_k$ clearly satisfies the conditions (M1), and (M2) in the definition of nice clique. It also clearly respects the lists ℓ. Hence, it remains to show that $A, B, C_1, \ldots, C_k$ forms a polar partition of G.

First, we show that A must be an independent set. This follows easily, because A is formed by the red vertices in $G - M$, and hence, from each connected component K of $G - M$, it can only contain at most one red vertex v_K (all other vertices of K must be blue by our precolouring). Similarly, it follows that B is a clique, since $B \subseteq M$, and M is a clique. Also, it can be seen that A is completely adjacent to B. This follows immediately, since by Propagation Rule 1, for any red vertex v in $G - M$, any non-neighbour of v in M will be coloured blue, and B consists of the red vertices in M. Finally, suppose that there exist blue vertices u, v, w in G forming a P_3, that is, $uv, vw \in E(G)$ but $uw \notin E(G)$. Clearly, the three vertices cannot be all in $G - M$, nor they can be all in M, which follows from the precolouring rules, and the fact that M is a clique. Hence, suppose that u, v belong to a connected component K of $G - M$, and $w \in M$. If $u \in C$, then clearly w must be red, since by the precolouring rules, any vertex of M that is non-adjacent to at least one vertex of C is coloured red, and w is not adjacent to u, a contradiction. Similarly, if $u \notin C$, then also $v \notin C$, since they are both blue and both in the same connected component of $G - M$. Hence, by Propagation Rule 2, v must be red, since $w \in M$ and v is a neighbour of w in $G - (M \cup C)$, again a contradiction. Hence, suppose that u is in K, and $v, w \in M$. By the

same argument, if $u \in C$, then w must be red, and if $u \notin C$, then by Propagation Rule 2, u must be red. Hence, it follows that u and w must be in $G - M$, and $v \in M$. Since u and w are not adjacent, they both cannot be in C; without loss of generality, suppose that $u \notin C$. Now, by Propagation Rule 2, u must be red, since $v \in M$, yielding again a contradiction. Therefore, the blue vertices must form a P_3-free graph, which proves that $A, B, C_1, \ldots, C_k$ is a polar partition of G.

On the other hand, let M be a nice clique, and let $A, B, C_1, \ldots, C_k$ be a fixed polar partition which respects ℓ and satisfies (M1), and (M2) for M. We show that M will be declared nice by Algorithm 3.5. We also show that the following invariant is maintained during the execution of the algorithm, for the choice of $C = C_1 \setminus M$.

$$\begin{array}{c} \{\textit{the red coloured vertices of } G\} \subseteq A \cup B \textit{ and} \\ \{\textit{the blue coloured vertices of } G\} \subseteq C_1 \cup \ldots \cup C_k \end{array} \qquad (*)$$

Note that at the beginning of the algorithm, no vertex is coloured, and hence $(*)$ holds. The algorithm starts by searching for vertices $v_K \in V(K)$, for each connected component K of $G - M$. Observe that it follows from $(M2)$ that any vertex of K, which also belongs to A, satisfies the conditions for v_K, and there can be at most one such vertex in K. So, if K contains a vertex a from A, then, for such K, the algorithm must successfully find $v_K = a$. Also $V(K - v_K) \subseteq C_1 \cup \ldots \cup C_k$, and hence, $K - v_K$ must be a disjoint union of cliques. On the other hand, if K does not contain any vertex from A, then $V(K) \subseteq C_1 \cup \ldots \cup C_k$, and K must be a clique. This shows that the algorithm will not reject M and will proceed to precolouring the components of $G - M$ at which point the condition $(*)$ will be satisfied.

Next, the algorithm colours by red the vertices of the set $X = \{v \in V(G) \mid \mathsf{b} \notin \ell(v)\}$, and by blue the vertices of the set $Y = \{v \in V(G) \mid \mathsf{r} \notin \ell(v)\}$. The fact that $A, B, C_1, \ldots, C_k$ respects ℓ immediately gives that $X \subseteq A \cup B$, and $Y \subseteq C_1 \cup \ldots \cup C_k$. This shows the algorithm will not reject M, and also that $(*)$ will be satisfied.

Now, let $C = C_1 \setminus M$, and let $T = C_1 \cap M$. Since C clearly induces a clique in $G - M$, C must completely belong to a connected component K of $G - M$. Note that any neighbour of C in $G - M$ must belong to A. Hence, either $C = \emptyset$, or $C = K$, or C is a connected component of $K - v_K$, since only v_K (if exists) can belong to A. It now follows that the algorithm will correctly find C as one of the possible choices it investigates. After that, the algorithm colours by blue the vertices of C, and colours by red the vertices of M which are not completely adjacent to C. Since $C_1 = C \cup T$ is a clique, we have that each vertex of

T is completely adjacent to C, so any vertex of M non-adjacent to at least one vertex in C must belong to B. (Recall that, by (M1), we have $M = B \cup T$.) This shows that after colouring C and its non-neighbourhood, the condition $(*)$ is again satisfied.

Now, the algorithm applies the propagation rules. We show, by induction on the number of applications of the rules, that $(*)$ will be maintained during this process. Clearly, before applying the rules, $(*)$ is true. Now, assume that $(*)$ is true, and a propagation rule is applied. First, suppose that Propagation Rule 1 is applied, that is, we have a red vertex in $G - M$, and a non-adjacent vertex u in M which is given blue colour by the rule. Since v is red, by $(*)$, we must have $v \in A$. Suppose that u is red before the application of the rule. By $(*)$, we have that $u \in B$. However, since the vertices of A and B are completely adjacent, we obtain a contradiction, since u and v are not adjacent. Hence, $u \in T$, and therefore $u \in C_1 \cup \ldots \cup C_k$. This shows that $(*)$ is true after the rule is applied. Now, suppose that Propagation Rule 2 is applied, that is, we have a blue vertex $v \in M$, and a neighbouring vertex u in $G - (M \cup C)$ which is coloured red by the rule. Since v is blue, by $(*)$, we have $v \in C_1$. Suppose that u is red before the rule is applied. By $(*)$, we must have $u \in C_1 \cup \ldots \cup C_k$. However, $u \notin M$ and $u \notin C$, and hence, $u \notin C_1$. Therefore, $u \in C_i$ for some $i \neq 1$. However, since the vertices of C_i and C_1 are non-adjacent, and v and u are adjacent, we obtain a contradiction. Hence, $u \in A$ which shows that $(*)$ is true after the rule is applied.

Now, it follows from $(*)$ that no vertex will receive both colours by the propagation rules, and hence, M will be declared nice by the algorithm, which concludes the proof. □

Proposition 3.14. *Algorithm 3.5 has time complexity $O(n(n+m))$.*

Proof. First, we find the connected components of $G - M$ and identify, which of them are cliques, easily in time $O(n+m)$. Then, for each connected component K of $G - M$, we find a vertex $x_K \in V(K)$, who has the most neighbours in M among the vertices of K. We then test whether $N(x_k) \cap M \supsetneq N(b) \cap M$ is true for each $b \in V(K)$, $b \neq x_K$. If the test succeeds, we set $v_K = x_K$, otherwise, we set $v_K = nil$. For any vertex b, finding the number of neighbours of b in M clearly takes $O(\deg(b))$. Testing $N(x_K) \cap M \supsetneq N(b) \cap M$ for $b \in V(K)$ also takes $O(\deg(b))$. It clearly follows that finding the vertices v_K for all components K of $G - M$ takes $O(n+m)$ time.

Next, it can be seen that there are at most n different choices for C. For each choice of C, we colour the vertices of C by blue, and colour by red any vertex of M which has a

non-neighbour in C. This can be done, by computing $X = \left(\bigcap_{u \in C} N(u)\right) \cap M$, and colouring by red all vertices of $M \setminus X$; both steps in time $O(n+m)$.

We now apply the propagation rules. In what follows, we show how to implement this step in time $O(n+m)$, which will imply the claim. We say that a vertex v is *weak*, if v is either a red vertex of $G - M$, and all its non-neighbours in M are blue, or v is a blue vertex of M, and all its neighbours in $G - M - C$ are red. Clearly, if a vertex is weak, there is no need to apply any propagation rules to it any more.

We start by computing the set S of all red vertices of $G - M$ in time $O(n)$. We then colour by blue each $u \in M$ such that $S \setminus N(u) \neq \emptyset$. Clearly, if, for $u \in M$, we have $v \in S \setminus N(u)$, then by Propagation Rule 1, u must be blue. Hence, after this, all vertices of S become weak. For each $u \in M$, testing $S \setminus N(u) \neq \emptyset$, can be easily done in $O(\deg(u))$ time by computing $S \cap N(u)$, and hence, $O(n+m)$ altogether for all $u \in M$.

Note that, at this point, each red vertex of $G - M$ is weak. We now choose a blue vertex v of M, and perform a series of operations after which v will become weak, and all red vertices in $G - M$ will again be weak. (Note that, in the process, some vertices of $G - M$ may become red, and some vertices of M may become blue.)

Hence, let v be any blue unprocessed vertex of M. We first compute, in time $O(\deg(v))$, the set $S_v = N(v) \setminus (M \cup C)$, and then remove from S_v all red coloured vertices, again, in time $O(\deg(v))$. If $S_v \neq \emptyset$, we colour the vertices of S_v by red, compute the set $X_v = \left(\bigcap_{u \in S_v} N(u)\right) \cap M$, and colour by blue each vertex $w \in M \setminus X_v$. Clearly, the set X_v can be computed in time $O\left(\sum_{u \in S_v} \deg(u)\right)$, and colouring $M \setminus X_v$ takes $O(\deg(v))$ time. After these steps, v becomes weak, because all neighbours of v in $G - (M \cup C)$ either belong to S_v or were already red. Also, each vertex u in S_v becomes weak, since each non-neighbour of u, which is in M, must belong to $M \setminus X_v$. Moreover, each $w \in M \setminus X_v$ must be non-adjacent to at least one $u \in S_v$, and hence, by Propagation rule 1, w must necessarily be blue. Finally, we observe that, again, all red vertices in $G - M$ are weak.

We repeat the above process until there are no more blue unprocessed vertices in M. We now look at the complexity of this procedure. Let $v_1, \ldots, v_t$ be all blue vertices we processed using the above procedure, in the order in which they were processed. Observe, that for each $i < j$, we must have $S_{v_i} \cap S_{v_j} = \emptyset$, since any $w \in S_{v_i} \cap S_{v_j}$ would have been given red colour when v_i was processed, and hence, it would have been removed from S_{v_j}, when processing v_j. As we discussed earlier, each v_i is processed in time $O\left(\deg(v_i) + \sum_{u \in S_{v_i}} \deg(u)\right)$. Hence, altogether $O\left(\sum_{i=1}^{t} \left(\deg(v_i) + \sum_{u \in S_{v_i}} \deg(u)\right)\right) \leq O\left(\sum_{v \in V(G)} \deg(v)\right) = O(n+m)$. $\square$

Algorithm 3.5: Testing whether a clique is nice.

Input: A clique M in a chordal graph G with list $\ell(v) \subseteq \{\mathsf{r}, \mathsf{b}\}$, for all $v \in V(G)$.
Output: Answer whether M is nice.

1 Find the connected components of $G - M$
2 **for** each connected component K of $G - M$ **do**
3 Find a vertex $v_K \in V(K)$ with $N(v_K) \cap M \supsetneq N(b) \cap M$ for all $b \in V(K - v_K)$
4 **if** v_K does not exist **then**
5 **if** K is not a clique **then**
 return "M is not nice"
6 $v_K \leftarrow nil$
7 **else** Colour red the vertex v_K
8 Colour blue the vertices of $K - v_K$
9 Colour red each $v \in V(G)$ with $\mathsf{b} \notin \ell(v)$
10 Colour blue each $v \in V(G)$ with $\mathsf{r} \notin \ell(v)$
11 **for** each choice of C — either $C = \emptyset$,
 or $C = V(K)$ for a clique component K of $G - M$,
 or C is a connected component of $K - v_K$ for a
 connected component of K of $G - M$ **do**
12 Colour blue the vertices of C
13 Colour red the vertices of M which are not completely adjacent to C
14 Apply Propagation Rules 1 and 2 as long as possible
15 **if** no vertex receives both colours **then**
16 Colour red all uncoloured vertices of M
17 Colour blue all uncoloured vertices of $G - M$
18 **return** "M is nice"
19 **return** "M is not nice"

The final algorithm for polarity of chordal graphs is summarized below.

Algorithm 3.6: Polar graph recognition.

Input: A chordal graph G with lists $\ell(v) \subseteq \{\mathsf{r}, \mathsf{b}\}$, for all $v \in V(G)$.
Output: Answer whether G is list polar.

1 Test if G admits a monopolar partition which respects ℓ
2 Test if G admits a unipolar partition which respects ℓ
3 **for** each pair of non-adjacent vertices u and v of G **do**
4 Test whether $N(u) \cap N(v)$ is an almost good clique
5 **for** each maximal clique M of G **do**
6 Test whether M is an almost nice clique
7 **if** any of the tests succeeds **then**
return "G is list polar"
8 **else return** "G is not list polar"

The correctness of this algorithm follows from Proposition 3.11. We now analyze its complexity. For unipolarity in general graphs, we have a $O(n^2m)$ time algorithm explained in Theorem 1.31. Similarly, for monopolarity in chordal graphs, we have an $O(n+m)$ time algorithm as shown by Theorem 3.10. Next, we test n^2 times a set for being an almost good clique. Each such test consists of $O(n)$ tests for being a good clique, which according to Proposition 3.12 amounts to testing for monopolarity. In all, we need $O(n^2 \times n \times (n+m))$ time. Finally, we test at most n sets for being an almost nice clique. Again, each such test consists of $O(n)$ tests for being a nice clique, which according to Proposition 3.14 requires time $O(n(n+m))$, and hence, $O(n^2 \times n(n+m))$ time for this step.

It follows that the total time complexity of this algorithm is $O(n^3(n+m))$.

Theorem 3.15. *There is a $O(n^3(n+m))$ time algorithm to test polarity of chordal graphs.* □

Chapter 4

Forbidden Subgraphs for Monopolarity

For cographs, it has been shown in [24] that monopolarity (and polarity) can be characterized by the absence of a finite set of forbidden induced subgraphs. By contrast, for chordal graphs, as we shall see, there are infinitely many minimal non-monopolar graphs. Additionally, since it can be seen that any polar graph of diameter more than six is necessarily monopolar, it will follow that there are also infinitely many minimal non-polar chordal graphs.

Interestingly, it turns out that all chordal minimal non-monopolar graphs can be generated by a simple recursive procedure (which unwinds the operation of the monopolarity recognition algorithm in case it rejects the graph). This turns out to be a particular example of the so-called *hyperedge replacement grammars* (see Section 2.4, and [58]). The grammar Γ constructs a tree-like structure (with at most three branches at any node) consisting of very simple blocks (with at most six vertices each). In fact, we shall prove a more general statement about minimal forbidden monopolar graphs by additionally considering lists.

Let G be a graph with two sets of lists $\ell(v)$ and $\ell'(v)$, for all $v \in V(G)$. We say that ℓ' is a *list extension* of ℓ, if $\ell(v) \subseteq \ell'(v)$, for all $v \in V(G)$. In particular, ℓ' is a *proper list extension* of ℓ, if ℓ' is a list extension of ℓ, and for some $v \in V(G)$, we have $\ell(v) \subsetneq \ell'(v)$. For a graph G with lists ℓ and a graph G' with lists ℓ', we say that G' is an *induced list extension* of G, if G' is an induced subgraph of G, and ℓ' is a list extension of ℓ on G'. Similarly, G' is a *proper induced list extension* of G, if G' is an induced list extension of G, and either G' is a proper induced subgraph of G, or ℓ' is a proper list extension of ℓ on G'.

We can immediately observe that the relation *"being an induced list extension"* is a partial order $\preccurlyeq$ on the graphs with lists. This partial order clearly generalizes the *"induced subgraph"* partial order on graphs (without lists). Note that there is no infinite descending chain in the (induced) subgraph order, but there exists an infinite descending chain in $\preccurlyeq$. However, this is no longer the case, if we only deal with graphs whose lists have bounded size. Therefore, we call the order $\preccurlyeq$ on graphs with lists of size at most k a *k-bounded* $\preccurlyeq$. Now, it follows from Proposition 2.1.1 in [26], that any set $\mathcal{C}$ of graphs (with lists) closed under k-bounded $\preccurlyeq$ for some k, is characterized by avoiding the set $\mathcal{F}_{\preccurlyeq}(\mathcal{C})$ of minimal forbidden $\preccurlyeq$-predecessors, that is, graphs with lists which are not in $\mathcal{C}$, but such that any proper induced list extension of any of them belongs to $\mathcal{C}$. We shall call such graphs *minimal list non-extendable forbidden induced subgraphs* for $\mathcal{C}$.

Now, let G be a graph with lists $\ell(v) \subseteq \{\mathsf{r}, \mathsf{b}\}$, for all $v \in V(G)$. We say that G with lists ℓ is *list monopolar*, if there exists a monopolar partition of G respecting ℓ.

Let $\mathcal{M}$ denote the class of all list monopolar chordal graphs. It can be observed that $\mathcal{M}$ is closed under 2-bounded induced list extension, since if a monopolar partition of a graph respects lists ℓ, it must clearly also respect any list extension ℓ' of ℓ on any induced subgraph. Hence, since the size of lists in $\mathcal{M}$ is at most two, it follows that $\mathcal{M}$ is characterized by the set $\mathcal{F}_{\preccurlyeq}(\mathcal{M})$ of minimal list non-extendable forbidden induced subgraphs. Note that these include all minimal forbidden induced subgraphs for $\mathcal{M}$ (in the usual sense); the graphs in $\mathcal{F}_{\preccurlyeq}(\mathcal{M})$ having lists $\{\mathsf{r}, \mathsf{b}\}$ for each vertex, are precisely the minimal forbidden induced subgraphs for $\mathcal{M}$.

In the remainder of this chapter, we shall completely describe the set $\mathcal{F}_{\preccurlyeq}(\mathcal{M})$ of minimal list non-extendable forbidden induced subgraphs for monopolarity of chordal graphs.

4.1 The two-connected case

First, we focus on 2-connected chordal graphs. Surprisingly, for 2-connected chordal graphs, there is only finitely many minimal list non-extendable forbidden induced subgraphs for monopolarity. The following observation is easy to see.

Observation 4.1. *All configuration in Figure 4.1 are minimal list non-extendable forbidden induced subgraphs for monopolarity.* □

For the proof of the next statement, we shall need the following lemma from [53].

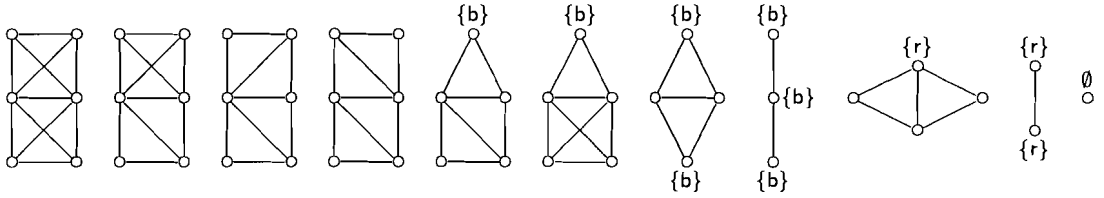

Figure 4.1: All minimal list non-extendable forbidden induced subgraphs for monopolarity of 2-connected chordal graphs.

Lemma 4.2. [53] *Let G be a chordal graph, and let $u_1, u_2, \ldots, u_k$ be a cycle in G. If u_1u_3 is not an edge of G, then u_2u_j is an edge of G for some $4 \leq j \leq k$.*

Theorem 4.3. *Let H be a 2-connected chordal graph with lists $\ell(v)$, $v \in V(H)$, which is not list monopolar. Then H contains one of the configurations in Figure 4.1.*

Proof. First, suppose that H is not monopolar (without considering the lists). Then by Proposition 3.3, H is not split. Recall that a graph that is not split either contains an induced cycle C_4 or C_5, or two independent edges $2K_2$. Hence, since H is chordal and not split, it must contain two independent edges.

Let xx' and yy' be two independent edges of H. We show that they must belong to a cycle of H. Using Menger's theorem (cf. [18]) on the vertices x, x' and y, y', we obtain that there must exist two induced vertex-disjoint paths P, P' between the sets $\{x, x'\}$ and $\{y, y'\}$. Clearly, both x and x' cannot belong to P nor both to P'. The same holds for y, y'. Hence, without loss of generality, we may assume that x, y belong to P, and x', y' belong to P'. It follows that $xPyy'(P')^{-1}x'$ is a cycle in H.

Now, assume that the edges xx' and yy' were picked such that the cycle C containing these edges is smallest possible. We show that it must have exactly six vertices. Suppose otherwise, and assume, without loss of generality, that P contains at least four vertices. Let u be the first vertex on P after x, and v be the last vertex on P before y. Since P contains at least four vertices, we must have $u \neq v$. Hence, since P is an induced path, v is not adjacent to x, and u is not adjacent to y.

Now suppose that x' is not adjacent to v, and that v is adjacent to y'. Let P_v be the subpath of P from x to v. It follows that $xP_vvy'(P')^{-1}x'$ is a cycle in H with two independent edges xx' and vy' whose length is less than the length of C, which contradicts the fact that C was chosen smallest possible. Hence, v is not adjacent to y'. But then, since

H is chordal and y is not adjacent to any vertex on P except v, it follows from Lemma 4.2 that y must be adjacent to a vertex v' of P' other than y'. Let $P'_{v'}$ be the subpath of P' from x' to v'. Again, we obtain that $xPyv'(P'_{v'})^{-1}x'$ is a cycle in H with two independent edges xx' and yv whose length is less than the length of C, which is a contradiction.

Therefore, x' must be adjacent to v. By symmetry, we have that y' must be adjacent to u. But then $xuy'yvx'$ is a cycle in H with independent edges xx' and yy' whose length is exactly six, which is, by our assumption, less than the length of C. This shows that C has length at most six, and clearly, since the edges xx' and yy' are independent, C cannot have length less than six. Finally, by chordality of H, it is easy to show that C is one of the first four configurations in Figure 4.1.

Now, suppose that H is monopolar (without lists) but not list monopolar with lists ℓ. First, if H contains a vertex v with $\ell(v) = \emptyset$, then v itself forms the last configuration in Figure 4.1. Hence, assume that $\ell(v) \neq \emptyset$, for all $v \in V(H)$, and denote by R the vertices v of H with $\ell(v) = \{\mathsf{r}\}$. If R contains adjacent vertices v and v', then H contains the second from the right configuration in Figure 4.1 on the vertices v, v'. Hence, we may assume that R is an independent set of H. Now, suppose that R contains a vertex v adjacent to distinct vertices u and u' such that uu' is not an edge. Then, since H is 2-connected, there must exist an induced path P from u to u' in $H - v$ (of length at least 2). By chordality, v must be adjacent to each vertex of P. Let w, w', w'' be some three consecutive vertices of P. Clearly, v is adjacent to all three of these vertices. Hence, H contains the third from the right configuration in Figure 4.1 induced on the vertices v, w, w', w''.

So, we may assume that the neighbourhood of each vertex of R induces a clique. Now, let B be the set of vertices v of H with $\ell(v) = \{\mathsf{b}\}$. We construct a graph H' from H as follows. Starting with $G = H$, we add, one by one, for each vertex $v \in B$, a new vertex v', and make it adjacent to v and all neighbours of v in (the current) G. The final graph G is the graph H'. Observe that, for any $v \in B$, the vertices v and v' are twins in H'. Hence, it follows from Proposition 7.12, that H' is chordal, since H is chordal. Clearly, H' is also 2-connected. Now, we define lists ℓ' for the vertices of H' as follows. For each $v \in V(H)$, if $\ell(v) = \{\mathsf{r}\}$, then we set $\ell'(v) = \{\mathsf{r}\}$, otherwise we set $\ell'(v) = \{\mathsf{r}, \mathsf{b}\}$, and also set $\ell'(v') = \{\mathsf{r}, \mathsf{b}\}$ if v' exists. (Recall that we assume that $\ell(v) \neq \emptyset$.)

We show that H' with lists ℓ' is list monopolar, if and only if, H with lists ℓ is list monopolar. For the forward direction, let $A \cup D$ be a monopolar partition of H' respecting ℓ'. Clearly, for each vertex $v \in B$, both v and v' cannot belong to A, since A is an

independent set. Hence, without loss of generality, we may assume that $v \in D$, for each $v \in B$. But then $A \cup (D \cap V(H))$ is clearly a monopolar partition of H respecting ℓ. On the other hand, if $A \cup D$ is a monopolar partition of H respecting ℓ, we must have $B \subseteq D$, and hence $A \cup (D \cup B')$, where B' are the vertices v' for $v \in B$, is a monopolar partition of H', which follows from the fact that adding a twin to a vertex of a clique also creates a clique.

Suppose now that H' is not monopolar (without lists). By the first five paragraph of this proof, we have that H' must contain an induced subgraph F' which is one of the first four configurations in Figure 4.1. Now, let F be the graph obtained from F' by contracting the edges vv', for all $v \in B$. Clearly, F is an induced subgraph of H, and it can be observed that F must be one of the first eight configurations in Figure 4.1.

Hence, we may assume that H' is monopolar. By Proposition 3.2, H' is also a split graph. We now show that this implies that H' with lists ℓ' is list monopolar, which by the above equivalence will lead to a contradiction. Since H' is split, there exists a split partition $A \cup D$ of H'. If $R \subseteq A$, then, clearly, $A \cup D$ respects ℓ', and we are done. Hence, we must have a vertex $v \in R$ which belongs to D. Since R is an independent set, no other vertex of R can belong to D. Now, if v is not adjacent to any vertex of A, then we obtain that $(A \cup \{v\}) \cup (D \setminus \{v\})$ is a monopolar partition of H' respecting ℓ'. Hence, v must be adjacent to $u \in A$. Since R is independent, $u \notin R$, and hence, $\ell'(u) = \{\mathsf{r}, \mathsf{b}\}$. Now, since the neighbours of v in H induce a clique, it can be seen that also the neighbours of v in H' must induce a clique. Hence, since $v \in D$, we obtain that u must be adjacent to each vertex of D. Now, this implies that $A' \cup D'$ is a monopolar partition of H' respecting ℓ', where $A' = A \setminus \{u\} \cup \{v\}$, and $D' = D \setminus \{v\} \cup \{u\}$. That concludes the proof. □

We now explain how to find in time $O(n + m)$ a minimal list non-extendable forbidden induced subgraph in a chordal 2-connected graph G with lists ℓ, if G is not list monopolar.

We first run in time $O(n + m)$ the algorithm for testing whether G (without lists) is a split graph. If G is not split, then the algorithm produces a minimal forbidden subgraph F. Since G is chordal, F must consists of two independent edges xx' and yy'. Now, by considering the maximal cliques that contain these edges, and exploring the (unique) path between these cliques in the clique-tree of G, one can find the vertex disjoint paths P and P' connecting xx' to yy', and hence the cycle $C = xPyy'(P')^{-1}x'$. Then using the argument of the proof, one can reduce the cycle C to one of the first four forbidden configurations in Figure 4.1. Since this step amounts to at most $O(n)$ tests for adjacent vertices, it follows

that this whole procedure can be easily implemented in time $O(n+m)$.

On the other hand, if G is a split graph, the algorithm provides a split partition $A \cup D$ of G. We construct the sets R and B as in the proof, and then we check whether R is an independent set. If not, the two neighbours in R give us a forbidden configuration. Otherwise, we test whether the neighbourhood of each vertex of R is a clique. We observe that it suffices to check those $v \in R$ which belong to D, since for $v \in R \cap A$, this is trivially true. Also, since D induced a clique, there will be at most one vertex to check, hence this can be easily implemented in time $O(n+m)$. If we find that a vertex $v \in R$ has two non-adjacent neighbours, we find the shortest path between these neighbours in $G - v$, and then pick some three consecutive vertices of this path together with v which gives us a forbidden configuration. Note that finding a shortest path between two vertices can be implemented in time $O(n+m)$ by breadth-first search. If no such vertex $v \in R$ is found, then we double every vertex in B as described in the proof to obtain G', and test whether G' is split. Clearly, G' has at most $2n$ vertices and $2m$ edges, and it can be constructed in time $O(n+m)$. Now, if G' is not split, then we find the forbidden graph the same way we did for G, and by (possibly) contracting some of its edges, we again obtain a forbidden configuration. On the other hand, if G' is split, we obtain a split partition $A' \cup D'$ of G', which will contain a split partition $A \cup D$ of H, and using the argument from the proof, we can find a monopolar partition of G by (possibly) exchanging up to two vertices between A and D. Again, this can be accomplished in time $O(n+m)$, which concludes the analysis.

We summarize this in the following theorem.

Theorem 4.4. *There exists an $O(n+m)$ time algorithm to test, for a given 2-connected chordal graph G with lists $\ell(v) \subseteq \{\mathsf{r}, \mathsf{b}\}$, whether G is list monopolar, that is, whether G admits a monopolar partition respecting ℓ. If G is not list monopolar, the algorithm produces a minimal list non-extendable forbidden induced subgraph contained in G.* □

4.2 The general case

Now, we turn our attention back to all (not necessarily 2-connected) chordal graphs. In this section, we shall describe a procedure generating all chordal minimal list non-extendable forbidden induced subgraphs using a graph grammar (see Section 2.4), and, as in the previous section, we also describe an $O(n+m)$ time algorithm to find a minimal list non-extendable forbidden induced subgraph in a chordal graph which is not list monopolar.

We use two hyperedge replacement grammars. The grammar Γ generates all minimal non-monopolar chordal graphs, and the grammar Γ' all minimal list non-extendable non-monopolar chordal graphs. The rules of both grammars are shown below. They are followed by a formal description. Before reading any further, we recommend that the reader revisits Section 2.4 in Chapter 2 to get better acquainted with the terminology of graph grammars.

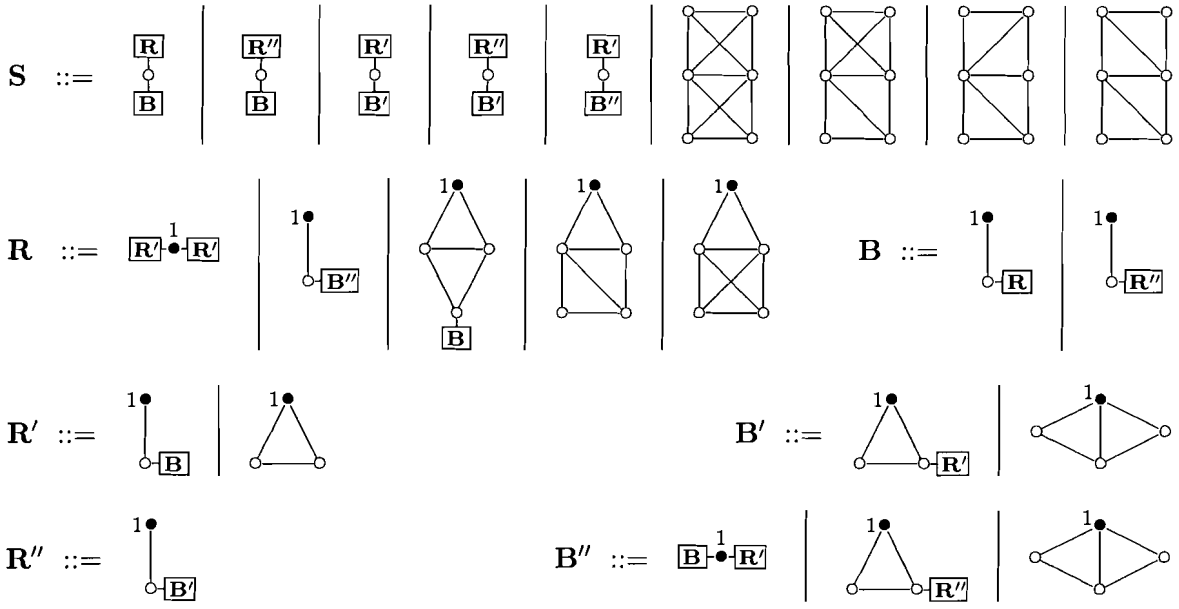

Figure 4.2: The rules of the grammar Γ.

Figure 4.3: Additional rules of the grammar Γ'.

The hyperedge replacement grammar Γ is a tuple $(N, T, P, \mathbf{S})$, where $N = \{\mathbf{S}, \mathbf{R}, \mathbf{R}', \mathbf{R}'', \mathbf{B}, \mathbf{B}', \mathbf{B}''\}$ are the nonterminal symbols, $T = \{\mathbf{X}\}$ are the terminal symbols, $\mathbf{S}$ is the starting symbol, the types (arities) of the symbols are $type(\mathbf{S}) = 0$, $type(\mathbf{R}) = type(\mathbf{R}') = type(\mathbf{R}'') = type(\mathbf{B}) = type(\mathbf{B}') = type(\mathbf{B}'') = 1$, and $type(\mathbf{X}) = 2$, and the production rules P are the rules in Figure 4.2. Similarly, the grammar Γ' is a tuple $(N, T', P', \mathbf{S})$ where N, and $\mathbf{S}$ are as before, $T' = T \cup \{\mathsf{r}, \mathsf{b}\}$ where $type(\mathsf{r}) = type(\mathsf{b}) = 1$, and P' are the rules P and the additional rules shown in Figure 4.3.

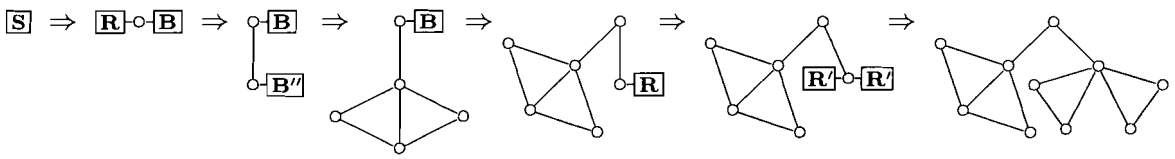

Figure 4.4: Example of a derivation of the grammar.

Recall that each production rule is a pair (Q, H) where $Q \in N$, and H is a hypergraph whose type is equal to $type(Q)$, that is, H has exactly $type(Q)$ distinct external nodes ext_H marked $1, \ldots, type(Q)$. The hypergraph H of any rule (Q, H) in Figures 4.2 and 4.3 is depicted as the (bipartite) incidence graph of its vertices and hyperedges. Recall, that this is a graph whose vertices are the vertices and hyperedges of the hypergraph, and whose edges are between vertices and hyperedges incident in the hypergraph. Note that for simplicity, all hyperedges of type (arity) 2 are depicted as regular edges; all these edges are labeled with the symbol $\mathbf{X}$, however, for simplicity, this is also not shown. Finally, observe that the hypergraphs in Figures 4.2 and 4.3 for the nonterminal $\mathbf{S}$ have no vertices marked, since the type of $\mathbf{S}$ is 0, and the rules for all other nonterminals have exactly one vertex marked 1, since their type is 1.

We say that a hypergraph H generated by a grammar is *terminal*, if H contains only hyperedges labeled with the terminal symbols of the grammar.

Now, let H be a terminal hypergraph that is generated by the grammar Γ', that is, $H \in L_{\mathbf{S}}(\Gamma')$. We define lists $\ell_H(v)$ for each vertex $v \in V(H)$ as follows. Initially, we set $\ell_H(v) = \{\mathsf{r}, \mathsf{b}\}$. Then, if v is incident to a hyperedge labeled r, we remove b from $\ell_H(v)$. If v is incident to a hyperedge labeled b, we remove r from $\ell_H(v)$. Hence, the final list for $\ell_H(v)$ can be either $\{\mathsf{r}, \mathsf{b}\}$, or $\{\mathsf{r}\}$, or $\{\mathsf{b}\}$, or empty. After we construct the lists, we remove all terminal hyperedges of type 1. Since the only remaining terminal hyperedges are of type 2, that is, they are ordinary edges, the resulting hypergraph will be a graph. In the subsequent text, whenever dealing with a terminal hypergraph H generated by the grammar Γ', we shall always assume that we have the lists ℓ_H as described above, and that H is a graph as explained.

Recall, that ext_H is the external vertex (or vertices) of a hypergraph H, and $N_H(v)$ is the set of neighbours of v in H. We have the following property of Γ'.

Lemma 4.5. *Let $Q \neq \mathbf{S}$, and H be a terminal hypergraph in $L_Q(\Gamma')$ with lists ℓ_H. Then,*

$(*)$ *there exists a monopolar partition $A \cup D$ of H respecting ℓ_H, and if $Q = \mathbf{R}'$, then $N_H(ext_H) \cup \{ext_H\} \subseteq D$.*

Moreover, for any monopolar partition $A \cup D$ of H respecting ℓ_H,

(i) if $Q \in \{\mathbf{R}, \mathbf{R}''\}$, then $ext_H \in A$,

(ii) if $Q = \mathbf{R}'$, then $N_H(ext_H) \cap D \neq \emptyset$,

(iii) if $Q = \mathbf{B}$, then $ext_H \in D$ and $N_H(ext_H) \subseteq A$,

(iv) if $Q \in \{\mathbf{B}', \mathbf{B}''\}$, then $ext_H \in D$ and $N_H(ext_H) \cap D \neq \emptyset$.

Additionally, for any $y \in V(H) \setminus \{ext_H\}$ (respectively any proper list extension $\hat{\ell}$ of ℓ_H), there exists a monopolar partition $\hat{A} \cup \hat{D}$ of $\hat{H} = H - y$ respecting ℓ_H (respectively of $\hat{H} = H$ respecting $\hat{\ell}$) such that

(i') if $Q = \mathbf{R}$, then $ext_H \in \hat{D}$,

(ii') if $Q = \mathbf{R}'$, then $N_{\hat{H}}(ext_H) \subseteq \hat{A}$,

(iii') if $Q = \mathbf{R}''$, then $ext_H \in \hat{D}$ and $N_{\hat{H}}(ext_H) \subseteq \hat{A}$,

(iv') if $Q \in \{\mathbf{B}, \mathbf{B}'\}$, then $ext_H \in \hat{A}$.

(v') if $Q = \mathbf{B}''$, then either $ext_H \in \hat{A}$ or $N_{\hat{H}}(ext_H) \subseteq \hat{A}$.

Proof. Let $Q^\bullet = H_0 \Rightarrow H_1 \Rightarrow \ldots \Rightarrow H_t = H$ be a derivation of H in Γ'. We prove the claim by induction on the length of the derivation of H.

If $t = 1$, then H is either the hypergraph of the 3^{rd} or the 4^{th} rule for $Q = \mathbf{R}$, or the hypergraph of the 2^{nd} rule for $Q = \mathbf{R}'$, or the hypergraph of the 2^{nd} or the 3^{rd} rule for $Q = \mathbf{B}'$ respectively $Q = \mathbf{B}''$, or the hypergraph of the additional rule for $Q \in \{\mathbf{R}'', \mathbf{B}\}$. If $Q = \mathbf{R}$ or $Q \in \{\mathbf{B}', \mathbf{B}''\}$, then the claim follows from Observation 4.1. If $Q = \mathbf{R}'$, then H is a triangle; hence, $(*)$ and (ii) follow. Also, removing any vertex from $N_H(ext_H)$ leaves a single vertex in $N_H(ext_H)$; hence, (ii') follows. (Note that in this case no proper list extension of ℓ_H exists.) Finally, if $Q \in \{\mathbf{R}'', \mathbf{B}\}$, then H is a single vertex x with $\ell_H(x) = \{\mathsf{r}\}$ respectively $\ell_H(x) = \{\mathsf{b}\}$; hence, $(*)$, (i), and (iii) follow, and for (the only) proper list extension $\ell'(x) = \{\mathsf{r}, \mathsf{b}\}$, the partitions $\hat{A} = \emptyset$, $\hat{D} = \{x\}$ respectively $\hat{A} = \{x\}$, $\hat{D} = \emptyset$ satisfy (iii') respectively (iv').

Now, let $t \geq 2$, and assume that the claim holds for $t - 1$. First, suppose that $Q = \mathbf{R}$. Then H_1 is either the hypergraph of the 1^{st}, the 2^{nd}, or the 3^{rd} rule for $\mathbf{R}$. Suppose that H_1

is the hypergraph of the 1st rule. Then $H = H_1[e/H', e'/H'']$, where $H', H'' \in L_{\mathbf{R}'}(\Gamma')$, and e, e' are the two hyperedges labeled $\mathbf{R}'$ incident to ext_H. Let $A \cup D$ be a monopolar partition of H respecting ℓ_H. Then $A \cup D$ induces monopolar partitions of H' and H'' respecting $\ell_{H'}$ and $\ell_{H''}$, respectively. Using the inductive hypothesis, it follows that there exists $u \in N_{H'}(ext_H) \cap D$ and $v \in N_{H''}(ext_H) \cap D$. Hence, $ext_H \in A$, since otherwise u, ext_H, v is a P_3 in $H[D]$. This proves (i). On the other hand, by the inductive hypothesis, there exists a monopolar partition $A' \cup D'$ of H' respecting $\ell_{H'}$ with $ext_H \in D'$ and $N_{H'}(ext_H) \subseteq D'$, and a monopolar partition $A'' \cup D''$ of H'' respecting $\ell_{H''}$ with $ext_H \in D''$ and $N_{H''}(ext_H) \subseteq D''$. Hence, $A \cup D$, where $A = A' \cup A'' \cup \{ext_H\}$ and $D = (D' \cup D'') \setminus \{ext_H\}$, is a monopolar partition of H respecting ℓ_H, which shows $(*)$. Now, let $y \in V(H) \setminus \{ext_H\}$. If $y \in V(H')$, then, by the inductive hypothesis, there exists a monopolar partition $A'_1 \cup D'_1$ of $H' - y = \hat{H}'$ respecting $\ell_{H'}$ with $N_{\hat{H}'}(ext_H) \subseteq A'_1$. Hence, $\hat{A} \cup \hat{D}$, where $\hat{A} = A'_1 \setminus \{ext_H\} \cup A''$ and $\hat{D} = D'_1 \cup D''$, is a monopolar partition of $H - y$ respecting ℓ_H with $ext_H \in \hat{D}$. The case when $y \in V(H'')$ is clearly symmetric. Now, let $\hat{\ell}$ be a proper list extension of ℓ_H. Clearly, there must exist a vertex $y \in V(H)$ such that $\ell_H(y) \subsetneq \hat{\ell}(y)$. Again, either $y \in V(H')$ or $y \in V(H'')$, and the proof is identical to the above. This shows (i').

Now, suppose that H_1 is the hypergraph of the 2nd rule for $\mathbf{R}$. Hence, $H = H_1[e/H']$, where $H' \in L_{\mathbf{B}''}(\Gamma')$, and e is the only hyperedge of H_1 labeled $\mathbf{B}''$. Let v denote the vertex incident to e, and let $A \cup D$ be a monopolar partition of H respecting ℓ_H. Again, by the inductive hypothesis, $v \in D$, and there exists $u \in N_{H'}(v) \cap D$. Hence, $ext_H \in A$, since otherwise u, v, ext_H is a P_3 in $H[D]$. On the other hand, by the inductive hypothesis, there exists a monopolar partition $A' \cup D'$ of H' respecting $\ell_{H'}$ with $v \in D'$. Hence, $A \cup D$, where $A = A' \cup \{ext_H\}$ and $D = D'$, is a monopolar partition of H respecting ℓ_H. Now, let $y \in V(H) \setminus \{ext_H\}$. If $y = v$, then $\hat{A} \cup \hat{D}$, where $\hat{A} = A'$ and $\hat{D} = D' \setminus \{v\} \cup \{ext_H\}$, is a monopolar partition of $H - y$ respecting ℓ_H with $ext_H \in \hat{D}$. Otherwise, $y \in V(H') \setminus \{v\}$; hence, by the inductive hypothesis, there exists a monopolar partition $A'_1 \cup D'_1$ of $H' - y = \hat{H}'$ respecting $\ell_{H'}$ with either $v \in A'_1$ or $N_{\hat{H}'}(v) \subseteq A'_1$. It follows that $\hat{A} \cup \hat{D}$, where $\hat{A} = A'_1$ and $\hat{D} = D'_1 \cup \{ext_H\}$, is a monopolar partition of $H - y$ respecting ℓ_H with $ext_H \in \hat{D}$. Now, if $\hat{\ell}$ is a proper list extension of ℓ_H, then there exists $y \in V(H')$ with $\ell_H(y) \subsetneq \hat{\ell}(y)$, and the proof again follows identically.

Now, suppose that H_1 is the hypergraph of the 3rd rule for $\mathbf{R}$. Hence, $H = H_1[e/H']$, where $H' \in L_{\mathbf{B}}(\Gamma')$, and e is the hyperedge of H_1 labeled $\mathbf{B}$. Let v be the vertex incident to e, and let u and w be the two remaining vertices of H_1. Now, suppose that $A \cup D$ is a

monopolar partition of H respecting ℓ_H. Again, by the inductive hypothesis, we have $v \in D$ and $N_{H'}(v) \subseteq A$. Hence, $ext_H \in A$, since otherwise either ext_H, u, v or ext_H, w, v is a P_3 in $H[D]$. On the other hand, there exists a monopolar partition $A' \cup D'$ of H' respecting $\ell_{H'}$ with $v \in D'$ and $N_{H'}(v) \subseteq A'$. Hence, $A \cup D$, where $A = A' \cup \{ext_H\}$ and $D = D' \cup \{u, w\}$, is a monopolar partition of H respecting ℓ_H. Now, let $y \in V(H) \setminus \{ext_H\}$. If $y = v$, then $\hat{A} \cup \hat{D}$, where $\hat{A} = A'$ and $\hat{D} = D' \setminus \{v\} \cup \{ext_H, u, w\}$, is a monopolar partition of $H - y$ respecting ℓ_H with $ext_H \in \hat{D}$. If $y = w$, then $\hat{A} \cup \hat{D}$, where $\hat{A} = A' \cup \{u\}$ and $D = D' \cup \{ext_H\}$, is a monopolar partition of H respecting ℓ_H with $ext_H \in \hat{D}$. Similarly if $y = u$. Finally, if $y \in V(H') \setminus \{v\}$, we have, by the inductive hypothesis, that there exists a monopolar partition $A'_1 \cup D'_1$ of $H' - y$ respecting ℓ_H with $v \in A'_1$. Hence, $\hat{A} \cup \hat{D}$, where $\hat{A} = A'_1$ and $\hat{D} = D'_1 \cup \{ext_H, u, w\}$, is a monopolar partition of $H - y$ respecting ℓ_H with $ext_H \in \hat{D}$. Again, the proof is identical for any proper list extension $\hat{\ell}$ of ℓ_H.

Now, suppose that $Q = \mathbf{B}$. Then H_1 is either the hypergraph of the 1st or the 2nd rule for $\mathbf{B}$. If H_1 is the hypergraph of the 1st rule, then $H = H_1[e/H']$, where $H' \in L_{\mathbf{R}}(\Gamma')$, and e is the hyperedge of H_1 labeled $\mathbf{R}$. Let v be the vertex incident to e, and let $A \cup D$ be a monopolar partition of H respecting ℓ_H. Then, by the inductive hypothesis, $v \in A$, and hence, $ext_H \in D$ and $N_H(ext_H) \subseteq A$. Also, by the inductive hypothesis, there exists a monopolar partition $A' \cup D'$ of H' respecting $\ell_{H'}$ with $v \in A'$, and hence, $A \cup D$, where $A = A'$ and $D = D' \cup \{ext_H\}$, is a monopolar partition of H respecting ℓ_H. Now, let $y \in V(H) \setminus \{ext_H\}$. If $y = v$, then $\hat{A} \cup \hat{D}$, where $\hat{A} = A' \setminus \{v\} \cup \{ext_H\}$ and $\hat{D} = D'$, is a monopolar partition of $H - y$ respecting ℓ_H with $ext_H \in \hat{A}$. If $y \in V(H') \setminus \{v\}$, then, by the inductive hypothesis, there exists a monopolar partition $A'_1 \cup D'_1$ of $H' - y$ respecting $\ell_{H'}$ with $v \in D'_1$. Hence, $\hat{A} \cup \hat{D}$, where $\hat{A} = A'_1 \cup \{ext_H\}$ and $\hat{D} = D'_1$, is a monopolar partition of H respecting ℓ_H with $ext_H \in \hat{A}$. Similarly for any proper list extension of ℓ_H. Now, if H_1 is the hypergraph of the 2nd rule for $\mathbf{B}$, the proof is identical to the above.

Now, suppose that $Q = \mathbf{R}'$. Then H_1 is the hypergraph of the 1st rule for $\mathbf{R}'$, and $H = H_1[e/H']$, where $H' \in L_{\mathbf{B}}(\Gamma')$, and e is the hyperedge of H_1 labeled $\mathbf{B}$. Let v be the vertex incident to e, and suppose that $A \cup D$ is a monopolar partition of H respecting ℓ_H. By the inductive hypothesis, $v \in D$, and hence, $N_H(ext_H) \cap D \neq \emptyset$. Also, by the inductive hypothesis, there exists a monopolar partition $A' \cup D'$ of H' respecting $\ell_{H'}$ with $v \in D'$ and $N_{H'}(v) \subseteq A'$. Hence, $A \cup D$, where $A = A'$ and $D = D' \cup \{ext_H\}$, is a monopolar partition of H respecting ℓ_H with $N_H(ext_H) \cup \{ext_H\} \subseteq D$. Now, let $y \in V(H) \setminus \{ext_H\}$. If $y = v$, then $\hat{A} \cup \hat{D}$, where $\hat{A} = A'$ and $\hat{D} = D' \setminus \{v\} \cup \{ext_H\}$, is a monopolar partition of $H - y = \hat{H}$

respecting ℓ_H with $N_{\hat{H}}(ext_H) \subseteq \hat{A}$. If $y \in V(H') \setminus \{v\}$, by the inductive hypothesis, there exists a monopolar partition $A'_1 \cup D'_1$ of $H' - y$ respecting $\ell_{H'}$ with $v \in A'_1$. Hence, $\hat{A} \cup \hat{D}$, where $\hat{A} = A'_1$ and $\hat{D} = D'_1 \cup \{ext_H\}$, is a monopolar partition of $H - y = \hat{H}$ respecting ℓ_H with $N_{\hat{H}}(ext_H) \subseteq \hat{A}$. Again, it follows similarly for any proper list extension of ℓ_H.

Now, suppose that $Q = \mathbf{R}''$. Hence, H_1 is the hypergraph of the 1st rule for $\mathbf{R}''$, and $H = H_1[e/H']$, where $H' \in L_{\mathbf{B}'}(\Gamma')$, and e is the hyperedge of H_1 labeled $\mathbf{B}'$. Let v be the vertex incident to e, and suppose that $A \cup D$ is a monopolar partition of H respecting ℓ_H. Then, by the inductive hypothesis, $v \in D$ and there exists $u \in N_{H'}(v) \cap D$. Hence, $ext_H \in A$, since otherwise ext_H, v, u is a P_3 in $H[D]$. On the other hand, by the inductive hypothesis, there exists a monopolar partition $A' \cup D'$ of H' respecting $\ell_{H'}$ with $v \in D'$. Hence, $A \cup D$, where $A = A' \cup \{ext_H\}$ and $D = D'$, is a monopolar partition of H respecting ℓ_H. Let $y \in V(H) \setminus \{ext_H\}$. If $y = v$, then $\hat{A} \cup \hat{D}$, where $\hat{A} = A'$ and $\hat{D} = D' \setminus \{v\} \cup \{ext_H\}$, is a monopolar partition of $H - y = \hat{H}$ respecting ℓ_H with $ext_H \in \hat{D}$ and $N_{\hat{H}}(ext_H) \subseteq \hat{A}$. If $y \in V(H') \setminus \{v\}$, then, by the inductive hypothesis, there exists a monopolar partition $A'_1 \cup D'_1$ of $H' - y$ respecting $\ell_{H'}$ with $v \in A'_1$. Hence, $\hat{A} \cup \hat{D}$, where $\hat{A} = A'_1$ and $\hat{D} = D'_1 \cup \{ext_H\}$, is a monopolar partition of $H - y = \hat{H}$ respecting ℓ_H with $ext_H \in \hat{D}$ and $N_{\hat{H}}(ext_H) \subseteq \hat{A}$. Similarly for any proper list extension of ℓ_H.

Now, suppose that $Q = \mathbf{B}'$. Hence, H_1 is the hypergraph of the 1st rule for $\mathbf{B}'$, and $H = H_1[e/H']$, where $H' \in L_{\mathbf{R}'}(\Gamma')$, and e is the hyperedge of H_1 labeled $\mathbf{R}'$. Let v be the vertex incident to e, and u be the remaining vertex of H_1. Let $A \cup D$ be a monopolar partition of H respecting ℓ_H. Then, by the inductive hypothesis, there exists $w \in N_{H'}(v) \cap D$. It follows that $v \in A$, since otherwise either ext_H, v, w or u, v, w is a P_3 in $H[D]$. Hence, $ext_H \in D$ and $N_H(ext_H) \cap D \neq \emptyset$. On the other hand, there exists a monopolar partition $A' \cup D'$ of H' respecting $\ell_{H'}$ with $N_{H'}(v) \cup \{v\} \subseteq D'$. Hence, $A \cup D$, where $A = A' \cup \{v\}$ and $D = D' \setminus \{v\} \cup \{ext_H, u\}$, is a monopolar partition of H respecting ℓ_H. Now, let $y \in V(H) \setminus \{ext_H\}$. If $y = v$, then $\hat{A} \cup \hat{D}$, where $\hat{A} = A' \cup \{ext_H\}$ and $\hat{D} = D' \setminus \{v\} \cup \{u\}$, is a monopolar partition of $H - y$ respecting ℓ_H with $ext_H \in \hat{A}$. If $y = u$, then $\hat{A} \cup \hat{D}$, where $\hat{A} = A' \cup \{ext_H\}$ and $\hat{D} = D'$, is a monopolar partition of $H - y$ respecting ℓ_H with $ext_H \in \hat{A}$. Finally, if $y \in V(H') \setminus \{v\}$, then, by the inductive hypothesis, there exists a monopolar partition $A'_1 \cup D'_1$ of $H' - y = \hat{H}'$ respecting $\ell_{H'}$ with $N_{\hat{H}'}(v) \subseteq A'_1$. Hence, $\hat{A} \cup \hat{D}$, where $\hat{A} = A'_1 \setminus \{v\} \cup \{ext_H\}$ and $\hat{D} = D'_1 \cup \{u, v\}$, is a monopolar partition of $H - y$ respecting ℓ_H with $ext_H \in \hat{A}$. Similarly for proper list extensions of ℓ_H.

Finally, suppose that $Q = \mathbf{B}''$. Then H_1 is either the hypergraph of the 1st, or the 2nd

rule for $\mathbf{B}''$. Suppose that H_1 is the hypergraph of the 1st rule. Hence, $H = H_1[e/H', e'/H'']$, where $H' \in L_{\mathbf{B}}(\Gamma')$, $H'' \in L_{\mathbf{R}'}(\Gamma')$, and e, e' are the hyperedges of H_1 labeled $\mathbf{B}$ and $\mathbf{R}'$ respectively. Let $A \cup D$ be a monopolar partition of H respecting ℓ_H. Then, by the inductive hypothesis, $ext_H \in D$, $N_{H'}(ext_H) \subseteq A$, and $N_{H''}(ext_H) \cap D \neq \emptyset$. Hence, $N_H(ext_H) \cap D \neq \emptyset$ and $ext_H \in D$. On the other hand, by the inductive hypothesis, there exists a monopolar partition $A' \cup D'$ of H' respecting $\ell_{H'}$ with $ext_H \in D'$ and $N_{H'}(ext_H) \subseteq A'$, and also a monopolar partition $A'' \cup D''$ of H'' respecting $\ell_{H''}$ with $N_{H''}(ext_H) \cup \{ext_H\} \subseteq D''$. Hence, $A \cup D$, where $A = A' \cup A''$ and $D = D' \cup D''$, is a monopolar partition of H respecting ℓ_H. Now, let $y \in V(H) \setminus \{ext_H\}$. If $y \in V(H')$, then, by the inductive hypothesis, there exists a monopolar partition $A'_1 \cup D'_1$ of $H' - y$ respecting $\ell_{H'}$ with $ext_H \in A'_1$. Hence, $\hat{A} \cup \hat{D}$, where $\hat{A} = A'_1 \cup A''$ and $\hat{D} = D'_1 \cup D'' \setminus \{ext_H\}$, is a monopolar partition of $H - y$ respecting ℓ_H with $ext_H \in \hat{A}$. If $y \in V(H'')$, then there exists a monopolar partition $A''_1 \cup D''_1$ of $H'' - y = \hat{H}''$ respecting $\ell_{H''}$ with $N_{\hat{H}''}(ext_H) \subseteq A''_1$. Hence, $\hat{A} \cup \hat{D}$, where $\hat{A} = A' \cup A''_1 \setminus \{ext_H\}$ and $\hat{D} = D' \cup D''_1$, is a monopolar partition of $H - y = \hat{H}$ respecting ℓ_H with $N_{\hat{H}}(ext_H) \subseteq \hat{A}$. Again, similarly for proper list extensions of ℓ_H.

Now, suppose that H_1 is the hypergraph of the 2nd rule for $\mathbf{B}''$. Then $H = H_1[e/H']$, where $H' \in L_{\mathbf{R}''}(\Gamma')$, and e is the hyperedge of H_1 labeled $\mathbf{R}''$. Let v be the vertex incident to e, and let u be the remaining vertex of H_1. Suppose that $A \cup D$ is a monopolar partition of H respecting ℓ_H. Then, by the inductive hypothesis, $v \in A$; hence, $ext_H \in D$ and $N_H(ext_H) \cap D \neq \emptyset$. Also, by the inductive hypothesis, there exists a monopolar partition $A' \cup D'$ of H' respecting $\ell_{H'}$ with $v \in A'$. Hence, $A \cup D$, where $A = A'$ and $D = D' \cup \{ext_H, u\}$, is a monopolar partition of H respecting ℓ_H. Now, let $y \in V(H) \setminus \{ext_H\}$. If $y = v$, then $\hat{A} \cup \hat{D}$, where $\hat{A} = A' \setminus \{v\} \cup \{ext_H\}$ and $\hat{D} = D' \cup \{u\}$, is a monopolar partition of $H - y$ respecting ℓ_H with $ext_H \in \hat{A}$. If $y = u$, then $\hat{A} \cup \hat{D}$, where $\hat{A} = A'$ and $\hat{D} = D' \cup \{ext_H\}$, is a monopolar partition of $H - y = \hat{H}$ respecting ℓ_H with $N_{\hat{H}}(ext_H) \subseteq \hat{A}$. If $y \in V(H') \setminus \{v\}$, then, by the inductive hypothesis, there exists a monopolar partition $A'_1 \cup D'_1$ of $H' - y = \hat{H}'$ respecting $\ell_{H'}$ with $v \in D'_1$ and $N_{\hat{H}'}(v) \subseteq A'_1$. Hence, $\hat{A} \cup \hat{D}$, where $\hat{A} = A'_1 \cup \{ext_H\}$ and $\hat{D} = D'_1 \cup \{u\}$, is a monopolar partition of $H - y$ respecting ℓ_H with $ext_H \in \hat{A}$. Again, it follows similarly for proper list extensions of ℓ_H.

The proof is now complete. □

Using this lemma, we can now prove that the grammar Γ' (and hence also Γ) generates only minimal chordal list non-extendable forbidden induced subgraphs for monopolarity.

Theorem 4.6. *All terminal graphs generated by Γ' are minimal list non-extendable forbidden induced subgraphs for monopolarity of chordal graphs.*

Proof. Let H with lists ℓ_H be a terminal graph generated by Γ'. It can be easily observed from the rules of Γ' that H must be chordal. Now, let $\mathbf{S}^\bullet = H_0 \Rightarrow H_1 \Rightarrow \ldots \Rightarrow H_t = H$ be a derivation of H in Γ'. It follows that H_1 is the hypergraph of one of the rules of Γ' for $\mathbf{S}$.

First, suppose that H_1 is the hypergraph of the 1st rule for $\mathbf{S}$. Then $H = H_1[e/H', e'/H'']$, where $H' \in L_{\mathbf{R}}(\Gamma')$, $H'' \in L_{\mathbf{B}}(\Gamma')$, and e, e' are the two hyperedges of H_1 labeled $\mathbf{R}$ and $\mathbf{B}$, respectively. Suppose that $A \cup D$ is a monopolar partition of H respecting ℓ_H. Using Lemma 4.5 for H', we have that $ext_H \in A$. However, applying Lemma 4.5 to H'' yields $ext_H \in D$, a contradiction. Hence, H is not list monopolar. Now, by Lemma 4.5, we also obtain that there exists a monopolar partition $A' \cup D'$ of H' respecting $\ell_{H'}$ with $ext_H \in A'$, and a monopolar partition $A'' \cup D''$ of H'' respecting $\ell_{H''}$ with $ext_H \in D''$ and $N_{H''}(ext_H) \subseteq A''$. Let $y \in V(H)$. If $y = ext_H$, then $\hat{A} \cup \hat{D}$, where $\hat{A} = A' \setminus \{ext_H\} \cup A''$ and $\hat{D} = D' \cup D'' \setminus \{ext_H\}$, is a monopolar partition of $H - y$ respecting ℓ_H. If $y \in V(H')$, then, by Lemma 4.5, there exists a monopolar partition $A'_1 \cup D'_1$ of $H' - y$ respecting $\ell_{H'}$ with $ext_H \in D'_1$. Hence, $\hat{A} \cup \hat{D}$, where $\hat{A} = A'_1 \cup A''$ and $\hat{D} = D'_1 \cup D''$, is a monopolar partition of $H - y$ respecting ℓ_H. If $y \in V(H'')$, then, by Lemma 4.5, there exists a monopolar partition $A''_1 \cup D''_1$ of $H'' - y$ respecting $\ell_{H''}$ with $ext_H \in A''_1$. Hence, $\hat{A} \cup \hat{D}$, where $\hat{A} = A''_1 \cup A'$ and $\hat{D} = D''_1 \cup D'$, is a monopolar partition of $H - y$ respecting ℓ_H. Now, let $\hat{\ell}$ be a proper list extension of ℓ_H. It follows that there must exist $y \in V(H)$ with $\ell_H(y) \subsetneq \hat{\ell}(y)$. Again, either $y \in V(H')$ or $y \in V(H'')$, and the proof is identical to the above.

Now, if H_1 is the hypergraph of the 2nd rule for $\mathbf{S}$, then the proof follows exactly as for the 1st rule. Hence, suppose that H_1 is the hypergraph of the 3rd rule for $\mathbf{S}$. Then $H = H_1[e/H', e'/H'']$, where $H' \in L_{\mathbf{R}'}(\Gamma')$, $H'' \in L_{\mathbf{B}'}(\Gamma')$, and e, e' are the two hyperedges of H_1 labeled $\mathbf{R}'$ and $\mathbf{B}'$, respectively. Suppose that $A \cup D$ is a monopolar partition of H respecting ℓ_H. Then, by Lemma 4.5, $ext_H \in D$, and there exists $u \in N_{H'}(ext_H) \cap D$ and $v \in N_{H''}(ext_H) \cap D$. Hence, u, ext_H, v is a P_3 in $H[D]$, a contradiction. On the other hand, by Lemma 4.5, there exists a monopolar partition $A' \cup D'$ of H' respecting $\ell_{H'}$ with $N_{H'}(ext_H) \cup \{ext_H\} \subseteq D'$, and a monopolar partition $A'' \cup D''$ of H'' respecting $\ell_{H''}$ with $ext_H \in D''$ and $N_{H''}(ext_H) \cap D'' \neq \emptyset$. Now, let $y \in V(H)$. If $y = ext_H$, then $\hat{A} \cup \hat{D}$, where $\hat{A} = A' \cup A''$ and $\hat{D} = (D' \cup D'') \setminus \{ext_H\}$, is a monopolar partition of $H - y$ respecting ℓ_H. If $y \in V(H')$, then, by Lemma 4.5, there exists a monopolar partition $A'_1 \cup D'_1$ of $H' - y = \hat{H}'$ with $N_{\hat{H}'}(ext_H) \subseteq A'_1$. Hence, $\hat{A} \cup \hat{D}$, where $\hat{A} = A'_1 \cup A''$ and $\hat{D} = D'_1 \cup D''$,

is a monopolar partition of $H - y$ respecting ℓ_H. If $y \in V(H'')$, then, by Lemma 4.5, there exists a monopolar partition $A_1'' \cup D_1''$ of $H'' - y$ respecting $\ell_{H''}$ with $ext_H \in A_1''$. Hence, $\hat{A} \cup \hat{D}$, where $\hat{A} = A' \cup A_1''$ and $\hat{D} = D' \setminus \{ext_H\} \cup D_1''$, is a monopolar partition of $H - y$ respecting ℓ_H. Now, if $\hat{\ell}$ is a proper list extension of ℓ_H, we again have $y \in V(H)$ with $\ell_H(v) \subsetneq \hat{\ell}(v)$, and the proof is identical to the above.

Finally, if H_1 is the hypergraph of the 4^{th} or the 5^{th} rule for $\mathbf{S}$, then the proof follows exactly as for the 1^{st} and the 3^{rd} rule, respectively, whereas if H_1 is the hypergraph of the 6^{th}, 7^{th}, 8^{th}, or 9^{th} rule for $\mathbf{S}$, then the proof follows from Observation 4.1. □

In the rest of this section, we prove that, conversely, any chordal graph with lists, which is not list monopolar, must necessarily contain some hypergraph generated by Γ'.

Let G be a graph with lists $\ell(v)$, for all $v \in V(G)$, and H be a terminal hypergraph generated by Γ'. We say that G *contains* H, if the graph associated with H is an induced subgraph of G, and $\ell_H(x) \supseteq \ell(x)$ for each $x \in V(H)$. Also, we say that G with lists ℓ and H are *corresponding*, if G is the graph associated with H, and $\ell_H(x) \supseteq \ell(x)$ for each $x \in V(G)$.

We shall need the following lemma. For a graph G, vertex v of G, and a block-vertex tree T of G, let T_v denote the subtree of T rooted at v, and G_v denote the subgraph of G formed by the blocks which are nodes of T_v.

Lemma 4.7. *Let G be a chordal graph with lists ℓ_0, let v be a vertex of G, and let T be a block-vertex tree of G. Let ℓ^* be the lists computed by Algorithm 3.2 on G, ℓ_0, and T. Suppose that $\ell^*(x) \neq \emptyset$ for all $x \in V(G)$.*

If $\ell^*(v) = \{\mathsf{b}\}$, then

- *(i) if $\ell_0(v) = \{\mathsf{b}\}$ or v has a trivial grandchild u with $\ell^*(u) = \{\mathsf{r}\}$, then*
  - *(a) G_v contains a hypergraph $F \in L_{\mathbf{B}}(\Gamma')$ with $ext_F = v$, and*
  - *(b) if, in addition, v either has a non-trivial child or has a trivial grandchild u' with $\ell^*(u') = \{\mathsf{b}\}$, then G_v contains a hypergraph $F' \in L_{\mathbf{B}''}(\Gamma')$ with $ext_{F'} = v$,*
- *(ii) otherwise, G_v contains a hypergraph $F \in L_{\mathbf{B}}(\Gamma') \cup L_{\mathbf{B}'}(\Gamma')$ with $ext_F = v$, and also contains a hypergraph $F' \in L_{\mathbf{B}'}(\Gamma') \cup L_{\mathbf{B}''}(\Gamma')$ with $ext_{F'} = v$,*

and if $\ell^*(v) = \{\mathsf{r}\}$, then

- *(i) G_v contains a hypergraph $F \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$ with $ext_F = v$, and*
- *(ii) if v has a trivial (or no) parent, then G_v also contains $F' \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$ with $ext_{F'} = v$.*

Proof. Let G be a fixed chordal graph with lists ℓ_0, let v be a vertex of G, and let T be a fixed block-vertex tree of G. We prove the claim by induction on the size of G_v.

Hence, we assume that the claim holds for any w having G_w smaller than v, and we prove the claim for v.

First, suppose that $\ell^*(v) = \{\mathsf{b}\}$. Recall that $\ell^*(x) \subseteq \ell_0(x)$ for all $x \in V(G)$. Hence, if $\ell_0(v) = \{\mathsf{b}\}$, then the vertex v with list $\ell_0(v)$ corresponds to the hypergraph F of the additional rule for $\mathbf{B}$, which, clearly, belongs to $L_{\mathbf{B}}(\Gamma')$, and $ext_F = v$. Hence, G_v contains the hypergraph F from $L_{\mathbf{B}}(\Gamma')$ with $ext_F = v$. On the other hand, if $\ell_0(v) = \{\mathsf{r}, \mathsf{b}\}$, then either Rule 1 or the Block Rule of Algorithm 3.2 must have removed r from the list of v. Suppose that Rule 1 removed r from the list of v. Then v has a a trivial grandchild u with $\ell^*(u) = \{\mathsf{r}\}$. Hence, u has a trivial parent, and we have, by the inductive hypothesis, that G_u contains a hypergraph F'_u such that $F'_u \in L_{\mathbf{R}}(\Gamma')$ or $F'_u \in L_{\mathbf{R}''}(\Gamma')$, and $ext_{F'_u} = u$. Let F be a hypergraph such that $F = H[e/F'_u]$, where H is the hypergraph of the 1st or the 2nd rule for $\mathbf{B}$, and e is the hyperedge of H labeled with $\mathbf{R}$ or $\mathbf{R}''$, respectively. Observe that F corresponds to the subgraph of G_v induced on v and the vertices of F'_u. Hence, $ext_F = v$, and it follows that G_v contains F, and $F \in L_{\mathbf{B}}(\Gamma')$.

Now, suppose that v has, in addition, a non-trivial child B. Then, since B is a 2-connected block, there must exist vertices w, w' in B such that v, w, w' forms a triangle in B. Let F' be the hypergraph $F' = H'[e'/F, e''/H'']$, where F is the hypergraph from one of the above cases, H' is the hypergraph of the 1st rule for $\mathbf{B}''$, e, e'' are the hyperedges of H' labeled with $\mathbf{B}$ and $\mathbf{R}'$, respectively, and H'' is the hypergraph of the 2nd rule for $\mathbf{R}'$. Clearly, F' corresponds to the subgraph of G induced on w, w' and the vertices of F. Hence, $ext_{F'} = v$, and G_v clearly contains $F' \in L_{\mathbf{B}''}(\Gamma')$. Finally, suppose that v has a trivial grandchild u' with $\ell^*(u') = \{\mathsf{b}\}$. We observe that u' has cannot have any non-trivial children, because otherwise b would have been removed from the list of v by Algorithm 3.2 using Rule 7. Also, u' has no trivial grandchildren u'' with $\ell^*(u'') = \{\mathsf{b}\}$, since, similarly, b would have been removed from the list of v by Algorithm 3.2 using Rule 2. Hence, either u' has no children at all, in which case we must clearly have $\ell_0(u') = \{\mathsf{b}\}$, or u' has a trivial grandchild u'' with $\ell^*(u'') = \{\mathsf{r}\}$. Therefore, by the inductive hypothesis, we have that $G_{u'}$ contains a hypergraph $F_{u'} \in L_{\mathbf{B}}(\Gamma')$ with $ext_{F_{u'}} = u'$. Now, let F'' be the hypergraph $F'' = H'[e'/F, e''/H'''][e'''/F_{u'}]$, where H', e', e'', and F are from the previous case, H''' is the hypergraph of the 1st rule for $\mathbf{R}'$, and e''' is the hyperedge of H''' labeled with $\mathbf{B}$. Clearly, F'' is corresponding to the subgraph of G induced on the vertices of F and the

vertices of $F_{u'}$. Therefore, $ext_{F''} = v$, and we have that G contains $F'' \in L_{\mathbf{B}''}(\Gamma')$.

Now, suppose that the Block Rule removed r from the list of v when processing a non-trivial child (block) B of v. Let ℓ' be lists constructed from ℓ^* by changing the list of v to $\{\mathsf{r}\}$, that is, $\ell'(v) = \{\mathsf{r}\}$, and $\ell'(x) = \ell^*(x)$ for all $x \neq v$. Since r was removed from the list of v by the Block Rule, it follows that B with lists ℓ' is not list monopolar. Hence, by Theorem 4.3, B with lists ℓ' contains a minimal list non-extendable forbidden induced subgraph F_0, which is one of the configurations in Figure 4.1. Suppose that v does not belong to F_0. Then all vertices of F_0 are in $B - v$, and hence, $B - v$ with lists ℓ' is not list-monopolar. But, since for all vertices x of $B - v$, we have $\ell'(x) = \ell^*(x)$, also $B - v$ with lists ℓ^* is not list-monopolar. However, by Theorem 3.9, G with lists ℓ^* is list monopolar, because $\ell^*(x) \neq \emptyset$ for all $x \in V(G)$, and hence, also B and $B - v$ with lists ℓ^* are list-monopolar, a contradiction. Thus, it follows that v must belong to F_0. Now, since $\ell'(v) = \{\mathsf{r}\}$ and B with lists ℓ^* is list monopolar, we have that F_0 is either the 9^{th} or 10^{th} configuration in Figure 4.1. In the former case, the hypergraph F' corresponding to F_0 with lists ℓ^* is precisely the hypergraph of the 2^{nd} rule for $\mathbf{B}'$ and the 3^{rd} rule for $\mathbf{B}''$, and $ext_{F'} = v$. Hence, $F' \in L_{\mathbf{B}'}(\Gamma') \cup L_{\mathbf{B}''}(\Gamma')$, and F' is contained in G_v. On the other hand, in the latter case, F_0 is formed by v and a child u of B with $\ell^*(u) = \{\mathsf{r}\}$. Hence, by the inductive hypothesis, G_u contains $F_u \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$ with $ext_{F_u} = u$. Now, since both u and v belong to B, which is a 2-connected block, there must exist w in B such that u, v, w forms a triangle. Also, from the properties of block-vertex trees, we have that u separates all vertices of $G_u - u$ from the vertices of $B - u$. Hence, w does not belong to G_u, and hence, w does not belong to F_u. Now, let F' be the hypergraph $F' = H[e/F_u]$, where H is the hypergraph of the 1^{st} rule for $\mathbf{B}'$ or the 2^{nd} rule for $\mathbf{B}''$, and e is the hyperedge of H labeled with $\mathbf{R}'$ or $\mathbf{R}''$, respectively. Clearly, $F' \in L_{\mathbf{B}'}(\Gamma') \cup L_{\mathbf{B}''}(\Gamma')$, and, again, F' corresponds to the subgraph of G_v induced on v, w and the vertices of G_u. Hence, $ext_{F'} = v$, and F' is contained in G_v. If, in fact, $F' \in L_{\mathbf{B}'}(\Gamma')$, then we also have $F = F' \in L_{\mathbf{B}}(\Gamma') \cup L_{\mathbf{B}'}(\Gamma')$ with $ext_F = v$. Otherwise, we must have $F_u \in L_{\mathbf{R}''}(\Gamma')$, and we let $F = H'[e'/F_u]$, where H' is the hypergraph of the 2^{nd} rule for $\mathbf{B}$, and e' is the hyperedge of H' labeled with $\mathbf{R}''$. Clearly, $F \in L_{\mathbf{B}}(\Gamma') \cup L_{\mathbf{B}'}(\Gamma')$, and F corresponds to the subgraph of G_v induced on v and the vertices of F_u. Hence, $ext_F = v$, and F is contained in G_v.

Next, suppose that $\ell^*(v) = \{\mathsf{r}\}$. Again, recall that $\ell^*(v) \subseteq \ell_0(v)$. Hence, if $\ell_0(v) = \{\mathsf{r}\}$, then the vertex v with list $\ell_0(v)$ corresponds to the hypergraph F of the additional rule for $\mathbf{R}''$, which, clearly, belongs to $L_{\mathbf{R}''}(\Gamma')$, and $ext_F = v$. Hence, G_v contains F, and

$F \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$, and also $F' = F \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$.

On the other hand, if $\ell_0(v) = \{\mathsf{r}, \mathsf{b}\}$, then either Rules 2 - 8 or the Block Rule of Algorithm 3.2 must have removed b from the list of v. Let us assume that the algorithm first applies Rules 6, 5, 8, 7, 4, 3, 2, in that order, and then applies the Block Rule.

Hence, suppose first that b was removed from the list of v by Rule 6. Then v has a non-trivial parent B and a non-trivial child B'. Since v belongs to B', and B' is a 2-connected block, there must exist vertices u, w in B such that u, v, w forms a triangle. Now, let F be the hypergraph of the 2^{nd} rule for $\mathbf{R}'$. Clearly, F corresponds to the subgraph of G_v induced on u, v, w, and hence, $ext_F = v$. Therefore, G_v contains F, and $F \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$.

Now, suppose that b was removed from the list of v by Rule 5. Then v has two non-trivial children B and B'. Observe that v must have a trivial or no parent, since otherwise Rule 6 would remove b from v before Rule 5. Again, we have vertices u, w in B and u', w' in B' such that u, v, w and u, v', w' are triangles. Let F be the hypergraph of the 2^{nd} rule for $\mathbf{R}'$, and let $F' = H[e/F, e'/F]$, where H is the 1^{st} rule for $\mathbf{R}$, and e, e' are the hyperedges of H labeled with $\mathbf{R}'$. Again, clearly, F and F' correspond to the subgraphs of G_v induced on the vertices u, v, w and u, v, w, u', w', respectively. Hence, $ext_F = v$, $ext_{F'} = v$, and we have that G_v contains both F and F', and $F \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$, and $F' \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$.

Next, suppose that b was removed from the list of v by Rule 8. Then v has a trivial grandchild u, a non-trivial parent B, and $\ell^*(u) = \{\mathsf{b}\}$. Hence, by the inductive hypothesis, G_u contains a hypergraph F_u such that either $F_u \in L_{\mathbf{B}}(\Gamma')$ or $F_u \in L_{\mathbf{B}}(\Gamma') \cup L_{\mathbf{B}'}(\Gamma')$. If $F_u \in L_{\mathbf{B}}(\Gamma')$, we let $F = H[e/F_u]$, where H is the hypergraph of the 1^{st} rule for $\mathbf{R}'$, and e is the hyperedge of H labeled with $\mathbf{B}$. Similarly, if $F_u \in L_{\mathbf{B}'}(\Gamma')$, then we let $F = H'[e'/F_u]$, where H' is the hypergraph of the 1^{st} rule for $\mathbf{R}''$, and e' is the hyperedge of H' labeled with $\mathbf{B}'$. In both cases, F corresponds to the subgraph of G induced on v and the vertices of F_u. Hence, $ext_F = v$, and we have that G_v contains F, and $F \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$.

Now, suppose that b was removed from the list of v by Rule 7. Then v has a trivial grandchild u, which has a non-trivial child B, and $\ell^*(u) = \{\mathsf{b}\}$. It can be seen that G_v contains the hypergraph F from the case for Rule 8 (above), $F \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$, and $ext_F = v$. Hence, we only need to find the hypergraph F'. Observe that v must have a trivial parent, since otherwise Rule 8 would have removed b from the list of v before Rule 7. Now, by the inductive hypothesis, we have that G_u contains a hypergraph F'_u such that $F'_u \in L_{\mathbf{B}''}(\Gamma')$ or $F'_u \in L_{\mathbf{B}'}(\Gamma') \cup L_{\mathbf{B}''}(\Gamma')$. If $F'_u \in L_{\mathbf{B}''}(\Gamma')$, then we let $F' = H[e/F'_u]$, where H is the hypergraph of the 2^{nd} rule for $\mathbf{R}$, and e is the hyperedge of H labeled with $\mathbf{B}''$.

Similarly, if $F'_u \in L_{\mathbf{B}'}(\Gamma')$, then we let $F' = H'[e'/F'_u]$, where H' is the hypergraph of the 1st rule for $\mathbf{R}''$, and e' is the hyperedge of H' labeled with $\mathbf{B}'$. In both cases, F' corresponds to the subgraph of G induced on v and the vertices of F'_u. Hence, $ext_{F'} = v$, and we have that G_v contains F', and $F' \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$.

Next, suppose that b was removed from the list of v by Rule 4. Then v has a trivial grandchild u, a non-trivial child B, and $\ell^*(u) = \{\mathsf{b}\}$. Again, G_v contains the hypergraph $F \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$ with $ext_F = v$ from the case for Rule 8. If, in fact, $F \in L_{\mathbf{R}''}(\Gamma')$, we let $F' = F$, and we obtain that G_v contains $F' \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$ with $ext_{F'} = v$. Otherwise, we must have $F \in L_{\mathbf{R}'}(\Gamma')$. Now, since B is a 2-connected block, there must exist vertices w, w' such that v, w, w' forms a triangle. Hence, we let $F' = H[e/F, e'/H']$, where H is the hypergraph of the 1st rule for $\mathbf{R}$, H' is the hypergraph of the 2nd rule for $\mathbf{R}'$, and e, e' are the hyperedges of H labeled with $\mathbf{R}'$. Clearly, F' corresponds to the subgraph of G induced on w, w' and the vertices of F. Hence, $ext_{F'} = v$, and we have that G_v contains F', and $F' \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$.

Now, suppose that b was removed from the list of v by Rule 3. Then v has trivial grandchildren u, w with $\ell^*(u) = \ell^*(w) = \{\mathsf{b}\}$. Again, applying the argument from the case for Rule 8 for both the branch for u and the branch for w, we obtain that G_v contains $F, F'' \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$ with $ext_F = ext_{F''} = v$, where F contains u and F'' contains w. Hence, if either $F \in L_{\mathbf{R}''}(\Gamma')$ or $F'' \in L_{\mathbf{R}''}(\Gamma')$, then we let $F' = F$ or $F' = F''$, respectively, and obtain that G_v contains $F' \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$ with $ext_{F'} = v$. Otherwise, we have $F, F'' \in L_{\mathbf{R}'}(\Gamma')$, and we let $F' = H[e/F, e'/F'']$, where H is the hypergraph of the 1st rule for $\mathbf{R}$, and e, e' are the hyperedges of H labeled with $\mathbf{R}'$. Clearly, F' corresponds to the subgraph of G induced on the vertices of F and the vertices of F''. Hence, $ext_{F'} = v$, and we have that G_v contains F', and $F' \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$.

Now, suppose that b was removed from the list of v by Rule 2. Then v has a trivial grandchild u, which has a trivial grandchild w, and $\ell^*(u) = \ell^*(w) = \{\mathsf{b}\}$. We observe that both u and w cannot have any non-trivial children, since otherwise b would have been removed from the list of u by Algorithm 3.2 using Rules 4 and 7, respectively. Also, either $\ell_0(u) = \{\mathsf{b}\}$ or u has a trivial grandchild u' with $\ell^*(u') = \{\mathsf{r}\}$. This follows from the fact that u has no non-trivial children, and only Rule 1 and the Block Rule can remove r from the lists. Hence, by the inductive hypothesis, we have that G_u contains a hypergraph $F_u \in L_{\mathbf{B}}(\Gamma')$ with $ext_{F_u} = u$, and a hypergraph $F'_u \in L_{\mathbf{B}''}(\Gamma')$ with $ext_{F'_u} = u$. Now, we let $F = H[e/F_u]$, where H is the hypergraph of the 1st rule for $\mathbf{R}'$, and e is the hyperedge

of H labeled with $\mathbf{B}$. Similarly, we let $F' = H'[e'/F'_u]$, where H' is the hypergraph of the 2^{nd} rule for $\mathbf{R}$, and e' is the hyperedge of H' labeled with $\mathbf{B}''$. Clearly, F (respectively F') corresponds to the subgraph of G induced on v and the vertices of F_u (respectively F'_u). Hence, $ext_F = ext_{F'} = v$, and we have that G_v contains $F \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$, and $F' \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$.

Finally, suppose that b was removed from the list of v by the Block Rule when processing a non-trivial child B of v. Let ℓ' be lists constructed from ℓ^* be changing the list of v to $\{\mathsf{b}\}$, that is, $\ell'(v) = \{\mathsf{b}\}$, and $\ell'(x) = \ell^*(x)$ for all $x \neq v$. Since b was removed from the list of v by the Block Rule, we have that B with lists ℓ' is not list monopolar. Hence, by Theorem 4.3, B with lists ℓ' contains a minimal list non-extendable forbidden induced subgraph F_0, which is one of the configurations in Figure 4.1. Again, we have that v must be in F_0, and B with lists ℓ^* is list monopolar, both of which follow from the fact that G with lists ℓ^* is list monopolar. Hence, since $\ell'(v) = \{\mathsf{b}\}$, F_0 must be either the 5^{th}, or the 6^{th}, or the 7^{th}, or the 8^{th} configuration in Figure 4.1. First, if F_0 is either the 5^{th} or the 6^{th} configuration in Figure 4.1, then the hypergraph F'_0 corresponding to F_0 is the hypergraph of either the 4^{th} or the 5^{th} rule, respectively, for $\mathbf{R}$, and $ext_{F'_0} = v$. Then, clearly, $F'_0 \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$, and F'_0 is contained in G_v. On the other hand, if F_0 is the 7^{th} configuration in Figure 4.1, then the hypergraph F'_0 corresponding to F_0 is the hypergraph of the 3^{rd} rule for $\mathbf{R}$, and $ext_{F'_0} = v$. Now, let u be the vertex of F_0 with $\ell'(u) = \ell^*(u) = \{\mathsf{b}\}$ (that is, the vertex opposite v in F_0). Observe that u cannot have any non-trivial children, since otherwise Algorithm 3.2 would have removed b from the list of u using Rule 6. Also, u cannot have any trivial grandchildren u' with $\ell^*(u') = \{\mathsf{b}\}$, since otherwise Algorithm 3.2 would have removed b from the list of u using Rule 8. Hence, either $\ell_0(u) = \{\mathsf{b}\}$ or u has a trivial grandchild w with $\ell^*(w) = \{\mathsf{r}\}$. Now, by the inductive hypothesis, we have that G_u contains a hypergraph $F_u \in L_{\mathbf{B}}(\Gamma')$ with $ext_{F_u} = u$. Hence, we let $F' = F'_0[e/F_u]$, where e is the hyperedge of F'_0 labeled with $\mathbf{B}$. (This is the hyperedge incident to u.) Clearly, F' corresponds to the subgraph of G induced on the vertices of F_0 and the vertices of F_u, and $ext_{F'} = v$. Therefore, G_v contains F', and $F' \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$. Finally, suppose that F_0 is the 8^{th} configuration in Figure 4.1. Then F_0 consists of three vertices u, w, z (one of which is v) with edges uw and wz. Observe that, since B is 2-connected, there must exist a shortest path P from u to z which avoids w. Then, since G (and hence also B) is chordal, and P is an induced (chordless) path in B, each vertex of P must be adjacent to w. Now, if the length of P is two, then either v is one of the vertices u, z and we obtain precisely the previous case (the 7^{th} configuration

in Figure 4.1), or $v = w$ and then B with lists ℓ^* contains F_0, and hence, B with lists ℓ^* is not list monopolar, which is a contradiction. Similarly, if the length of P is three, then either $u \neq v$ or $z \neq v$, and in both cases, B with lists ℓ^* contains F_0 (the 5^{th} configuration in Figure 4.1), and hence, it is not list monopolar, a contradiction. Finally, if the length of P is at least four, then B (even without lists) is not monopolar, since F_0 then contains the 3^{rd} configuration in Figure 4.1, and hence, again, a contradiction.

The proof is now complete. □

We need to make one more observation. The following is easy to check.

Observation 4.8. *Each hypergraph in Figures 4.5, 4.6, and 4.7 is generated by* Γ'. □

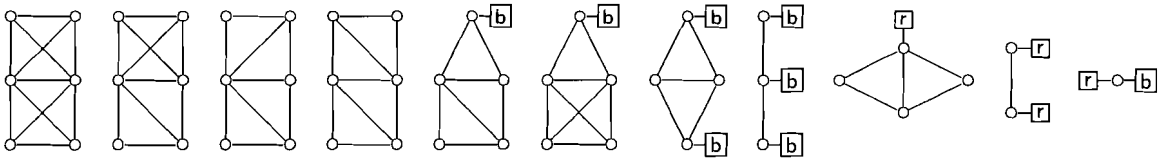

Figure 4.5: Hypergraphs (part 1) for the proof of Theorem 4.9.

We are now finally ready to prove the main theorem of this section.

Theorem 4.9. *The grammar* Γ' *generates (terminal hypergraphs corresponding to) all minimal list non-extendable forbidden induced subgraphs for monopolarity of chordal graphs.*

Proof. By Theorem 4.6, we have that each hypergraph generated by Γ' corresponds to a chordal graph with lists, which is minimal list non-extendable forbidden induced subgraph for monopolarity. Hence, it remains to prove that each chordal minimal list non-extendable forbidden induced subgraph for monopolarity corresponds to a hypergraph, which is generated by Γ'. In what follows, we prove a more general statement, namely that each chordal graph G with lists ℓ_0, which is not list monopolar, contains a hypergraph generated by Γ'.

Hence, let G be a chordal graph with lists $\ell_0(v) \subseteq \{\mathsf{r}, \mathsf{b}\}$, for all $v \in V(G)$, which is not list monopolar. If G is 2-connected, then, by Theorem 4.3, G must contain a minimal list non-extendable forbidden induced subgraph F_0, which is one of the configurations in Figure 4.3. Let F be the hypergraph corresponding to F_0. It can be immediately seen that F is one of the hypergraphs in Figure 4.5. Hence, by Observation 4.8, F is generated by Γ', and therefore, G contains a hypergraph generated by Γ'.

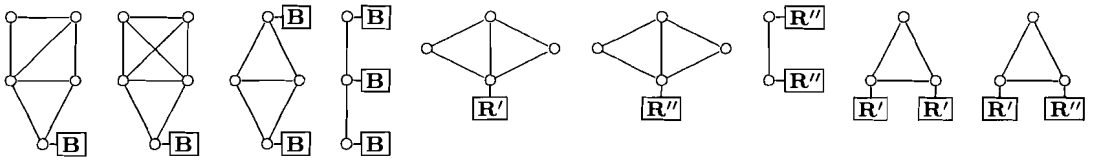

Figure 4.6: Hypergraphs (part 2) for the proof of Theorem 4.9.

Now, let T be a fixed block-vertex tree of G, and let ℓ^* be the lists computed by Algorithm 3.2 on G, ℓ_0, and T. Since G with lists ℓ_0 is not list monopolar, we have, by Theorem 3.9, that there must exist a vertex v of G such that $\ell^*(v) = \emptyset$, and $\ell^*(x) \neq \emptyset$ for all $x \in V(T_v) \setminus \{v\}$. If also $\ell_0(v) = \emptyset$, then the vertex v with its list $\ell_0(v)$ corresponds precisely the right-most hypergraph F in Figure 4.5. Hence, by Observation 4.8, F is generated by Γ', and therefore, G contains a hypergraph generated by Γ'.

Thus, we assume that $\ell_0(v) \neq \emptyset$. Suppose first that $\ell_0(v) = \{\mathsf{r}, \mathsf{b}\}$. Since $\ell^*(v) = \emptyset$, there are two rules among Rules 1 - 8 and the Block Rule, one of which removed r from the list of v and the other removed b from the list of v.

First, suppose that both r and b were removed from the list of v by the Block Rule when processing a non-trivial child B of v. It follows that B with lists ℓ^* is not list monopolar. Hence, by Theorem 4.3, B contains a minimal list non-extendable forbidden induced subgraph F_0, which is one of the configurations in Figure 4.1. If F_0 is one of the first four configuration in Figure 4.1, then, by Observation 4.8, the hypergraph corresponding to F_0 is generated by Γ', and hence, G contains a hypergraph generated by Γ'.

If F_0 is either the 5^{th} or the 6^{th} configuration in Figure 4.1, then F_0 contains a vertex u with list $\ell^*(u) = \{\mathsf{b}\}$. Clearly, u is a child of B. Hence, we observe that u cannot have any non-trivial children, since otherwise b would have been removed from the list of u by Algorithm 3.2 using Rule 6. Also, u has no trivial grandchildren u' with $\ell^*(u') = \{\mathsf{b}\}$, since otherwise b would have been removed from the list of u by Algorithm 3.2 using Rule 8. Hence, either $\ell_0(u) = \{\mathsf{b}\}$, or u has a trivial grandchild u' with $\ell^*(u') = \{\mathsf{r}\}$. Thus, by Lemma 4.7, we have that G_u contains a hypergraph $F_u \in L_{\mathbf{B}}(\Gamma')$ with $ext_{F_u} = u$. Therefore, we let $F = H[e/F_u]$, where H is the 1^{st} or the 2^{nd} hypergraph in Figure 4.6, and e is the hyperedge of H labeled with $\mathbf{B}$. Clearly, F corresponds to the subgraph of G induced on the vertices of F_u and F_0, and $F \in L_{\mathbf{S}}(\Gamma')$, since, by Observation 4.8, H is generated by Γ'. Hence, G contains a hypergraph generated by Γ'.

Similarly, if F_0 is the 7^{th} configuration in Figure 4.1, then F_0 has vertices u, w with $\ell^*(u) = \ell^*(w) = \{\mathsf{b}\}$, which are both children of B. Again, both u and w have no non-trivial children and no trivial grandchildren u' with $\ell^*(u') = \{\mathsf{b}\}$. Hence, by Lemma 4.7, G_u contains $F_u \in L_{\mathbf{B}}(\Gamma')$ with $ext_{F_u} = u$, and G_w contains $F_w \in L_{\mathbf{B}}(\Gamma')$ with $ext_{F_w} = w$. Thus, we let $F = H[e/F_u, e'/F_w]$, where H is the 3^{rd} hypergraph in Figure 4.6, and e, e' are the hyperedges of H labeled with $\mathbf{B}$. Clearly, F corresponds to the subgraph of G induced on the vertices of F_u, F_w and F_0, and we have $F \in L_{\mathbf{S}}(\Gamma')$. Hence, G contains a hypergraph generated by Γ'. Finally, if F_0 is the 8^{th} configuration in Figure 4.1, then F_0 has vertices u, w, z with $\ell^*(u) = \ell^*(w) = \ell^*(z) = \{\mathsf{b}\}$. Again, it follows that u, w, z are children of B, and G_u, G_v and G_w contain $F_u, F_w, F_z \in L_{\mathbf{B}}(\Gamma')$, respectively. Hence, $F = H[e/F_u, e'/F_w, e''/F_z]$, where H is the 4^{th} hypergraph in Figure 4.6, and e, e', e'' are the hyperedges of H labeled with $\mathbf{B}$, is generated by Γ' and contained in G.

Now, if F_0 is the 9^{th} configuration in Figure 4.1, then F_0 contains a vertex u with $\ell^*(u) = \{\mathsf{r}\}$, which is a child of B. Hence, By Lemma 4.7, we have that G_u contains $F_u \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$ with $ext_{F_u} = u$. Thus, we let $F = H[e/F_u]$, where H is either the 5^{th} or the 6^{th} hypergraph in Figure 4.6, and e is the hyperedge of H labeled with $\mathbf{R}'$ or $\mathbf{R}''$, respectively. Clearly, F corresponds to the subgraph of G induced on the vertices of F_u and F_0, and $F \in L_{\mathbf{S}}(\Gamma')$. Hence, G contains a hypergraph generated by Γ'.

Finally, if F_0 is the 10^{th} configuration in Figure 4.1, then F_0 contains vertices u, w with $\ell^*(u) = \ell^*(w) = \{\mathsf{r}\}$, which are both children of B, and hence, by Lemma 4.7, we obtain that G_u and G_w contain $F_u, F_w \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$, respectively. If, in fact, $F_u, F_w \in L_{\mathbf{R}''}(\Gamma')$, then we let $F = H[e/F_u, e'/F_w]$, where H is the 7^{th} hypergraph in Figure 4.6, and e, e' are the hyperedges of H labeled with $\mathbf{R}''$. Clearly, F corresponds to the subgraph of G induced on the vertices of F_u and F_w, and $F \in L_{\mathbf{S}}(\Gamma')$, and hence, G contains a hypergraph generated by Γ'. Otherwise, $F_u \in L_{\mathbf{R}'}(\Gamma')$ or $F_w \in L_{\mathbf{R}'}(\Gamma')$. Now, since B is a 2-connected block, there must exist a vertex z in B such that u, w, z forms a triangle. Hence, we let $F = H'[e/F_u, e'/F_w]$, where H' is the 8^{th} or the 9^{th} hypergraph in Figure 4.6, and e, e' are hyperedges of H' labeled with $\mathbf{R}'$ or $\mathbf{R}''$ (depending on where F_u and F_w belong). Again, F clearly corresponds to the subgraph of G induces on z and the vertices of F_u and F_w, and we have $F \in L_{\mathbf{S}}(\Gamma')$. Hence, G contains a hypergraph generated by Γ'.

From now on, we can assume that no non-trivial child of v caused the removal of both r and b from the list of v. Hence, suppose next that r (but not b) was removed from v by the Block Rule when processing a child B of v. Let G_B be the subgraph of G_v induced on v, the

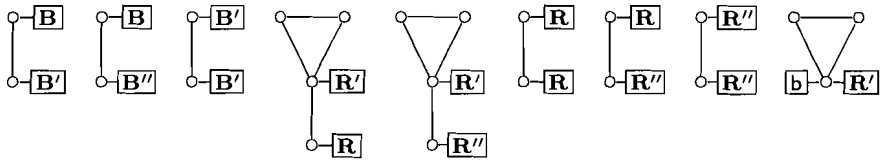

Figure 4.7: Hypergraphs (part 3) for the proof of Theorem 4.9.

vertices of B, and all blocks in the subtree rooted at B. (Observe that $v \in V(G_B)$.) Also, let ℓ' be lists constructed from ℓ^* by changing the list of v to $\{\mathsf{b}\}$. Since the Block Rule did not remove b from the list of v when processing B, it now follows that B with lists ℓ' is list monopolar, and hence, also G_B with lists ℓ' is list monopolar. Thus, we can apply Lemma 4.7 on G_B with lists ℓ', and we obtain that G_B contains a hypergraph $F_B \in L_{\mathbf{B}}(\Gamma') \cup L_{\mathbf{B}'}(\Gamma')$ with $ext_{F_B} = v$, and a hypergraph $F'_B \in L_{\mathbf{B}'}(\Gamma') \cup L_{\mathbf{B}''}(\Gamma')$ with $ext_{F'_B} = v$.

First, suppose that v has a non-trivial parent B', or a non-trivial child other B' than B. Hence, b was removed from the list of v by Algorithm 3.2 using Rules 6 or 5, respectively. Since B' is a 2-connected block, there must exist vertices w, z in B' such that v, w, z forms a triangle in B'. Thus, we let $F = H[e/F'_B, e'/H']$, where H is the hypergraph of the 3^{rd} or the 5^{th} rule for $\mathbf{S}$, H' is the hypergraph of the 2^{nd} rule for $\mathbf{R}'$, and e, e' are the hyperedges of H labeled with $\mathbf{B}'$ or $\mathbf{B}''$ and $\mathbf{R}'$, respectively. Clearly, F corresponds to the subgraph of G induced on v, w, z and the vertices of F'_B, and we have $F \in L_{\mathbf{S}}(\Gamma')$. Hence, G contains a hypergraph generated by Γ'.

Now, suppose that v has no non-trivial children other than B, and has trivial (or no) parent. Since b was removed from the list of v, it follows that v must have a trivial grandchild u with $\ell^*(u) = \{\mathsf{b}\}$. Hence, by Lemma 4.7, G_u contains a hypergraph F_u, such that $F_u \in L_{\mathbf{B}}(\Gamma')$ or $F_u \in L_{\mathbf{B}}(\Gamma') \cup L_{\mathbf{B}'}(\Gamma')$, and $ext_{F_u} = u$. If $F_u \in L_{\mathbf{B}}(\Gamma')$, then we let $F = H[e/F_u, e'/F'_B]$, where H is the 1^{st} or the 2^{nd} hypergraph in Figure 4.7, and e, e' are the hyperedges of H' labeled with $\mathbf{B}$ and $\mathbf{B}'$ or $\mathbf{B}''$, respectively. If $F_u \in L_{\mathbf{B}'}(\Gamma')$, then we let $F = H'[e/F_u, e'/F_B]$, where H' is the 1^{st} or the 3^{rd} hypergraph in Figure 4.7, and e, e' are the hyperedges of H labeled with $\mathbf{B}'$ and $\mathbf{B}$ or $\mathbf{B}'$, respectively. In both cases, F corresponds to the subgraph of G induced on the vertices of F_u and F_B, respectively, F'_B, and we have $F \in L_{\mathbf{S}}(\Gamma')$. Hence, G contains a hypergraph generated by Γ'.

From now on, we can assume that no non-trivial child of v caused the removal of r from the list of v. Hence, r must have been removed from the list of v by Rule 1, and it follows that v has a trivial grandchild u with $\ell^*(u) = \{\mathsf{r}\}$. Thus, by Lemma 4.7, G_u contains a

hypergraph $F'_u \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$. Now, let X denote the set of all trivial grandchildren u' of v with $\ell^*(u') = \{\mathsf{r}\}$, and let G'_v be the graph constructed from G_v by removing the subgraphs G_u for each $u \in X$. Also, let ℓ' be lists constructed from ℓ^* be changing the list of v to $\{\mathsf{r}\}$. We observe that v no longer has any trivial grandchildren u' with $\ell^*(u') = \{\mathsf{r}\}$ in G'_v, and hence, G'_v with lists ℓ' must be list monopolar. First, suppose that v has a non-trivial parent B. Then, since B is a 2-connected block, there exist vertices w, z in B such that v, w, z forms a triangle. Now, we can apply Lemma 4.7 to G'_v with lists ℓ', and we obtain that G'_v contains a hypergraph $F_v \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$ with $ext_{F_v} = v$. If $F_v \in L_{\mathbf{R}'}(\Gamma')$, then we let $F = H[e/F'_u, e'/F_v]$, where H is the 4^{th} or 5^{th} hypergraph in Figure 4.7, and e, e' are the hyperedges of H labeled with $\mathbf{R}$ or $\mathbf{R}''$ and $\mathbf{R}'$, respectively. Clearly, F corresponds to the subgraph of G induced on w, z and the vertices of F'_u and F_v, and we have $F \in L_{\mathbf{S}}(\Gamma')$. Hence, G contains a hypergraph generated by Γ'. On the other hand, if $F_v \in L_{\mathbf{R}''}(\Gamma')$, then we let $F = H'[e/F'_u, e'/F_v]$, where H' is either the 7^{th}, or the 8^{th} hypergraph in Figure 4.7, and e, e' are the hyperedges of H' labeled with $\mathbf{R}$ or $\mathbf{R}''$ and $\mathbf{R}''$, respectively. Clearly, F corresponds to the subgraph of G induced on the vertices of F'_u and F_v, and we have $F \in L_{\mathbf{S}}(\Gamma')$. Finally, if v has a trivial (or no) parent, then, by Lemma 4.7, G'_v additionally contains $F'_v \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$ with $ext_{F'_v} = v$. Thus, we let $F = H''[e/F'_u, e'/F'_v]$ where H'' is either the 6^{th}, the 7^{th}, or the 8^{th} hypergraph in Figure 4.7, and e, e' are the hyperedges of H'' labeled with $\mathbf{R}$ or $\mathbf{R}''$ (depending on where F'_u and F'_v belong). Clearly, F corresponds to the subgraph of G induced on the vertices of F'_u and F'_v, and we have $F \in L_{\mathbf{S}}(\Gamma')$. Hence, G contains a hypergraph generated by Γ'.

Now, we can assume that either $\ell_0(v) = \{\mathsf{r}\}$ or $\ell_0(v) = \{\mathsf{b}\}$. Let ℓ'_0 be lists constructed from ℓ_0 by changing the list of v to $\{\mathsf{r}, \mathsf{b}\}$. If G_v with lists ℓ'_0 is not list monopolar, then we can consider G_v with lists ℓ'_0 instead of ℓ_0, and repeat the above analysis. Hence, we can assume that G_v with lists ℓ'_0 is list monopolar. First, suppose that $\ell_0(v) = \{\mathsf{r}\}$, and let ℓ' be lists constructed from ℓ^* by changing the list of v to $\{\mathsf{b}\}$. Since G_v with lists ℓ'_0 is list monopolar, it follows that G_v with lists ℓ' is also list monopolar. Thus, by Lemma 4.7, G_v contains a hypergraph F_v such that $F_v \in L_{\mathbf{B}}(\Gamma')$, or $F_v \in L_{\mathbf{B}}(\Gamma') \cup L_{\mathbf{B}'}(\Gamma')$ with $ext_{F_v} = v$. Hence, we let $F = H[e/F_v, e'/H']$, where H is either the hypergraph of either the 2^{nd} or the 4^{th} rule for $\mathbf{S}$, H' is the hypergraph of the additional rule for $\mathbf{R}''$, and e, e' are the hyperedges of H labeled with $\mathbf{B}$ or $\mathbf{B}'$ and $\mathbf{R}''$, respectively. Clearly, F corresponds the subgraph of G induced on the vertices of F_v, and we have $F \in L_{\mathbf{S}}(\Gamma')$. Hence, G contains a hypergraph generated by Γ'.

Now, suppose that $\ell_0(v) = \{\mathsf{b}\}$, and let ℓ' be lists constructed from ℓ^* by changing the list of v to $\{\mathsf{r}\}$. Again, since G_v with lists ℓ'_0 is list monopolar, it follows that G_v with lists ℓ' is also list monopolar. Hence, by Lemma 4.7, G_v contains a hypergraph $F_v \in L_{\mathbf{R}'}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$ with $ext_{F_v} = v$. Suppose that v has a non-trivial parent B. Then, since B is 2-connected block, there exist vertices w, z in B such that v, w, z forms a triangle. Hence, if $F_v \in L_{\mathbf{R}'}(\Gamma')$, we let $F = H[e/F_v]$, where H is the 9^{th} hypergraph in Figure 4.7, and e is the hyperedge of H labeled with $\mathbf{R}'$. Clearly, F corresponds to the subgraph of G induced on w, z and the vertices of F_v, and $F \in L_{\mathbf{S}}(\Gamma')$. On the other hand, if $F_v \in L_{\mathbf{R}''}(\Gamma')$, then we let $F = H[e/F_v, e'/H'']$, where H is the hypergraph of the 2^{nd} rule for $\mathbf{S}$, H'' is the hypergraph of the additional rule for $\mathbf{B}$, and e, e' are the hyperedges of H labeled with $\mathbf{R}''$ and $\mathbf{B}$, respectively. Clearly, F corresponds to the subgraph of G induced on the vertices of F_v, and we have $F \in L_{\mathbf{S}}(\Gamma')$. Finally, suppose that v has a trivial (or no) parent, and hence, by Lemma 4.7, G_v additionally contains a hypergraph $F'_v \in L_{\mathbf{R}}(\Gamma') \cup L_{\mathbf{R}''}(\Gamma')$ with $ext_{F'_v} = v$. Hence, we let $F = H[e/F'_v, e'/H'']$, where H is the hypergraph of the 1^{st} or the 2^{nd} rule for $\mathbf{S}$, H'' is as before, and e, e' are the hyperedges of H labeled with $\mathbf{R}$ or $\mathbf{R}''$ and $\mathbf{B}$, respectively. Clearly, F corresponds to the subgraph of G induced on the vertices of F'_v, and we have $F \in L_{\mathbf{S}}(\Gamma')$. Hence, G contains a subgraph generated by Γ'.

The proof is now complete. □

As a corollary, we immediately obtain the following theorem.

Theorem 4.10. *The grammar Γ generates all minimal non-monopolar chordal graphs.* □

In closing, we remark that the proofs of Lemma 4.7 and Theorem 4.9 immediately imply an $O(n + m)$ time algorithm for finding a minimal list non-extendable forbidden induced subgraph for monopolarity in a chordal graph with lists, which is not list monopolar.

Hence, we have the following strengthening of Theorem 3.10.

Theorem 4.11. (Monopolarity of chordal graphs - Certifying version)
There is an $O(n + m)$ time algorithm to decide, for a chordal graph G with lists ℓ, whether G admits a monopolar partition which respects ℓ, and to find such a partition if one exists. If such a partition does not exist, the algorithm produces in time $O(n + m)$ an induced subgraph G' of G with lists $\ell'(v) \supseteq \ell(v)$, for each $v \in V(G')$, such that G' has $O(n)$ edges, and G' with lists ℓ' is a minimal list non-extendable forbidden induced subgraph for monopolarity.

Chapter 5

Subcolourings of Chordal Graphs

Recall, that a *k-subcolouring* of a graph G is a partition of the vertices of G into k sets $V_1 \cup V_2 \cup \ldots \cup V_k$, such that each V_i induces a disjoint union of *cliques* (complete graphs) in G, that is, each V_i induces a P_3-free graph. A graph G is called *k-subcolourable*, if there exists a k-subcolouring of G. The smallest integer k such that G is k-subcolourable is called the *subchromatic number* of G, and is denoted by $\chi_s(G)$.

Subcolourings and the subchromatic number were first introduced in [2]. Initially, the main focus was the properties of the subchromatic number $\chi_s(G)$. More recently, the complexity of recognizing k-subcolourable graphs has become a focus of attention. As we remark in Chapter 1, in general graphs this problem is NP-complete for all $k \geq 2$. Interestingly, the problem remains NP-complete for all $k \geq 2$, even if the graph is triangle-free and of maximum degree four [31] (and also [37]), or if the graph is a comparability graph [7]. On the other hand, there are several natural classes of graphs for which the problem has a polynomial time solution for any fixed k, e.g., graphs of bounded treewidth [31], or interval graphs [7], for which an $O(n^{2k+1})$ time algorithm is known (even for the list case).

In [7], the authors formulated the following open problem. Determine the complexity of the k-subcolouring problem for the class of chordal graphs. In the following sections, we completely answer this question by giving a polynomial time algorithm for $k = 2$, and showing NP-completeness for all $k \geq 3$.

5.1 2-subcolourings

In this section, we describe an $O(n^3)$ time algorithm for testing list 2-subcolourability of chordal graphs. In comparison, the best known algorithm for this problem to date for interval graphs is the algorithm of [7] which runs in time $O(n^5)$. Therefore, our algorithm also improves the complexity of this problem for the smaller class of interval graphs.

Instead of considering a k-subcolouring of G as a partition $V(G) = V_1 \cup V_2 \cup \ldots \cup V_k$, one can view it as a mapping $c : V(G) \to \{1, 2, \ldots, k\}$, where for every $i \in \{1, 2, \ldots, k\}$, the vertices of V_i are mapped to i. Therefore, we shall employ the terminology of colourings and refer to c as a colouring of G, and refer to the elements of $\{1, 2, \ldots, k\}$ as colours. (Note that c is not necessarily a proper colouring.) In particular, for a 2-subcolouring $V(G) = V_1 \cup V_2$, the associated colouring c is a mapping $c : V(G) \to \{\mathsf{r}, \mathsf{b}\}$. We shall refer to the elements of V_1 and V_2 as red and blue vertices and use letters r and b for the two colours.

5.1.1 Subcolouring digraph

Observe first that a graph G is k-subcolourable, if and only if, each connected component of G is k-subcolourable. Hence, let G be a connected chordal graph, and let T be a fixed clique-tree of G. (Clique-trees are defined in Section 1.2.) Let C_u for $u \in V(T)$ denote the maximal clique of G associated with u, and let $\mathcal{C}(X)$ denote the union of cliques associated with the vertices of a set $X \subseteq V(T)$, that is, $\mathcal{C}(X) = \bigcup_{u \in X} C_u$. In this section, we shall not consider T to be rooted. We shall use parentheses $(,)$ to denote the edges of T, to distinguish them from the edges of G. The removal of an edge (u, v) splits T into two subtrees; we shall denote by $T_{u,v}$ the subtree containing the vertex v, and by $T_{v,u}$ the subtree containing the vertex u. We denote $G_{u,v} = \mathcal{C}(T_{u,v})$ and $G_{v,u} = \mathcal{C}(T_{v,u})$.

Observe that in any k-subcolouring c of G, a vertex a in a clique C of G can have neighbours of the same colour as a in at most one connected component of $G \setminus C$. Based on this, we construct a multidigraph $\mathcal{D}_c(G)$ with coloured edges to capture the properties of the colouring c. We shall refer to $\mathcal{D}_c(G)$ as a *subcolouring digraph* for c. To avoid ambiguity, the edges of $\mathcal{D}_c(G)$ shall be referred to as *arcs* and denoted using angle brackets $\langle , \rangle$ to distinguish them from the edges of T and the edges of G. In particular, $\langle u, v \rangle_i$ shall denote an arc from u to v coloured i, and we shall write $\langle u, v \rangle$ for an arc from u to v (of some colour). The digraph $\mathcal{D}_c(G)$ is constructed as follows. The vertices of $\mathcal{D}_c(G)$ are the vertices of T, and for vertices u, v that are adjacent in T, there is an arc $\langle u, v \rangle_i$ in $\mathcal{D}_c(G)$, if there

exist vertices $a \in C_u$ and $b \in G_{u,v} \setminus C_u$ such that $ab \in E(G)$ and both a and b have the same colour i in c. Note that we have arcs in $\mathcal{D}_c(G)$ only between vertices that are adjacent in T. The construction is illustrated in Figure 5.1.

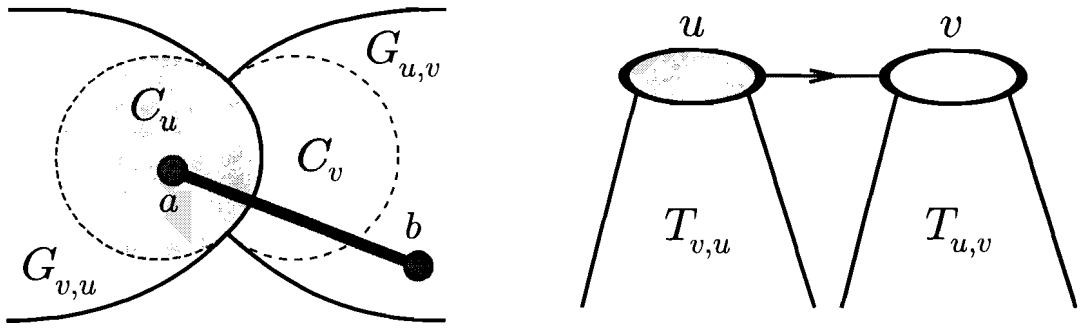

Figure 5.1: Illustrating the case when there is an arc $\langle u, v\rangle$ in $\mathcal{D}_c(G)$.

Formally, we define $\mathcal{D}_c(G)$ as follows.

(i) $V(\mathcal{D}_c(G)) = V(T)$

(ii) $$E(\mathcal{D}_c(G)) = \left\{ \langle u,v\rangle_i \;\middle|\; \begin{array}{ll} & (u,v) \in E(T) \\ \exists a \in C_u & ab \in E(G) \\ \exists b \in G_{u,v} \setminus C_u & c(a) = c(b) = i \end{array} \right\}$$

We have the following observations about $\mathcal{D}_c(G)$.

Proposition 5.1. *Let u, v, w be vertices of $\mathcal{D}_c(G)$ and let i be a colour from $\{1, 2, \ldots, k\}$. If c is a k-subcolouring then $\mathcal{D}_c(G)$*

(i) cannot contain both the arc $\langle u, v\rangle_i$ and the arc $\langle v, u\rangle_i$,

(ii) cannot contain both the arc $\langle u, v\rangle_i$ and the arc $\langle u, w\rangle_i$,

(iii) cannot contain all of the arcs $\langle u, v\rangle_1, \langle u, v\rangle_2, \ldots, \langle u, v\rangle_k$.

Proof. Suppose that (i) is false. Let a, b and a', b' be the vertices of G that caused the arcs $\langle u, v\rangle_i$ and $\langle v, u\rangle_i$ respectively to appear in $\mathcal{D}_c(G)$. It is not difficult to see that $a, a' \in C_u \cap C_v$ and $bb' \notin E(G)$. Hence, the graph induced on a, b, a', b' must contain an induced P_3 coloured i, a contradiction. One can easily repeat this same argument for pairs a, b and a', b' that falsify (ii).

Finally, let a_1, b_1, a_2, b_2, ..., a_k, b_k be the pairs of vertices that falsify *(iii)*. Since C_u and C_v are two different maximal cliques of G, there exists a vertex $d \in C_u \setminus C_v$. Again, it is not difficult to see that $a_i \in C_u \cap C_v$ for all i, and d is not a neighbour of any b_i. Now, any colour j assigned to d creates an induced P_3 coloured j on the vertices d, a_j, b_j, which yields a contradiction. □

Now, we turn to 2-subcolourings. In what follows, we shall assume that G is a connected 2-subcolourable chordal graph, T a fixed clique-tree of G, and c is a 2-subcolouring of G. Recall that we have red and blue vertices in G, and r and b denote the two colours.

We shall call an edge (u, v) in T a *strong* edge of T, if $\mathcal{D}_c(G)$ contains both $\langle u, v\rangle_{\mathsf{r}}$ and $\langle v, u\rangle_{\mathsf{b}}$, or both $\langle u, v\rangle_{\mathsf{b}}$ and $\langle v, u\rangle_{\mathsf{r}}$. We shall call an edge (u, v) in T a *weak* edge of T, if there is at most one arc between u and v in $\mathcal{D}_c(G)$. It follows from Proposition 5.1 that every edge in T must be either strong or weak.

Let u, v be adjacent vertices in T. Let $I_{u,v} = C_u \cap C_v$, and let $N_{u,v}$ be the set of all vertices of $G_{u,v} \setminus C_u$, which are neighbours of $C_u \cap C_v$. Furthermore, let $L_{u,v} = G_{v,u} \setminus C_v$ and $R_{u,v} = G_{u,v} \setminus (I_{u,v} \cup N_{u,v})$. (Note that $C_v \subseteq I_{u,v} \cup N_{u,v}$.) We have the following observation.

Proposition 5.2. *Let u, v be adjacent vertices in T.*

(i) If $\langle u, v\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$, or $\langle u, v\rangle_{\mathsf{b}} \in E(\mathcal{D}_c(G))$, then the vertices of $C_u \setminus C_v$ are all blue, or all red, respectively.

(ii) If $\langle u, v\rangle \notin E(\mathcal{D}_c(G))$ then the vertices of $I_{u,v}$ and $N_{u,v}$ are all red and all blue respectively, or all blue and all red respectively.

(iii) G has no induced P_3 having both a vertex of $L_{u,v}$ and a vertex of $R_{u,v}$.

Proof. For *(i)*, suppose that $\langle u, v\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$ (the other case is clearly symmetric), and let a, b be the red vertices that caused this arc. It is easy to see, that $a \in C_u \cap C_v$ and b is not adjacent to any vertex in $C_u \setminus C_v$. Hence, no vertex d of $C_u \setminus C_v$ can be red, since otherwise d, a, b is an induced red P_3.

For *(ii)*, let $a \in I_{u,v}$ and $b \in N_{u,v}$ be adjacent. Then a and b must have different colours, since otherwise we would have an arc $\langle u, v\rangle \in E(\mathcal{D}_c(G))$. Since C_u and C_v are different maximal cliques, there exists $d \in C_v \setminus C_u$. Now, the claim follows, because d is adjacent to all vertices of $I_{u,v}$, and hence any $a' \in I_{u,v}$ must have different colour from d.

Finally, for *(iii)*, let b, a, d be an induced P_3 in G (with edges ba and ad), which contains

both a vertex of $L_{u,v}$ and a vertex of $R_{u,v}$. Now, since $L_{u,v}$ and $R_{u,v}$ are completely non-adjacent, it follows that $a \notin L_{u,v} \cup R_{u,v}$. Hence, we can assume that $b \in L_{u,v}$, and $d \in R_{u,v}$. Now, if $a \in I_{u,v}$, then we must have $d \in N_{u,v}$, but $N_{u,v} \cap R_{u,v} = \emptyset$. Hence, $a \in N_{u,v}$, and $b \in I_{u,v}$, but $L_{u,v} \cap I_{u,v} = \emptyset$. Therefore, no such vertices b, a, d exist in G. □

Note that this observation allows us to consider independently the subgraphs of T, which no longer contain any weak edges. For strong edges in T we have the following observations.

Observation 5.3. *Every vertex of T has at most two incident strong edges, that is, the connected components formed by the strong edges of T are paths.*

Proof. It is not difficult to see that if a vertex u in T has three adjacent strong edges, then for at least two of them, say (u, v) and (u, w), we have arcs $\langle u, v\rangle$ and $\langle u, w\rangle$ of the same colour in $\mathcal{D}_c(G)$. By Proposition 5.1(ii) this is not possible. □

Proposition 5.4. *Let u be a vertex in T with distinct neighbours v, w, z.*

(i) If (v, u) is a strong edge and $\langle w, u\rangle \notin E(\mathcal{D}_c(G))$, then (z, u) is not a strong edge.

(ii) If (v, u) is a strong edge, $\langle w, u\rangle \notin E(\mathcal{D}_c(G))$, and $\langle z, u\rangle \notin E(\mathcal{D}_c(G))$, then $I_{u,w} = I_{u,z}$.

(iii) If $\langle v, u\rangle \notin E(\mathcal{D}_c(G))$, $\langle w, u\rangle \notin E(\mathcal{D}_c(G))$, and $\langle z, u\rangle \notin E(\mathcal{D}_c(G))$, then $I_{u,v} = I_{u,w}$ or $I_{u,w} = I_{u,z}$ or $I_{u,z} = I_{u,v}$.

Proof. For (i) suppose that the edges (v, u) and (z, u) are strong and that $\langle w, u\rangle \notin E(\mathcal{D}_c(G))$. Since G is connected, there exists $a \in I_{u,w}$. Without loss of generality, we may assume that $\langle u, z\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$. Hence, by Proposition 5.2(i), we obtain that $C_z \setminus C_v$ is all red, $C_v \setminus C_z$ is all blue, and $C_u \subseteq C_v \cup C_z$. Also, since C_u, C_z and C_v are different maximal cliques, we have $d \in C_z \setminus C_u$ and $b \in C_v \setminus C_u$. Now, clearly, $a \in C_v \cup C_z$. Hence, if a is red, then $a \in C_z$, and hence $\langle w, u\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$, and if a is blue, then $a \in C_v$, and hence $\langle w, u\rangle_{\mathsf{b}} \in E(\mathcal{D}_c(G))$, a contradiction.

For (ii) suppose that (v, u) is strong, $\langle w, u\rangle \notin E(\mathcal{D}_c(G))$, and $\langle z, u\rangle \notin E(\mathcal{D}_c(G))$ but $I_{u,w} \neq I_{u,z}$. Without loss of generality, we may assume that $I_{u,w} \not\subseteq I_{u,z}$ and that $\langle u, v\rangle_{\mathsf{r}}, \langle v, u\rangle_{\mathsf{b}} \in E(\mathcal{D}_c(G))$. Hence, there must exist a vertex $a \in I_{u,w} \setminus I_{u,z}$, a vertex $b \in I_{u,z}$ (since G is connected), and a vertex $d \in C_v \setminus C_u$ (since the cliques are maximal). Clearly, $c(a) \neq c(b)$, otherwise we have $\langle z, u\rangle \in E(\mathcal{D}_c(G))$. Now, since $\langle v, u\rangle_{\mathsf{b}} \in E(\mathcal{D}_c(G))$, it follows that the vertex d is red. Similarly, since $\langle u, v\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$, we have that $C_u \setminus C_v$ is all

blue. Hence, if a is red, then $a \in I_{u,v}$, and hence $\langle w,u\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$, and if b is red, then $b \in I_{u,v}$, and hence $\langle z,u\rangle_{\mathsf{r}} \in E(\mathcal{D}_c(G))$, a contradiction.

Finally, for (iii) suppose that none of $\langle v,u\rangle$,$\langle w,u\rangle$, and $\langle z,u\rangle$ is in $E(\mathcal{D}_c(G))$, but the three sets $I_{u,v}$, $I_{u,w}$ and $I_{u,z}$ are pairwise different. Without loss of generality, we may assume that $I_{u,v} \not\subseteq I_{u,w} \not\subseteq I_{u,z}$, and either $I_{u,v} \not\subseteq I_{u,z}$, or $I_{u,v} \not\supseteq I_{u,z}$. If $I_{u,v} \not\subseteq I_{u,z}$, suppose first that $J \neq \emptyset$, where $J = I_{u,v} \setminus (I_{u,w} \cup I_{u,z})$. It follows that we must have a vertex $a \in J$, a vertex $b \in I_{u,w} \setminus I_{u,z}$ and a vertex $c \in I_{u,z}$. Now, at least two of the vertices a,b,c must have the same colour. That gives us one of the arcs $\langle v,u\rangle$, $\langle w,u\rangle$, $\langle z,u\rangle$ in $E(\mathcal{D}_c(G))$, and hence, a contradiction. If $J = \emptyset$, we similarly obtain a contradiction for vertices $a \in (I_{u,w} \cap I_{u,v}) \setminus I_{u,z}$, $b \in (I_{u,z} \cap I_{u,v}) \setminus I_{u,w}$, and $c \in C_u \setminus C_v$. Now, if $I_{u,v} \not\supseteq I_{u,z}$, it follows that we have a vertex $a \in I_{u,v} \setminus I_{u,w}$, a vertex $b \in I_{u,w} \setminus I_{u,z}$, and $c \in I_{u,z} \setminus I_{u,v}$, and again a contradiction follows. □

Let $P_{u,v}$ denote the (unique) path from u to v in T. We shall call the path $P_{u,v}$ *strong*, if it is formed only by strong edges of T. Note that we also allow paths of zero length (paths $P_{u,v}$ with $u = v$); all such paths are trivially strong. A strong path is *maximal*, if it is not properly contained in another strong path. A vertex z in T adjacent to a vertex u is a *special neighbour* of u, if $\langle z,u\rangle \notin E(\mathcal{D}_c(G))$. For strong paths, we have the following claim.

Proposition 5.5. *For any strong path $P_{u,v}$ in T (possibly with $u = v$), there exist sets $A_{u,v}$ and $A'_{u,v}$ (both possibly empty) such that for any special neighbour s of some $t \in P_{u,v}$, we have $I_{s,t} = A_{u,v}$ or $I_{s,t} = A'_{u,v}$.*

Proof. If $u \neq v$, then, by Proposition 5.4(i), only u and v can have special neighbours. Hence, if u has a special neighbour z, we let $A_{u,v} = I_{z,u}$, and $A_{u,v} = \emptyset$ otherwise. If v has a special neighbour w, we let $A'_{u,v} = I_{w,v}$, and $A'_{u,v} = \emptyset$ otherwise. Now, the claim follows from Proposition 5.4(ii).

If $u = v$ and u has two special neighbours z and w with $I_{z,u} \neq I_{w,u}$, we define $A_{u,v} = I_{z,u}$, and $A'_{u,v} = I_{w,u}$. Otherwise, we let $A_{u,v} = I_{z,u}$ and $A'_{u,v} = \emptyset$, if u has a special neighbour z, but does not satisfy the previous condition. Finally, we let $A_{u,v} = A'_{u,v} = \emptyset$, if u has no special neighbours. Now, the claim follows from Proposition 5.4(iii). □

Finally, we give a complete characterization of the structure of the colouring c on the vertices of $\mathcal{C}(P_{u,v})$ of the strong paths $P_{u,v}$. Let $B_{u,v}$ and $B'_{u,v}$ denote the sets of neighbours of $A_{u,v}$ and $A'_{u,v}$ in $\mathcal{C}(P_{u,v})$, respectively.

Theorem 5.6. *For any strong path $P_{u,v}$ in T (possibly with $u = v$), we have*

(i) $\mathcal{C}(P_{u,v}) = C_u \cup C_v$, *and the vertices of* $C_u \setminus C_v$ *and* $C_v \setminus C_u$ *are all red and all blue respectively, or all blue and all red respectively,*

(ii) for every weak edge (s,t) *incident to* $P_{u,v}$, *the vertices of* $I_{s,t}$ *are all red or all blue,*

(iii) the vertices of $A_{u,v} \cup B'_{u,v}$ *and* $A'_{u,v} \cup B_{u,v}$ *are all red and all blue respectively, or all blue and all red respectively, and*

(iv) if in addition $P_{u,v}$ *is maximal, then any colouring* c' *of* $\mathcal{C}(P_{u,v})$ *satisfying* $(i) - (iii)$ *is a 2-subcolouring of* $\mathcal{C}(P_{u,v})$ *and can be extended to a 2-subcolouring of* G.

Proof. We prove (i) by induction on the length of the path $P_{u,v}$. If $u = v$, then there is nothing to prove. Hence, let w be the neighbour of v on $P_{u,v}$, and assume that $\mathcal{C}(P_{u,w}) = C_u \cup C_w$ and that the vertices of $C_u \setminus C_w$ and $C_w \setminus C_u$ are all red and all blue, respectively. Since (w,v) is a strong edge, by Proposition 5.2(i), we have that $C_v \setminus C_w$ is all blue, and $C_w \setminus C_v$ is all red. From this we deduce $C_w \setminus (C_u \cup C_v) = \emptyset$, which implies $C_w \subseteq C_u \cup C_v$, and the claim follows.

Now, claims (ii) and (iii) follow directly from Proposition 5.2(ii) and Lemma 5.5, since for any special neighbour s of $t \in P_{u,v}$, we have $I_{s,t} = A_{u,v}$ and $N_{s,t} \cap \mathcal{C}(P_{u,v}) = B_{u,v}$, or $I_{s,t} = A'_{u,v}$ and $N_{s,t} \cap \mathcal{C}(P_{u,v}) = B'_{u,v}$.

Finally, let c' be any colouring of $\mathcal{C}(P_{u,v})$ satisfying $(i) - (iii)$. Let c'' be a colouring of G constructed from the colouring c as follows. First, we exchange the colours red and blue on the vertices of $G_{t,s}$ for each neighbour $s \notin P_{u,v}$ of $t \in P_{u,v}$, so that the colours of $I_{s,t}$ match the colouring c'. (Note that since $P_{u,v}$ is maximal, the edge (s,t) is weak.) Then we replace the colours of $\mathcal{C}(P_{u,v})$ by c'. Clearly, c'' extends c'. We show that c'' is a 2-subcolouring of G. Suppose otherwise, and let b, a, d be an induced P_3 in G with edges ba and ad such that $c''(b) = c''(a) = c''(d)$. If $b, a, d \in \mathcal{C}(P_{u,v})$, then, by (i), it follows that the vertices b, a, d are all in C_u, or all in C_v, but that is not possible, since $bd \notin E(G)$. On the other hand, if $a \notin \mathcal{C}(P_{u,v})$, then it follows from the construction of c'' that $c(b) = c(a) = c(d)$, which is not possible, since c is a 2-subcolouring. Hence, for some neighbour $s \notin P_{u,v}$ of $t \in P_{u,v}$, we have that $a \in I_{t,s}$ and $b \in N_{t,s}$ and $d \in N_{s,t} \cap \mathcal{C}(P_{u,v})$. Now, if $\langle t, s \rangle \notin E(\mathcal{D}_c(G))$, then, by Proposition 5.2(ii), we have that $c(a) \neq c(b)$, and hence $c''(a) \neq c''(b)$, because $a, b \in G_{t,s}$. On the other hand, if $\langle t, s \rangle \in E(\mathcal{D}_c(G))$, then s must be a special neighbour of t, but then, by (iii), we have that $a \in A_{u,v}$ and $d \in B_{u,v}$, or $a \in A'_{u,v}$ and $d \in B'_{u,v}$, and hence, $c''(a) \neq c''(d)$, a contradiction. □

5.1.2 Algorithm

Now, we are ready to describe the algorithm for deciding (list) 2-subcolourability for chordal graphs. We assume that we are given a chordal graph G and a fixed clique-tree T of G, and we want to decide whether or not G is 2-subcolourable. Later, we describe how to obtain a list version of the algorithm.

This time, we consider T rooted at an arbitrary fixed vertex r. Therefore, we write $p[v]$ to denote the parent of a vertex v in T. For a vertex v in T, we denote by T_v the subtree of T rooted at v. We shall say that T_v is $(-)$ colourable, if there exists a 2-subcolouring c_v of $\mathcal{C}(T_v)$ such that the vertices of $I_{p[v],v}$ are all red or all blue. Similarly, T_v is $(+)$ colourable, if there exists a 2-subcolouring c_v of $\mathcal{C}(T_v)$ such that the vertices of $I_{p[v],v}$ and $N_{p[v],v}$ are all red and all blue, respectively, or all blue and all red, respectively. In the special case of the root r, when $p[v]$ does not exist, we shall say that T_r is $(-)$ colourable, if there exists a 2-subcolouring c_r of $\mathcal{C}(T_r) = G$.

Note that, by Lemma 5.5, for every strong path, we only need to consider up to two special neighbours z and w. If the path has only one such neighbour (or none), we use nil as the value of z or w. Therefore, we always view a path $P_{u,v}$ having two special neighbours z and w of u and v, respectively, but allow one (or both) of z and w to be nil. We shall say that the path $P_{u,v}$ is (z,w)-colourable, if there exists a 2-subcolouring $c_{u,v}$ of $\mathcal{C}(P_{u,v})$ such that for every edge (s,t) incident to $P_{u,v}$ in T, the vertices of $I_{s,t}$ are all red or all blue, and such that if z is not nil (w is not nil), then the vertices of $N_{z,u}$ ($N_{w,v}$ respectively) in $\mathcal{C}(P_{u,v})$ are all red or all blue.

The algorithm works as follows. It processes the vertices of T in a bottom-up order and identifies which edges of T could be weak for some 2-subcolouring of G. This is done by testing and recording for every vertex v in T, whether or not the subtree T_v is $(+)$ colourable, and whether or not the subtree T_v is $(-)$ colourable. (Note that T_v must be either $(+)$ colourable or $(-)$ colourable if $(v,p[v])$ is a weak edge in T for some 2-subcolouring of G.)

For a vertex x of T, the test for colourability of T_x is done as follows. First, if we are testing $(-)$ colourability, we precolour the vertices of $I_{p[x],x}$ by red or blue, otherwise we precolour the vertices of $I_{p[x],x}$ and $N_{p[x],x}$ by red and blue, respectively, or by blue and red, respectively. Then we choose a strong path $P_{u,v}$ in T_x that passes through x, and we choose special neighbours z and w of u and v, respectively. (See the above remark about special neighbours.) Then we test for the colourability of $P_{u,v}$ with respect to the chosen special

neighbours by applying Theorem 5.6. Finally, we recursively test, for every special neighbour y of $P_{u,v}$, whether or not the corresponding tree T_y is $(-)$ colourable or $(+)$ colourable, and for all other neighbours of $P_{u,v}$, whether or not their corresponding trees are $(+)$ colourable. We declare T_x $(-)$ colourable (or $(+)$ colourable, depending on the particular case), if and only if, the above tests succeed for some choice of u, v, and some choice of special neighbours of u and v. Note that, since we process the vertices in a bottom-up order, each recursive call amounts to a constant time table look-up.

If the algorithm succeeds to declare T_r $(-)$ colourable, then the graph G is 2-subcolourable, otherwise G is not 2-subcolourable. The correctness of this algorithm follows directly from Proposition 5.2 and Theorem 5.6. A more precise description of the algorithm can be found below (see Algorithms 5.1–5.4). Now, we shall discuss some implementation details.

Note that, in the procedure for testing colourability of a strong path $P_{u,v}$ (Algorithm 5.2), we need to precolour for each edge (s,t) incident to $P_{u,v}$, the vertices of the set $I_{s,t}$ either all red or all blue (as follows directly from Theorem 5.6). However, we cannot do this independently for each edge (s,t), since if two such sets $I_{s,t}$ and $I_{s',t'}$ intersect, and we want that in each set all vertices must have the same colour, then also all vertices in their union $I_{s,t} \cup I_{s',t'}$ must have the same colour. Similarly, by transitivity, if there are sets $I_{s,t} = I_{s_1,t_1}, I_{s_2,t_2}, \ldots, I_{s_k,t_k} = I_{s',t'}$ such that $I_{s_i,t_i} \cap I_{s_{i+1},t_{i+1}} \neq \emptyset$ for each $1 \leq i \leq k$, then also all vertices of $I_{s_1,t_1} \cup \ldots \cup I_{s_k,t_k}$ must have the same colour; we say that the sets $I_{s,t}$ and $I_{s',t'}$ are *conflicting*.

Hence, instead, our algorithm constructs a special collection $\mathcal{L}$ of sets with the property that no two sets of $\mathcal{L}$ intersect, each set $L \in \mathcal{L}$ is a union of pairwise conflicting sets $I_{s,t}$, and for any edge (s,t) incident to $P_{u,v}$, there is a set $L \in \mathcal{L}$ which contains all vertices of $I_{s,t}$. Once $\mathcal{L}$ is constructed, we can independently choose the red or blue colour for each set $L \in \mathcal{L}$, and colour all vertices of L using that colour. Since different sets from $\mathcal{L}$ are not intersecting, this procedure never colours a vertex both red and blue. Also, since each set $I_{s,t}$ is a subset of some $L \in \mathcal{L}$, this way we colour the vertices of $I_{s,t}$ all blue or all red for each edge (s,t) incident to $P_{u,v}$.

We obtain the sets $\mathcal{L}$ using a variant of the Union-find algorithm as follows. Initially, we let $\mathcal{L}$ be empty. Then we process the edges incident to $P_{u,v}$ one by one, and for each such edge (s,t), we first find all sets $L \in \mathcal{L}$ that intersect $I_{s,t}$. Then we remove all these sets and take their union augmented with the vertices of $I_{s,t}$, and add this newly created set into $\mathcal{L}$. Then we process next edge. It is not difficult to see that this procedure creates the sets $\mathcal{L}$

as described above. We implement this procedure as follows. We assign to each vertex of G a label indicating to which set of $\mathcal{L}$ that vertex belongs (or no label if it does not belong to any set of $\mathcal{L}$). When processing an edge (s,t), we find the sets of $\mathcal{L}$ that intersect $I_{s,t}$ by computing the set S of different labels on the vertices of $I_{s,t}$. This can be done using a table of size $O(n)$ in time $O(n)$. After that, we create the new set $L \supseteq I_{s,t}$ by labeling with a new label i all vertices whose label belongs to S and also all vertices of $I_{s,t}$. This can be again accomplished in time $O(n)$. After all edges (s,t) are processed, we construct the sets $L \in \mathcal{L}$ by scanning the labels of all vertices again in time $O(n)$.

Additionally, it is not difficult to see that this algorithm can be easily extended to solve the list 2-subcolouring problem in chordal graphs. Recall that this is the problem where each vertex v has a (non-empty) list $\ell(v)$ of admissible colours, in our case colours red and blue, and the task is to decide whether G has a 2-subcolouring that respects these lists. To obtain a modified algorithm solving the list version of this problem, we only need to modify the procedure for testing strong paths (Algorithm 5.2). In the procedure, we add an initial step in which we precolour by red all vertices v whose list contains only the red colour, and precolour by blue all vertices v whose list contains only the blue colour. Then in steps 3, 4, and 5, we choose the colours of the sets precoloured in these steps so that no vertex is precoloured both red and blue. If we cannot make such a choice, we declare $P_{u,v}$ not (z,w)-colourable. The correctness of this (modified) algorithm is straightforward.

Finally, we briefly explain how to construct a 2-subcolouring of G, if the algorithm declares G 2-subcolourable. To do that we modify the algorithm so that anytime it declares T_x $(+)$-colourable or $(-)$-colourable, we store the vertices u, v, z, w for which the test in Algorithms 5.3, 5.4 was successful. Using this additional information, we can construct a 2-subcolouring of G by backtracking from the root and colouring the vertices of strong paths $P_{u,v}$ that we used when declaring the subtrees of T $(+)$- respectively $(-)$-colourable, using the colourings constructed for $P_{u,v}$ by Algorithm 5.2. Since in the tests we always make sure that the colourings of different strong paths are compatible in the sense of Proposition 5.2, it can be seen that this procedure indeed gives a 2-subcolouring of G.

We summarize the main result of this section in the following theorem.

Theorem 5.7. *There exists an $O(n^3)$ time algorithm deciding, for a given chordal graph G (with lists $\ell(v)$, for all $v \in V(G)$), whether G is (list) 2-subcolourable; the algorithm also constructs a 2-subcolouring of G if one exists.*

Proof. As noted before, one can determine the maximal cliques of G and construct a clique-tree T of G in time $O(n+m)$. Also, it follows from the above remark that for any pair of vertices u, v and a choice of special neighbours z and w of u and v, respectively, one can determine (z,w)-colourability of the path $P_{u,v}$ in time $O(n^2)$. In the first part of the algorithm, this test is performed for every pair of vertices (including the choice of their special neighbours). This step can be implemented more efficiently by reusing the results for the subpaths, that is, starting from some vertex v and computing all paths from v, altogether in time $O(n^2)$. Therefore, the total running time for the first part of the algorithm is $O(n^3)$. In the second part, note that during the course of the algorithm (in the procedures for testing the colourability of a subtree T_x, Algorithms 5.3 and 5.4), we consider every path (including the choice of special neighbours) in T only once. Each path is processed in time $O(n)$, so in total, we have $O(n^3)$ time. Finally, constructing a 2-subcolouring of G (if exists) as describe above the theorem, can be easily done by reusing the colourings of strong paths we computed earlier, which only takes $O(n)$ time. □

Algorithm 5.1: The test for 2-subcolourability of G.

Input: A chordal graph G and a clique tree T of G rooted at r
Output: Decide whether G is 2-subcolourable

```
1  for every two vertices u, v in T do
2      for every neighbour z and w (including nil) of u and v respectively do
3          test and record whether P_{u,v} is (z, w)-colourable
4  initialize S ← ∅    (S is the set of processed vertices)
5  while S ≠ V(T) do
6      pick a vertex v ∉ S whose all children are in S
7      test and record whether T_v is (−) colourable
8      test and record whether T_v is (+) colourable (if v ≠ r)
9      S ← S ∪ {v}
10 if T_r is (−) colourable then
       return "G is 2-subcolourable"
11 else return "G is not 2-subcolourable"
```

Algorithm 5.2: The test whether $P_{u,v}$ is (z,w)-colourable.

Input: Vertices u, v of T, vertices z, w neighbours of u, v respectively or nil
Output: Decide whether $P_{u,v}$ is (z,w)-colourable

1 compute $\mathcal{C}(P_{u,v})$
2 **if** $\mathcal{C}(P_{u,v}) \neq C_u \cup C_v$ **then return** *"$P_{u,v}$ is not (z,w)-colourable"*
3 precolour the vertices of $C_u \setminus C_v$ and $C_v \setminus C_u$ by red and blue respectively (or blue and red respectively)
4 **if** $z \neq$ nil **then** precolour the vertices of $N_{z,u}$ red (or blue)
5 **if** $w \neq$ nil **then** precolour the vertices of $N_{w,v}$ blue (or red)
6 initialize the set $\mathcal{L} \leftarrow \emptyset$
7 **for** each edge (s,t) incident to $P_{u,v}$ **do**
8 compute the set $\mathcal{L}_s$ consisting of those sets from $\mathcal{L}$ which intersect $I_{s,t}$
9 $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathcal{L}_s \cup \left\{ I_{s,t} \cup \left(\bigcup_{L \in \mathcal{L}_s} L \right) \right\}$
10 **if** some $L \in \mathcal{L}$ contains both a precoloured red and a precoloured blue vertex **then return** *"$P_{u,v}$ is not (z,w)-colourable"*
11 **else return** *"$P_{u,v}$ is (z,w)-colourable"*

Algorithm 5.3: The test whether T_x is $(-)$ colourable.

Input: A vertex x in T
Output: Decide whether T_x is $(-)$ colourable

1 **for** each $u, v \in T_x$ such that $x \in P_{u,v}$ **do**
2 **for** every child z and w (including nil) of u and v respectively **do**
3 **if** $P_{u,v}$ is (z,w)-colourable and for each child $s \neq w, z$ of $P_{u,v}$ the tree T_s is $(+)$ colourable and either $z =$ nil (resp. $w =$ nil) or T_z (resp. T_w) is $(+)$ or $(-)$ colourable **then return** *"T_x is $(-)$ colourable"*
4 **return** *"T_x is not $(-)$ colourable"*

Algorithm 5.4: The test whether T_x is (+) colourable.

Input: A vertex x in T

Output: Decide whether T_x is (+) colourable

```
1 for each u ∈ T_x do
2     for every child z (including nil) of u do
3         if P_{u,x} is (z, p[x])-colourable
            and for each child s ≠ z of P_{u,x} the tree T_s is (+) colourable
             and either z = nil or T_z is (+) or (−) colourable
              then return " T_x is (+) colourable"
4 return " T_x is not (+) colourable"
```

5.2 3-subcolourings

In this section, we continue our study of complexity of the k-subcolouring problem in chordal graphs by showing, in contrast to the previous section, that it is NP-complete to decide, for a chordal graph, whether it has a 3-subcolouring.

To show this, we construct a reduction from the following auxiliary problem.

Problem: *AUX*

Input: Given $(S, z_0, z_1, \mathcal{T})$ where S is a set, z_0, z_1 are elements of S, and $\mathcal{T}$ is a set of ordered triples of distinct elements of S.

Query: Decide whether there exists a partition of S into two sets S_0 and S_1 with $z_0 \in S_0$ and $z_1 \in S_1$ such that for any triple $(x_1, x_2, x_3) \in \mathcal{T}$ and $j \in \{0, 1\}$, if $x_1, x_2 \in S_j$, then $x_3 \in S_j$. (We say that the triples $\mathcal{T}$ are *satisfied.*)

Proposition 5.8. *The problem AUX is NP-complete.*

Proof. It is easy to observe that AUX is in the class NP since, given a partition of S, one can easily in polynomial time verify that this partition forms a solution to the problem. Now, we show that the problem AUX is also NP-hard. We reduce the problem *NAE-3-SAT* (not all equal 3-SAT) to the problem AUX. Consider an instance to *NAE-3-SAT*, namely a formula φ in conjunctive normal form with exactly three distinct literals per clause. Let

$C_1, C_2, \ldots, C_m$ be the clauses of φ, and $v_1, v_2, \ldots, v_n$ be the variables of φ. Let $w_1, w_2, \ldots, w_n$ be a set of new variables. For any k, define $\eta(v_k) = v_k$ and $\eta(\neg v_k) = w_k$.

An instance $\mathcal{I}_\varphi = (S, z_0, z_1, \mathcal{T})$ to the problem AUX is constructed as follows. The set $S = \{v_1, \ldots, v_n, w_1, \ldots, w_n, x_F, x_T\}$, $z_0 = x_F$, $z_1 = x_T$, and the set of triples $\mathcal{T}$ consists of triples $(\eta(l_1^i), \eta(l_2^i), \eta(\neg l_3^i))$ for each clause $C_i = l_1^i \vee l_2^i \vee l_3^i$, and triples (v_i, w_i, x_T) and (v_i, w_i, x_F) for each variable v_i.

We show that there exists a satisfying truth assignment τ to φ such that in no clause all three literals evaluate to *true* by τ, if and only if, there exists a solution to the problem AUX for $\mathcal{I}_\varphi$. Suppose that there exists a truth assignment τ to φ as described above. The partition of $S = S_0 \cup S_1$ is constructed as follows. Put x_F into S_0, x_T into S_1, and for each variable v_i, if $\tau(v_i) = true$, then put v_i into S_1 and w_i into S_0, otherwise put v_i into S_0 and w_i into S_1. Now, we consider the triples of $\mathcal{T}$. Triples (v_i, w_i, x_T) and (v_i, w_i, x_F) are clearly satisfied, since v_i is never in the same set as w_i. Also, for a triple $(\eta(l_1^i), \eta(l_2^i), \eta(\neg l_3^i))$ of $\mathcal{T}$, since τ satisfies φ (in the sense described above), we have that at least one but at most two of l_1^i, l_2^i, l_3^i are evaluated to *true* by τ. Now, if $\eta(l_1^i), \eta(l_2^i) \in S_1$, then l_1^i and l_2^i are both *true*, and hence, l_3^i must be $false$, and $\neg l_3^i$ is *true*. Hence, by the above, it follows that $\eta(\neg l_3^i)$ belongs to S_1. Similarly, if $\eta(l_1^i), \eta(l_2^i) \in S_0$, we obtain that $\eta(\neg l_3^i)$ belongs to S_0. That proves that $S_0 \cup S_1$ is a solution for $\mathcal{I}_\varphi$.

Now, consider a solution $S = S_0 \cup S_1$ for $\mathcal{I}_\varphi$. Since $(v_i, w_i, x_F) \in \mathcal{T}$ and $(v_i, w_i, x_T) \in \mathcal{T}$, we have that v_i and w_i never belong to the same set S_j, $j \in \{0, 1\}$, since otherwise both x_F and x_T would have to belong to S_j which is not possible.

We construct the truth assignment τ as follows. We let $\tau(v_i) = true$, if $v_i \in S_1$, otherwise we let $\tau(v_i) = false$. Consider a clause $C_i = l_1^i \vee l_2^i \vee l_3^i$ of φ. Since $(\eta(l_1^i), \eta(l_2^i), \eta(\neg l_3^i)) \in \mathcal{T}$, we have that, if $\eta(l_1^i), \eta(l_2^i) \in S_j$, then $\eta(\neg l_3^i) \in S_j$; hence, $\eta(l_3^i) \in S_{1-j}$. Thus, either l_1^i, l_2^i are both *true* and l_3^i is $false$, or l_1^i, l_2^i are both $false$ and l_3^i is *true*, or l_1^i, l_2^i have different truth values. In any of these cases, at least one and at most two literals of C_i are *true*. It follows that τ is the required satisfying truth assignment for φ, and that concludes the proof. □

In fact, it can be seen that the above problem AUX can be easily reformulated in the language of boolean constraint satisfaction. By a result of Shaefer [60], it is known that all problems of boolean constraint satisfaction are either polynomial time solvable or NP-complete, and there is a concrete description which problems fall into which category. This provides us with an alternative proof of NP-completeness for AUX.

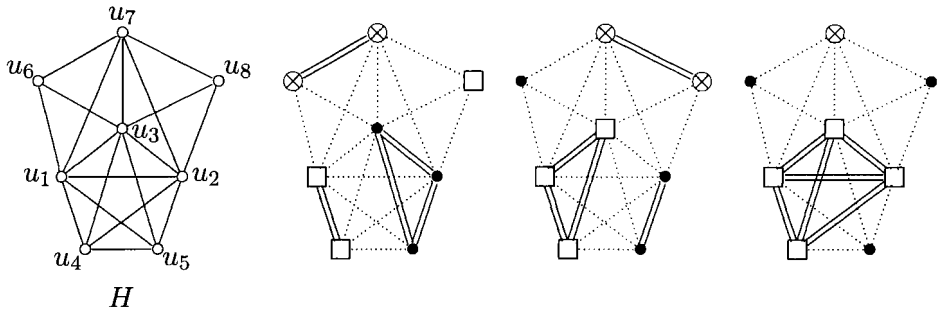

Figure 5.2: The graph H and sample 3-subcolourings of H.

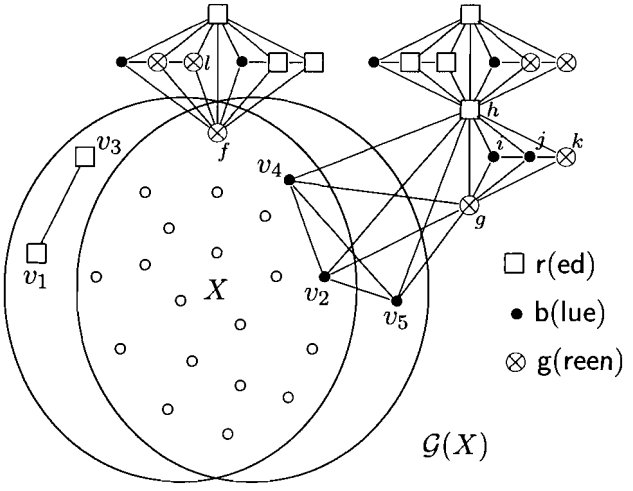

Figure 5.3: The graph $\mathcal{G}(X)$ with a sample 3-subcolouring.

We remark that, just like in the previous section, we shall use the terminology of colourings, refer to the three sets of a subcolouring partition as red, blue, and green vertices, and denote the three colours by r, b, and g, respectively.

Lemma 5.9. *Let H be the graph in Figure 5.2. Then, for any choice of colours c_1, c_2, c_3, $c_4, c_5 \in \{\mathsf{r}, \mathsf{b}\}$, there exists a 3-subcolouring c of H with $c(u_1) = c_1$, $c(u_2) = c_2$, $c(u_3) = c_3$, $c(u_4) = c_4$, $c(u_5) = c_5$, if and only if, $c_1 = c_3$ or $c_2 = c_3$ or $c_4 = c_5$.*

Proof. Consider some choice of colours for c_1, c_2, c_3, c_4, and c_5. If $c_1 = c_3 = \mathsf{r}$, we let $c(u_6) = \mathsf{b}$, and $c(u_7) = c(u_8) = \mathsf{g}$, and if $c_1 = c_3 = \mathsf{b}$, then we let $c(u_6) = \mathsf{r}$, and $c(u_7) = c(u_8) = \mathsf{g}$. We proceed similarly if $c_2 = c_3$. If $c_1 \neq c_3 \neq c_2$, then clearly $c_1 = c_2$, and we consider the following cases. If $c_1 = c_2 = c_4 = c_5$, then we let $c(u_6) = c_3$, and $c(u_7) = c(u_8) = \mathsf{g}$, and if $c_1 = c_2 \neq c_4 = c_5$, then we let $c(u_7) = c_1 = c_2$, and $c(u_6) = c(u_8) = \mathsf{g}$.

It can observed that in any of the above cases, we obtain a 3-subcolouring of H. Finally, if $c_4 \neq c_5$, then either $c_1 = c_2 = c_4$ and $c_3 = c_5$, or $c_1 = c_2 = c_5$ and $c_3 = c_4$. In both cases, we have that the vertices u_6, u_7 and u_8 must be coloured by g, since otherwise we have a P_3 in H in colour r or b. However, then the vertices u_6, u_7, u_8 form a P_3 in colour g. Hence, in these cases, a 3-subcolouring does not exist, and that concludes the proof. □

Now, for a set of vertices X, let $\mathcal{G}(X)$ be the graph in Figure 5.3; note that any two vertices in the same circle are adjacent, in other words, the circles represent cliques, and the set X are the vertices shown as ○.

Lemma 5.10. *The graph $\mathcal{G}(X)$ is chordal, and in any 3-subcolouring c of $\mathcal{G}(X)$, we have $c(v_1) = c(v_3) \neq c(v_2) = c(v_4) = c(v_5)$ and $c(x) \neq c(f)$ for each $x \in X \cup \{v_1, v_2, v_3, v_4, v_5\}$.*

Proof. It is not difficult to observe that the graph $\mathcal{G}(X)$ is indeed chordal. Hence, suppose that c is a 3-subcolouring of $\mathcal{G}(X)$, and let N be the set of neighbours of the vertex f which appear above f in Figure 5.3. The vertices of N form a 3-subcolourable graph which is not 2-subcolourable (by Proposition 5.14). Therefore, in any 3-subcolouring of $\mathcal{G}(X)$, the vertices of N must use all three colours. Hence, there must be a vertex $l \in N$ with $c(f) = c(l)$. It follows that no neighbour of f below f can use the colour of f, in other words, we obtain $c(f) \neq c(x)$ for $x \in X \cup \{v_1, v_2, v_3, v_4, v_5\}$. By the same argument, we obtain that no neighbour of the vertex h below h uses the same colour as h, and, in particular, this is true for the vertices g, i, j, k. Note that the vertices i, j, k form a P_3. Hence, at least one of them, say k, must use the colour of g, that is, $c(g) = c(k)$. Now, it follows that $c(v_2) = c(v_4) = c(v_5)$, since the vertices v_2, v_4, v_5 must be coloured differently from both g and h, and there are only three colours available. Finally, since v_1, v_2, v_5 is a P_3 and v_3, v_2, v_5 is a P_3, we obtain $c(v_1) = c(v_3) \neq c(v_2)$. That concludes the proof. □

Now, we are ready to prove the main result of this section. It is easy to observe that the problem of 3-subcolouring is in the class NP, since, given a colouring of the input graph G, one can easily check whether each colour class induces a disjoint union of cliques. To prove that it is also NP-hard, we show a reduction from the problem AUX. Consider an instance to the problem AUX, namely a 4-tuple $(S, z_0, z_1, \mathcal{T})$, where S is a set, $z_0, z_1 \in S$, and $\mathcal{T}$ is a set of ordered triples of distinct elements of S. We construct a graph $G_{\mathcal{I}}$ as follows. We start with the graph $G = \mathcal{G}(X_S)$, where $X_S = \{x_s \mid s \in S \setminus \{z_0, z_1\}\}$, in which we relabel the vertices v_1, v_2 to x_{z_0}, x_{z_1}, respectively. Then, for each triple $(t_1, t_2, t_3) \in \mathcal{T}$, we add

into G a copy H_{t_1,t_2,t_3} of the graph H, and identify the vertices u_1, u_2, u_3, u_4, u_5 of H_{t_1,t_2,t_3} with the vertices $x_{t_1}, x_{t_2}, x_{t_3}, v_3, v_4$ of G in that order. The final graph G we obtain after considering all triples of $\mathcal{T}$ is the graph $G_{\mathcal{I}}$.

The following observation is easy to check.

Proposition 5.11. *The graph $G_{\mathcal{I}}$ is chordal.* □

Proposition 5.12. *The graph $G_{\mathcal{I}}$ is 3-subcolourable, if and only, if there exists a solution to the problem AUX for $\mathcal{I}$.*

Proof. Let $S = S_0 \cup S_1$ be a solution for $\mathcal{I}$. We construct a 3-subcolouring c of $G_{\mathcal{I}}$ as follows. First, we colour the vertices of the subgraph $\mathcal{G}(X_S)$ of $G_{\mathcal{I}}$ as shown in Figure 5.3. Then, we colour the vertices of $X_0 = \{x_s \mid s \in S_0\}$ by r, and the vertices of $X_1 = \{x_s \mid s \in S_1\}$ by b. It is easy to see that this way we obtain a 3-subcolouring of $\mathcal{G}(X_S)$. Now, we colour the vertices of H_{t_1,t_2,t_3}, for each triple $(t_1, t_2, t_3) \in \mathcal{T}$, as follows. Recall that, for each $j \in \{0, 1\}$, if $t_1, t_2 \in S_j$, then also $t_3 \in S_j$, and hence, if $c(x_{t_1}) = c(x_{t_2})$, then $c(x_{t_1}) = c(x_{t_2}) = c(x_{t_3})$. It follows (and also by Lemma 5.10) that we can colour the vertices of H_{t_1,t_2,t_3} using one of the colourings shown in Figure 5.2 without introducing a monochromatic P_3 in $G_{\mathcal{I}}$. After considering all triples of $\mathcal{T}$, we clearly obtain a 3-subcolouring of $G_{\mathcal{I}}$.

Conversely, let c be a 3-subcolouring of $G_{\mathcal{I}}$. Then, by Lemma 5.10, we have that $c(x_{z_0}) \neq c(x_{z_1})$. We define $S_0 = \{s \in S \mid c(x_s) = c(x_{z_0})\}$, and $S_1 = \{s \in S \mid c(x_s) = c(x_{z_1})\}$. Observe that S_0 and S_1 form a partition of S, and that $z_0 \in S_0$ and $z_1 \in S_1$. Now, consider a triple $(t_1, t_2, t_3) \in \mathcal{T}$, and suppose that $t_1, t_2 \in S_j$ for $j \in \{0, 1\}$. Hence, $c(x_{t_1}) = c(x_{t_2}) = c(x_{z_j})$. Also, by Lemma 5.10, we have $c(v_3) \neq c(v_4)$, and therefore, using Lemma 5.9, we obtain $c(x_{t_1}) = c(x_{t_2}) = c(x_{t_3})$. Hence, $t_3 \in S_j$, and therefore, it follows that $S_0 \cup S_1$ is a solution for $\mathcal{I}$, which concludes the proof. □

We have proved the following theorem.

Theorem 5.13. *It is NP-complete to decide, for a given chordal graph G, whether or not G admits a 3-subcolouring.* □

5.3 k-subcolourings

In this final section of this chapter, we complete the answer to the question of [7] by showing NP-completeness of the k-subcolouring problem in chordal graphs for all $k \geq 3$.

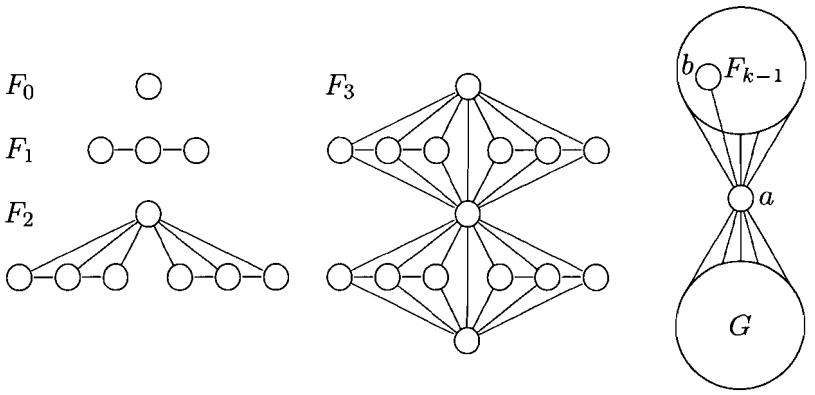

Figure 5.4: The graphs F_k for small k, and the construction from Theorem 5.15.

Let F_0 be the graph with a single vertex. For $k \geq 1$, let F_k be the graph constructed from the disjoint union of two copies of F_{k-1} by adding a new vertex a and making it adjacent to all other vertices. The graphs F_0, F_1, F_2 and F_3 are depicted in Figure 5.4. We have the following observation.

Proposition 5.14. *For any $k \geq 0$, the graph F_k is chordal and minimal non k-subcolourable.*

Proof. It is easy to observe that F_k is indeed a chordal graph. We show that it is not k-subcolourable, but any proper induced subgraph of F_k is k-subcolourable. We proceed by induction on k. For $k \leq 1$, the claim is trivial. Hence, let $k \geq 2$, let F' and F'' be the two copies of F_{k-1} in F_k, and let a be the vertex dominating both F' and F''. By the inductive hypothesis, we have that both F' and F'' are not $(k-1)$-subcolourable, but for any $x' \in V(F')$ and $x'' \in V(F'')$, the graphs $F' - x'$ and $F'' - x''$ are $(k-1)$-subcolourable. In particular, both F' and F'' are k-subcolourable, and in any k-subcolouring of F' or F'', all k colours must be used. Hence, for any choice of the colour for a, there is a vertex in F' and a vertex in F'' of that colour, which creates a monochromatic P_3, and therefore, F_k is not k-subcolourable. Now, let x be any vertex in F_k. If $x = a$, then $F_k - a$ is just the disjoint union of F' and F'', and hence, it is k-subcolourable. On the other hand, if $x \in V(F')$, then, since F' is minimal non $(k-1)$-subcolourable, it follows that $F' - x$ is $(k-1)$-subcolourable. Therefore, we can colour $F' - x$ and $F'' - x''$ with $k-1$ colours, and colour a and x'' with a new colour. Clearly, we get a k-subcolouring of $F_k - x$. By symmetry, we have that $F_k - x$ for $x \in V(F'')$ is also k-subcolourable. Therefore, this proves that F_k is a minimal non k-subcolourable graph as claimed. □

Finally, we are ready to proof the main theorem of this section.

Theorem 5.15. *For any $k \geq 3$, it is NP-complete to decide, for a given chordal graph G, whether or not G admits a k-subcolouring.*

Proof. For any $k \geq 3$, the problem of k-subcolouring of chordal graphs is clearly in NP. We show that it is also NP-hard. We proceed by induction on k. For $k = 3$, we obtain the claim by Theorem 5.13. Hence, let $k \geq 4$, and assume that the problem of $(k-1)$-subcolouring of chordal graphs is NP-hard. We reduce this problem to the problem of k-subcolouring of chordal graphs, which will imply the claim. Consider an instance to the former problem, namely a chordal graph G. Let G' be the graph constructed from the disjoint union of G and F_{k-1} by adding a new vertex a and making it adjacent to all other vertices (see Figure 5.4). We now show that G is $(k-1)$-subcolourable, if and only if, G' is k-subcolourable. Let c be a $(k-1)$-subcolouring of G. Also, by Proposition 5.14, let c'' be a $(k-1)$-subcolouring of $F_{k-1} - x$ for some $x \in V(F_{k-1})$. The colouring c' of G' is obtained by combining the colourings c and c'', and then colouring a and x with a new colour. It is easy to see that c' is a k-subcolouring of G'. Now, let c' be a k-subcolouring of G'. Again, by Proposition 5.14, it follows that the vertices of F_{k-1} must use all k colours. Hence, let b be a vertex of F_{k-1} in G' which uses the same colour as a. If some vertex $d \in V(G)$ also uses the colour of a, then b, a, d forms a monochromatic P_3. Hence, the vertices of G use at most $k-1$ colours, which implies that the colouring c' restricted to the vertices of G must be a $(k-1)$-subcolouring of G. That concludes the proof. □

Chapter 6

P_k-transversals of Chordal Graphs

Recall that an H-transversal of a graph G is a subset S of the vertices of G that intersects each induced copy of H in G, that is, $G - S$ is H-free. In this section, we study the complexity of several types of P_k-transversal problems in chordal graphs, namely the P_j-free and the K_j-free P_k-transversal problems. For both types of these problems, we show that it is NP-complete to decide, for a chordal graph, whether it has such P_k-transversal. This is summarized by the following theorem.

Theorem 6.1. *Let $2 \leq j \leq k$ and $4 \leq k$. Then it is NP-complete to decide, for a given chordal graph G, whether G has a P_j-free P_k-transversal. It is also NP-complete to decide, for a given chordal graph G, whether G has a K_j-free P_k-transversal.*

6.1 Stable P_4-transversals

First, we prove Theorem 6.1 for the case $j = 2$, $k = 4$. This problem is also known as the stable P_4-transversal problem. (Note that any P_2- or K_2-free graph contains no edges, and hence, it must be a stable set, whence the name). We remark that this problem is known to be NP-complete for comparability graphs (and hence also for perfect graphs) [45].

We describe a polynomial time reduction from the problem $3SAT$. Consider an instance to the problem $3SAT$; namely, a formula φ in a conjunctive normal form with n variables $v_1, \ldots, v_n$ and m clauses $C_1, \ldots, C_m$ such that each clause C_j has exactly three distinct literals. (Recall that literal is either a variable v_i or its negation $\neg v_i$.) Let J_i^+ denote the indices j of the clauses C_j which contain the literal v_i, and let J_i^- denote the indices j of

the clauses C_j which contain the literal $\neg v_i$.

We now describe a chordal graph G_φ which will have a stable P_4-transversal, if and only if, the formula φ is satisfiable. For any $j \in \{1 \dots m\}$, let Y_j be the graph shown in Figure 6.1 with the three distinguished vertices l_1^j, l_2^j, and l_3^j. Let $G_0 = Y_1 \cup \dots \cup Y_m$ be the disjoint union of the graphs $Y_1, \dots, Y_m$, and for each $i \geq 1$, let G_i be the graph constructed as follows (see Figure 6.1). Starting from G_{i-1}, we add two adjacent vertices v_i and $\overline{v}_i$, and make them completely adjacent to all vertices of G_{i-1}. Then, for every $j \in J_i^+$, if v_i is the k-th literal of the clause C_j, we add a vertex v_i^j, and make it adjacent to v_i and to the vertex l_k^j of Y_j. Also, for every $j \in J_i^-$, if $\neg v_i$ is the k-th literal of the clause C_j, we add a vertex $\overline{v}_i^j$, and make it adjacent to $\overline{v}_i$ and to the vertex l_k^j in Y_j. (Note that we assume that a literal occurs in a clause only once.) Finally, we let G_φ be the graph G_n.

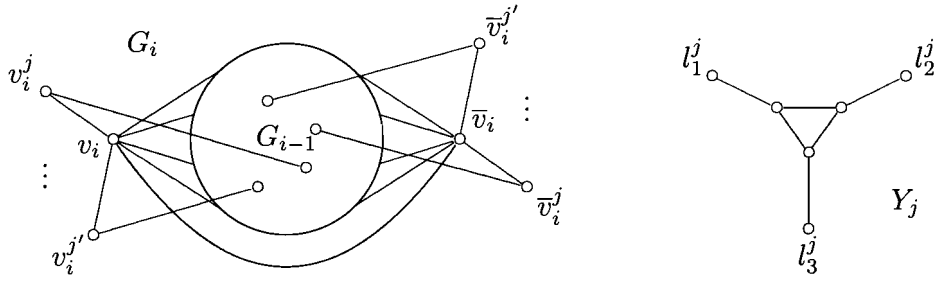

Figure 6.1: The graphs G_i and Y_j.

We can also describe the graph G_φ in the following (non-inductive) way.

(i) The vertex set of G_φ consists of the vertices $v_i, \overline{v}_i$ for each $1 \leq i \leq n$, the vertices v_i^j (respectively $\overline{v}_i^j$) for each occurrence of the literal v_i (respectively $\neg v_i$) in the clause C_j, and the vertices of the graphs Y_j for each $1 \leq j \leq m$, which include the distinguished vertices l_1^j, l_2^j, and l_3^j.

(ii) The vertex v_i (respectively $\overline{v}_i$) is adjacent to all vertices of Y_j for each $j \in \{1 \dots m\}$, to the vertices $v_{i'}, \overline{v}_{i'}$ for all i', to all vertices v_i^j (respectively $\overline{v}_i^j$) that may exist, and to all vertices $v_{i'}^j, \overline{v}_{i'}^j$ that may exist for all $1 \leq i' < i$.

(iii) The vertex v_i^j (respectively $\overline{v}_i^j$) is adjacent to the vertex v_i (respectively $\overline{v}_i$), to the vertex l_k^j, if v_i (respectively $\neg v_i$) is the k-th literal of the clause C_j, and to the vertices $v_{i'}, \overline{v}_{i'}$ for all $i < i' \leq n$.

(iv) The vertex l_k^j is adjacent to its only neighbour in Y_j, to the vertex v_i^j (respectively $\overline{v}_i^j$), where v_i (respectively $\neg v_i$) is the k-th literal of the clause C_j, and to the vertices

$v_{i'}, \overline{v}_{i'}$ for all $1 \leq i' \leq n$.

(v) The remaining vertices of Y_j are only adjacent to their respective neighbours in Y_j and to the vertices $v_i, \overline{v}_i$ for all $1 \leq i \leq n$.

First, we prove that the graph G_φ corresponding to the formula φ is chordal.

Proposition 6.2. *For each $0 \leq i \leq n$, the graph G_i is chordal. Hence, $G_\varphi = G_n$ is chordal.*

Proof. We prove the claim by induction. For $i = 0$, we observe that the graph Y_j is chordal, and hence G_0 is chordal. Therefore, let $i \geq 1$, and suppose that G_{i-1} is chordal; let π be a perfect elimination ordering of its vertices. It is straightforward to verify that v_i^1, v_i^2, ..., $\overline{v}_i^1$, $\overline{v}_i^2$, ..., π, v_i, $\overline{v}_i$ is a perfect elimination ordering of G_i, and the claim follows. □

We have the following observations about the graphs G_φ and Y_j.

Observation 6.3. *Every stable P_4-transversal of the graph Y_j contains at least one of the vertices l_1^j,l_2^j or l_3^j. Every maximal stable set of Y_j is a P_4-transversal.* □

Proposition 6.4. *Let S be a stable P_4-transversal of G_i, where $1 \leq i \leq n$. Then the vertices v_i and $\overline{v}_i$ are not in S, and if $v_i^j \notin S$ for some j, then $\overline{v}_i^{j'} \in S$ for all (possible) j'.*

Proof. First, observe that the vertex v_i is adjacent to all vertices of the graph Y_j for all j. Hence, if v_i belongs to the stable set S, then all vertices of Y_j must be in $G_i - S$, and hence $G_i - S$ contains an induced P_4, contrary to S being a P_4-transversal. The same holds for $\overline{v}_i$. Now, suppose that $v_i^j \notin S$ and also $\overline{v}_i^{j'} \notin S$ for some j, j'. Then by the previous argument also $v_i \notin S$ and $\overline{v}_i \notin S$, and hence S cannot be a P_4-transversal, since the vertices $v_i^j, v_i, \overline{v}_i, \overline{v}_i^{j'}$ induce a P_4 in $G_i - S$. □

Proposition 6.5. *The formula φ is satisfiable, if and only if, G_φ has a stable P_4-transversal.*

Proof. First suppose that τ is a satisfying truth assignment for φ. We use τ to construct a stable P_4-transversal of $G_\varphi = G_n$.

Let S_0^j be any maximal stable set in Y_j with the following property. For all k, the vertex $l_k^j \in S_0^j$, if and only if, v_i is the k-th literal of the clause C_j and $\tau(v_i) = true$, or $\neg v_i$ is the k-th literal of the clause C_j and $\tau(v_i) = false$. Clearly, since τ satisfies φ and hence satisfies the clause C_j, we have that at least one of the vertices l_1^j, l_2^j, l_3^j must belong to S_0^j.

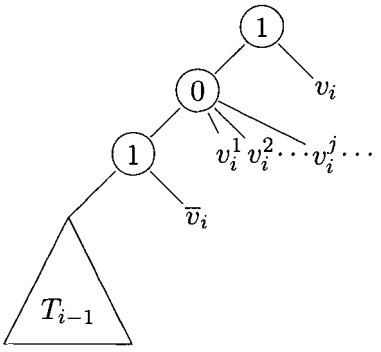

Figure 6.2: The cotree for the graph $G_i - S_i$.

Also, by Observation 6.3, S_0^j is a P_4-transversal of Y_j. Now, let $S_0 = S_0^1 \cup \ldots \cup S_0^m$. Since the graphs Y_j in G_0 are vertex disjoint, it follows that S_0 is a stable P_4-transversal of G_0.

Now, let $S = S_0 \cup S_1^+ \cup \ldots \cup S_n^+$, where $S_i^+ = \{\overline{v}_i^j \mid j \in J_i^-\}$, if $\tau(v_i) = true$, and $S_i^+ = \{v_i^j \mid j \in J_i^+\}$, if $\tau(v_i) = false$. We show that S is a stable P_4-transversal of G_φ. First, let $S_i = S_0 \cup S_1^+ \cup \ldots \cup S_i^+$. By definition, $S_i = S_{i-1} \cup S_i^+$ and $S_{i-1} \subseteq S_i$. We now show by induction that S_i is a stable P_4-transversal of G_i.

For $i = 0$, the claim follows from the above. Hence, suppose that $i \geq 1$, and assume that S_{i-1} is a stable P_4-transversal of the graph G_{i-1}. Without loss of generality, we may assume that $\tau(v_i) = true$. Hence, by definition, $S_i = S_{i-1} \cup \{\overline{v}_i^j \mid j \in J_i^-\}$. Now, we observe that each vertex $\overline{v}_i^j$ is only adjacent to the vertex $\overline{v}_i$ and the vertex l_k^j, if $\neg v_i$ is the k-th literal of the clause C_j. Hence, if $\neg v_i$ is the k-th literal of the clause C_j, then $l_j^k \notin S_0^j \subseteq S_0 \subseteq S_i$, because $\tau(v_i) = true$. Also, $\overline{v}_i \notin S_i$, by definition. This proves that S_i is a stable set.

It remains to show that S_i is a P_4-transversal of G_i, that is, $G_i - S_i$ is a P_4-free graph. By the inductive hypothesis, the graph $G_{i-1} - S_{i-1}$ is already P_4-free. Therefore, there exists a cotree T_{i-1} of this graph. (Cotrees are defined in Section 1.3.) To show the claim, we construct a cotree for $G_i - S_i$. Recall that, since $\tau(v_i) = true$, we have $l_k^j \in S_0^j \subseteq S_i$, if v_i is the k-th literal of the clause C_j. Hence, since l_k^j is the only neighbour of v_i^j in G_{i-1}, the vertex v_i^j has no neighbours in $G_{i-1} - S_{i-1}$. Therefore, it follows that the tree in Figure 6.2 is a tree representation (a cotree) of $G_i - S_i$, which shows that $G_i - S_i$ is P_4-free; hence, $S = S_n$ is a stable P_4-transversal of $G_\varphi = G_n$.

Now, suppose that G_φ has a stable P_4-transversal S. We construct a truth assignment τ for the formula φ in the following way. For every variable v_i, we set $\tau(v_i) = true$, if for some j, the vertex $v_i^j \notin S$; otherwise, we set $\tau(v_i) = false$. We show that τ satisfies φ.

Consider the clause C_j of φ. Since S is a stable P_4-transversal of G_φ, the set $S \cap Y_j$ is a stable P_4-transversal of Y_j. It follows from Observation 6.3 that there exists $\hat{k} \in \{1,2,3\}$ such that $l^j_{\hat{k}} \in S \cap Y_j \subseteq S$. Now, if v_i is the $\hat{k}$-th literal of the clause C_j, then v^j_i is adjacent to $l^j_{\hat{k}}$, which implies $v^j_i \notin S$. Therefore, $\tau(v_i) = true$, and hence τ satisfies C_j. Similarly, if $\neg v_i$ is the $\hat{k}$-th literal of the clause C_j, then we deduce $\overline{v}^j_i \notin S$. Now, by Proposition 6.4, we must have $v^{j'}_i \in S$ for all (possible) j'. Hence, by definition, $\tau(v_i) = false$, and we again conclude that τ satisfies C_j. Finally, since τ satisfies all clauses C_j of φ, it also satisfies the formula φ itself, and that concludes the proof. □

Now, the proof of Theorem 6.1 for $j = 2$, $k = 4$ is complete, once we observe that the graph G_φ can be constructed in time polynomial in the size of φ. We summarize this below.

Theorem 6.6. *It is NP-complete to decide, for a given chordal graph G, whether G has a stable P_4-transversal.* □

6.2 Forcing graphs

For the proof of the general case of Theorem 6.1, we shall need the following special "forcing" graphs. We first describe their structure, and then their properties.

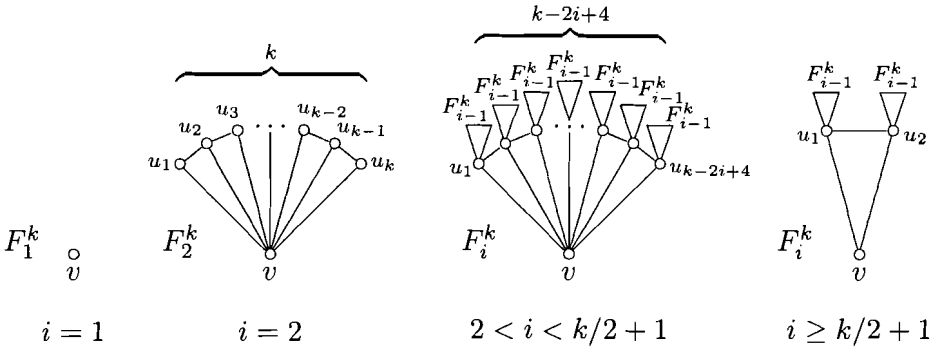

Figure 6.3: Forcing graphs for the P_k-transversal problem.

For every $1 \leq i \leq k$, the graph F^k_i with a distinguished vertex v is defined inductively as follows. For $i = 1$, the graph F^k_1 is a single vertex v with no edges. For $2 \leq i \leq k$, let $t = \max\{2, k - 2i + 4\}$; then the graph F^k_i is formed by t disjoint copies of F^k_{i-1} whose vertices v are joined into a chordless path $u_1, \ldots, u_t$ dominated by a new vertex v. This construction is illustrated above in Figure 6.3.

Proposition 6.7. *Every induced path of F_i^k ending in v has length at most $i-1$.*

Proof. By induction on i. If $i = 1$, the claim is trivially true. For $i > 1$, suppose that F_i^k contains an induced path P of length at least i that ends in v, and let u_j be the last vertex on P before v. If for some $j' \neq j$, we have $u_{j'}$ in P, then the vertices u_j and $u_{j'}$ must be adjacent, and hence $v, u_j, u_{j'}$ induces a triangle in P, contradicting that P is an induced path. Hence, it follows that $P - v$ forms an induced path in the copy of F_{i-1}^k in F_i^k attached to u_j. Since $P - v$ ends in u_j, using the induction hypothesis, we obtain that $P - v$ has length at most $i-2$, and hence, P has length at most $i-1$, a contradiction. □

For a graph G, let $\mathcal{P}_j^k(G)$ denote the set of P_j-free P_k-transversals of G.

Proposition 6.8. *Suppose that $1 \leq i \leq j < k$ or $1 \leq i < j \leq k$. Then for each $S \in \mathcal{P}_j^k(F_i^k)$, either $F_i^k[S]$ or $F_i^k - S$ contains an induced path of length $i-1$ ending in v. Furthermore, there exists $S \in \mathcal{P}_j^k(F_i^k)$ with $v \notin S$, and if $i < j$, there exists $S \in \mathcal{P}_j^k(F_i^k)$ with $v \in S$.*

Proof. By induction on i. If $i = 1$ and $1 < j \leq k$, then $\mathcal{P}_j^k(F_1^k) = \{\emptyset, \{v\}\}$. Clearly, for $S = \emptyset$ respectively $S = \{v\}$, we have that v is an induced path in $F_1^k - \emptyset = F_1^k$ respectively $F_1^k[\{v\}] = F_1^k$ of length 0 ending in v. Also, we have $\emptyset = S \in \mathcal{P}_j^k(F_1^k)$ with $v \notin S$, and since $i = 1 < j$, we have $\{v\} = S \in \mathcal{P}_j^k(F_1^k)$ with $v \in S$.

If $i = 1$ and $1 = j < k$, then $\mathcal{P}_j^k(F_1^k) = \{\emptyset\}$, and again, for $S = \emptyset$, we have that v is an induced path in $F_1^k - \emptyset = F_1^k$ of length 0 ending in v, and for $\emptyset = S \in \mathcal{P}_j^k(F_1^k)$, we have $v \notin S$. (Note that $1 \not< j$.)

This proves the claim for $i = 1$. Therefore, let $i > 1$, and let $S \in \mathcal{P}_j^k(F_i^k)$. Let $t = \max\{2, k - 2i + 4\}$. Note that by definition of t, for any $i > 1$, the graph F_i^k contains the vertices $u_1, \ldots, u_t$. Let $G_1, \ldots, G_t$ be the copies of F_{i-1}^k in F_i^k attached to $u_1, \ldots, u_t$, respectively. Using the inductive hypothesis, for each $1 \leq r \leq t$, let Q_r be an induced path of length $i-2$ ending in u_r that either belongs to $G_r[S]$ or $G_r - S$. We show that there must exist $1 \leq \ell \neq \ell' \leq t$ with $u_\ell \in S$ and $u_{\ell'} \notin S$. Otherwise either $\{u_1, \ldots, u_t\} \cap S = \emptyset$ or $\{u_1, \ldots, u_t\} \subseteq S$. In the former case, $P = Q_1 u_2 \ldots u_{t-1} (Q_t)^{-1}$ forms an induced path of length $t - 1 + 2(i-2) \geq k - 2i + 4 - 1 + 2i - 4 = k - 1$ in $F_i^k - S$, contradicting $S \in \mathcal{P}_j^k(F_i^k)$. In the latter case, $P = Q_1 u_2 \ldots u_{t-1} (Q_t)^{-1}$ forms an induced path of length at least $k - 1 \geq j - 1$ in $F_i^k[S]$, again contradicting $S \in \mathcal{P}_j^k(F_i^k)$. Hence, if $v \in S$, then $Q_\ell v$ is an induced path in $F_i^k[S]$ of length $i-1$, and if $v \notin S$, then $Q_{\ell'} v$ is an induced path in $F_i^k - S$ of length $i-1$. This proves the first part of the claim.

Now, we show the second part of the claim. Using the inductive hypothesis and that $i-1<j$, we have that, for $1 \leq r \leq t$, there exists $S_r \in \mathcal{P}_j^k(G_r)$ with $u_r \in S_r$ if $r=1$, and $u_r \notin S_r$ if $2 \leq r \leq t$. Let $S = S_1 \cup \ldots \cup S_r$. We show that $S \in \mathcal{P}_j^k(F_i^k)$. First, observe that $F_i^k[S]$ is the disjoint union of $G_1[S_1], \ldots, G_t[S_t]$. Since for each $1 \leq r \leq t$, we have $S_r \in \mathcal{P}_j^k(G_r)$, it follows that $F_i^k[S]$ is P_j-free. Now, suppose that $F_i^k - S$ contains a path P of length at least $k-1$.

First, assume that $v \in P$. Hence, there exist induced paths P', P'' (both possibly empty) such that $P = P'vP''$. By Proposition 6.7, both $P'v$ and vP'' are of length at most $i-1$. If either P' or P'' is empty, then P has length at most $i-1<k-1$, a contradiction. Hence, let u_ℓ be the last vertex of P', and let $u_{\ell'}$ be the first vertex of P''. If $i \leq k/2$, we obtain that P has length at most $2(i-1) \leq k-2<k-1$, a contradiction. Otherwise, if $k/2<i<k/2+1$, then $t \leq 3$, because $k-2i+4<k-2(k/2)+4=4$, and since $u_1 \in S$, it follows that u_ℓ and $u_{\ell'}$ must be adjacent, contradicting that P is an induced path. Finally, if $i \geq k/2+1$, we have $t=2$ since $k-2i+4 \leq k-2(k/2+1)+4=2$, and again since $u_1 \in S$, it implies that either P' or P'' is empty, a contradiction.

Therefore, $v \notin P$. If $P \cap \{u_1, \ldots, u_t\} = \emptyset$, then $P \subseteq G_r - S_r$ for some $1 \leq r \leq t$, which contradicts $S_r \in \mathcal{P}_j^k(G_r)$. Hence, let u_ℓ respectively $u_{\ell'}$ be the first respectively the last vertex of $\{u_1, \ldots, u_t\}$ that appears on P. If $\ell = \ell'$, then $P \subseteq G_\ell - S_\ell$, contradicting that $S_\ell \in \mathcal{P}_j^k(G_\ell)$. Hence, without loss of generality, we may assume that $\ell < \ell'$. Let P', Q, P'' be (possibly empty) induced paths in F_i^k such that $P = P'u_\ell Q u_{\ell'} P''$. Clearly, we must have $P' \subseteq G_\ell$, and $P'' \subseteq G_{\ell'}$. Now, since Q is an induced path, $v \notin P$, and each u_r, $1 \leq r \leq t$, is a cut-vertex in F_i^k, we must have $Q = u_{\ell+1}u_{\ell+2}\ldots u_{\ell'-1}$. Using Proposition 6.7, we obtain that both $P'u_\ell$ and $P''u_{\ell'}$ have length at most $i-2$. Now, since $u_1 \in S$, we have that $\ell \geq 2$, and hence $u_\ell Q u_{\ell'}$ has length at most $k-2i+2$. Altogether, we obtain that P has length at most $k-2i+2+2(i-2) = k-2<k-1$, a contradiction.

This proves that $S \in P_j^k(F_i^k)$. Now, assuming $i<j$, we also show that $S \cup \{v\} \in P_j^k(F_i^k)$ which will conclude the proof. Clearly, since $S \in P_j^k(F_i^k)$, we have that $F_i^k - S$ is P_k-free, so is $F_i^k - (S \cup \{v\})$. It remains to show that $F_i^k[S \cup \{v\}]$ is P_j-free. Let P be an induced path in $F_i^k[S \cup \{v\}]$ of length at least $j-1$. If $v \notin P$, then, since $u_r \notin S$ for $2 \leq r \leq t$, we must have $P \subseteq G_r[S_r]$ for some $1 \leq r \leq t$, contradicting that $S_r \in \mathcal{P}_j^k(G_r)$. Hence, $v \in P$, but then v must be the end-vertex of P, since only one neighbour of v, namely u_1, is in S. It follows, by Proposition 6.7, that P has length at most $i-1<j-1$, a contradiction. □

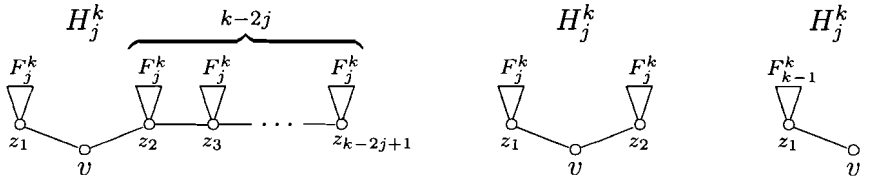

Figure 6.4: Forcing graphs H_j^k for the proof of Theorem 6.1.

For $2 \leq j < k$, let $t = \max\{2, k-2j+1\}$. Then the graph H_j^k with a distinguished vertex v is defined as the graph formed by t disjoint copies of the graph F_j^k whose vertices v are $z_1, \ldots, z_t$ such that $z_2, \ldots, z_t$ is a chordless path, and the vertices z_1, z_2 are adjacent to a new vertex v. Moreover, the graph H_k^k is defined as the graph F_{k-1}^k whose vertex v is z_1, and z_1 is adjacent to a new vertex v. This construction is illustrated in Figure 6.4.

Proposition 6.9. *Let $2 \leq j < k$. Then for each $S \in \mathcal{P}_j^k(H_j^k)$, we have $v \in S$. Furthermore, there exists $S \in \mathcal{P}_j^k(H_j^k)$ with $N(v) \cap S = \emptyset$. Additionally, for each $S \in P_k^k(H_k^k)$, we have that either $v \in S$ and $N(v) \cap S = \emptyset$, or $v \notin S$ and $N(v) \subseteq S$. Also, there exists $S, S' \in P_k^k(H_k^k)$ with $v \in S$, and $v \notin S'$.*

Proof. Let $S \in \mathcal{P}_j^k(H_j^k)$, and let $t = \max\{2, k-2j+1\}$. Note that H_j^k contains vertices $z_1, \ldots, z_t$. Let $G_1, \ldots, G_t$ be the copies of F_j^k in H_j^k attached to $z_1, \ldots, z_t$ respectively. Using Proposition 6.8, for each $1 \leq r \leq t$, we have that there exists an induced path Q_r either in $G_r[S]$ or in $G_r - S$ of length $j-1$ ending in z_r. If for some $1 \leq \ell \leq t$, we have $z_\ell \in S$, then Q_ℓ is a path of length $j-1$ in $H_j^k[S]$, contradicting $S \in \mathcal{P}_j^k(H_j^k)$. Hence, we must have $\{z_1, \ldots, z_t\} \cap S = \emptyset$. Now, it follow that $v \in S$, since otherwise $Q_1, v, z_2, \ldots, z_{t-1}, (Q_t)^{-1}$ forms an induced path of length $t + 2(j-1) \geq k - 2j + 1 + 2(j-1) = k-1$ in $H_j^k - S$, and that contradicts $S \in \mathcal{P}_j^k(H_j^k)$.

Now, by Proposition 6.8, for each $1 \leq r \leq t$, we have that there exists $S_r \in \mathcal{P}_j^k(G_r)$ with $z_r \notin S_r$. Let $S = S_1 \cup \ldots \cup S_t \cup \{v\}$. Clearly, $N(v) \cap S = \{z_1, z_2\} \cap S = \emptyset$. We show that $S \in \mathcal{P}_j^k(H_j^k)$. Observe that $H_j^k[S]$ is the disjoint union of $G_1[S_1], \ldots, G_t[S_t]$, and v. Hence, it follows that $H_j^k[S]$ is P_j-free. It remains to show that $H_j^k - S$ is P_k-free. Let P be an induced path of length at least $k-1$ in $H_j^k - S$. If $P \cap \{z_1, \ldots, z_t\} = \emptyset$, then $P \subseteq G_r - S$ for some $1 \leq r \leq t$, contradicting that $S_r \in \mathcal{P}_j^k[G_r]$. Hence, let z_ℓ respectively $z_{\ell'}$ be the first respectively the last vertex of $\{z_1, \ldots, z_t\}$ that appears on P. If $\ell = \ell'$, then $P \subseteq G_\ell - S$,

contradicting $S_\ell \in \mathcal{P}_j^k[G_\ell]$. Hence, we may assume that $\ell < \ell'$. Since $v \in S$, this also implies $\ell' \geq 3$, and hence $t = k - 2j + 1$. Let P', Q, P'' be (possibly empty) induced paths in H_j^k such that $P = P' z_\ell Q z_{\ell'} P''$. Clearly, $P' \subseteq G_\ell$ and $P'' \subseteq G_{\ell'}$. Since, for each $1 \leq r \leq t$, the vertex z_r is a cut-vertex of H_j^k, we must have $Q = z_{\ell+1} z_{\ell+2} \dots z_{\ell'-1}$. Using Proposition 6.7, we obtain that paths $P' z_\ell$ and $z_{\ell'} P''$ both have length at most $j - 1$. Altogether, it follows that P has length at most $t - 2 + 2(j-1) = k - 2j + 1 - 2 + 2j - 2 = k - 3 < k - 1$, a contradiction.

Finally, let $S \in \mathcal{P}_k^k(H_k^k)$. Using Proposition 6.8, we have that there exists an induced path P either in $H_k^k[S]$ or in $H_k^k - S$ of length $k-2$ ending in z_1. Now, if $z_1, v \in S$, then Pv forms an induced path of length $k-1$ in $H_k^k[S]$, and if $z_1, v \notin S$, then Pv forms an induced path of length $k-1$ in $H_k^k - S$, both contradicting $S \in \mathcal{P}_k^k(H_k^k)$. Hence, either $z_1 \in S$ and $v \notin S$, or $z_1 \notin S$ and $v \in S$. Now, by Proposition 6.8, there must exist $S' \in \mathcal{P}_k^k(H_k^k - v)$ with $z_1 \in S'$. It follows that $S' \in \mathcal{P}_k^k(H_k^k)$, and also that $S = V(H_k^k) \setminus S' \in \mathcal{P}_k^k(H_k^k)$. Clearly, we must have $v \in S$ and $v \notin S'$, which concludes the proof. □

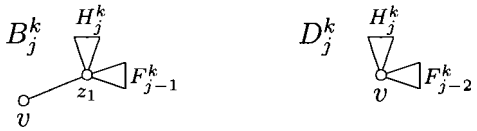

Figure 6.5: Forcing graphs B_j^k and D_j^k for the proof of Theorem 6.1.

Finally, for $2 \leq j < k$, the graph B_j^k with a distinguished vertex v is the graph formed by the disjoint union of a copy of the graph H_j^k and a copy of the graph F_{j-1}^k whose vertices v are identified and adjacent to a new vertex v. The graph D_j^k with a distinguished vertex v is the graph formed by the disjoint union of a copy of the graph H_j^k and a copy of the graph F_{j-2}^k whose vertices v are identified and are the distinguished vertex v of D_j^k. Both constructions are illustrated in Figure 6.5.

Proposition 6.10. *Let $2 \leq j < k$. Then for each $S \in \mathcal{P}_j^k(B_j^k)$, we have $v \notin S$. Furthermore, there exists $S \in \mathcal{P}_j^k(B_j^k)$ with $N(v) \subseteq S$.*

Proof. Let G_1 and G_2 denote the copies of H_j^k and F_{j-1}^k in B_j^k, respectively, and let $S \in \mathcal{P}_j^k(B_j^k)$. Then, by Proposition 6.9, we must have $z_1 \in S$. Hence, by Proposition 6.8, $B_j^k[S]$ contains an induced path P of length $j-2$ ending in z_1. Now, it follows that $v \notin S$, since otherwise Pv is an induced path of length $j-1$ in $B_j^k[S]$, contradicting $S \in \mathcal{P}_j^k(B_j^k)$. On

the other hand, using Propositions 6.8 and 6.9, we can obtain $S_1 \in \mathcal{P}_j^k(G_1)$ and $S_2 \in \mathcal{P}_j^k(G_2)$ such that $z_1 \in S_1 \cap S_2$, and all neighbours of z_1 in G_1 do not belong to S_1. We let $S = S_1 \cup S_2$, and it easily follows that $S \in \mathcal{P}_j^k(B_j^k)$ with $N(v) \subseteq S$. □

Using a similar proof, we obtain the following.

Proposition 6.11. *Let $3 \leq j < k$. Then for each $S \in \mathcal{P}_j^k(D_j^k)$, the graph $G[S]$ contains an induced path of length $j-3$ ending in v. Furthermore, there exists $S \in \mathcal{P}_j^k(D_j^k)$ such that every induced path of $G[S]$, which ends in v, has length at most $j-3$.* □

6.3 Stable P_5-transversals

Now, using a similar proof as in Section 6.1, we prove Theorem 6.1 for $j = 2$, $k = 5$. Again, we construct a polynomial time reduction from the problem $3SAT$. We shall explain how to modify the construction from the previous proof. Hence, let us assume that we have the formula φ with variables $v_1, \ldots, v_n$ and clauses $C_1, \ldots, C_m$ as before, and let Y_j' be the graph in Figure 6.6. Let $G_0' = Y_1' \cup \ldots \cup Y_m'$ be the disjoint union of graphs $Y_1', \ldots, Y_m'$, and for $i \geq 1$, let G_i' be the graph constructed as follows. Starting with G_{i-1}', we add the vertices v_i, $\overline{v}_i$, v_i^j, and $\overline{v}_i^j$ exactly as we did in the construction of G_i. Then we additionally add vertices $a, b, c, d, e, x_i, \overline{x}_i, z_i, \overline{z}_i$ as shown in Figure 6.6, and attach to the vertices a, b, c, d, e individual copies of the graph B_2^5. Again, we let $G_\varphi' = G_n'$.

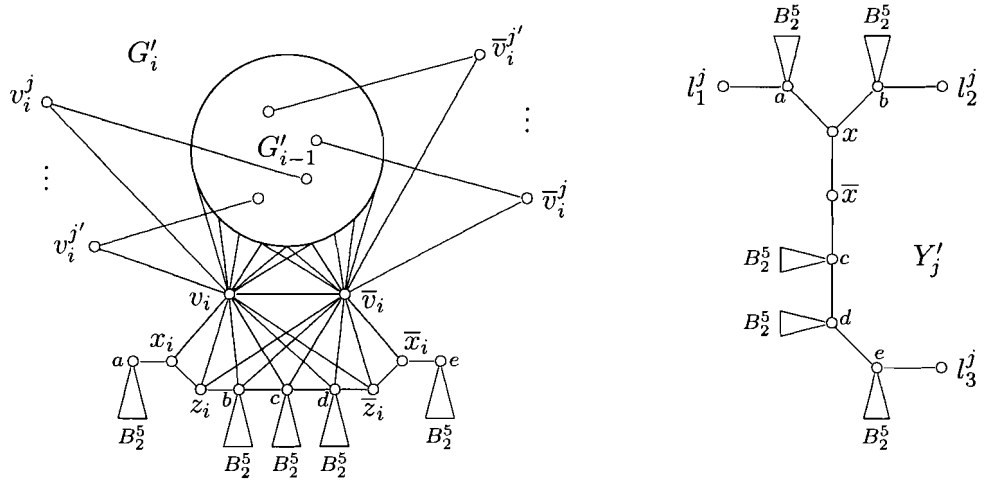

Figure 6.6: The graphs Y_j' and G_i'.

Observation 6.12. *For each $1 \leq i \leq n$, the graph G'_i is chordal; hence $G'_\varphi = G'_n$ is chordal.*

Proof. We prove the claim by induction. For $i = 0$, the claim follows from the chordality of Y'_j. Hence, let $i \geq 1$, and assume that G'_{i-1} is chordal, and let π be a perfect elimination ordering G'_{i-1}. Also, let $\pi_a, \pi_b, \pi_c, \pi_d, \pi_e$ be perfect elimination orderings of the copies of B_2^5 attached to the vertices a, b, c, d, e, respectively, such that π_a ends in a, π_b ends in b, π_c ends in c, π_d ends in d, and π_e ends in e. Then it is not difficult to verify that π_a, π_e, x_i, $\overline{x}_i$, z_i, $\overline{z}_i$, π_b, π_d, π_c, v_i^1, v_i^2, ..., $\overline{v}_i^1$, $\overline{v}_i^2$, ..., π, v_i, $\overline{v}_i$ is a perfect elimination ordering of G'_i. □

Proposition 6.13. *Every stable P_5-transversal of the graph Y'_j contains at least one of the vertices l_1^j,l_2^j or l_3^j. For each non-empty subset $X \subseteq \{1,2,3\}$, there exists a stable P_5-transversal S_X of Y'_j such that $l_k^j \in S_X$, if and only if, $k \in X$.*

Proof. Let S be a stable P_5-transversal of Y'_j. Suppose that none of the vertices l_1^j, l_2^j, l_3^j belongs to S. Then, by Proposition 6.10, none of the vertices a, b, c, d, e belongs to S. It follows that $x \in S$, since otherwise l_1^j, a, x, b, l_2^j is an induced P_5 in $Y'_j - S$. Also, $\overline{x} \in S$, since otherwise $\overline{x}, c, d, e, l_3^j$ is an induced P_5 in $Y'_j - S$. But then S is not an independent set, since $x, \overline{x} \in S$ and $x\overline{x}$ is an edge of Y'_j, a contradiction.

On the other hand, let $X \subseteq \{1,2,3\}$ be a non-empty set. Let B be the disjoint union of the copies of B_2^5 in Y'_j. Using Proposition 6.10, we obtain that there exists a stable P_5-transversal Z of B such that $\{a, b, c, d, e\} \cap Z = \emptyset$, but all neighbours of a, b, c, d, e in B belong to Z. Now, if $1 \in X$ or $2 \in X$, we let $S = \{l_i^j \mid i \in X\} \cup \{\overline{x}\} \cup Z$, and if $3 \in X$, we let $S = \{l_i^j \mid i \in X\} \cup \{x\} \cup Z$. In both cases, S is clearly a stable P_5-transversal of Y'_j. □

Proposition 6.14. *Let S be a stable P_5-transversal of G'_i, where $1 \leq i \leq n$. Then the vertices v_i and $\overline{v}_i$ are not in S, and if $v_i^j \notin S$ for some j, then $\overline{v}_i^{j'} \in S$ for all (possible) j'.*

Proof. Since v_i is adjacent to all vertices of Y'_j, if v_i belongs to S, then all vertices of Y'_j must be in $G'_i - S$, and hence $G'_i - S$ contains an induced P_5. Similarly for $\overline{v}_i$. Now, suppose that $v_i^j \notin S$ and also $\overline{v}_i^{j'} \notin S$ for some j, j'. Using Proposition 6.10, we obtain that none of the vertices a, b, c, d, e is in S. Hence, since also $v_i, \overline{v}_i \notin S$, we must have $x_i \in S$ and $\overline{x}_i \in S$, since otherwise $v_i^j, v_i, \overline{v}_i, \overline{x}_i, e$ respectively $\overline{v}_i^{j'}, \overline{v}_i, v_i, x_i, a$ is an induced P_5 in $G'_i - S$. But that implies $z_i, \overline{z}_i \notin S$, and hence, $z_i, b, c, d, \overline{z}_i$ is an induced P_5 in $G'_i - S$. □

Proposition 6.15. *The formula φ is satisfiable, if and only if, the graph G'_φ has a stable P_5-transversal.*

Proof. Again, let τ be a satisfying truth assignment for φ, and let X_j be the set of indices k from $\{1,2,3\}$ such that $k \in X_j$, if and only if, v_i is the k-th literal of C_j and $\tau(v_i) = true$, or $\neg v_i$ is the k-th literal of C_j and $\tau(v_i) = false$. Let S_0^j be the P_5-transversal S_{X_j} of Y'_j obtained using Proposition 6.13. Again, let $S'_0 = S_0^1 \cup \ldots \cup S_0^m$, and let $S'_i = S'_{i-1} \cup S_i^+$ where $S_i^+ = \{\overline{v}_i^j \mid j \in J_i^-\} \cup \{\overline{x}_i, z_i\} \cup Z_i$, if $\tau(v_i) = true$, and $S_i^+ = \{v_i^j \mid j \in J_i^+\} \cup \{x_i, \overline{z}_i\} \cup Z_i$, if $\tau(v_i) = false$, where Z_i is the union of stable P_5-transversals of the five copies of B_2^5 in G'_i. (Note that, by Proposition 6.10, there exists a P_5-transversal of B_2^5.)

We show by induction that $S' = S'_n$ is a stable P_5-transversal of G'_i. For $i = 0$, the claim follows immediately. Hence, let $i \geq 1$, and assume that S'_{i-1} is a stable P_5-transversal of G'_{i-1}. Without loss of generality, suppose that $\tau(v_i) = true$. Hence, $S'_i = S'_{i-1} \cup \{\overline{x}_i, z_i\} \cup \{\overline{v}_i^j \mid j \in J_i^-\} \cup Z_i$. Now, using the same argument as in the proof of Proposition 6.5, it follows that S'_i is a stable set. We now show that $G'_i - S'_i$ is P_5-free. Suppose otherwise, and let A be an induced P_5 in $G'_i - S'_i$. Let B denote the disjoint union of the copies of B_2^5 in G'_i, and let $C = B - \{a, b, c, d, e\}$. First, we have that A does not completely belong to G'_{i-1} or to B, since S'_{i-1} and Z_i are stable P_5-transversals of the respective graphs. Additionally, no vertex of C can belong to A, since by Proposition 6.10, the neighbours of a, b, c, d, e in C belong to Z_i. Also, e is not in A, since $\overline{x}_i \in S'_i$. Hence, we have that all vertices of $G'_i - S'_i$ except a, e, and the vertices of C, are adjacent to v_i, which implies that v_i is not in A. Hence, a and x_i are not in A, since v_i is not in A and $z_i \in S'_i$. Also, v_i^j is not in A, since v_i is not in A and $l_k^j \in S'_i$ (the only two neighbours of v_i^j), because v_i is the k-th literal of the clause C_j for some $k \in \{1,2,3\}$, and hence $k \in X_j$, which implies $l_k^j \in S_{X_j} \subseteq S'_0 \subseteq S'_i$. It remains to observe that all vertices, which we have not yet been ruled out, are adjacent to $\overline{v}_i$, which implies that $\overline{v}_i$ is not in A. That leaves only the vertices $b, c, d, \overline{z}_i$, which clearly do not form an induced P_5, and hence, we obtain a contradiction. This proves that S' is a stable P_5-transversal of G'_φ.

Now, let S' be a stable P_5-transversal of G'_φ. We need to show that φ is satisfiable. The proof is exactly as in Proposition 6.5, but we use Propositions 6.13 and 6.14 in place of Observation 6.3 and Proposition 6.4, respectively. That concludes the proof. □

Again, we summarize this in a theorem.

Theorem 6.16. *It is NP-complete to decide, for a given chordal graph G, whether G has a stable P_5-transversal.* □

6.4 P_j-free P_k-transversals

Now, we extend the results from the previous sections by showing a reduction from the stable P_4- and P_5-transversal problems in chordal graphs to the P_j-free P_k-transversal problem in chordal graphs; thus, completely proving the first part of Theorem 6.1.

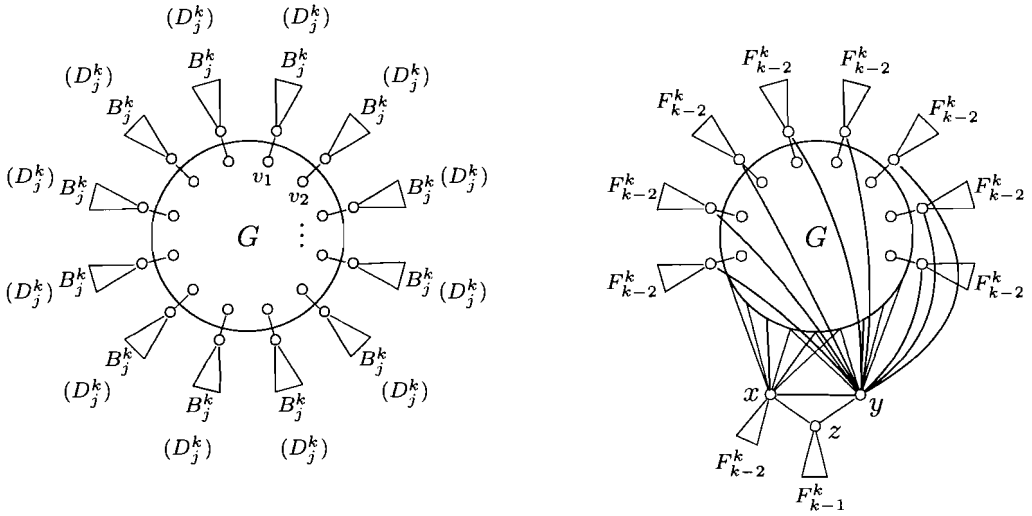

Figure 6.7: The graphs G' (G'') and G''' for the P_j-free P_k-transversal problem.

Hence, let $2 \leq j \leq k$, and let G be a graph with vertices $v_1, \ldots, v_n$. The graph G' (respectively G'') is the graph constructed from the graph G by adding n disjoint copies $G_1, \ldots, G_n$ of the graph B_j^k (respectively D_j^k), and for each $1 \leq i \leq n$, connecting v_i to the vertex v of G_i. Similarly, the graph G''' is the graph constructed from G by adding n disjoint copies $G_1, \ldots, G_n$ of the graph F_{k-2}^k, a copy G_x of F_{k-2}^k, a copy G_z of F_{k-1}^k, and vertices x, y, z such that xy,xz,yz are edges, x and z are identified with the vertices v of G_x and G_z, respectively, and for each $1 \leq i \leq n$, the vertex v of G_i is connected to v_i and y. The constructions are illustrated in Figure 6.7.

We have the following properties of the graphs G', G'', and G'''.

Observation 6.17. *The graphs G', G'', and G''' are chordal provided G is chordal.* □

Proposition 6.18. *For $2 \leq j < k$, the graph G has a P_j-free P_{k-2}-transversal, if and only if, the graph G' has a P_j-free P_k-transversal.*

Proof. Let $v_1, \ldots, v_n$ be the vertices of G, and $G_1, \ldots, G_n$ be the copies of B_j^k in G' adjacent to $v_1, \ldots, v_n$, respectively. Let $v_1', \ldots, v_n'$ be the vertices v of $G_1, \ldots, G_n$. (Note that $v_i v_i'$ is an edge for each $1 \leq i \leq n$.)

First, let S be a P_j-free P_{k-2}-transversal of G. By Proposition 6.10, we have that, for each $1 \leq i \leq n$, there exists $S_i \in \mathcal{P}_j^k(G_i)$ with $v_i' \notin S_i$, and $N(v_i') \cap V(G_i) \subseteq S_i$. Let $S' = S \cup S_1 \cup \ldots \cup S_n$. We show that $S' \in \mathcal{P}_j^k(G')$. First, it is easy to see that $G'[S']$ is P_j-free, since it is the disjoint union of $G[S], G_1[S_1], \ldots, G_n[S_n]$. On the other hand, suppose that P is an induced path of length $k-1$ in $G' - S'$. Clearly, since for each $1 \leq i \leq n$, the edge $v_i v_i'$ is a bridge of G', it follows that there can be at most two vertices of $\{v_1', \ldots, v_n'\}$ in P. Also, for any $1 \leq i \leq n$, if v_i' is in P, then it must be an end-vertex of P, since $N(v_i') \cap V(G_i) \subseteq S'$. Hence, if P contains two vertices v_i', v_j' with $i \neq j$, then P is an induced path from v_i' to v_j', and therefore, $P - \{v_i', v_j'\}$ is an induced path of length $k-3$ in $G - S$, which contradicts $S \in \mathcal{P}_j^{k-2}(G)$. Similarly, we get a contradiction with $S \in \mathcal{P}_j^{k-2}(G)$, if P contains only one, respectively, no vertex v_i', and $P - v_i'$, respectively, P completely belongs to $G - S$. Therefore, it follows that P must completely belong to $G_i - S_i$ for some $1 \leq i \leq n$, which also leads to a contradiction since $S_i \in \mathcal{P}_j^k(G_i)$. Hence, it shows that $G' - S'$ is P_k-free, and therefore, $S' \in \mathcal{P}_j^k(G')$.

On the other hand, let S' be a P_j-free P_k-transversal of G', and let $S = S' \cap V(G)$. We show that $S \in \mathcal{P}_j^{k-2}(G)$. Clearly, $G[S]$ is P_j-free, since $G'[S']$ is P_j-free. Now, suppose that $G - S$ contains an induced path P of length $k-3$. Let v_i and v_j be the first and the last vertex of P, respectively. By Proposition 6.10, we have $v_i' \notin S'$ and $v_j' \notin S'$. Hence, $v_i' P v_j'$ is an induced path of length $k-1$ in $G' - S'$, a contradiction. Therefore, $G - S$ is P_{k-2}-free, and hence, $S \in \mathcal{P}_j^{k-2}(G)$, which concludes the proof. □

Proposition 6.19. *For $3 \leq j < k$, the graph G has a stable P_k-transversal, if and only if, the graph G'' has a P_j-free P_k-transversal.*

Proof. Let $v_1, \ldots, v_n$ be the vertices of G, and $G_1, \ldots, G_n$ be the copies of D_j^k in G'' adjacent to $v_1, \ldots, v_n$, respectively. Let $v_1', \ldots, v_n'$ be the vertices v of $G_1, \ldots, G_n$, respectively.

First, let S be a stable P_k-transversal of G. By Proposition 6.11, for each $1 \leq i \leq n$, there exists $S_i \in \mathcal{P}_j^k(D_j^k)$ with $v_i' \in S_i$ and the property that every induced path of $G_i[S_i]$,

which ends in v_i', has length at most $j-3$. Let $S' = S \cup S_1 \cup \ldots \cup S_n$. We show that $S' \in \mathcal{P}_j^k(G'')$. It is easy to observe that $G'' - S'$ is P_k-free, since $G'' - S'$ is the disjoint union of $G - S, G_1 - S_1, \ldots, G_n - S_n$. Now, suppose that $G''[S']$ contains an induced path P of length $j-1$. Then, since S is a stable set, and $v_i v_i'$ is a bridge for each $1 \leq i \leq n$, we have that at most one vertex of S can be in P. Hence, if, for some $1 \leq i \leq n$, the vertex $v_i \in S$ is in P, then v_i must be an end-vertex of P, and it follows that $P - v_i$ is an induced path of length $j-2$ in $G_i[S_i]$, which ends in v_i', a contradiction. Similarly, we obtain a contradiction, if no vertex of S appears in P, since then P belongs completely to $G_i[S_i]$ for some $1 \leq i \leq n$. Hence, this shows that $G''[S']$ is P_j-free, and thus, $S' \in \mathcal{P}_j^k(G'')$.

Now, let S' be a P_j-free P_k-transversal of G'', and let $S = S' \cap V(G)$. We show that S is a stable P_k-transversal of G. Clearly, $G - S$ is P_k-free, since $G'' - S'$ is P_k-free. On the other hand, by Proposition 6.10, we have that, for each $1 \leq i \leq n$, $G_i[S']$ contains an induced path P_i of length $j-3$ ending in v_i'. Hence, if S contains adjacent vertices v_i, v_j, then $P_i v_i v_j (P_j)^{-1}$ is an induced path in $G''[S']$ of length $2(j-3)+3 = 2j-3 \geq j$ (since $j \geq 3$), a contradiction. Hence, S is an independent set, and we have $S \in \mathcal{P}_2^k(G)$, which concludes the proof. □

Proposition 6.20. *For $3 \leq k$, the graph G has a stable P_k-transversal, if and only if, the graph G''' has a P_k-free P_k-transversal.*

Proof. Let $v_1, \ldots, v_n$ be the vertices of G, and let $G_1, \ldots, G_n$ be the copies of F_{k-2}^k in G''' adjacent to $v_1, \ldots, v_n$, respectively. Let $v_1', \ldots, v_n'$ be the vertices v of $G_1, \ldots, G_n$. Also, let G_x be the copy of F_{k-2}^k attached to x, and G_z be the copy of F_{k-1}^k attached to z. (Note that we have edges $v_i v_i'$ and $v_i' y$ for each $1 \leq i \leq n$.)

Let S be a stable P_k-transversal of G. By Proposition 6.8, we have that, for each $1 \leq i \leq n$, there exists $S_i \in \mathcal{P}_k^k(G_i)$ with $v_i' \in S_i$, and also there exists $S_x \in \mathcal{P}_k^k(G_x)$ and $S_z \in \mathcal{P}_k^k(G_z)$ with $x \notin S_x$ and $z \in S_z$, respectively. Let $S' = S \cup S_1 \cup \ldots \cup S_n \cup S_x \cup S_z$. We show that $S' \in \mathcal{P}_k^k(G''')$. Suppose that $G'''[S']$ contains an induced path P of length $k-1$. Since $x, y \notin S'$, it follows that P belongs to $G'''[S \cup S_1 \cup \ldots \cup S_n]$. Also, S is stable, and hence, either P or $P - v_i$ is completely in $G_i[S_i]$ for some $1 \leq i \leq n$. In the former case, we have a contradiction, since $S_i \in \mathcal{P}_k^k(G_i)$. In the latter case, v_i' is an end-vertex of $P - v_i$. Hence, by Proposition 6.7, $P - v_i$ has length at most $k-3 < k-2$, which is, again, a contradiction. Now, suppose that $G''' - S'$ contains an induced path P' of length $k-1$. Since $z \in S'$ and $v_i' \in S$ for each $1 \leq i \leq n$, we have that P must belong to $G'''[V(G-S) \cup V(G_x - S_x) \cup \{y\}]$.

Hence, clearly, P must contain x, since both $G-S$ and G_x-S_x are P_k-free, and y is adjacent to each vertex of $G-S$. But then either P, or $P-v$ for some $v \in V(G) \cup \{y\}$, completely belongs to $G_x - S_x$. In the former case, we get a contradiction since $S_x \in \mathcal{P}_k^k(G_x)$. In the latter cases, x is an end-vertex of $P-v$, and hence, by Proposition 6.7, we have that $P-v$ has length at most $k-3 < k-2$, again, a contradiction. This proves that $S' \in \mathcal{P}_k^k(G''')$.

Now, let S' be a P_k-free P_k-transversal of G'''. Without loss of generality, we may assume that $z \in S'$, since otherwise we can take $V(G''') \setminus S'$ for S' instead. By Proposition 6.8, we have that $G_z[S']$ contains an induced path P_z of length $k-2$ which ends in z. Hence, $x, y \notin S'$, since otherwise $P_z\, x$ or $P_z\, y$ is an induced path of length $k-1$ in $G'''[S']$. Therefore, again by Proposition 6.8, $G_x - S'$ contains an induced path P_x of length $k-3$ which ends in x. This implies that, for each $1 \leq i \leq n$, we have $v_i' \in S'$, since otherwise $P_x\, y\, v_i'$ is an induced path of length $k-1$ in $G''' - S'$. Hence, by Proposition 6.8, for each $1 \leq i \leq n$, we have an induced path P_i' in $G_i[S']$ of length $k-3$ which ends in v_i'.

Now, let $S = S' \cap V(G)$. We show that S is a stable P_k-transversal of G. Clearly, $G-S$ is P_k-free, since $G''' - S'$ is P_k-free. On the other hand, S must be an independent set, since if S contains adjacent vertices v_i, v_j, then $P_i' v_i v_j (P_j')^{-1}$ is an induced path in $G'''[S']$ of length $2(k-3)+3 \geq k$ (since $k \geq 3$), which contradicts $S' \in \mathcal{P}_k^k(G''')$. Hence, S is a stable P_k-transversal of G, which concludes the proof. □

Theorem 6.21. *Let $2 \leq j \leq k$ and $4 \leq k$. Then it is NP-complete to decide, for a given chordal graph G, whether G has a P_j-free P_k-transversal.*

Proof. The problem is clearly in NP. We prove that it is also NP-hard. First, we prove the claim for $j = 2$ and all $k \geq 4$ using induction on k. For $k = 4$ and $k = 5$, the claim is proved in Theorems 6.6 and 6.16, respectively. Hence, let $k \geq 6$, and assume that the claim holds for all $4 \leq k' < k$. We show the claim for k by reduction from the stable P_{k-2}-transversal problem in chordal graphs. Let G be an instance (a chordal graph) to the stable P_{k-2}-transversal problem in chordal graphs. Let G' be the graph constructed from G as described below Figure 6.3. By Observation 6.17, G' is chordal, and, by Proposition 6.18, G has a stable P_{k-2}-transversal, if and only if, G' has a stable P_k-transversal. By the inductive hypothesis, the stable P_{k-2}-transversal problem in chordal graphs is NP-hard. Hence, the stable P_k-transversal problem in chordal graphs is also NP-hard.

Now, for $j \geq 3$ and $k \geq 4$, we prove the claim by reduction from the stable P_k-transversal problem, which, by the previous paragraph, is NP-hard. Let G be an instance (a chordal

graph) to the stable P_k-transversal problem in chordal graphs. Let G'' and G''' be the graphs constructed from G as described below Figure 6.3. By Observation 6.17, both G'' and G''' are chordal. Now, if $j < k$, then, by Proposition 6.19, we have that G has a stable P_k-transversal, if and only if, G'' has a P_j-free P_k-transversal. Similarly, if $j = k$, we have, by Proposition 6.20, that G has a stable P_k-transversal, if and only if, G''' has a P_k-free P_k-transversal. In both cases, since the stable P_k-transversal problem in chordal graphs is NP-hard, it follows that also the P_j-free P_k-transversal problem is NP-hard. That concludes the proof. □

6.5 K_j-free P_k-transversals

Finally, we finish the proof of Theorem 6.1 by reducing the problem of stable P_k-transversal in chordal graphs to the problem of K_j-free P_k-transversal in chordal graphs for each $j \geq 2$.

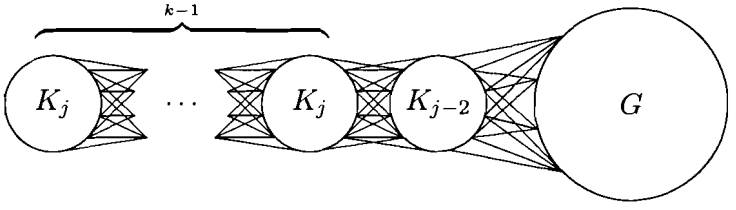

Figure 6.8: The graph G^* for the K_j-free P_k-transversal problem.

First, note that, since $K_2 = P_2$, the case $j = 2$ is already proved in the previous chapter. Hence, let $j \geq 3$, and let G be graph. The graph G^* is the graph constructed from G by adding $k-1$ disjoint copies $G_1, \ldots, G_{k-1}$ of the complete graph K_j, and adding a copy G_k of K_{j-2} such that the vertices of G_k are completely adjacent to the vertices of G, and for each $1 \leq i \leq k-1$, the vertices of G_i are completely adjacent to the vertices of G_{i+1}. The construction is illustrated in Figure 6.8.

Proposition 6.22. *For all $j \geq 3$, all $k \geq 3$, the graph G has a stable P_k-transversal, if and only if, the graph G^* has a K_j-free P_k-transversal.*

Proof. Let $G_1, \ldots, G_{k-1}$ be the copies of K_j, and G_k be the copy of K_{j-2} in G', where, for each $1 \leq i \leq k-1$, the clique G_i is completely adjacent to G_{i+1}, and G_k is completely adjacent to G. First, we show that $G^*[V(G_1) \cup \ldots \cup V(G_{k-1})]$ is P_k-free. Suppose otherwise, and let P be an induced path on k vertices in $G^*[V(G_1) \cup \ldots \cup V(G_{k-1})]$. By the pigeonhole principle, there must exist u, v on P such that $u, v \in V(G_i)$ for some $1 \leq i \leq k-1$. Hence, u and

v are twins in G', and hence, they are also twins in P, since P is induced in G'. However, P is a chordless path on $k \geq 3$ vertices, and hence, contains no twins, a contradiction.

Now, let S be a stable P_k-transversal of G. Let $S^* = S \cup V(G_k)$. We show that S^* is a K_j-free P_k-transversal of G^*. First, we observe that $G^* - S^*$ is the disjoint union of $G - S$ and $G^*[V(G_1) \cup \ldots \cup V(G_{k-1})]$. Since $G - S$ is P_k-free, and also, by the previous paragraph, $G^*[V(G_1) \cup \ldots \cup V(G_{k-1})]$ is P_k-free, we have that $G^* - S^*$ is P_k-free. Now, suppose that $G^*[S^*]$ contains a clique K of size j. Since S is an independent set, K contains at most one vertex of S. Hence, either K or $K - v$ for some $v \in S$, is completely contained in G_k. But G_k has only $j - 2$ vertices, a contradiction. This proves that S^* is a K_j-free P_k-transversal of G^*.

Now, let S^* be a K_j-free P_k-transversal of G^*. We observe that for each $1 \leq i \leq k-1$, there exists a vertex $u_i \in G_i$ such that $u_i \notin S^*$, since otherwise G_i is a clique of size j in $G^*[S^*]$, which contradicts that S^* is K_j-free. Also, we must have $v \in S^*$ for each $v \in V(G_k)$, since otherwise $u_1, \ldots, u_{k-1}, v$ is an induced path of length $k-1$ in $G^* - S^*$, but $G^* - S^*$ is P_k-free. Now, let $S = S^* \cap V(G)$. Clearly, $G - S$ is P_k-free, since $G^* - S^*$ is P_k-free. On the other hand, $G[S]$ must be an independent set, since if u, v are two adjacent vertices in $G[S]$, then $V(G_k) \cup \{u, v\}$ induces a clique of size j in $G^*[S^*]$. Hence, we have that S is a stable P_k-transversal of G, which concludes the proof. □

Theorem 6.23. *Let $2 \leq j \leq k$ and $4 \leq k$. Then it is NP-complete to decide, for a given chordal graph G, whether G has a K_j-free P_k-transversal.*

Proof. The problem is clearly in NP. We prove that it is also NP-hard. If $j = 2$, then the K_j-free P_k-transversal problem is the stable P_k-transversal problem, and the claim follows from Theorem 6.21. For $j \geq 3$, we prove the claim by reduction from the stable P_k-transversal problem in chordal graphs. Let G be an instance to this problem (a chordal graph). Let G^* be the graph constructed from G as described below Figure 6.8. It is easy to observe that, since G is chordal, also G^* is chordal. Also, by Proposition 6.22, G has a stable P_k-transversal, if and only if, G^* has a K_j-free P_k-transversal. By Theorem 6.21, the stable P_k-transversal problem in chordal graphs is NP-hard. Hence, the K_j-free P_k-transversal problem in chordal graphs is also NP-hard, and that concludes the proof. □

Chapter 7

Other cases

In this chapter, we further investigate the GCOL problem in the class of chordal graphs in a number of special cases for which we establish their complexity. Then, in the second part, we extend the results from the previous chapter to strongly chordal graphs, and discuss the complexity of the P_4-free P_4-transversal problem in chordal comparability graphs.

The following is a series of results which establish complexity of a number of cases of the GCOL problem in the class of chordal graphs.

$$M_1 = \begin{pmatrix} 0 & * & 1 & * \\ * & 0 & * & 1 \\ 1 & * & 1 & * \\ * & 1 & * & 1 \end{pmatrix} \qquad M_2 = \begin{pmatrix} \overbrace{0 & \cdots & *}^{k-1} & * & * \\ \cdot & \ddots & \cdot & * & * \\ * & \cdots & 0 & * & * \\ * & * & * & 0 & 1 \\ * & * & * & 1 & 1 \end{pmatrix} \qquad M_3 = \left(\begin{array}{ccc|cccc} & & & * & * & * & * \\ & M & & * & * & * & * \\ & & & * & * & * & * \\ \hline * & * & * & 1 & 0 & \cdots & 0 \\ * & * & * & 0 & 1 & \cdots & 0 \\ * & * & * & \cdot & \cdot & \ddots & \cdot \\ * & * & * & 0 & 0 & \cdots & 1 \end{array}\right)$$

Figure 7.1: Matrices for selected GCOL problems in chordal graphs.

Theorem 7.1. *The $\overline{P}_3$-free $\overline{P}_3$-transversal problem is solvable in the class of chordal graphs in time $O(n^{10})$.*

Proof. Recall that a chordal graph is $\overline{P}_3$-free, if and only if, it is a join of a clique and an independent set. Let M_1 be the matrix in Figure 7.1. It clearly follows that a chordal graph G has a $\overline{P}_3$-free $\overline{P}_3$-transversal, if and only if, G admits an M_1-partition. For the matrix M_1, it was shown in [9] that the list M_1-partition problem admits a polynomial time solution already in general graph. We now explain how this algorithm works.

Recall that an $(k+\ell) \times (k+\ell)$ matrix M is an (A,B,C)-block matrix, if M is of the form $M = \left(\begin{array}{c|c} A & C^T \\ \hline C & B \end{array}\right)$, where A is a symmetric $k \times k$ matrix with all zeroes on the main diagonal, B is a symmetric $\ell \times \ell$ matrix with all ones on the diagonal, and C is a $k \times \ell$ matrix. For such matrices A and B, if there is a graph H which is both A-partitionable and B-partitionable, then H has at most $k \times \ell$ vertices. This is due to the fact that, if H is A-partitionable, we have $\chi(H) \leq k$, and if H is B-partitionable, we have $\alpha(H) \leq \ell$. Hence, by Theorem 2.1, it follows that we can enumerate in time $O(n^{2k\ell+2}T(n))$ all $n^{2k\ell}$ partitions of an n-vertex graph G into an A-partitionable graph and a B-partitionable graph, where $T(n)$ is the complexity of the A-partition respectively B-partition problem. Clearly, G must admit such a partition, if G is M-partitionable.

Hence, we apply the above to M_1, and obtain, in time $O(n^{10}(n+m))$, all $O(n^8)$ partitions of G into an A_1-partitionable graph $G[X]$ and a B_1-partitionable graph $G[Y]$, where the matrices A_1 and B_1 correspond to bipartite and co-bipartite graphs, respectively. In fact, we can improve this complexity, if we assume that G is chordal. As follows from the proof of Theorem 2.1, it takes only $O(n^{2k\ell}T(n))$ time to enumerate all sparse-dense partitions, if we know at least one such partition. If G is chordal, a partition of G into a bipartite and a co-bipartite graph can be obtained easily by trying all pairs of maximal cliques C, C' of G (possibly also $C' = C$, or $C = C' = \emptyset$), and testing whether $G - (C \cup C')$ is bipartite. The complexity of this is clearly $O(n^2(n+m))$, since there are at most n maximal cliques in G, if G is chordal. Hence, the above partitions of a chordal G can be found in time $O(n^{10})$.

For each such partition $X \cup Y$, we find sets $V_1 \cup V_2 = X$ and $V_3 \cup V_4 = Y$ such that $V_1 \cup V_2 \cup V_3 \cup V_4$ is an M_1-partition of G. To find these sets, we construct an instance of $2SAT$ which will have solution, if and only if, such partition for $X \cup Y$ exists.

We assign a variable x_v to each vertex $v \in X$, and a variable y_v to each vertex $v \in Y$. For each $u, v \in X$ with $uv \in E(G)$, we add clauses $x_u \vee x_v$ and $\neg x_u \vee \neg x_v$. Similarly, for each $u, v \in Y$ with $uv \notin E(G)$, we add clauses $y_u \vee y_v$ and $\neg y_u \vee \neg y_v$. Finally, for each $u \in X$ and $v \in Y$ with $uv \notin E(G)$, we add clauses $x_u \vee y_v$ and $\neg x_u \vee \neg y_v$.

We show that the above clauses are satisfiable, if and only if, there exists an M_1-partition $V_1 \cup V_2 \cup V_3 \cup V_4$ of G with $X = V_1 \cup V_2$ and $Y = V_3 \cup V_4$. Suppose that such partition exists; set $x_v = true$, if $v \in V_1$, and $x_v = false$, if $v \in V_2$, and set $y_v = true$, if $v \in V_3$, and $y_v = false$, if $v \in V_4$. Consider clauses $x_u \vee x_v$ and $\neg x_u \vee \neg x_v$ for $u, v \in X$ with $uv \in E(G)$. Since both V_1 and V_2 are independent sets, it follows that $u \in V_1$ and $v \in V_2$, or $u \in V_2$ and

$v \in V_1$. Hence, either $x_u = true$ and $x_v = false$, or $x_u = false$ and $x_v = true$. In either case, the two clauses are satisfied. By symmetry, also the clauses $y_u \vee y_v$ and $\neg y_u \vee \neg y_v$ are satisfied for each $u, v \in Y$ with $uv \notin E(G)$. Finally, consider $u \in X$ and $y \in Y$ with $uv \notin E(G)$. Since we have all edges between V_1 and V_3, and also all edges between V_2 and V_4, it follows that either $u \in V_2$ and $y \in V_3$, or $u \in V_1$ and $y \in V_4$. Hence, either $x_u = false$ and $y_v = true$, or $x_u = true$ and $y_v = false$. Again, in either case the clauses $x_u \vee y_v$ and $\neg x_u \vee \neg y_v$ are satisfied. This shows that all clauses are satisfied.

Now, suppose that we have a truth assignment which satisfies the above clauses. We put $v \in X$ into V_1, if $x_v = true$, and into V_2 if $x_v = false$. Also, we put $v \in Y$ into V_3, if $y_v = true$, and into V_4, if $y_v = false$. Clearly, $V_1 \cup V_2 = X$, $V_3 \cup V_4 = Y$, and $V_1 \cup V_2 \cup V_3 \cup V_4$ is a partition of $V(G)$. We show that it is, in fact, an M_1-partition. First, if V_1 contains adjacent vertices $u, v \in X$, then we must have $x_u = x_v = true$, but then the clause $\neg x_u \vee \neg x_v$ is not satisfied. Similarly, if V_2 contains adjacent vertices $u, v \in X$, then the clause $x_u \vee x_v$ is not satisfied. This proves that both V_1 and V_2 are independent sets, and, by symmetry, we also have that V_3 and V_4 are cliques. Now, consider $u \in V_1$ and $v \in V_3$ such that $uv \notin E(G)$. Hence, $x_u = true$ and $y_v = true$, but then the clause $\neg x_v \vee \neg y_v$ is not satisfied. Similarly, if $u \in V_2$ and $v \in V_4$, the clause $x_v \vee y_v$ is not satisfied. Therefore, this proves that $V_1 \cup V_2 \cup V_3 \cup V_4$ is indeed an M_1-partition of G.

We now analyze the complexity. It is known that $2SAT$ for m clauses C with n variables $v_1, \ldots, v_n$ can be solved by constructing a digraph D whose vertices are the variables and their negations, and which has arcs $(\neg x, y)$ and $(\neg y, x)$ for each clause $x \vee y$ in C. (Note that $x \in \{v_i, \neg v_i\}$ and $y \in \{v_j, \neg v_j\}$ for some $i, j \in \{1 \ldots n\}$.) Now, the clauses C are satisfiable, if and only if, there is no variable $i \in \{1 \ldots n\}$ such that there is a directed path from v_i to $\neg v_i$, and a directed path from $\neg v_i$ to v_i in D. In other words, clauses C are satisfiable, if and only if, for each $i \in \{1 \ldots n\}$, the vertices v_i and $\neg v_i$ belong to different strong components of D. The complexity now follows from the result of Tarjan [65], who showed that computing strong components of any graph with n vertices and m edges can be done in time $O(n + m)$. Hence, also, solving $2SAT$ for m clauses on n vertices can be done in time $O(n + m)$. (Observe that D has $2n$ vertices and $2m$ edges.)

In our instance of $2SAT$, we have a variable for each vertex of G, and at most one clause for each pair of variables, that is, n variables, and $O(n^2)$ clauses.

Now, since we have $O(n^8)$ choices for sets X and Y, this gives total running time $O(n^{10})$. (Recall that enumerating sets X, Y also takes $O(n^{10})$ since G is chordal.) □

Theorem 7.2. *The $\overline{P}_3$-free K_k-transversal problem is solvable in the class of chordal graphs in time $O\big(n^{2k}(nk^3(4k)^k + m)\big)$.*

Proof. Again, recall that a chordal graph G is $\overline{P}_3$-free, if and only if, G is a join of a clique and an independent set. Also, a chordal graph G is K_k-free, if and only if, $\chi(G) \leq k-1$. Now, let M_2 be the $(k+1) \times (k+1)$ matrix in Figure 7.1. It follows that a chordal graph G has a $\overline{P}_3$-free K_k-transversal, if and only if, G is M_2-partitionable.

As in the above proof, we have that M_2 is (A_2, B_2, C_2)-block matrix, where A_2 corresponds to all k-colourable graphs, and B_2 corresponds to all cliques. Recall that a matrix M is crossed, if each non-$*$ element of M belongs to a row or a column of non-$*$ elements. It can be clearly seen that C_2 is crossed. Hence, by Theorem 2.3, there exists a polynomial time algorithm for the list M_2-partition problem. We now explain details of this algorithm.

Similarly, as in the above proof, we have that using the sparse-dense algorithm of Theorem 2.1, we can enumerate in time $O(n^{2k+2}(n+m))$ all $O(n^{2k})$ partitions of G into an A_2-partitionable graph $G[X]$ and a B_2-partitionable graph $G[Y]$. In fact, since G is chordal, we can find one such partition in time $O(n(n+m))$ by testing each maximal clique C of G whether $\chi(G-C) \leq k$. Hence, we only need $O(n^{2k}(n+m))$ time to enumerate all such partitions.

For each such partition $X \cup Y$, we assign to each vertex v of X a list $\ell(v) = \{1 \ldots k\}$, and remove k from the list $\ell(v)$ of each vertex $v \in X$ which has a non-neighbour in Y. Then we use the algorithm from Theorem 2.2 to decide whether $G[X]$ has an A_2-partition which respects the lists ℓ. This has complexity $O(nk^3(4k)^k)$. If such partition $V_1 \cup \ldots \cup V_k$ exists, we have that $V_1 \cup \ldots \cup V_k \cup Y$ is a M_2-partition of G, since all vertices of V_k are necessarily adjacent to all vertices of Y. (Each vertex $v \in V_k$ has $k \in \ell(v)$, and hence, by the above, v must be adjacent to each vertex of Y.) Conversely, if G admits an M_2-partition $V_1 \cup \ldots \cup V_k \cup V_{k+1}$ such that $Y = V_{k+1}$, then all vertices of V_k are completely adjacent to Y, and hence, the above procedure will not remove k from their lists. It follows that $V_1 \cup \ldots \cup V_k$ is an A_2-partition of $G[X]$ which respects the lists ℓ, and hence, the algorithm from Theorem 2.2 must successfully produce an A_2-partition of $G[X]$.

Altogether, the total complexity of the above algorithm is $O(n^{2k}(nk^3(4k)^k + m))$, and that concludes the proof. □

In fact, using the above proof, one can show the following more general result.

Theorem 7.3. *Let M be a $k \times k$ matrix with entries $\{0, 1, *\}$ and all 0 on the diagonal. Then the problem of deciding, whether a chordal graph admits a partition into an M-partitionable graph and a $\overline{P}_3$-free graph, is solvable in time $O\big(n^{2k+2}(nk^5(4k)^k + m)\big)$.* □

Theorem 7.4. *The K_j-free K_k-transversal problem is solvable in the class of chordal graphs in time $O(n + m)$.*

Proof. First, we show that a chordal graph G has a K_j-free K_k-transversal, if and only if, $\chi(G) \leq k + j - 2$. First, suppose that G has a K_j-free K_k-transversal S. Since $G[S]$ is K_j-free and $G - S$ is K_k-free, we have $\omega(G[S]) \leq j - 1$ and $\omega(G - S) \leq k - 1$. Hence, also $\chi(G[S]) \leq j - 1$ and $\chi(G - S) \leq k - 1$, since G is chordal (and hence perfect). Therefore, $\chi(G) \leq k + j - 2$. On the other hand, if $\chi(G) \leq k + j - 2$, let $S_1 \cup \ldots \cup S_{k+j-2}$ be a partition of the vertex set of G into $k + j - 2$ independent sets (in other words, a proper colouring of G). Let $S = S_1 \cup \ldots \cup S_{j-1}$. Clearly, $\chi(G[S]) \leq j - 1$ and $\chi(G - S) \leq k - 1$. Hence, $\omega(G[S]) \leq j - 1$ and $\omega(G - S) \leq k - 1$, which shows that S is a K_j-free K_k-transversal of G.

Now, it follows that testing whether a chordal graph G has a K_j-free K_k-transversal amounts to computing $\chi(G)$ and testing whether $\chi(G) \leq k + j - 2$, which can be easily done in $O(n + m)$ time. □

Theorem 7.5. *Let M be a $k \times k$ matrix with entries $\{0, 1, *\}$ and all 1 on the diagonal, and let $\mathcal{P}$ be an induced hereditary class of graphs recognizable in time $T(n, m)$. Then the problem of deciding, whether a chordal graph admits a partition into an M-partitionable graph and a graph from $\mathcal{P}$, is solvable in time $O\big(n^{2k}(T(n, m) + nk^2)\big)$, or in $O\big(T(n, m) \cdot n\big)$ if $k = 1$.*

Proof. The algorithm for this problem is a modification of the algorithm from Theorem 2.2 for matrix partitions of chordal graphs for 1-diagonal matrices. It works as follow. First, it finds a perfect elimination ordering π of the input graph G, and assigns to each vertex $v \in V(G)$ a list $\ell(v) = \{1 \ldots k\}$. Next, for each $1 \leq i \leq k$, it either decides that the set U_i is empty, or chooses two vertices x_i, y_i of G, and sets $\ell(x_i) = \ell(y_i) = \{i\}$. Then it removes i from the list $\ell(v)$ of each vertex v which appears in π before x_i or after y_i, and, for each $z \in V(G)$ and each $j \in \{1 \ldots k\}$, it removes j from $\ell(z)$, if zx_i or zy_i is an edge and $M_{i,j} = 0$, or if zx_i or zy_i is not an edge and $M_{i,j} = 1$.

After that, the algorithm finds the set X of all vertices $v \in V(G)$ with $\ell(v) \neq \emptyset$, and tests whether $G - X$ belongs to $\mathcal{P}$. If this succeeds, the algorithm returns $X \cup V(G - X)$ as a partition of G into an M-partitionable graph and a graph from $\mathcal{P}$. If this does not

succeed for all of the above choices (of vertices x_i, y_i), the algorithm announces that no such partition of G exists.

We now argue correctness of this algorithm. Suppose that the algorithm returns a partition $X \cup V(G - X)$. Clearly, $G - X$ is in $\mathcal{P}$. Recall that X consists of all vertices v with $\ell(v) \neq \emptyset$. For each $v \in X$, we pick $i \in \ell(v)$, and put $v \in U_i$. We show that $U_1, \ldots, U_k$ is an M-partition of $G[X]$. Suppose otherwise, and let $u \in U_i$ and $v \in U_j$ be distinct vertices such that either $uv \in E(G)$, and $M_{i,j} = 0$, or $uv \notin E(G)$, but $M_{i,j} = 1$. Assume the former, and without loss of generality, let $\pi(u) < \pi(v)$. Then, clearly, $v \neq y_i$ since $j \notin \ell(y_i)$, and $y_i v \notin E(G)$, because otherwise the algorithm would have removed j from $\ell(v)$. Hence, $u \neq y_i$, and $\pi(u) < \pi(y_i)$. Now, since $M_{i,i} = 1$, we must have $y_i u \in E(G)$, but then, also $y_i v \in E(G)$, since $\pi(u) < \pi(y_i)$, and $\pi(u) < \pi(v)$, a contradiction. (Note that π is a perfect elimination ordering.) Similarly, in the latter case, we obtain $\pi(x_j) < \pi(u) < \pi(v)$, and $x_j v \in E(G)$, $x_j u \in E(G)$, which gives $uv \in E(G)$, and hence, a contradiction.

On the other hand, suppose that G admits a partition into an M-partitionable graph $G[X^*]$ and a graph $G - X^* \in \mathcal{P}$, and let π be the perfect elimination ordering of G used by the algorithm. Let $U_1^*, \ldots, U_k^*$ be an M-partition of $G[X^*]$. For each $1 \leq i \leq k$ with $U_i^* \neq \emptyset$, let x_i^* to be the first and y_i^* to be the last vertex of U_i^* in the ordering π. Now, consider the step of the algorithm in which it considers U_i empty for those i such that $U_i^* = \emptyset$, and considers $x_i = x_i^*$ and $y_i = y_i^*$ for all other i. We show that the algorithm will successfully return a partition of G. Let ℓ be the lists computed by the algorithm. First, we observe that for each $1 \leq i \leq k$, and each $v \in U_i^*$, since $U_1^*, \ldots, U_k^*$ is an M-partition of $G[X^*]$, we have that v is adjacent to x_j, y_j for those j such that $M_{i,j} = 1$, and v is not adjacent to x_j, y_j for those j such that $M_{i,j} = 0$. (Note that $x_j, y_j \in U_j^*$.) Hence, the algorithm will not remove i from $\ell(v)$, which implies that $\ell(v) \neq \emptyset$.

It now follows that $X^* \subseteq X$, where X is the set of vertices v with $\ell(v) \neq \emptyset$, which is constructed by the algorithm. Hence, $G - X$ is an induced subgraph of $G - X^*$, and, since $\mathcal{P}$ is an induced hereditary class, we have $G - X \in \mathcal{P}$. This proves that the algorithm will successfully return a valid partition of G.

Finally, we argue the complexity. Clearly, for each $1 \leq i \leq k$, we have $1 + n + \binom{n}{2} \leq n^2$ choices for the vertices x_i, y_i (including the choice when $U_i = \emptyset$). Hence, altogether, there are n^{2k} choices. For each such choice, we compute the lists ℓ in time $O(k^2 n)$ by testing, for each vertex v, the edges and non-edges between v and vertices $x_1, y_1, \ldots, x_k, y_k$. Then we find the set X in time $O(kn)$ by examining the lists ℓ, and test whether $G - X \in \mathcal{P}$ in time $T(n, m)$.

It clearly follows that the running time of the above algorithm is $O\big(n^{2k}(T(n,m)+nk^2)\big)$. On the other hand, if $k=1$, we necessarily have $M=(1)$, and instead, we find all maximal cliques of G in time $O(n+m)$, and, for each such clique C, we test whether $G-C\in\mathcal{P}$. The correctness of this procedure follows easily, and the complexity is clearly $O(T(n,m)\cdot n)$, since there are at most n maximal cliques in G. That concludes the proof. □

As a corollary, we obtain the following theorem.

Theorem 7.6. *The P_3-free (the $\overline{P}_3$-free) $\overline{K}_k$-transversal problem is solvable in the class of chordal graphs in time $O(n^{2k-2}(nk^2+m))$, or $O(n(n+m))$ if $k=2$.*

Proof. Let G be a chordal graph, and let X be the set of all dominating vertices of G. We show that G is $\overline{P}_3$-free, if and only if, $G-X$ is an independent set. Clearly, if $G-X$ is an independent set, then G is a join of a clique and an independent set, and hence, G is $\overline{P}_3$-free. (Observe that the vertices of X induce a clique in G.) On the other hand, suppose that G is $\overline{P}_3$-free, but $G-X$ contains adjacent vertices u,v. Then, since both u and v are not dominating G, there exist vertices u' and v' such that $uu'\notin E(G)$ and $vv'\notin E(G)$. If also $uv'\notin E(G)$, then u,v,v' is an induced $\overline{P}_3$ in G. Similarly, if $u'v\notin E(G)$, then u,v,u' is an induced $\overline{P}_3$ in G. Hence, we must have $uv'\in E(G)$, and $u'v\in E(G)$. Now, if $u'v'\notin E(G)$, then u',v,v' is an induced $\overline{P}_3$. Hence, $u'v'\in E(G)$, which implies that u,v,u',v' is an induced C_4 in G, but G is chordal, a contradiction.

Hence, it clearly follows that the above implies an $O(n+m)$ time algorithm for recognizing chordal $\overline{P}_3$-free graphs. Also, recognizing P_3-free graphs G can be done in time $O(n+m)$ by computing the connected components of G.

The claim now follows immediately from Theorem 7.5. □

Theorem 7.7. *Let M be a $k\times k$ matrix with entries $\{0,1,*\}$ and all 0 on the main diagonal. Then the problem of deciding, whether a chordal graph admits a partition into an M-partitionable graph and a P_3-free graph, is solvable in time $O(n^k(nk^3 4^k))$.*

Proof. For an input chordal graph G on n vertices, let M_3 be the $(n+k)\times(n+k)$ matrix depicted in Figure 7.1, which is constructed from M and $n\times n$ matrix whose all diagonal entries are 1, and all off-diagonal entries are 0. Recall a graph is P_3-free, if and only if, it is a disjoint union of cliques. Hence, it follows that G admits a partition into an M-partitionable

graph and a P_3-free graph, if and only if, G is M_3-partitionable. (Note that for each size of G, we have a different matrix.)

Now, recall the algorithm from Theorem 2.2 for the list M-partition problem on graphs of treewidth at most $k-1$, which we describe in detail in Section 2.2. There, we remark that this algorithm works, in fact, for any matrix M, not only those M with all diagonal entries 0. Hence, we can use this algorithm to decide whether or not G admits an M_3-partition. All we need to argue is the complexity.

Unfortunately, the treewidth of G can be as large as n, and the matrix M_3 is of size $n+k$, and hence, the analysis from Section 2.2 gives a running time of $O(n^4(n+k)^n\, 4^{n+k})$. That, clearly, is not polynomial in n, and it is due to the fact that the analysis assumes that the number of possible pairs (Ξ, S) is $2^{n+k}(n+k)^n$. However, this is not the case for M_3.

In what follows, we show that the number of pairs (Ξ, S), which are explored by the algorithm, is polynomial in n. First, since G is chordal, for any M_3-partition $\Sigma = V_1 \cup \ldots \cup V_{n+k}$ of G, and any $v \in V(T)$, there is at most k vertices in $X(v)$ which belong to $V_1 \cup \ldots \cup V_k$ (the parts corresponding to the matrix M), since $V_1, \ldots, V_k$ are independent sets. This gives n^k possibilities for such vertices. Moreover, $X(v)$ can contain vertices of at most one set V_j, where $j \in \{k+1 \ldots n+k\}$, since for each $i, i' \in \{k+1 \ldots n+k\}$, we have no edges between V_i and $V_{i'}$, and $X(v)$ is a clique. This gives n choices for j, but, in fact, it suffices to investigate only one such choice, because all other choices are obtained by renaming the sets V_j.

In addition, when processing a forget node v, every time we are adding a pair (Ξ, S) into $F(v)$ (recall that $\Xi = V_1 \cup \ldots \cup V_{n+k}$ is an M_3-partition of $X(v)$), we remove from S any $j \in \{k+1 \ldots n+k\}$ such that $V_j = \emptyset$. Clearly, no vertex of G considered later in the algorithm will be adjacent to any vertex of $G_v - X(v)$, and hence, for such j, there will be nothing to check, so we can safely remove it from S. This gives that any such S will contain at most one $j \in \{k+1 \ldots n+k\}$, and again, by symmetry, it suffices to investigate only two choices (either there is such j in S or there is no such j in S). Hence, 2^{k+1} possibilities for S.

Altogether, we have that there are only $2(2n)^k$ possible pairs (Ξ, S) for any $F(v)$. Therefore the total running time of this algorithm is $O(n^2k^2(4n)^k)$, and, in fact, a more careful analysis gives $O(nk^3(4n)^k)$, which concludes the proof. □

As a corollary, we obtain the following theorem.

Theorem 7.8. *For each k, the P_3-free K_k-transversal problem is solvable in the class of chordal graphs in time $O(n^k k^3 4^k)$.* □

Finally, we prove the following theorem whose special case has been shown in [3]. Recall that the symbol $\uplus$ denotes the operation of disjoint union of graphs.

Theorem 7.9. *Let $\mathcal{P}$ be an additive induced hereditary class of (chordal) graphs, and H be any graph. Then the problem of deciding, whether a (chordal) graph admits a partition into an H-free graph and a graph from $\mathcal{P}$, can be polynomially reduced to the problem of deciding, whether a (chordal) graph admits a partition into a $(H \uplus K_1)$-free graph and a graph from $\mathcal{P}$.*

Proof. Consider an instance to the former problem, namely, a graph G. Let F be any minimal forbidden subgraph for $\mathcal{P}$, and let G' be the disjoint union of G and F. Observe that if G is chordal and $\mathcal{P}$ is a class of chordal graphs, then also G' is also chordal. Now, the claim will follow once we show that G admits a partition into an H-free graph and a graph from $\mathcal{P}$, if and only if, G' admits a partition into an $(H \uplus K_1)$-free graph, and a graph from $\mathcal{P}$. First, let $V_1 \cup V_2$ be a partition of G such that $G[V_1]$ is H-free, and $G[V_2] \in \mathcal{P}$. Let v be any vertex of F. Clearly, $F - v$ is in $\mathcal{P}$, since F is a minimal forbidden subgraph for $\mathcal{P}$. Hence, also $G[V_2] \uplus (F - v)$ is in $\mathcal{P}$, since $\mathcal{P}$ is additive. Moreover, $G[V_1] \uplus \{v\}$ is clearly $(H \uplus K_1)$-free, since G is H-free. Therefore, $V_1' = V_1 \cup \{v\}$ and $V_2' = V_2 \cup V(F - v)$ is the required partition for G'. Now, suppose that G' admits a partition $V_1 \cup V_2$ such that $G'[V_1]$ is $(H \uplus K_1)$-free, and $G'[V_2] \in \mathcal{P}$. Clearly, the vertices of the subgraph F of G' cannot all belong to V_2, since F is a forbidden subgraph for $\mathcal{P}$. Hence, there must exist a vertex v in F with $v \in V_1$. It follows that, if $G[V_1 \cap V(G)]$ contains H as an induced subgraph, then $G'[V_1]$ is not $(H \uplus K_1)$-free, since the copy of H together with the vertex v give a copy of $H \uplus K_1$. Hence, it follows that $V_1' = V_1 \cap V(G)$ and $V_2' = V_2 \cap V(G)$ is the required partition of G, and that concludes the proof. □

As a corollary, we obtain the following theorem.

Theorem 7.10. *For each $k \geq 4$, the $\overline{P}_3$-free P_k-transversal problem is NP-complete in the class of chordal graphs.*

Proof. The proof follows easily from the previous theorem. The problem is clearly in NP. By Theorem 6.1, the P_2-free P_k-transversal problem is NP-hard in the class of chordal graphs. Hence, by Theorem 7.9, also the $\overline{P}_3$-free P_k-transversal problem in NP-hard in the class of chordal graphs, which shows the claim. □

7.1 Strongly Chordal Graphs

In this section, we extend the results of Chapter 6 to the class of strongly chordal graphs.

We say that a perfect elimination ordering $\prec$ of a graph G is a *strong elimination ordering*, if for any vertices $u \prec v \prec w \prec z$, if uz, uw and vw are edges of G, then also vz is an edge. A graph G is called *strongly chordal*, if there exists a strong elimination ordering of the vertices of G. It can be seen that the class of strongly chordal graphs is a subclass of chordal graphs, but also a superclass of the class of interval graphs and of the class of chordal comparability graphs (the graphs both chordal and comparability).

We remark that for strongly chordal graphs, a forbidden induced subgraph characterization is known due to Farber [25]. A *k-sun* is a graph formed by a cycle $v_0, v_1, \ldots, v_{k-1}$ with edges v_iv_{i+1} (and possibly other edges), and an independent set $w_0, w_1, \ldots w_{k-1}$, where w_i is adjacent only to v_i and v_{i+1} (all indices are taken modulo k).

Theorem 7.11. *[25] A graph G is strongly chordal, if and only if, for all $k \geq 3$, G does not contain a k-sun as an induced subgraph.*

Recall that a block of a graph G is a (set) maximal induced subgraph that cannot be disconnected by a removal of a single vertex, a cutvertex of G is a vertex of G whose removal disconnects G, a dominating vertex of G is a vertex of G adjacent to all other vertices of G, and a twin of a vertex of G is an adjacent vertex having the same set of neighbours. We have the following simple observations (not only) about strongly chordal graphs.

Proposition 7.12. *Let $\mathcal{C}$ be a induced hereditary class of graphs.*

(i) If all minimal forbidden induced subgraphs of $\mathcal{C}$ have no cutvertices, Then for any graph G, the graph G is in $\mathcal{C}$, if and only if, all blocks of G are in $\mathcal{C}$.

(ii) If all minimal forbidden induced subgraphs of $\mathcal{C}$ are connected, then for any graph G, the graph G is in $\mathcal{C}$, if and only if, all connected components of G are in $\mathcal{C}$.

(iii) If all minimal forbidden induced subgraphs of $\mathcal{C}$ have no dominating vertices, then for any graph G, the graph G is in $\mathcal{C}$, if and only if, a graph constructed from G by adding a dominating vertex is in $\mathcal{C}$.

(iv) If all minimal forbidden induced subgraphs of $\mathcal{C}$ have no twins, then for any graph G, the graph G is in $\mathcal{C}$, if and only if, a graph constructed from G by adding a twin of a vertex of G is in $\mathcal{C}$.

Proof. Suppose that G is in $\mathcal{C}$. Then, clearly, since $\mathcal{C}$ is induced hereditary, all blocks of G must be in $\mathcal{C}$. On the other hand, if G is not in $\mathcal{C}$, then G must contain a minimal forbidden induced subgraph F of $\mathcal{C}$. If F contains a cutvertex v of G, then v must be a cutvertex of F as well, since F is an induced subgraph of G. But, by (i), F contains no cutvertices. Hence, F must be contained in a block of G, which proves the claim.

Similarly, if G is in $\mathcal{C}$, then all connected components of G must be in $\mathcal{C}$, since $\mathcal{C}$ is induced hereditary, and if G is not in $\mathcal{C}$, then it contains a minimal forbidden induced subgraph F, which, by (ii), is connected, and hence it completely belongs to a connected component of G.

Now, suppose that the graph G' constructed from G by adding a dominating vertex v is in $\mathcal{C}$. Since G is an induced subgraph of G', also G must be in $\mathcal{C}$. On the other hand, if G' is not in $\mathcal{C}$, then G' must contain a minimal forbidden induced subgraph F of $\mathcal{C}$. Suppose that v is in F. Since v is adjacent to all vertices of G, and F is an induced subgraph of G', it follows that, in F, the vertex v is adjacent to all other vertices of F. But, by (iii), F contains no dominating vertex. Therefore, v is not in F, and hence, F is an induced subgraph of G which implies that G is not in $\mathcal{C}$.

Finally, suppose that the graph G' constructed from G by adding a twin v of a vertex u of G is in $\mathcal{C}$. Then, similarly, since G is an induced subgraph of G', G must also be in $\mathcal{C}$. On the other hand, if G' is not in $\mathcal{C}$, then G' contains a minimal forbidden induced subgraph F of $\mathcal{C}$. Suppose that F contains both u and v. Again, since F is an induced subgraph of G', u and v must also be twins in F. But, by (iv), F contains no twins. Hence, either u is not in F or v is not in F. In both cases, since $G' - u$ is clearly isomorphic to $G' - v = G$, we obtain that G also contains F as an induced subgraph, and that proves the claim. □

We observe that all minimal forbidden induced subgraphs of chordal graphs and strongly chordal graphs, by Theorem 7.11, are connected and contain no cutpoints, dominating vertices, or twins. Hence, the previous observation applies to both classes.

We now prove that all graphs used in the proofs of Theorem 6.1 are strongly chordal, which will prove the following extension of this theorem.

Theorem 7.13. *Let $2 \leq j \leq k$ and $4 \leq k$. Then it is NP-complete to decide, for a given strongly chordal graph G, whether G has a P_j-free P_k-transversal. It is also NP-complete to decide, for a given strongly chordal graph G, whether G has a K_j-free P_k-transversal.*

Proof. To prove this theorem, we show that the graphs in Figures 6.1, 6.3, 6.4, 6.5, 6.6, 6.7, and 6.8, are all strongly chordal.

We start with the graphs G_i and Y_j in Figure 6.1. We observe that the graph Y_j is clearly strongly chordal, since it does not contain any k-sun. This proves that G_0 is strongly chordal. For all other $i \geq 1$, we prove by induction on i that G_i is also strongly chordal. Hence, assume that G_{i-1} is strongly chordal, and consider the graph G_i. Let π be a strong elimination ordering of G_{i-1}. We show that $\pi' = v_i^1, v_i^2, \ldots, \overline{v}_i^1, \overline{v}_i^2, \ldots, \pi, v_i, \overline{v}_i$ is a strong elimination ordering of G_i. First, it is easy to see that π' is a perfect elimination ordering. Now, consider vertices u, v, w, z appearing in π' in this order such that uw, uz, vw are edges of G_i, but vz is not an edge. Since both v_i and $\overline{v}_i$ are adjacent to all vertices of G_{i-1}, it follows that u must be either a vertex v_i^j or a vertex $\overline{v}_i^j$ for some j. Hence, w is a vertex of G_{i-1}, and z is either v_i or $\overline{v}_i$. Now, by the construction of G_i, w is not adjacent to any other vertex $v_i^{j'}$ or $\overline{v}_i^{j'}$. Hence, v must be in G_{i-1}, so we have an edge between v and z, a contradiction. This proves that π' is a strong elimination ordering of G_i, and hence G_i is strongly chordal.

Now, consider the graphs F_i^k in Figure 6.3. The graph F_1^k is trivially strongly chordal. Also, the graph F_2^k is strongly chordal, which follows, by Proposition 7.12, from the fact that v is a dominating vertex of F_2^k, and the chordless path $u_1, \ldots, u_k$ contains no k-sun. Now, by induction, suppose that F_{i-1}^k is strongly chordal. Observe that all blocks of F_i^k are either copies of F_{i-1}^k or the graph $F_2^{k'}$, where $k' = \max\{2, k - 2i + 4\}$. Hence, by Proposition 7.12, also F_i^k is strongly chordal. Similarly, since the blocks of the graph H_j^k in Figure 6.5 are either the graphs F_j^k or paths, we have that the graph H_j^k is strongly chordal. By the same argument, the graphs B_j^k and D_j^k in Figure 6.5 are also strongly chordal.

Now, consider the graphs G_i' and Y_j' in Figure 6.6. Again, we have that the blocks of Y_j' are either copies of B_2^5 or paths, which shows that both Y_j' and G_0' are strongly chordal. Now, by induction, assume for $i \geq 1$ that G_{i-1}' is strongly chordal. Let H be the disjoint union of G_{i-1}' and the chordless path $z_i, b, c, d, \overline{z}_i$ of G_i', and H' be the graph constructed from H by adding the vertices $x_i, \overline{x}_i$ and $v_i^j, \overline{v}_i^j$ for all j. Observe that H is strongly chordal, and it is an induced subgraph of H' dominated by v_i and $\overline{v}_i$. Now, the argument from first paragraph of this proof for $G_{i-1} = H$ and $G_i = H'$, precisely gives that H' is strongly chordal. Since H' is also a block of G_i', and all other blocks of G_i' are either copies of B_2^5 or paths, it follows that G_i' is also strongly chordal.

Next, consider the graphs G', G'', and G'' in Figure 6.7. Assuming that G is strongly

chordal, we immediately have that both G' and G'' are also strongly chordal, since their blocks are the blocks of G, the copies of B_j^k or D_j^k, and paths, and, clearly, each is strongly chordal. For G''', consider the block H of G''' containing G. Observe that y is a dominating vertex of H, and hence, it suffices to show that $H - y$ is strongly chordal. But the only non-path block of $H - y$ is formed by the graph G dominated by x, and hence, since G is strongly chordal, we have that H is also strongly chordal. Now, clearly, also G''' is strongly chordal, since all other blocks of G''' are copies of F_{k-2}^k or F_{k-1}^k.

Finally, consider the graph G^* in Figure 6.8. Let H be the graph constructed from G by adding a dominating vertex v, and a chordless path $u_1, \ldots, u_k = v$ attached to v. Again, by Proposition 7.12, if G is strongly chordal, H is also strongly chordal. Now, we observe that G^* is precisely the graph that is obtained from H by adding $j - 1$ twins of each of $u_1, \ldots, u_{k-1}$, and $j - 3$ twins of v. By Proposition 7.12, this implies that G^* is strongly chordal, and that concludes the proof. □

7.2 Chordal Comparability Graphs

Recall that a graph G is comparability, if there exists an orientation F of its edges $E(G)$ such that F is transitive. The class of chordal comparability graphs is defined as the class of all graph which are both chordal and comparability.

In this section, we prove that the problem of P_4-free P_4-transversals is trivial in chordal comparability graphs, that is, we show that any chordal comparability graph has a P_4-free P_4-transversal. This is in contrast to both Theorem 6.1 and Theorem 7.13, since the class of chordal comparability graphs is a subclass of both chordal and strongly chordal graphs. We prove this theorem by proving an interesting property of perfect elimination orderings of chordal comparability graphs.

A perfect elimination ordering $\prec$ of a graph G is called a *simple elimination ordering*, if for any $u \prec v \prec w$, either the neighbours $x \succ u$ of w are also neighbours of v, or the neighbours $x \succ u$ of v are also neighbours of w. It is known that a graph G has a simple elimination ordering, if and only if, G is strongly chordal [25]. In fact, any strong elimination ordering is also a simple elimination ordering, but not conversely, although, it is known that one can in time $O(n + m)$ transform any simple elimination ordering into a strong elimination ordering [59]. Since chordal comparability graphs are strongly chordal,

we also have that there exists a simple elimination ordering for any chordal comparability graph. It turns out that there exists an $O(n+m)$ time algorithm [6] which, given a chordal comparability graph G, constructs a simple elimination ordering of G. This ordering can be then used to solve a number of combinatorial problems, such as the maximum matching problem, efficiently. In contrast, note that, at the time of writing, no linear time algorithm for constructing a simple elimination ordering of a strongly chordal graph is known.

The actual algorithm which constructs a simple elimination ordering of a chordal comparability graph is known as Cardinality Lexicographic Breadth-First Search (CLexBFS), which is a modification of the usual LexBFS (see Section 1.2). This algorithm works almost exactly as LexBFS, but instead of choosing any vertex with lexicographically largest label, it always chooses a vertex of the largest degree among the vertices with lexicographically largest label. We summarize this algorithm below as Algorithm 7.1.

Algorithm 7.1: Cardinality Lexicographic Breadth-First search.

Input: A graph G

Output: An elimination ordering π

```
1 set label(v) ← ∅ for all v ∈ V(G)
2 for i ← n downto 1 do
3     pick an unnumbered vertex v with the largest degree among
          the vertices with lexicographically largest label
4     π(i) ← v /* the vertex v becomes numbered */
5     for each unnumbered w adjacent to v do
6         append i to label(w)
```

We shall make use of the following property of CLexBFS.

Proposition 7.14. *Let $\prec$ be an elimination ordering of a graph G computed by CLexBFS. Then, for any two vertices u, v of G with $N(u) \subsetneqq N(v)$ or $N[u] \subsetneqq N[v]$, we have $u \prec v$.*

Proof. Let u and v be two vertices of G with $N(u) \subsetneqq N(v)$ or $N[u] \subsetneqq N[v]$. Observe that at any step of the CLexBFS algorithm, the label of any vertex is formed by the numbers assigned to the processed neighbours of that vertex. Since any neighbour of u is also a neighbour of v, it clearly follows that until one of the two vertices is chosen by the algorithm,

their labels are identical. Now, suppose that $v \prec u$, and consider the step of the algorithm at which u is chosen to be the next processed vertex. This is a step at which u is a vertex with the largest degree among the vertices with lexicographically largest label. But since the degree of u is smaller than the degree of v, and, by the above, the labels of u and v at that point in the algorithm must be identical, the algorithm must choose v instead as the next vertex, a contradiction. Hence, $u \prec v$ as claimed. □

Now, let F be an orientation of the edges of a graph G, and $\pi = (v_1, v_2, \ldots, v_n)$ be any permutation of the vertices of G. Let G_F be the oriented graph formed by the vertices of G and the arcs F. We shall call a vertex v_i *smooth* with respect to F and π, if v_i is neither a source nor a sink in $G_F[v_1, v_2, \ldots, v_i]$, that is, v_i has both an incoming edge in F from some v_j, $j < i$, and an outgoing edge in F to some v_k, $k < i$.

Let $\sigma_\pi(F)$ be the number of vertices in G which are smooth with respect to F and π. We say that F is a *uniform* orientation of G with respect to π, if $\sigma_\pi(F) = 0$.

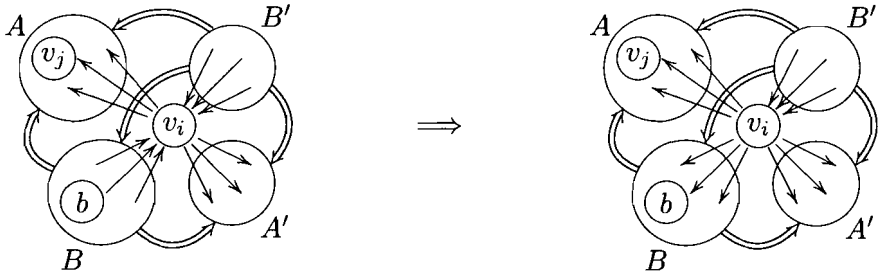

Figure 7.2: Transforming a transitive orientation into a uniform transitive orientation.

Lemma 7.15. *For any chordal comparability graph G, there exists a transitive orientation F of G which is uniform with respect to some simple elimination ordering π of the vertices of G. In addition, there exists an $O(n + m)$ time algorithm for constructing F.*

Proof. Let π be an elimination ordering of G computed by CLexBFS, and let F be a transitive orientation of G with least possible value of $\sigma_\pi(F)$. By [6], the ordering π is, in fact, a simple elimination ordering of G. We show that F must be uniform with respect to π, that is, $\sigma_\pi(F) = 0$.

If not, then let i be the largest index such that v_i is smooth with respect to F and π. Let A be the neighbours v_k of v_i with $v_iv_k \in F$ and $k < i$, and A' be the neighbours v_k of v_i with $v_iv_k \in F$ and $k > i$. Similarly, let B be the neighbours v_k of v_i with $v_kv_i \in F$ and $k < i$, and B' be the neighbours v_k of v_i with $v_kv_i \in F$ and $k > i$ (see Figure 7.2a). Since v_i is smooth, we must have $A \neq \emptyset$ and $B \neq \emptyset$. Moreover, since v_i has the largest index among the smooth vertices, we have that no vertex in $A' \cup B'$ is smooth. Also, since F is transitive, we must have $ba \in F$ for every $b \in B \cup B'$ and $a \in A \cup A'$. Now, let v_j be the vertex with the smallest index j among the neighbours of v_i. Without loss of generality, we may assume that $v_j \in A$ (otherwise, we use F^{-1} in place of F). Clearly, every neighbour v_k of v_i has $k \geq j$. Moreover, v_j is simplicial in $G[v_j, v_{j+1}, \ldots, v_n]$, since π is a perfect elimination ordering. Hence, $B \cup B'$ must be a clique, and since no vertex of B' is smooth, we must have $b'b \in F$ for all $b' \in B'$ and $b \in B$. Now, consider any vertex b in B. From the above, b is adjacent to all vertices of A, B, A', and B', and hence, we have $N[b] \supseteq N[v_i]$. However, b appears in π before v_i, and hence, by Proposition 7.14, $N[b] = N[v_i]$.

Now, let F' be the orientation of G obtained from F by reversing the direction of the arcs between v_i and the vertices of B (see Figure 7.2b). We show that F' is a transitive orientation of G. Suppose that x, y, z is a transitive violation in F', that is, $xy \in F'$, $yz \in F'$ but $xz \notin F'$. Since F is transitive, we must have that either $x = v_i$, $y \in B$ and $z \in A \cup A'$, or $x \in B'$, $y = v_i$ and $z \in B$. But in both cases, we have $xz \in F'$, a contradiction. Hence, this shows that F' must be transitive. However, v_i is no longer smooth with respect to F' and π, and, clearly, any vertex which was not smooth with respect to F and π is also not smooth with respect to F' and π. This contradicts the choice of F, and the claim follows.

Now, we explain how we can construct a uniform transitive orientation of G with respect to some simple elimination ordering π of G. First, we obtain a transitive orientation of G. This can be accomplished in time $O(n + m)$ using the algorithm of [46]. Then we run the algorithm CLexBFS on G to obtain a simple elimination ordering π. After that, we process the vertices of G in the reverse of π. If we encounter a smooth vertex v_i, we scan its neighbourhood and pick a vertex v_j with the smallest index j. Then, again by scanning the neighbourhood of v_i, we orient the edges between v_i and its neighbours v_k, where $k < i$, so that they have the same direction as the arc between v_i and v_j in F. By the above argument, this cannot create a transitive violation. After that, we process next vertex. Clearly, each vertex v_i is processed in time $O(|N(v_i)|)$. Hence, the total running time is $O(n + m)$. □

Now, as a consequence of the above, we obtain the main theorem of this section.

Theorem 7.16. *Every chordal comparability graph G has a P_4-free P_4-transversal. This transversal can be found in time $O(n+m)$.*

Proof. By Lemma 7.15, we have that there exists a simple elimination ordering $\pi = v_1, \ldots, v_n$ of G and a transitive orientation F of G, which is uniform with respect to π. We partition the vertices of G into two sets X and Y with the property that for any two adjacent vertices v_i, v_j with $i < j$, if v_i and v_j are both in X, then $v_iv_j \in F$, and if v_i and v_j are both in Y, then $v_jv_i \in F$.

We process the vertices of G in the order given by π. A vertex v_k is placed into X, if there is no $v_i \in X$ with $i < k$ which is adjacent to v_k and $v_kv_i \in F$. Similarly, v_k is placed into Y, if there is no $v_j \in Y$ with $j < k$ which is adjacent to v_k and $v_jv_k \in F$. Since F is uniform, the vertex v_k must be either a sink or a source in $G[v_1, \ldots, v_k]$, and hence it follows that one of the above steps is always possible.

Now, we show that both X and Y induce P_4-free subgraphs in G, which will give the result. Suppose that v_i, v_j, v_k, v_l with edges v_iv_j, v_jv_k and v_kv_l is a P_4 in X. Without loss of generality, suppose that $v_iv_j \in F$. Then, since F is transitive, we must have $v_kv_j \in F$, and $v_kv_l \in F$, since otherwise we would either have $v_iv_k \in F$, or $v_lv_j \in F$, and v_iv_k, v_jv_l are not edges of G. Hence, by the above property of X, we must have $i < j$, $k < j$ and $k < l$. But π is a perfect elimination ordering, which gives that v_j and v_l must be adjacent, a contradiction. A similar argument for Y shows that also Y induces P_4-free subgraph.

Finally, we discuss the complexity of the above procedure. First, using Lemma 7.15, we obtain F and π. Then we process the vertices of G in the order of π, and for each vertex v_k, we look in the neighbourhood of v_k for a vertex v_i with $i < k$. If $v_iv_k \in F$, we put v_k in X, otherwise we put v_k in Y. Since F is uniform, we can pick any such vertex v_i, and the outcome will be the same. This processing, clearly, takes only $O(|N(v_i))|$ time, and hence, $O(n+m)$ for the whole procedure. □

We close this section by mentioning the following open problems.

Problem 7.17. *Determine the complexity of the stable P_4-transversal problem in the class of chordal comparability graphs.*

Problem 7.18. *Determine the complexity of the P_3-free P_4-transversal problem in the class of chordal comparability graphs.*

Chapter 8

Injective Colourings

In this chapter, we investigate properties of injective colourings in chordal graphs.

An *injective colouring* of a graph G is a colouring c of the vertices of G that assigns different colours to any pair of vertices that have a common neighbour. (That is, for any vertex v, if we restrict c to the (open) neighbourhood of v, this mapping will be injective; whence the name.) Note that injective colouring is not necessarily a proper colouring, that is, it is possible for two adjacent vertices to receive the same colour. The injective chromatic number of G, denoted $\chi_i(G)$, is the smallest integer k such that G can be injectively coloured with k colours.

Injective colourings are closely related to (but not identical with) the notions of locally injective colourings [32] and $L(h,k)$-labellings [5, 8, 40]. In particular, $L(0,1)$-labellings, unlike injective colourings, assign distinct colours only to non-adjacent vertices with a common neighbour.

Injective colourings were introduced by Hahn, Kratochvíl, Širáň and Sotteau in [42]. They attribute the origin of the concept to complexity theory on Random Access Machines. They prove several interesting bounds on $\chi_i(G)$, and also show that, for $k \geq 3$, it is NP-complete to decide whether the injective chromatic number of a graph is at most k. Here we look at the complexity of this problem when the input graphs G are restricted to be chordal.

In the following sections, we show that determining $\chi_i(G)$ is still difficult when restricted to chordal graphs. In fact, it is not only NP-hard, but unless $NP = ZPP$, the injective chromatic number of a chordal graph cannot be efficiently approximated within a factor of $n^{1/3-\epsilon}$, for any $\epsilon > 0$. (Here ZPP is the class of languages decidable by a randomized algorithm that makes no errors and whose expected running time is polynomial.)

For split graphs, this is best possible since we show an $\sqrt[3]{n}$-approximation algorithm for the injective chromatic number of a split graph. Utilizing a result of [48], we show that this is also true for all chordal graphs.

On the positive side, we show that for any fixed k, one can in time $O(n \cdot k \cdot k^{(k/2+1)^2})$ determine whether a chordal graph can be injectively coloured using no more than k colours. Moreover, we describe large subclasses of chordal graphs that allow computing the injective chromatic number efficiently. We show that for a chordal graph G, one can efficiently compute the injective chromatic number of G from the chromatic number of the square of $G - \mathcal{B}(G)$, that is, the graph G with its bridges $\mathcal{B}(G)$ removed. It follows that for strongly chordal graphs and power chordal graphs (the graphs whose powers are all chordal) the problem is polynomial time solvable.

8.1 Basic properties

We start with the following simple observation.

Observation 8.1. *For any graph G, we have $\chi_i(G) \geq \Delta(G)$ and $\chi(G^2) \geq \Delta(G) + 1$.*

Proof. For any two neighbours u, w of v, the vertex v is their common neighbour; hence, u, w must have different colours in any injective colouring of G. Similarly, since the distance of u and w is at most two (u, v, w is a path of length two connecting them), they must have different colours in any proper colouring of G^2, and in addition the colours of u, w must be different from the colour of v, since they are adjacent to v. □

For trees this is also an upper bound.

Proposition 8.2. *For any tree T, we have $\chi_i(T) = \Delta(T)$ and $\chi(T^2) = \Delta(T) + 1$.*

Proof. Let u be a leaf in T, and let v the parent of u. Then we clearly have that $\chi(T^2) = \max\{\deg(v) + 1, \chi((T-u)^2)\}$ and $\chi_i(T) = \max\{\deg(v), \chi_i(T-u)\}$. The claim now follows by induction on the size of T. □

Now, we look at the general case. Let $G^{(2)}$ be the *common neighbour graph* of a graph G, that is, the graph on the vertices of G, in which two vertices are adjacent, if they have a common neighbour in G. It is easy to see that the injective chromatic number of G is exactly the chromatic number of $G^{(2)}$. In general, as we shall see later, properties of the graph $G^{(2)}$

can be very different from those of G. For instance, even if G is efficiently colourable, e.g. if G is perfect, it may be difficult to colour $G^{(2)}$. Note that any edge of $G^{(2)}$ must be also an edge of G^2 (but not conversely). This yields the following inequality.

Proposition 8.3. *For any graph G, we have $\chi_i(G) \leq \chi(G^2)$.* □

In fact, this inequality can be strengthened. Let $\mathcal{F}(G)$ be the set of edges of G that do not lie in any triangle. Note that an edge of G is also an edge of $G^{(2)}$ if and only if it belongs to a triangle of G. This proves the following proposition.

Proposition 8.4. *For any graph G, we have $\chi_i(G) = \chi(G^2 - \mathcal{F}(G))$.* □

Now we turn to chordal graphs. The following is a direct consequence of Proposition 3.4.

Observation 8.5. *Any edge in a bridgeless chordal graph lies in a triangle.* □

Let $\mathcal{B}(G)$ be the set of bridges of G. Since a bridge of a graph can never be in a triangle, we have the following fact.

Proposition 8.6. *For any chordal graph G, we have $\chi_i(G) = \chi(G^2 - \mathcal{B}(G))$.* □

Now, since $\mathcal{B}(G - \mathcal{B}(G)) = \emptyset$, we have the following corollary.

Corollary 8.7. *For any chordal graph G, we have $\chi_i(G - \mathcal{B}(G)) = \chi\big((G - \mathcal{B}(G))^2\big)$.* □

It turns out that there is a close connection between $\chi_i(G - \mathcal{B}(G))$ and both the injective chromatic number $\chi_i(G)$ of G and the chromatic number $\chi(G^2)$ of the square of G.

Proposition 8.8. *For any graph G, we have $\chi(G^2) = \max\big\{\Delta(G) + 1, \chi\big((G - \mathcal{B}(G))^2\big)\big\}$*

Proof. Let $k = \max\big\{\Delta(G) + 1, \chi\big((G - \mathcal{B}(G))^2\big)\big\}$. It follows from Observation 8.1 and Corollary 8.7 that $\chi(G^2) \geq k$. We fix a set of k colours ($k \geq \chi\big((G - \mathcal{B}(G))^2\big)$), and consider a colouring of $(G - \mathcal{B}(G))^2$ using these k colours. Now, using this colouring, we add the bridges of G one by one, modifying the colouring accordingly. Let uv be a bridge of G and let X and Y be the connected components which become connected by the addition of uv. Suppose that $u \in X$ and $v \in Y$. We can permute the colours of X and Y independently so that u and v obtain the same colour i. Since we have $k \geq \Delta(G) + 1$ colours, there must be a colour $j \neq i$ not used in the neighbourhood of v in Y. Using the same argument for u, we may assume that j is not used in the neighbourhood of u in X. Finally, we exchange

in X the colours i and j. It is easy to see that after adding all bridges of G one by one, we obtain a proper colouring of G^2. □

A similar argument proves the next proposition.

Proposition 8.9. *For any split graph G, we have $\chi_i(G) = \max\{\Delta(G), \chi_i(G - \mathcal{B}(G))\}$*

Proof. As in the above proof, we let $k = \max\{\Delta(G), \chi_i(G - \mathcal{B}(G))\}$ and assume that we have an injective colouring of $G - \mathcal{B}(G)$ using some fixed k colours. Now, if uv is a bridge of G with $\deg(u) \geq \deg(v)$, then either G is a tree and the claim follows from Proposition 8.2, or $\deg(v) = 1$. Hence, if the colour of u is not used in the neighbourhood of u, we can use this colour to colour v. Otherwise, since we have $k \geq \Delta(G)$ colours, there must exist a colour not used in the neighbourhood of u, and we can use it to colour v. This way we add all bridges of G one by one, and clearly obtain an injective colouring of G. □

Finally, combining Corollary 8.7, and Propositions 8.8 and 8.3, we obtain the following tight upper and lower bound on the injective chromatic number of a chordal graph.

Proposition 8.10. *For any chordal graph G, we have*

$$\chi(G^2) - 1 \leq \max\{\Delta(G), \chi_i(G - \mathcal{B}(G))\} \leq \chi_i(G) \leq \chi(G^2)$$

8.2 Hardness and approximation results

In this section, we focus on hardness results for the injective chromatic number problem. We begin by observing that it is NP-hard to compute the injective chromatic number of a split graph. This also follows from a similar proof in [42]; we include our construction here, since we shall extend it to prove an accompanying inapproximability result in Theorem 8.13.

Theorem 8.11. *It is NP-complete for a given split (and hence chordal) graph G and an integer k, to decide whether the injective chromatic number of G is at most k.*

Proof. First, we observe that the problem is clearly in NP. We show it is also NP-hard. Consider an instance of the graph colouring problem, namely a graph G and an integer l. We may assume that G is connected and contains no bridges. Let H_G be the graph constructed from G by first subdividing each edge of G and then connecting all the new vertices. That is,

$$V(H_G) = V(G) \cup \{x_{uv} \mid uv \in E(G)\}$$

$$E(H_G) = \{ux_{uv}, vx_{uv} \mid uv \in E(G)\} \cup \{x_{st}x_{uv} \mid uv, st \in E(G)\}$$

The graph H_G can clearly be constructed in polynomial time. It is not difficult to see that H_G is a split graph, hence it is also chordal. Moreover, one can check that the subgraph of H_G^2 induced on the vertices of G is precisely the graph G. Since G is bridgeless, H_G is also bridgeless, hence using Proposition 8.6 we have the following.

$$\chi_i(H_G) = \chi(H_G^2) = \chi(G) + m$$

Therefore, $\chi_i(H_G)$ is at most $k = l + m$, if and only if, $\chi(G)$ is at most l. That concludes the proof. □

By Proposition 8.10, for any chordal graph G, the injective chromatic number of G is either $\chi(G^2)$ or $\chi(G^2) - 1$. Interestingly, merely distinguishing between these two cases is already NP-complete.

Theorem 8.12. *It is NP-complete to decide, for a given split (and hence chordal) graph G, whether $\chi_i(G) = \chi(G^2) - 1$.*

Proof. To show that the problem is in NP, one has to observe that by Propositions 8.8 and 8.9, for a split graph G, $\chi_i(G) = \chi(G^2) - 1$, if and only if, $\chi_i(G - \mathcal{B}(G)) \leq \Delta(G)$. For chordal graphs, it follows similarly from Theorem 8.20. We now show that the problem is also NP-hard. It is known [47] that it is NP-complete to decide whether a planar graph has a 3-colouring. Consider an instance of this problem, a planar graph G. By the Four Colour Theorem [18], we have $\chi(G) \leq 4$. Also, we can assume that $\chi(G) \geq 3$, since whether G is bipartite can be determined in polynomial time. As in the proof of Theorem 8.11, we assume that G is bridgeless. Now, let H be the graph H_G from the proof of Theorem 8.11 augmented with a vertex z adjacent to all x_{uv} and with four vertices a, b, c, d adjacent only to z. Now, since H_G is bridgeless, the edges az, bz, cz, dz are the only bridges of H. Hence, $\chi_i(H - \mathcal{B}(H)) = \chi(G) + m + 1 \geq m + 4$. It is easy to see that $\Delta(H) = m + 4$, since any x_{uv} is adjacent to exactly two non-clique vertices. So, by Propositions 8.8 and 8.9, we have the following.

$$\chi(G) + m + 1 = \chi_i(H) \leq \chi(H^2) = \max\{\Delta(G) + 1, \chi(G) + m + 1\} = m + 5$$

Therefore, $\chi_i(H) = \chi(H^2) - 1$, if and only if, G is 3-colourable, which is NP-hard to decide. That concludes the proof. □

We remark that a 4-colouring of a planar graph G can always be efficiently constructed [55]. Hence, in the proof above, an optimal colouring of H^2 is always efficiently constructible.

Therefore, it follows that Theorem 8.12 holds even if, along with G, we are provided with an optimal colouring of G^2.

Now, we extend the proof of Theorem 8.11 to show that under a certain complexity assumption, it is not tractable to approximate the injective chromatic number of a split (chordal) graph within a factor of $n^{1/3-\epsilon}$ for all $\epsilon > 0$.

Theorem 8.13. *Unless $NP = ZPP$, for any $\epsilon > 0$, it is not possible to efficiently approximate $\chi(G^2)$ and $\chi_i(G)$ within a factor of $n^{1/3-\epsilon}$, for any split (and hence chordal) graph G.*

Proof. In [30], it was shown that for any fixed $\epsilon > 0$, unless $NP = ZPP$, the problem of deciding whether $\chi(G) \leq n^\epsilon$ or $\alpha(G) < n^\epsilon$ for a given graph G is not solvable in polynomial time. Consider an instance of this problem, namely a graph G. Again, as in the proof of Theorem 8.11, we may assume that G is connected and bridgeless. Let $H_{k,G}$ be the split graph constructed from k copies of H_G (the graph used in the proof Theorem 8.11) by identifying, for each $uv \in E(G)$, all copies of x_{uv}. That is, if $v_1, v_2, \ldots, v_n$ are the vertices of G, we have

$$V(H_{k,G}) = \bigcup_{i=1}^{k} \{v_1^i, v_2^i, \ldots, v_n^i\} \cup \{x_{uv} \mid uv \in E(G)\}$$
$$E(H_{k,G}) = \bigcup_{i=1}^{k} \{u^i x_{uv},\ v^i x_{uv} \mid uv \in E(G)\} \cup \{x_{uv}x_{st} \mid uv, st \in E(G)\}$$

Now, since G is bridgeless, $H_{k,G}$ is also bridgeless. Consider an independent set I of $H_{k,G}^2$. It is not difficult to check that either I trivially contains only a single vertex x_{uv}, or for each pair of vertices $u^i, v^j \in I$, the vertices u and v are not adjacent in G. Hence, it follows that from any colouring of $H_{k,G}^2$, one can construct a fractional k-fold colouring of G (that is, a collection of independent sets covering each vertex of G at least k times) by projecting each non-trivial colour class of $H_{k,G}^2$ to G, that is, mapping each u^i to u. Using this observation, we obtain the following inequalities.

$$\frac{k \cdot n}{\alpha(G)} + m \leq k \cdot \chi_f(G) + m \leq \chi(H_{k,G}^2) = \chi_i(H_{k,G}) \leq k \cdot \chi(G) + m$$

Therefore, if $\chi(G) \leq n^\epsilon$, then $\chi(H_{k,G}^2) \leq k \cdot n^\epsilon + m$, and if $\alpha(G) < n^\epsilon$ then $\chi(H_{k,G}^2) > k \cdot n^{1-\epsilon} + m$. Now, we fix $k = m$, and denote by N the number of vertices in $H_{m,G}$. For $n \geq 2^{1/\epsilon}$, we obtain the following.

$$\frac{m \cdot n^{1-\epsilon} + m}{m \cdot n^\epsilon + m} \geq \frac{1}{2} n^{1-2\epsilon} \geq n^{1-3\epsilon} \geq (m \cdot n + m)^{\frac{1}{3}(1-3\epsilon)} = N^{\frac{1}{3}-\epsilon}$$

Hence, if we can efficiently $(N^{\frac{1}{3}-\epsilon})$-approximate the colouring of $H^2_{m,G}$, then we can decide whether $\chi(G) \leq n^\epsilon$ or $\alpha(G) < n^\epsilon$. That concludes the proof. □

Note that a seemingly stronger result appeared in [1]. Namely, the authors claim that the chromatic number of the square of a split graph is not $(n^{1/2-\epsilon})$-approximable for all $\epsilon > 0$. However, this result is not correct. In fact, we show below that there exists a polynomial time algorithm $\sqrt[3]{n}$-approximating the chromatic number of the square of a split graph G, and also $\sqrt[3]{n}$-approximating the injective chromatic number of G. Note that this is also a strengthening of best known $\sqrt{n}$-approximation algorithm for the chromatic number of the square in general graphs (cf. [1]). We need the following lemma.

Lemma 8.14. *For chordal graphs, the injective chromatic number is α-approximable if and only if the chromatic number of the square is α-approximable.* □

Proof. First, suppose that we have an α-approximation algorithm A for the chromatic number of the square of a chordal graph. Observe that, if G is chordal, then also the graph $G - \mathcal{B}(G)$ is chordal. Hence, we can use A to α-approximately colour the square of $G - \mathcal{B}(G)$ using $k \leq \alpha \cdot \chi\big((G - \mathcal{B}(G))^2\big)$ colours, which also gives us an injective colouring of $G - \mathcal{B}(G)$. Then, using Corollary 8.27 and Theorem 8.31, we can extend this colouring to an injective colouring of G using no more than $\max\{\chi_i(G), k\}$ colours. This, clearly, is an α-approximation of $\chi_i(G)$, since $k \leq \alpha \cdot \chi_i(G - \mathcal{B}(G)) \leq \alpha \cdot \chi_i(G)$. The converse follows similarly, since, by Proposition 8.8, we can similarly extend any injective colouring of $G - \mathcal{B}(G)$ using k colours into a colouring of G^2 using no more than $\max\{k, \Delta(G) + 1\}$ colours. □

Theorem 8.15. *There exists a polynomial time algorithm that given a split graph G approximates $\chi(G^2)$ and $\chi_i(G)$ within a factor of $\sqrt[3]{n}$.*

Proof. Let G be a connected split graph with a clique X and an independent set Y. Denote by H the subgraph of G^2 induced on Y. Let $p = |X|$, $N = |V(H)|$, and $M = |E(H)|$. Clearly, $\chi(G^2) = p + \chi(H)$. Consider an optimal colouring of H with colour classes $V_1, V_2, \ldots, V_{\chi(H)}$. Let E_{ij} be the edges of H between V_i and V_j. Clearly, for each edge $uv \in E_{ij}$, there must exist a vertex x_{uv} in X adjacent to both u and v. Moreover, for any two edges $uv, st \in E_{ij}$, we have $x_{uv} \neq x_{st}$, since otherwise we obtain a triangle in $H[V_i \cup V_j]$, which is bipartite. Hence, $p \geq |E_{ij}|$, and considering all pairs of colours in H, we conclude that $p \geq M/\binom{\chi(H)}{2} \geq 2M/\chi^2(H)$.

Now, a simple edge count shows that any graph with t edges can be coloured with no more than $1/2 + \sqrt{2t + 1/4}$ colours. Such a colouring can be found by a simple greedy algorithm [18]. We can apply this algorithm to H, and use additional p colours to colour the vertices of X. This way we obtain a colouring c of G^2 using at most $p + 1 + \sqrt{2M}$ colours. Using the lower bound from the previous paragraph, we prove the following inequalities (assuming $M \geq 6$).

$$\frac{p+1+\sqrt{2M}}{\chi(G^2)} \leq \frac{p+1+\sqrt{2M}}{p+\sqrt{\frac{2M}{p}}} \leq (2M)^{1/6} \leq N^{1/3} \leq n^{1/3}$$

We concentrate on the second inequality (the other inequalities are obvious). We assume $p \geq 0$, $M \geq 0$, and substitute $z^2 = p$ and $a^6 = 2M$. We observe that the denominator in the fraction on the left is always positive; hence, we can multiply the whole inequality by the denominator without affecting the direction of the inequality sign. Hence, we have

$$z^2 + 1 + a^3 \leq a(z^2 + \frac{a^3}{z})$$

Also, since $z \geq 0$, we can multiply both sides with z. By rearranging the terms, we obtain

$$(a-1)z^3 - (a^3+1)z + a^4 \geq 0$$

Now, we fix sufficiently large $a > 1$, and consider the function $f(z)$ on the left-hand side. By taking the derivative of $f(z)$, we can compute the extremes $z_1 = \sqrt{\frac{1}{3}(a^3+1)/(a-1)}$, and $z_2 = -\sqrt{\frac{1}{3}(a^3+1)/(a-1)}$. Using the second derivative, we obtain that $f(z)$ achieves minimum in $z_1 \geq 0$ and maximum in $z_2 \leq 0$. Hence, if $f(z_1) \geq 0$, then $f(z) \geq 0$ for all $z \geq 0$. By taking sufficiently large a, we can guarantee that $f(z_1) \geq 0$, and the above inequality follows. (In fact, a more careful analysis reveals that $a \geq 1.5$, and hence, $M \geq 6$ suffices.)

Hence, the colouring c is an $\sqrt[3]{n}$-approximation of $\chi(G^2)$, and using Lemma 8.14, we can also obtain a colouring which is a $\sqrt[3]{n}$-approximation of $\chi_i(G)$. □

It turns out that the above result can be strengthened and its proof simplified using the following result from [48].

Theorem 8.16. [48] *Let G be a chordal graph with maximum degree $\Delta \geq 1$. Then:*

$$\chi(G^k) \leq \left\lfloor \sqrt{\frac{91k-118}{384}}(\Delta+1)^{(k+1)/2} \right\rfloor + \Delta + 1 = O(\sqrt{k}\Delta^{(k+1)/2})$$

We remark that, in fact, the proof of the above result implies a simple polynomial time greedy algorithm for obtaining a proper colouring of G^2 achieving the bound.

Now, we can prove the strengthening of Theorem 8.15.

Theorem 8.17. *There exists a polynomial time algorithm that given a chordal graph G approximates $\chi(G^2)$ and $\chi_i(G)$ within a factor of $\sqrt[3]{n}$.*

Proof. Let G be a chordal graph on n vertices with maximum degree $\Delta \geq 1$. First, suppose that $\Delta \geq n^{2/3} - 1$. Hence, by Observation 8.1, $\chi(G^2) \geq \Delta + 1 \geq n^{2/3}$. Now, a colouring that assigns a different colour to each vertex of G uses exactly n colours, and it is clearly a proper colouring of G^2, which achieves approximation ratio $n/\chi(G^2) \leq n/n^{2/3} = n^{1/3}$. On the other hand, if $\Delta < n^{2/3} - 1$, then, by the previous theorem, we can obtain in polynomial time a proper colouring of G^2 using at most $\frac{1}{2}(\Delta+1)^{3/2}+\Delta+1$ colours. Again, $\chi(G^2) \geq \Delta + 1$, and hence, (assuming $n \geq 8$) the colouring achieves approximation ratio

$$\frac{\frac{1}{2}(\Delta+1)^{3/2}+\Delta+1}{\chi(G^2)} \leq \frac{\frac{1}{2}(\Delta+1)^{3/2}+\Delta+1}{\Delta+1} = \frac{1}{2}(\Delta+1)^{1/2}+1 < \frac{1}{2}n^{1/3}+1 \leq n^{1/3}.$$

The second part of the claim again follows by Lemma 8.14. □

8.3 Exact algorithmic results

Now, we focus on algorithms for injective colouring of chordal graphs. Although, computing the injective chromatic number of a chordal graph is hard, the associated decision problem with a fixed number of colours has a polynomial time solution, that is, the problem is fixed parameter tractable. We need the following lemma.

Lemma 8.18. *For any chordal graph G, the treewidth of G^2 is at most $\frac{1}{4}\Delta(G)^2 + \Delta(G)$.*

Proof. Let (T, X) be a clique-tree of G and define $X'(u) = X(u) \cup N\big(X(u)\big)$ for each $u \in V(T)$. It is not difficult to check that (T, X') is a tree decomposition of G^2. We now bound the size of $X'(u)$ for all $u \in V(T)$, and thus, bound the treewidth of G^2.

$$\big|X'(u)\big| = \big|X(u)\big| + \sum_{x \in X(u)} \Big(\deg(x) - \big|X(u)\big| + 1\Big) \leq \big|X(u)\big|\Big(\Delta(G) - \big|X(u)\big| + 2\Big) \leq \left(\frac{\Delta(G)}{2} + 1\right)^2.$$

The inequalities above follow from the fact that $X(u)$ is a clique, $\deg(x) \leq \Delta(G)$, and that the third expression attains its maximum when $|X(u)| = \Delta(G)/2 + 1$. □

Theorem 8.19. *Given a chordal graph G and a fixed integer k, one can decide in time $O\big(n \cdot k \cdot k^{(k/2+1)^2}\big)$ whether $\chi_i(G) \leq k$ and also whether $\chi(G^2) \leq k$.*

Proof. It is easy to see that if $\chi_i(G) \leq k$ or if $\chi(G^2) \leq k$, then $\Delta(G)$ must be at most k. Thus, if $\Delta(G) > k$, we can reject G immediately. Using Lemma 8.18, we can construct in time $O(nk^2)$ a tree decomposition (T, X) of G^2 whose width is at most $k^2/4+k$. Now, using standard dynamic programming techniques on the tree T (cf. [18, 28] and Theorem 2.4), we can decide in time $O(n \cdot k \cdot k^{(k/2+1)^2})$ whether $\chi(G^2) \leq k$ and whether $\chi_i(G) \leq k$. □

Now, we show that for certain subclasses of chordal graphs, the injective chromatic number can be computed in polynomial time (in contrast to Theorem 8.11). First, we summarize the results; the details are presented in subsequent sections.

We call a graph G a *power chordal* graph, if all powers of G are chordal. Recall that in Propositions 8.8 and 8.9, we showed how, from the chromatic number of the square of the graph $G - \mathcal{B}(G)$, one can compute $\chi(G^2)$ for any graph G, respectively $\chi_i(G)$ for a split graph G. The following theorem describes a similar property for the injective chromatic number in chordal graphs. The proof will follow from Corollary 8.27 and Theorem 8.31, which we prove in Sections 8.3.2 and 8.3.4, respectively.

Theorem 8.20. *There exists an $O(n + m)$ time algorithm that computes $\chi_i(G)$ given a chordal graph G and $\chi_i(G - \mathcal{B}(G))$. Using this algorithm one can also construct an optimal injective colouring of G from an optimal injective colouring of $G - \mathcal{B}(G)$ in time $O(n+m)$.*

Proof. Let G be a chordal graph, and let k be the injective chromatic number of $G - \mathcal{B}(G)$. By Corollary 8.7, $\chi_i(G) \geq k$. Now, using Corollary 8.27, we can obtain an injective clique decomposition (T, X) of G such that the vertices of T can be partitioned into two sets V_b and V_t, where for each vertex $v \in V_b$, the graph $G[X(v)]$ is bridgeless, and for each vertex $v \in V_t$, the graph $G[X(v)]$ is perfectly tree-dominated. For $v \in V_b$, the graph $G[X(v)]$ must clearly be a subgraph of $G - \mathcal{B}(G)$, and hence $\chi_i\big(G[X(v)]\big) \leq k$. For $v \in V_t$, we can compute $\chi_i\big(G[X(v)]\big)$ and an optimal injective colouring of $G[X(v)]$ using Theorem 8.31. Now, by Proposition 8.24, we can obtain $\chi_i(G)$ by taking the maximum of k and the values of $\chi_i\big(G[X(v)]\big)$ for $v \in V_t$. By the same argument, if we have an optimal injective colouring of $G - \mathcal{B}(G)$, we can obtain an optimal injective colouring of G. □

For induced-hereditary subclasses of chordal graphs, we have the following property.

Proposition 8.21. *Let $\mathcal{C}$ be an induced-hereditary subclass of chordal graphs. Then the following statements are equivalent.*

(i) One can efficiently compute $\chi(G^2)$ for any $G \in \mathcal{C}$.

(ii) One can efficiently compute $\chi_i(G - \mathcal{B}(G))$ for any $G \in \mathcal{C}$.

(iii) One can efficiently compute $\chi_i(G)$ for any $G \in \mathcal{C}$.

This follows from Theorem 8.20, Proposition 8.8, and the fact that each connected component of $G - \mathcal{B}(G)$ must be in $\mathcal{C}$. In some cases, e.g., in the class of power chordal graph, this is true even if $\mathcal{C}$ is not induced-hereditary. The following corollary will follow from Theorem 8.20 and Corollary 8.29, which we prove in Section 8.3.3.

Corollary 8.22. *The injective chromatic number of a power chordal graph can be computed in polynomial time.*

This implies that the injective chromatic number of a strongly chordal graph can also be computed in polynomial time.

Finally, observe that due to Theorem 8.12, one cannot expect the property from Proposition 8.21 to hold for any subclass of chordal graphs.

8.3.1 Injective structure

In order to prove Theorem 8.20, we investigate the structural properties of graphs G that allow efficient computation of $\chi_i(G)$. In this section, G refers to an arbitrary connected graph (not necessarily chordal).

We say that a vertex separator of G is a *clique separator*, if it induces a clique in G. A tree decomposition (T, X) of G is a *decomposition by clique separators*, if for any $uv \in E(T)$, the set $X(u) \cap X(v)$ induces a clique in G. This type of decomposition of graphs was introduced and studied by Tarjan [65]. The decomposition turns out to be particularly useful for the graph colouring problem; namely, one can efficiently construct an optimal colouring of G from optimal colourings of $G[X(u)]$ for all $u \in V(T)$. We define and study a similar concept for the injective colouring problem. Recall that $G^{(2)}$ denotes the common neighbour graph of G defined in Section 8.1.

We say that a subset S of vertices of G is *injectively closed*, if for any two vertices $x, y \in S$ having a common neighbour in G, there exists a common neighbour of x and y that belongs to S. A subset S of vertices of G is called an *injective clique*, if S induces a clique in $G^{(2)}$.

Note that an injective clique is not necessarily injectively closed in G. A subset of vertices S of G is called an *injective separator* of G, if S is injectively closed in G, S is a separator of $G^{(2)}$, and $G^{(2)}$ is connected. Note that $G^{(2)}$ can be disconnected even if G is connected, e.g., if G is bipartite. An *injective decomposition* of G is a tree decomposition (T, X) of G such that for any $uv \in E(T)$, the set $X(u) \cap X(v)$ is an injective separator of G. An injective separator S is an *injective clique separator*, if S is also an injective clique. An *injective clique decomposition* is an injective decomposition (T, X) such that for any $uv \in E(T)$, the set $X(u) \cap X(v)$ is an injective clique. Note that any injective clique decomposition of G is a decomposition of $G^{(2)}$ and G^2 by clique separators.

We have the following properties.

Lemma 8.23. *Let (T, X) be an injective decomposition of a graph G. Then for each $u \in V(T)$, the set $X(u)$ is injectively closed.*

Proof. Let x, y be vertices of $X(u)$ having a common neighbour z in G. Suppose that $z \notin X(u)$. Then, since (T, X) is a tree decomposition, the vertices u' for which $z \in X(u')$ all belong to a connected component C of $T - u$. Let v be the (only) neighbour of u in C. Now, since xz, yz are edges, there must exist $w, w' \in V(T)$ such that $x, z \in X(w)$ and $y, z \in X(w')$. Clearly, $w, w' \in C$. Hence, it follows that $x, y \in X(v)$. But, since (T, X) is an injective decomposition, the set $X(u) \cap X(v)$ must be injective closed, and hence, there must exists a common neighbour $z' \in X(u) \cap X(v) \subseteq X(u)$ of x, y. □

Theorem 8.24. *Let (T, X) be an injective clique decomposition of a graph G. Then*

$$\chi_i(G) = \chi(G^{(2)}) = \max_{u \in V(T)} \chi\Big(G^{(2)}\big[X(u)\big]\Big) = \max_{u \in V(T)} \chi_i\Big(G\big[X(u)\big]\Big).$$

Proof. The first equality is by definition. We obtain the second equality from the fact that (T, X) is a decomposition of $G^{(2)}$ by clique separators. The last equality follows easily, since by Lemma 8.23, we have $G^{(2)}[X(u)] = G[X(u)]^{(2)}$, and by definition $\chi\big(G[X(u)]^{(2)}\big) = \chi_i\big(G[X(u)]\big)$. □

8.3.2 Computing $\chi_i(G)$ in chordal graphs

In this section, we focus on injective clique decompositions of chordal graphs. We have the following simple fact.

Observation 8.25. *Let H be a bridgeless graph having a dominating vertex. Then H is an injective clique.*

Proof. Let u be a vertex that dominates H. Clearly, u is a common neighbour of any two vertices of $H-u$. Now, if v is a vertex of $H-u$, then v must have a neighbour w in $H-u$, hence w is a common neighbour of u and v. □

We say that a graph G is *injectively decomposable*, if G contains an an injective clique separator S; we say that S *injectively decomposes* G. A graph G is *injectively indecomposable*, if it is not *injectively decomposable*. A graph G is called *perfectly tree-dominated*, if G contains an induced tree T, such that any vertex and any connected component of $G - V(T)$ has exactly one neighbour in T. For such T, we say that T *perfectly dominates* G, or that G is *perfectly dominated* by T.

The following statement relates injectively indecomposable chordal graphs and perfectly tree-dominated graphs.

Proposition 8.26. *Any perfectly tree-dominated graph is injectively indecomposable. Any injectively indecomposable chordal graph is either perfectly tree-dominated or bridgeless.* □

Proof. Let G be perfectly dominated by a tree T. We can assume that $G \neq T$ since otherwise $G^{(2)}$ is disconnected, and hence has no separators. Suppose that G has an injective clique separator S. It is easy to check that S contains at most one vertex of T. Suppose that S does not contain any vertex of T. Since S is an injective clique, $G[S]$ must be connected, and S belongs completely to some connected component of $G - V(T)$. Hence, by definition, there exists some vertex u in T that is adjacent to all vertices of S. But then S cannot be a separator, since any vertex of $G - S$ is reachable from u. Now, suppose that S contains a vertex u of T. It follows similarly that u must be adjacent to all vertices of $S \setminus \{u\}$. Consider any vertex x of $G - S$ that is adjacent to some vertex of S. It easily follows that x must be adjacent to u. But then S cannot be an injective separator, since u is adjacent to all components of $G - S$.

Now, let G be an injectively indecomposable chordal graph that contains a bridge but is not perfectly tree-dominated. Let F be the subgraph of G induced on the vertices of the bridges $\mathcal{B}(G)$ of G, and let T be any connected component of F. Clearly, T is a tree. Since G is not perfectly tree-dominated, there must exist a vertex x in G that is not adjacent to any of the vertices of T. Since G is connected, consider the vertex u of T that is closest to x in G.

Let v be any neighbour of u in T, and let $S = \{u\} \cup S_0$, where $S_0 = N(u) \setminus V(T)$. Since G is chordal, $G[S]$ must be bridgeless, and it is dominated by u. Hence, by Observation 8.25, S is an injective clique in G, and S is injectively closed. Now, since S clearly separates x from v in $G^{(2)}$, we have that S is an injective clique separator, therefore S decomposes G, however, G is injectively indecomposable. □

The property above has an important corollary.

Corollary 8.27. *For any chordal graph G, there exists an injective clique decomposition (T, X) of G, such that for any $u \in V(T)$, the set $X(u)$ induces either a bridgeless graph or a perfectly tree-dominated graph. This decomposition can be constructed in time $O(n + m)$.*

Proof. First, we find the bridges $\mathcal{B}(G)$ of G. Then, we construct a tree decomposition (T, X) of G such that for $u \in V(T)$, the set $X(u)$ is either a connected component of $G - \mathcal{B}(G)$, or a connected component T of $G[\mathcal{B}(G)]$ augmented with the neighbours of T in G. It follows from the proof of Proposition 8.26 that (T, X) is a injective clique decomposition. □

8.3.3 Bridgeless chordal graphs

In this section, we describe some classes of chordal graphs G that allow efficiently computing the chromatic number $\chi(G^2)$ of the square of G.

We focus on chordal graphs whose square is also a chordal graph. Clearly, for any such graph G, one can efficiently colour its square. Chordal graphs whose powers are also chordal were already studied in the past. In particular, it was shown by Duchet [22] that for any k, if G^k is chordal, then also G^{k+2} is chordal. Therefore, if a chordal graph G has a chordal square, then any power of G must be chordal, that is, G is power chordal. Interestingly, many known subclasses of chordal graphs, e.g. trees, interval graphs, and strongly chordal graphs, were shown to be power chordal [1]. Moreover, Laskar and Shier [51] found the following subgraph characterization of power chordal graphs. Recall that a k-sun is a graph formed by a cycle $v_0, v_1, \ldots, v_{k-1}$ with edges $v_i v_{i+1}$ (and possibly other edges), and an independent set $w_0, w_1, \ldots w_{k-1}$, where w_i is adjacent only to v_i and v_{i+1} (all indices are taken modulo k). A k-sun of a graph G is *suspended* in G, if there exists a vertex z in G adjacent to w_i and w_j, where $j \neq i$ and $j \neq i \pm 1$ modulo k.

Theorem 8.28. [51] *A graph G is power chordal, if and only if, any k-sun of G, $k \geq 4$, is suspended.*

Using this characterization, we obtain the following corollary.

Corollary 8.29. *If G is power chordal, the graph $G - \mathcal{B}(G)$ is also power chordal.*

Proof. Let $v_0, \ldots, v_{k-1}, w_0, \ldots, w_{k-1}$ be a k-sun S in G suspended by z, that is, there exist $i < j$ such that z is adjacent to w_i and w_j. It follows that neither zw_i nor zw_j is a bridge, since $w_i, v_{i+1}, v_{i+2}, \ldots, v_j, w_j, z$ is a cycle in G. Hence, S is also suspended in $G - \mathcal{B}(G)$. □

Note that, by Theorem 8.28, strongly chordal graphs are trivially power chordal, since no strongly chordal graph can contain an induced k-sun, $k \geq 3$ [25]. Also, notice that the class of power chordal graphs is not induced-hereditary (closed under taking induced subgraphs), since a graph that contains a k-sun can be power chordal, but the k-sun itself (taken as an induced subgraph) is not.

8.3.4 Perfectly tree-dominated graphs

In this final section, we show how to efficiently compute the injective chromatic number of a perfectly tree-dominated graph.

First, we have the following property.

Proposition 8.30. *For any perfectly tree-dominated graph G, we have*

$$\Delta(G) \leq \chi_i(G) \leq \chi(G^2) = \Delta(G) + 1.$$

Proof. Let G be perfectly dominated by T. Observe that any two vertices from $G - V(T)$ that are adjacent must be both adjacent to the same vertex of T. Hence, we can construct a graph H from G by removing all edges between the vertices of $G - V(T)$, and it follows that $\chi(G^2) = \chi(H^2)$. Now, clearly, H is a tree, and from Observation 8.1 and Proposition 8.2, we obtain $\Delta(G) + 1 \leq \chi(G^2) = \chi(H^2) = \Delta(H) + 1 \leq \Delta(G) + 1$. □

Now, let G be a perfectly tree-dominated graph. If G is a tree, then by Proposition 8.2, we have $\chi_i(G) = \Delta(G)$, and a greedy injective colouring of G will be optimal. Otherwise, let T_G be a minimal tree perfectly dominating G. We define a tree decomposition (T, X) of G as follows. We set $T = T_G$, and for $u \in V(T)$, we set $X(u) = N(u) \cup \{u\}$. Clearly,

$X(u) \cap X(v) = \{u, v\}$ is injectively closed, and the set $X(u) \cap X(v)$ is a separator of $G^{(2)}$. Hence, it follows that (T, X) is an injective decomposition of G. Using this decomposition, we can find $\chi_i(G)$ in time $O(n + m)$ as we show in the following theorem

Theorem 8.31. *The injective chromatic number $\chi_i(G)$ and an optimal injective colouring of a perfectly tree-dominated graph G can be computed in time $O(n + m)$.*

Proof. The algorithm follows from Theorem 2.4. A straightforward application of this theorem gives time complexity $O(\Delta^2 \cdot n)$, where Δ is the maximum degree in T. Unfortunately, Δ can be as large as $\Theta(n)$. To make the algorithm run in $O(n + m)$, we need the following simple observation.

Let $p_1, p_2, \ldots, p_t$ be integers and for any i let $m_i = \max\{p_j \mid j \neq i\}$. Then the values m_i for all i can be computed in time $O(t)$.

The proof is as follows. Let p_{l_1} and p_{l_2} be the largest and the second largest element among $p_1, \ldots, p_t$ respectively. Clearly, if $l_1 \neq i$ then $m_i = l_1$, otherwise $m_i = l_2$.

Recall that, in the colouring algorithm from Theorem 2.4, we tested, for each colouring c of a bag $X(v)$, whether there are matching colourings c_w of $X(v) \cap X(w)$ for the children w of v, and then we took the maximum of the number of colours used c, and m, which was the maximum of $m(c_w)$ among all children w of v, where $m(c_w)$ is the minimum colours needed to extend c_w to the graph G_w. Computing these numbers turns out to be costly in our case. In what follows, we show how to compute these numbers faster, using the above observation.

Let $u_1, \ldots, u_k$ be the children of v in T. Since $G^{(2)}[X(v) \cap X(u_i)]$ is just a pair of isolated vertices, there are only two different colourings, that is, either both vertices have the same colour or they have different colours; let c_i^e and c_i^d, respectively, denote these two colourings. Similarly for v, the bag $X(v)$ induces in $G^{(2)}$ a disjoint union of a clique of size k and an isolated vertex, and hence, either the isolated vertex has a colour different from the colours of the clique, or has the same colour as some vertex in the clique. It follows that each such colouring c of $G^{(2)}[X(u)]$ matches with at most one colouring c_i^e for some i, and, for all other $j \neq i$, it matches only with c_j^d. Hence, we use the observation above for the numbers $m(c_1^d), \ldots, m(c_k^d)$ to obtain, in time $O(k)$, the numbers $m_i = \max\{m(c_j^d) \mid j \neq i\}$ for each $i \in \{1 \ldots k\}$. This also gives us the value $d = \max\{m(c_i^d) \mid i \in \{1 \ldots k\}\}$ in the same time. Now, it can be seen that the number of colours $m(c)$ is the minimum of d and the numbers $\max\{m_i, m(c_i^e)\}$ for all $i \in \{1 \ldots k\}$. This clearly takes another $O(k)$ time to compute.

Therefore, for all vertices v, the complexity is $\sum_{v \in V(T)} O(\deg(v)) = O(|V(T)|) = O(n)$. Finally, we observe that one can obtain the tree decomposition (T, X) by computing the blocks of G in time $O(n + m)$. Hence, the theorem is proved. □

Chapter 9

Conclusions

$X \backslash Y$	K_2	$\overline{K_2}$	P_3	$\overline{P_3}$	K_k	$\overline{K_k}$	P_4	P_i
K_2	P							
$\overline{K_2}$	P	P						
P_3	$\frac{1P}{N}$	^2P	$\frac{3P}{N}$					
$\overline{P_3}$	^2P	$\frac{5P}{N}$	$\frac{1P}{N}$	$\frac{4P}{N}$				
K_l	$\frac{5P}{N}$	^6P	$\frac{7P}{N}$	$\frac{5P}{^bN}$	$\frac{5P}{N}$			
$\overline{K_l}$	^6P	$\frac{5P}{N}$	$\frac{^aP}{^bN}$	$\frac{5P}{N}$	^6P	$\frac{5P}{N}$		
P_4	^8N	$\frac{^aP}{N}$	^8N	^bN	^9N	$\frac{^aP}{N}$	^8N	
P_j	^9N	$\frac{^aP}{N}$	^9N	^bN	^9N	$\frac{^aP}{N}$	^9N	^9N

Notes: $k, l \geq 3$ and $i, j \geq 4$

[1] [23] and Theorems 3.10, 3.15
[2] [67] and Theorem 1.31
[3] [62] and Theorem 5.7
[4] [27] and Theorem 7.1
[5] [28] and Theorems 7.2, 7.5
[6] [3, 27] and Theorem 1.3
[7] Theorems 7.7, 7.8
[8] [63] and Theorems 6.1, 6.6,
[9] Theorem 6.1
[a] Theorem 7.5
[b] Theorems 7.9

$P \equiv$ Polytime in general
$N \equiv NP$-complete in chordal
$\frac{P}{N} \equiv \frac{\text{P in chordal graphs}}{\text{NPc in general graphs}}$

Table 9.1: A summary of complexity results in chordal graphs and general graphs.

In Table 9.1, we summarize our results from previous chapters. The entry in column X and row Y in the table indicates the complexity (both in general graphs and in chordal graphs) of the problem of partitioning a vertex set of a graph into two parts, where one part induces a X-free subgraph and the other induces a Y-free subgraph. Here, P stands for polynomial time and N stands for NP-complete. The fraction $\frac{P}{N}$ indicates that the problem

is NP-complete in general but polynomial time solvable in chordal graphs, whereas an entry with P respectively N indicates a polynomial time solvable respectively NP-complete problem in both classes.

As one could predict, some problems difficult in general graphs are polynomial time solvable in chordal graphs, while other ones remain NP-complete. As in the case of general graphs (Theorems 1.1 and 1.2), one would hope to obtain a complete characterization of complexity of partitioning problems for chordal graphs. In this thesis, we made an important step towards solving this by establishing complexities in a number of cases of partitioning problems in chordal graphs, which were previously not known. Based on these results, we would like to propose some generalizations (possibly even) leading to dichotomy theorems for these problems in chordal graphs. Although, it is probably too early to conjecture a complete dichotomy, it seems likely that a dichotomy for special types of these problems, just like the ones we study in this thesis, can be proved. In particular, we conjecture the following.

Conjecture 9.1. *Let H be a connected graph. Then the stable H-transversal problem in the class of chordal graphs is NP-complete, if H itself is chordal and contains an induced P_4, and polynomial time solvable otherwise.*

We remark that the conjecture has been confirmed in all the special cases we have addressed in this thesis; however, there is more evidence for the conjecture.

It can be seen, by examining the proof of Theorem 6.6, that this proof, in many ways, relies on the fact that P_4-free graphs are closed under the operation of join. (Actually, in chordal graphs, this is equivalent to being closed under adding a dominating vertex.) In general, if H does not contain a dominating vertex, then the class of H-free graphs is closed under adding a dominating vertex (see Proposition 7.12). If, in addition, H contains an induced P_4, it appears that it should be possible to adapt the proof of Theorem 6.6 to prove NP-completeness of the stable H-transversal problem in chordal graphs for such H as predicted by the conjecture.

On the other hand, if a connected chordal graph H is P_4-free, then it can be seen that H must contain a dominating vertex, and therefore, the class of H-free graphs is not closed under adding a dominating vertex. This suggest that it is not immediately possible to carry out the steps of the proof of Theorem 6.6 for such H. This provides additional evidence for the conjecture. Finally, the case of H with a P_4 and a dominating vertex is supported by the following theorem.

Theorem 9.2. *Let H be a chordal graph having no dominating vertex. Then the stable H-transversal problem in chordal graphs is polynomially reducible to the stable H'-transversal problem in chordal graphs, where H' is obtained from H by adding a dominating vertex.*

Proof. Let H be a chordal graph with no dominating vertex, and let H' be the graph constructed from H by adding a dominating vertex v. Also, let H'' be the graph constructed from H' by adding a twin to each vertex of H' with the exception of v. By Proposition 7.12, both H' and H'' are chordal. Now, we observe that each stable H'-transversal of H'' must contain the vertex v, and also $H'' - v$ is H'-free, since H has no dominating vertex.

Hence, let G be a graph, and let G' be the graph constructed from G by (i) adding a dominating vertex u into G, then (ii) taking a disjoint union with H'', and finally, (iii) adding an edge between u and v. Clearly, G' is chordal. In fact, it is easy to see that G has a stable H-transversal, if and only, if G' has a stable H'-transversal. Clearly, if S is a stable H-transversal of G, then $S \cup \{v\}$ is a stable H'-transversal of G', since $H'' - v$ is H'-free. On the other hand, if S' is a stable H'-transversal of G', then we must have $v \in S'$, and hence $u \notin S'$. If $G - S'$ contains a copy of H, then $G' - S'$ contains a copy of H', since u dominates G and $u \notin S'$. This shows that $S' \cap V(G)$ is a stable H-transversal of G, which concludes the proof. □

Note that, in the above theorem, H is not necessarily connected. Finally, we close by mentioning two similar problems we have not discussed in this thesis and whose complexity is unknown at the time of writing.

Problem 9.3. *Determine the complexity of the following problem. Given a chordal graph G, decide whether the vertex set of G admits a partition into sets $V_1 \cup V_2 \cup V_3$ such that $G[V_1]$ is an independent set, and both $G[V_2]$ and $G[V_3]$ are disjoint unions of cliques.*

Problem 9.4. *For any fixed symmetric 0-diagonal matrix M with entries $\{0, 1, *\}$, determine the complexity of the following problem. Given a chordal graph G, decide whether the vertex set of G admits a partition into sets $V_1 \cup V_2 \cup V_3$ such that $G[V_1]$ is M-partitionable, and both $G[V_2]$ and $G[V_3]$ are disjoint unions of cliques.*

Bibliography

[1] AGNARSSON, G., GREENLAW, R., AND HALLDÓRSSON, M. M. On powers of chordal graphs and their colorings. *Congress Numerantium 100* (2000), 41–65.

[2] ALBERTSON, M. O., JAMISON, R. E., HEDETNIEMI, S. T., AND LOCKE, S. C. The subchromatic number of a graph. *Discrete Mathematics 74* (1989), 33–49.

[3] ALEKSEEV, V. E., FARRUGIA, A., AND LOZIN, V. V. New results on generalized graph colorings. *Discrete Mathematics and Theoretical Computer Science 6* (2004), 215–222.

[4] BERGE, C. Färbung von graphen deren sämtliche bzw. deren ungerade kreise starr sind. *Wissenschaftliche Zeitschrift, Martin Luther Universitt Halle-Wittenberg, Mathematisch-Naturwissenschaftliche Reihe 10* (1961), 114–115.

[5] BODLAENDER, H. L., KLOKS, T., TAN, R. B., AND VAN LEEUWEN, J. Approximations for λ-coloring of graphs. *The Computer Journal 47* (2004), 193–204.

[6] BORIE, R. B., AND SPINRAD, J. P. Construction of a simple elimination scheme for a chordal comparability graph in linear time. *Discrete Applied Mathematics 91* (1999), 287–292.

[7] BROERSMA, H., FOMIN, F. V., NEŠETŘIL, J., AND WOEGINGER, G. J. More about subcolorings. *Computing 69* (2002), 187–203.

[8] CALAMONERI, T. The $L(h,k)$-labelling problem: A survey and annotated bibliography. *The Computer Journal 49* (2006), 585–608.

[9] CAMERON, K., ESCHEN, E. M., HOÀNG, C. T., AND SRITHARAN, R. The list partition problem for graphs. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)* (2004), pp. 391–399.

[10] CHUDNOVSKY, M., ROBERTSON, N., SEYMOUR, P., AND THOMAS, R. The strong perfect graph theorem. *Annals of Mathematics 164* (2006), 51–229.

[11] CORNEIL, D. G. Lexicographic breadth first search – a survey. In *Graph-Theoretic Concepts in Computer Science (WG 2004), Lecture Notes in Computer Science 3353* (2004), Springer, pp. 1–19.

[12] CORNEIL, D. G., AND KRUEGER, R. A unified view of graph searching. *SIAM Journal on Discrete Mathematics*. to appear.

[13] CORNEIL, D. G., LERCHS, H., AND STEWART, L. Complement reducible graphs. *Discrete Applied Mathematics 3* (1981), 163–174.

[14] COURCELLE, B. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation 85* (1990), 12–75.

[15] COURCELLE, B. On the expression of monadic second-order graph properties without quantifications over sets of edges. In *Fifth Annual IEEE Symposium on Logic in Computer Science (LICS)* (1990), pp. 190–196.

[16] COURCELLE, B. The monadic second-order logic of graphs III: Tree-decompositions, minors and complexity issues. *Informatique Théoretique et Applications 26* (1992), 257–286.

[17] DECHTER, R. Constraint networks. In *Encyclopedia of Artificial Intelligence*. 1992.

[18] DIESTEL, R. *Graph Theory*, 3rd ed., vol. 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, 2005.

[19] DIRAC, G. A. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg 25* (1961), 71–76.

[20] DREWES, F. NP-completeness of k-connected hyperedge-replacement languages of order k. *Information Processing Letters 45* (1993), 89–94.

[21] DREWES, F. Recognising k-connected hypergraphs in cubic time. *Theoretical Computer Science 109* (1993), 83–122.

[22] DUCHET, P. Classical perfect graphs. *Annals of Discrete Mathematics 21* (1984), 67–96.

[23] EKIM, T., HELL, P., STACHO, J., AND DE WERRA, D. Polarity of chordal graphs. *Discrete Applied Mathematics*. to appear.

[24] EKIM, T., MAHADEV, N. V. R., AND DE WERRA, D. Polar cographs. manuscript.

[25] FARBER, M. Characterizations of strongly chordal graphs. *Discrete Mathematics 43* (1983), 173–189.

[26] FARRUGIA, A. *Uniqueness and complexity in generalised colouring.* PhD thesis, University of Waterloo, 2003. `http://etd.uwaterloo.ca/etd/afarrugia2003.pdf`.

[27] FEDER, T., HELL, P., KLEIN, S., AND MOTWANI, R. List partitions. *SIAM Journal on Discrete Mathematics 16* (2003), 449–478.

[28] FEDER, T., HELL, P., KLEIN, S., NOGUEIRA, L. T., AND PROTTI, F. List matrix partitions of chordal graphs. *Theoretical Computer Science 349* (2005), 52–66.

[29] FEDER, T., AND VARDI, M. Y. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM Journal on Computing 28* (1999), 57–104.

[30] FEIGE, U., AND KILLIAN, J. Zero knowledge and the chromatic number. *Journal of Computer and System Sciences 57* (1998), 187–199.

[31] FIALA, J., JANSEN, K., LE, V. B., AND SEIDEL, E. Graph subcoloring: Complexity and algorithms. In *Graph-Theoretic Concepts in Computer Science (WG 2001), Lecture Notes in Computer Science 2204* (2001), Springer, pp. 154–165.

[32] FIALA, J., AND KRATOCHVÍL, J. Partial covers of graphs. *Discussiones Mathematicae Graph Theory 22* (2002), 89–99.

[33] FULKERSON, D. R., AND GROSS, O. A. Incidence matrices and interval graphs. *Pacific Journal of Mathematics 15* (1965), 835–855.

[34] GAGARIN, A. V. Chordal $(1, \beta)$-polar graphs. *Vestsi Nats. Akad. Navuk Belarusi Ser. Fiz.-Mat. Navuk* (1999), 115–118. (in Russian).

[35] GALLAI, T. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungaricae 18* (1967), 25–66.

[36] GAVRIL, F. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory B 16* (1974), 47–56.

[37] GIMBEL, J. G., AND HARTMAN, C. Subcolorings and the subchromatic number of a graph. *Discrete Mathematics 272* (2003), 139–154.

[38] GOLUMBIC, M. C. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.

[39] GOLUMBIC, M. C., AND JAMISON, R. E. The edge intersection graphs of paths in a tree. *Journal of Combinatorial Theory B 38* (1985), 8–22.

[40] GRIGGS, J. R., AND YEH, R. K. Labelling graphs with a condition at distance 2. *SIAM Journal on Discrete Mathematics 5* (1992), 586–595.

[41] GRÖTSCHEL, M., LOVÁSZ, L., AND SCHRIJVER, A. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica 1* (1981), 169–197. Corrigendum Combinatorica 4 (1984), 291–295.

[42] HAHN, G., KRATOCHVÍL, J., ŠIRÁŇ, J., AND SOTTEAU, D. On the injective chromatic number of graphs. *Discrete Mathematics 256* (2002), 179–192.

[43] HELL, P., KLEIN, S., NOGUEIRA, L. T., AND PROTTI, F. Partitioning chordal graphs into independent sets and cliques. *Discrete Applied Mathematics 141* (2004), 185–194.

[44] HELL, P., RASPAUD, A., AND STACHO, J. On injective colourings of chordal graphs. In *Proceedings of The 8th Latin American Theoretical Informatics Symposium (LATIN 2008).* to appear.

[45] HOÀNG, C. T., AND LE, V. B. On P_4-transversals of perfect graphs. *Discrete Mathematics 216* (2000), 195–210.

[46] HSU, W.-L., AND MA, T.-H. Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs. *SIAM Journal on Computing 28* (1999), 1004–1020.

[47] JENSEN, T. R., AND TOFT., B. *Graph coloring problems.* Wiley Interscience Series in Discrete Mathematics. Willey, New York, 1995.

[48] KRÁĽ, D. Coloring powers of chordal graphs. *SIAM Journal on Discrete Mathematics 18* (2005), 451–461.

[49] KUMAR, V. Algorithms for constraint-satisfaction problems. *AI Magazine 13* (1992), 32–44.

[50] LANGE, K.-J., AND WELZL, E. String grammars with disconnecting or a basic root of the difficulty in graph grammar parsing. *Discrete Applied Mathematics 16* (1987), 17–30.

[51] LASKAR, R., AND SHIER, D. On powers and centers of chordal graphs. *Discrete Applied Mathematics 6* (1983), 139–147.

[52] LAUTEMANN, C. The complexity of graph languages generated by hyperedge replacement. *Acta Informatica 27* (1990), 399–421.

[53] LEKKERKERKER, C. G., AND BOLAND, J. C. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae 51* (1962), 45–64.

[54] MESEGUER, P. Constraint satisfaction problem: an overview. *AICOM 2* (1989), 3–16.

[55] ROBERTSON, N., SANDERS, D. P., SEYMOUR, P., AND THOMAS, R. Efficiently four-coloring planar graphs. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (STOC)* (1996), pp. 571–575.

[56] ROSE, D. J. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications 32* (1970), 579–609.

[57] ROSE, D. J., TARJAN, R. E., AND LEUKER, G. S. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing 5* (1976), 266–283.

[58] ROZENBERG, G., Ed. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations.* World Scientific, 1997.

[59] SAWADA, J., AND SPINRAD, J. P. From a simple elimination ordering to a strong elimination ordering in linear time. *Information Processing Letters 86* (2003), 299–302.

[60] SHAEFER, T. J. The complexity of satisfiability problems. In *Proceedings of 10th ACM Symposium on Theory of Computing (STOC)* (1978), pp. 216–226.

[61] SPINRAD, J. P. *Efficient Graph Representations.* American Mathematical Society, 2003.

[62] STACHO, J. On 2-subcolourings of chordal graphs. In *Proceedings of The 8th Latin American Theoretical Informatics Symposium (LATIN 2008).* to appear.

[63] STACHO, J. On P_4-transversals of chordal graphs. *Discrete Mathematics.* to appear.

[64] TARJAN, R. E. Maximum cardinality search and chordal graphs. Unpublished Lecture Notes CS 259, Stanford University.

[65] TARJAN, R. E. Depth first search and linear graph algorithms. *SIAM Journal on Computing 1* (1972), 146–160.

[66] TARJAN, R. E. Decomposition by clique separators. *Discrete Mathematics 55* (1985), 221–232.

[67] TYSHKEVICH, R. I., AND CHERNYAK, A. A. Algorithms for the canonical decomposition of a graph and recognizing polarity. *Izvestia Akad. Nauk BSSR, ser. Fiz. Mat. Nauk. 6* (1985), 16–23. (in Russian).

[68] WEST, D. *Introduction to Graph Theory.* Prentice Hall, 1996.