Convexity in Graphs: Vertex Order Characterisations and Graph Searching

Von der Fakultät 1 – MINT – Mathematik, Informatik, Physik, Elektro- und Informationstechnik der Brandenburgischen Technischen Universität Cottbus–Senftenberg genehmigte Dissertation zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

vorgelegt von

Jesse Beisegel

geboren am 03.12.1987 in Bonn

Tag der mündlichen Prüfung: 05.03.2020

Abstract

We study convexities designed to characterise some of the most fundamental classes of graphs. To this end, we present some known results on this topic in a slightly different form, so as to give a homogeneous representation of a very disparate field. Furthermore, we present some new results on the Carathéodory number of interval graphs and also give a more or less exhaustive account of everything that is known in this context on AT-free graphs, including new results on characterising linear vertex orders and the structure of the intervals of this class.

We introduce the new class of bilateral AT-free graphs which is motivated by the linear order characterisation and the convexity used to describe AT-free graphs. We discuss their relation to other known classes and consider the complexity of recognition. Furthermore, as a consequence of notions from abstract convexity we present algorithmic results with regards to some natural subclasses of these.

As an application of notion of an extreme vertex of a convex geometry, we discuss structural aspects of avoidable vertices in graphs, which form a generalisation of simplicial vertices. This includes a characterisation of avoidable vertices as simplicial vertices in some minimal triangulation of the graph and a new proof of the existence result. Furthermore, we discuss the algorithmic issues regarding the problem of efficient computation of avoidable vertices in a given graph. This is complemented by an algorithmic application of the concept of avoidable vertices to the maximum weight clique problem, by identifying a rather general class of graphs in which every avoidable vertex is bisimplicial. This leads to a polynomial-time algorithm for the maximum weight clique problem in this class of graphs. Implications of this approach for digraphs are also discussed. All of these results lead to a conjecture concerning the generalisation of avoidable vertices to avoidable paths and we prove this conjecture for paths of length less or equal to two.

Finally, we analyse the properties of many different and widely used forms of graph search. Here, we discuss the problem of recognising whether a given vertex can be the last vertex visited by some fixed graph search. Moreover, we present some new aspects of the problem of deciding whether a given spanning tree of a graph is a graph search tree of a particular type of search. We generalise the concept of such trees to many well-known searches and give a broad analysis of the computational complexity of this problem. Both of these discussions are motivated by the use of graph searches in the context of computing properties of convexity.

Zusammenfassung

In dieser Arbeit betrachten wir Konvexitäten, die entworfen wurden, um einige der grundlegendsten Graphenklassen zu charakterisieren. Dazu präsentieren wir einige bekannte Resultate zu diesem Thema in einer abgeänderten Form, um eine homogene Darstellung eines diversen Felds zu bieten. Außerdem, geben wir neue Resultate über die Carathéodory Zahl von Intervallgraphen, sowie einen weitestgehend vollständigen Überblick über alle Ergebnisse bezüglich der charakterisierenden Konvexität von AT-freien Graphen, welcher auch neue Ergebnisse über charakterisierende Knotenordnungen und die Struktur der Intervalle dieser Klasse umfasst.

Wir führen die neue Klasse der bilateral AT-freien Graphen ein, welche durch die charakterisierende Knotenordnung und Konvexität der AT-freien Graphen motiviert ist. Wir diskutieren das Verhältnis dieser Graphen zu anderen Unterklassen der AT-freien Graphen und untersuchen die Komplexität ihrer Erkennung. Außerdem geben wir einige algorithmische Ergebnisse zu Unterklassen von bilateral AT-freien Graphen, welche aus der Analyse ihrer Konvexität folgen.

Als Anwendung des Begriffs eines Extremknoten einer konvexen Geometrie diskutieren wir einige strukturelle Aspekte von vermeidbaren Knoten, welche eine Verallgemeinerung der simplizialen Knoten darstellen. Dies beinhaltet eine Charakterisierung von vermeidbaren Knoten als simpliziale Knoten einer minimalen Triangulierung eines Graphen, sowie einen neuen Beweis über deren Existenz. Wir analysieren die algorithmischen Aspekte des Erkennungsproblems von vermeidbaren Knoten eines gegebenen Graphen. Diese Ergebnisse verwenden wir, um das Konzept eines vermeidbaren Knotens zur Berechnung von Cliquen maximalen Gewichts algorithmisch auszunutzen, indem eine Klasse ermittelt wird, für die jeder vermeidbare Knoten bisimplizial ist. Dies führt zu einem Polynomialzeitalgorithmus zur Berechnung einer Clique maximalen Gewichts auf dieser Klasse. Die Konsequenzen dieses Ansatzes werden auch für gerichtete Graphen analysiert. Alle diese Ergebnisse geben den Anlass zu einer Vermutung, die die Existenz vermeidbarer Knoten zu der Existenz vermeidbarer Pfade ausweitet; diese Vermutung wird für Pfade der Länge 1 und 2 bewiesen.

Schließlich betrachten wir die Eigenschaften einiger unterschiedlicher und häufig genutzter Graphensuchen. Wir diskutieren das Problem der Erkennung von Endknoten dieser Suchen. Außerdem präsentieren wir neue Ergebnisse über die Erkennung von Suchbäumen verschiedener Graphensuchen. Wir verallgemeinern das Konzept solcher Suchbäume, um weitere komplexere Suchstrategien abzufangen, und betrachten die Komplexität der Erkennung solcher Bäume. Diese Untersuchungen sind motiviert durch die häufige Verwendung von Graphensuchen, um Eigenschaften von Konvexitäten algorithmisch zu ermitteln.

Acknowledgements

First, I would like to thank my advisor Ekki for introducing me to and guiding me through some of the key subjects of this thesis and giving me the opportunity to research freely wherever my interest lay. This work would not have been possible without the great working atmosphere that I have enjoyed during the production of this text and I wish to thank all the colleagues I have worked with throughout the years, especially Carolin, Martin and Robert. Furthermore, a big part of this thesis was developed as part of a bilateral project with the University of Primorska, and I want to thank Martin, Matjaž, Nina and Nevena for their hospitality during my many stays in Koper.

Some results presented in this thesis were part of a team effort, and I would like to thank all my co-authors for their input and helpful remarks that have made this manuscript a much easier read: Maria Chudnovsky, Carolin Denkert, Vladimir Gurvich, Ekkehard Köhler, Matjaž Krnc, Martin Milanič, Nevena Pivač, Robert Scheffler, Mary Servatius and Martin Strehler

Finally, I wish to thank my parents for proofreading this incomprehensible gibberish, as well as Fabienne for putting up with my constant complaints about broken proofs and other mathematicians' problems and for encouraging me to keep going.

Cottbus, October 8th 2019 Jesse Beisegel

Contents

Introduction

Since its beginnings in the 1960s and especially after the publication of the classical survey by Golumbic, *Algorithmic Graph Theory and Perfect Graphs* [75] in 1980, the field of algorithmic graph theory has grown rapidly. Countless different graph classes have been introduced, analysed and categorised. In 1999, another survey, *Graph Classes* by Brandstädt et al. [22], covered as many as 200 different classes, while the website `graphclasses.org` counts as many as 1600 in its database (although a significant amount of these are probably equivalent). These efforts in research have been rewarded with many striking results and a host of very efficient algorithms for problems that are $\mathcal{NP}$-hard on general graphs. Some of the most prominent examples are the recognition of perfect graphs achieved by Chudnovsky et al. [32, 33] and the solution to the colouring problem on these using the ellipsoid method given by Grötschel et al. [78], as well as the linear time recognition of chordal [126, 136] and interval graphs [18, 102] which yield very simple linear time algorithms for such optimisation problems as maximum independent set and colouring [75, 63].

These results have been achieved using a vast amount of different tools, including forbidden (induced) minor characterisations (see for example Kuratowskis characterisation of planar graphs [108]), forbidden induced subgraph characterisations (see Gallais characterisation of comparability graphs [67]), as well as such algorithmic techniques as modular decomposition (Gallai [67]), or PQ-trees (Booth and Lueker [18]). In this text, we will concentrate on one particular construction known as a *characterising linear vertex ordering*. This can be seen as a linear order $\sigma = (v_1, \ldots, v_n)$ of the vertices of a graph, as well as some property $\mathcal{P}_\mathcal{G}$ of a linear ordering, such that a graph G belongs to a given graph class $\mathcal{G}$ if and only if there exists a linear ordering σ of the vertices of G with property $\mathcal{P}_\mathcal{G}$.

One of the earliest examples of such an ordering was given by Rose [123] for chordal graphs, i.e., the class of graphs which do not contain an induced cycle of size greater or equal four. Rose characterised a graph G as being chordal if and only if it has a *perfect elimination ordering*, i.e., an ordering $\sigma = (v_1, \ldots, v_n)$ of the vertices of G, such that for any v_i, v_j and v_k with $i < j < k$ the fact that $v_iv_k, v_jv_k \in E$ implies that v_i and v_j are adjacent. On a chordal graph a perfect elimination ordering can be found in linear time by performing a special graph search algorithm based on breadth first search and it is not only useful to characterise these graphs with a very compact certificate, but also a crucial ingredient to many different optimisation algorithms. For example, if we are given a perfect elimination ordering, an optimal colouring of the corresponding graph can be achieved by a simple greedy colouring strategy from right to left on that order [75].

The possibility of being able to describe a graph class with a linear vertex order hints at an underlying structure of that graph which is best described as some form of convexity. In fact, there have been several attempts to do just this, as can be seen for example in the survey on *convex geometries* by Edelman and Jamison [56] or the book by Korte et al. [101] which gives an equivalent concept named an *anti-matroid*.

The concept of *convexity* is at the heart of mathematical optimisation in all its forms, whether this be linear or integer programming, non-linear optimisation, multi-criterial optimisation or optimal control. In all these fields, convexity ensures that optimal solutions need only be searched for in a restricted area of all possible solutions, for example in linear programming an optimal solution can be found among the *vertices* of the polyhedron underlying a linear program (Dantzig [47]).

Convexity used in this way is motivated through geometry, a certain shape of the solution space which lends itself to optimisation. However, since this concept was properly axiomatised in a much more general form throughout the 20th century, it has been possible to transfer these structural concepts to other areas where a geometric intuition does not necessarily apply. One of these is the field of combinatorics and, in particular, graph theory.

Using the example of a particular family of graph classes, we will argue that abstract convexity can be used to unify several important concepts: chordal graphs, interval graphs and AT-free graphs form an instructive example to this end. AT-free graphs are defined as the graphs which do not contain an independent triple of vertices, such that each two of them are joined by a path that avoids the neighbourhood of the third. The class of interval graphs forms one of the oldest and most famous graph classes. Defined as the intersection graph of intervals on the real line, they form the intersection of AT-free graphs and chordal graphs, i.e., a graph is an interval graph if and only if it is AT-free and chordal (Lekkerkerker and Boland [109]). This family of classes possesses many interesting properties and has a very rich algorithmic structure which makes it possible to solve many classical $\mathcal{NP}$-complete graph problems on these efficiently. Furthermore, they are also related to another important class: the comparability graphs. This is the class of graphs representing some partial order. It can be shown that the intersection between chordal graphs and cocomparability graphs, those graphs which form the complement of a comparability graph, is again the class of interval graphs.

Due to the geometric nature of the definition of interval graphs, it is not surprising that they convey a convex structure. However, we will see that such a structure can also be found in the much more abstract AT-free graphs. We will present tailor-made convexities for each of these three classes which can be used to characterise them. Some of the most important structural properties of these classes will directly correspond to some of the standard concepts of their respective convexity. In particular, it can be shown that the convexity defining interval graphs can be constructed by somehow intersecting the convexities of AT-free graphs and chordal graphs.

While chordal graphs and interval graphs have been studied so exhaustively that one cannot hope to attain many new insights, we will argue that for AT-free graphs, the study of their convexity is a useful tool to find further structural properties. In particular, we present a new vertex order characterisation of this class based on a convexity.

To make algorithmic use of this concept of convexity in a graph, it would be of a great advantage to have some standardised routine to compute important structures of the convexity, such as convex sets and extreme points. For most graph convexities (and for all discussed in this text), such a routine is given by a *graph search*. A generalised graph search can be roughly described as a procedure visiting all vertices of a graph one by one, while always maintaining a *connected* induced subgraph on the visited vertices. We will see that such searches can be constructed to maintain not only connected but also convex subsets of the vertices by furnishing it with appropriate rules for vertex selection. Furthermore, it will also be possible to find extreme vertices in the form of the vertices visited last by such a search.

In the light of such results it becomes important to analyse graph searches very thoroughly with regard to their algorithmic properties and the structures which are computed by them. For example, if a vertex visited last by a search is an extreme point of a convexity, it becomes important to be able to decide whether a given vertex can be such a vertex visited last by a search. Furthermore, as a graph search is able to maintain a convex set throughout its run, it is interesting to analyse the structure of these sets with regard to the performed search. This can be done not only by examining the exact order of the visited vertices but also by the study of the so-called graph search tree. This tree can be defined in different ways (and we will discuss some of these possibilities in a later chapter), but one well-known example can be easily explained in the form of the BFS-tree. This tree is computed by the *breadth-first search algorithm* and adds an edge from every vertex of a graph to its neighbour visited first by the search. This tree contains all shortest paths between the start vertex (the vertex visited first by the search) and all other vertices of the graph, where a shortest path is understood to be a path with the least amount of edges.

Results

In the following we give an overview of the most important new results presented in this text structured by the respective subject matter. As most of these results have already been published in some form or other, we clearly state a corresponding reference at the beginning of each section.

Graph Convexity and Subfamilies of AT-free Graphs

Some of the results stated here can be found in a published extended abstract [6].

1. Graph Convexities Characterising Graph Classes. We give an overview of results which characterise graph classes with a matching convex geometry and bring this into context with vertex order characterisations for these. In addition to well known convexities such as monophonic convexity (used to characterise chordal graphs), we also present a very recent convex geometry characterising AT-free graphs and a new convex geometry called the interception convexity. We present all of these results in a uniform manner using the language of interval convexity and show how this theory can be used to give a homogeneous representation of some of the most crucial properties in these classes.

2. Convexity on Interval Graphs and a Tight Bound for the Carathéodory Number. Using a Theorem by Lekkerkerker and Boland [109], which characterises the class of interval graphs as the intersection of AT-free and chordal graphs, we present a new formulation of a characterising convexity which coincides with the construction given by Alcón et al. [3] (Theorem 1.4.13). Using this and with the help of a technique designed by Chvátal [34], we give a new proof that this convexity is in fact a convex geometry. Furthermore, we show that the Carathéodory number of this convexity is at most 2 and that this bound is tight (Corollary 1.4.14).

3. Convexity on AT-free Graphs and a new BFS-based Vertex Order Characterisation. Building on the work of Chang et al. [28] who defined a characterising convex geometry for AT-free graphs, we give new results on the structure of this convexity. We make some progress towards giving a tight upper bound for the Carathéodory number of this convexity. Furthermore, we use these results to state a polynomial time BFS-based algorithm which computes a new characterising vertex order of AT-free graphs that always coincides with a BFS order, settling an open question due to Corneil and Stacho [38] (Theorem 1.5.10).

4. The Recognition of Bilateral AT-free Graphs and Subfamilies. Motivated by vertex order characterisation and the convexity of AT-free graphs, we introduce the class of bilateral AT-free graphs. After comparing this new class with other well-known subclasses of AT-free graphs such as cocomparability graphs, we show that the recognition of this class is $\mathcal{NP}$-complete (Theorem 2.2.9). For some subclasses of this class we give linear time algorithms to compute AT-free and bilateral AT-free orders by using multisweep graph search algorithms (Theorems 2.3.5 and 2.3.10).

Avoidable Vertices and Paths

The results given here are joint work with Maria Chudnovsky, Vladimir Gurvich, Martin Milanič and Mary Servatius. A published extended abstract of this work can be found in [7].

1. Characterisation, existence, and computation of avoidable vertices. Following the work of Ohtsuki et al. [117], we revisit the connection between avoidable vertices and minimal triangulations of graphs by characterising avoidable vertices in a graph G as exactly the simplicial vertices in some minimal triangulation of G (Theorem 3.1.1). Using properties of Lexicographic Breadth First Search that follow from works of Berry and Bordat [13] and Aboulker et al. [1], we show that every graph with at least two vertices contains a diametral pair of avoidable vertices (Theorem 3.2.4). The same approach shows that a pair of distinct (though not necessarily diametral) avoidable vertices in a given graph G with at least two vertices can be computed in linear time (Theorem 3.2.5).

2. New polynomially solvable cases of the maximum weight clique problem. A graph is 1-perfectly orientable if its edges can be oriented so that the out-neighbourhood of every vertex induces a tournament, and hole-cyclically orientable if its edges can be oriented so that each induced cycle of length at least four is oriented cyclically. We connect the structural and algorithmic properties of avoidable vertices with the concept of bisimplicial vertices to develop an efficient algorithm for the maximum-weight clique problem in the class of 1-perfectly orientable graphs and, more generally, in the class of hole-cyclically orientable graphs (Theorem 3.3.6). These results generalize the well known fact that the maximum-weight clique problem is polynomial-time solvable in the classes of chordal graphs and circular-arc graphs.

3. Existence of avoidable edges. We show that for every graph G and every non-universal vertex $v \in V(G)$ there exists an avoidable vertex in the non-neighbourhood of v (Theorem 3.1.4). While this result clearly follows from Theorem 3.2.4, we give a direct proof that is not based on any graph search (such as LBFS). We then adapt the approach to prove the existence of two avoidable edges in any graph with at least two edges (Theorem 3.4.8). This settles in the affirmative the case $k = 2$ of Conjecture 3.0.8 and generalizes the case $k = 2$ of Theorem 3.0.6.

4. Implications for vertex- and edge-transitive graphs. We derive some consequences of existence results for avoidable vertices and edges for highly symmetric graphs. More specifically, we show that in a vertex-transitive graph every induced two-edge path closes to an induced cycle (Corollary 3.5.1), while in an edge-transitive graph every 3-edge path closes to a cycle (Corollary 3.5.2) and every induced 3-edge path closes to an induced cycle (Corollary 3.5.3). While these structural results are straightforward consequences of the results on avoidable vertices and edges, we are not aware of any statement of these results in the literature. For all the three statements, we give examples showing that analogous statements fail for longer paths.

End-Vertices and Graph Search Trees

The results given here have been achieved in joint work with Carolin Denkert, Ekkehard Köhler, Matjaž Krnc, Nevena Pivač, Robert Scheffler and Martin Strehler. The results on end-vertices of graph searches are given in [9]. A published extended abstract concerning graph search trees can be found in [8].

1. The End Vertex Problem for MCS and MNS. Drawing on work by Corneil et al. [44] and Charbit et al. [30], we investigate the problem of deciding whether a given vertex in a graph is the last vertex in some execution of a particular type of graph search. Complementing the results given there on BFS, LBFS and LDFS, we show that this problem is also $\mathcal{NP}$-complete for Maximum Cardinality Search (Theorem 4.2.1) and $\mathcal{NP}$-complete on weakly chordal graphs for Maximal Neighbourhood Search (Theorem 4.1.2).

2. Polynomial Algorithms for the End-Vertex Problem on Interval, Unit Interval and Split Graphs. For the class of Chordal Graphs we give an improved running time analysis for the end-vertex problem on MNS, reducing it to linear time (Corollary 4.3.4). Among other results, we present a linear time decision algorithm for the end-vertex problem of DFS for interval graphs (Corollary 4.3.12), as well as linear time algorithms for unit interval and split graphs for LDFS, MCS and MNS (Proposition 4.3.3, Corollary 4.3.4 and Corollary 4.3.7).

3. The Graph Search Tree Recognition Problem. We give a new formulation for the problem of deciding whether a given spanning tree is a graph search tree for some execution of a particular graph search. This formulation distinguishes two different variants of defining a graph search tree: A first-in tree ($\mathcal{F}$-tree) which is based on the construction of BFS trees, and a last-in tree ($\mathcal{L}$-tree) which is derived from the DFS tree. Adding to results by Korach and Ostfeld [100] and Manber [112], who studied this question for the cases of DFS and BFS, we show that recognising graph search trees is $\mathcal{NP}$-complete in the case of $\mathcal{F}$-trees for the searches LBFS, LDFS, MCS and MNS even on the class of weakly chordal graphs (Theorems 5.2.1 and 5.3.1).

4. Polynomial Algorithms for Recognising Graph Search Trees. Giving a nice boundary of complexity, we present polynomial time algorithms to solve the $\mathcal{F}$-tree recognition problem for LBFS, LDFS, MCS and MNS on chordal graphs (Theorem 5.4.7 and Corollary 5.4.10) and a linear time algorithm for split graphs for those same searches (Corollary 5.5.2). In the case of the $\mathcal{L}$-tree recognition, we show that recognition can be solved in polynomial time for LDFS on all graphs (Theorem 5.1.4). In addition, we give linear time algorithms for this problem on chordal graphs in the cases of LBFS, LDFS, MCS and MNS (Theorem 5.4.14 and Corollary 5.4.16).

Structure of the Thesis

In Chapter 0, we give a brief overview of most of the notation and concepts needed in the course of this work. We begin with some basic notation and definitions on graphs most frequently used here. Then we proceed to a short summary of some important decision and optimisation problems which form the basis and motivation for algorithmic graph theory. This is followed by short introductions to the fields of algorithmic graph theory, graph classes and graph searching. We conclude Chapter 0 with an elementary survey of some of the most important concepts (at least in the context of graph theory) of (discrete) abstract convexity theory.

Chapter 1 is dedicated to the analysis of convexities designed to characterise some of the most fundamental classes of graphs. Here, we present some known results on this topic in a slightly different form, so as to give a homogeneous representation of a very disparate field. Furthermore, we present some new results on the Carathéodory number of interval graphs and also give a more or less exhaustive account of everything that is known in this context on AT-free graphs, including new results on characterising linear vertex orders and the structure of the intervals of this class.

In Chapter 2, we introduce the new class of bilateral AT-free graphs which is motivated by the linear order characterisation and the convexity used to describe AT-free graphs. We discuss their relation to other known classes and consider the complexity of recognition. Furthermore, as a consequence of the results of Chapter 1 we present algorithmic results with regards to some natural subclasses of these.

In Chapter 3, we discuss structural aspects of avoidable vertices in graphs, which form a generalisation of simplicial vertices. This includes a characterisation of avoidable vertices as simplicial vertices in some minimal triangulation of the graph and a new proof of the existence result. Furthermore, we discuss the algorithmic issues regarding the problem of efficient computation of avoidable vertices in a given graph. This is complemented by an algorithmic application of the concept of avoidable vertices to the maximum weight clique problem, by identifying a rather general class of graphs in which every avoidable vertex is bisimplicial. This leads to a polynomial-time algorithm for the maximum weight clique problem in this class of graphs. Implications of this approach for digraphs are also discussed. Finally, we state a conjecture concerning the generalisation of avoidable vertices to avoidable paths and prove this conjecture for paths of length less or equal to two.

Chapters 4 and 5 are concerned with the properties of many different and widely used forms of graph search. Here, we discuss the problem of recognising whether a given vertex can be the last vertex visited by some fixed graph search. Moreover, we present some new aspects of the problem of deciding whether a given spanning tree of a graph is a graph search tree of a particular type of search. We generalise the concept of such trees to many well-known searches and give a broad analysis of the computational complexity of this problem. Both of these discussions are motivated by the use of graph searches in the context of computing properties of convexity.

0 Preliminaries

In this chapter, we recall some of the basic notation and definitions used throughout this text. We give short overviews of some $\mathcal{NP}$-complete graph problems, as well as introducing some of the most well known graph classes in use throughout this text. In this context, we also present a selection of the most important graph searches, as they will be used frequently throughout this text. Finally, we end this chapter with a brief introduction to the field of abstract convexity with an emphasis on interval convexity and convex geometry. For anything not defined here or anywhere else in this text, we refer to classic textbooks in these various fields: for any basic graph theoretic notions we recommend [48, 147], for topics in algorithmic graph theory [22, 75], for graph searching [37, 65] and finally, for any questions on abstract convexity [56, 144]

0.1 General Notation

In this section, we summarize the definitions of some of the most frequently used notions in this text. All graphs in this text will be finite but may be either undirected or directed. We will refer to an undirected graph simply as a *graph* and denote it as $G = (V, E)$ where V is the vertex set and $E \subseteq V \times V$ the edge set. A directed graph will be called a *digraph* and denoted as $D = (V, A)$ where V is again the set of vertices and A the set of arcs. Graphs and digraphs in this text will always be simple, that is, without loops or multiple edges (with pairs of oppositely oriented arcs in digraphs allowed). Given a graph $G = (V, E)$, we denote by n and m the number of vertices and edges in G, respectively. Unless stated otherwise we use standard graph and digraph terminology and notation. In particular, an edge in a graph connecting two vertices u and v will be denoted as $\{u, v\}$ or uv and in this case u and v are said to be *adjacent*. Furthermore, the vertices u and v are said to be incident to the edge uv. An arc in a digraph pointing from u to v will be denoted as (u, v) or uv. The set of all vertices adjacent to a vertex v in G, i.e., its *neighbourhood*, is denoted by $N_G(v)$ and the cardinality of this set, the *degree* of v, by $d_G(v)$. Similarly, the *closed neighbourhood* $N_G(v) \cup \{v\}$ is written as $N_G[v]$. The *minimal degree* of the vertices of a graph G is denoted by $\delta(G)$ and the *maximal degree* by $\Delta(G)$. A vertex with degree 0 is said to be *isolated*. If every vertex of a graph G is of degree exactly k, then that graph is said to be *k-regular*.

We say that a graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Given a graph G and a set S of its vertices, we denote by $G[S]$ the *subgraph of G induced by S*, that is, the graph with vertex set S and edge set $\{uv \in E(G) : u, v \in S\}$. By $G-S$ we denote the subgraph of G induced by $V(G)\setminus S$, and when $S = \{v\}$ we also write $G - v$. Given an edge set $F \subseteq E(G)$, we denote by $G - F$ the graph $(V(G), E(G) \setminus F)$; when $F = \{f\}$, we also write $G - f$. Similarly, if u and v are nonadjacent vertices of

G, then $G + uv$ is obtained from G by connecting u and v with an edge. A *clique* in a graph G is a set of pairwise adjacent vertices and an *independent set* in G is a set of pairwise nonadjacent vertices. By P_n, C_n, and K_n we denote the path, cycle, and the complete graph with n vertices, respectively. A *claw* is a graph which consists of one vertex adjacent to exactly three vertices of degree one. A *net* consists of a C_3, whose vertices are adjacent to three different vertices of degree one.

A *walk* of length k in a graph G is a sequence of vertices $(v_1, \dots, v_{k+1})$ such that $v_i v_{i+1} \in E$ for all $i \in \{1, \dots, k\}$. This walk is called a *trail* if all its edges are distinct. If a trail has the additional property that all vertices v_i for $i \in \{1, \dots k\}$ are pairwise distinct, we call it a *path* (in G). We will sometimes denote a path $(v_1, \dots, v_k)$ as $v_1 - \dots - v_k$. A trail whose end-vertices coincide is called a *circuit*. A walk $(v_1, \dots, v_k)$ in G such that all vertices v_i for $i \in \{1, \dots k\}$ are pairwise distinct, except that $v_1 = v_k$, is called a *cycle* (in G). A *chord* of a path (cycle) is an edge between two vertices on the path that is itself not part of this path (cycle). A path or a cycle in G is said to be *chordless* if it does not have any chords. We say that a path (respectively, induced path), say $(v_1, \dots, v_k)$, in a graph G *closes* to a cycle (respectively, induced cycle) if there is a cycle (respectively, induced cycle) in G of the form $(v_1, \dots, v_k, u_1, \dots, u_p)$. Given a path P in a graph G and vertices $x, y \in V(P)$, we denote by $x - P - y$ or $P_{[x,y]}$ the subpath of P from x to y. Furthermore, in analogy to open and closed intervals, we will denote by $P_{(x,y)}$ the subpath of P from x to y excluding x and y, and by $P_{(x,y]}$ and $P_{[x,y)}$ the subpaths excluding x and y, respectively. Concatenations of such paths into longer paths or cycles will be denoted similarly, by $x - P - y - Q - z$, etc. We say that a path P *avoids* a vertex v if the intersection of $N[v]$ and $V(P)$ is empty, and we say that v *intercepts* P otherwise.

Two graphs are *isomorphic* if there is a correspondence between their vertex sets that preserves adjacency. Thus, $G = (V, E)$ is isomorphic to $G' = (V, E')$ if there is a bijection $\phi : V \to V'$ such that $uv \in E$ if and only if $\phi(u)\phi(v) \in E'$. Given a family of graphs $\mathcal{F}$, we say that a graph is *$\mathcal{F}$-free* if no induced subgraph of G is isomorphic to a graph in $\mathcal{F}$.

An *automorphism of a graph* G is a bijection from the vertex set of G to itself that maps edges to edges and non-edges to non-edges. A graph G is said to be *vertex-transitive* if for every two vertices $u, v \in V(G)$ there exists an automorphism of G mapping u to v. Similarly, G is said to be *edge-transitive* if for every two edges $e, f \in E(G)$ there exists an automorphism of G mapping e to f.

An *orientation* of a graph $G = (V, E)$ is a digraph obtained by assigning each edge of G a direction. A *tournament* is an orientation of the complete graph. Given a digraph $D = (V, A)$, the *in-neighbourhood* of a vertex v in D, denoted by $N_D^-(v)$, is the set of all vertices w such that $(w, v) \in A$. Similarly, the *out-neighbourhood* of v in D, denoted by $N_D^+(v)$, is the set of all vertices w such that $(v, w) \in A$.

A graph is said to be *connected* if for every pair of distinct vertices u and v there is a path from u to v and *disconnected* otherwise. A maximal connected subgraph is a *connected component* of the graph. In a connected graph a subset of vertices $S \subseteq V$ is called an *a-b-separator* for two nonadjacent vertices $a, b \in V$ if a and b are contained in different components of $G - S$. Furthermore, a set $S \subseteq V$ is a *separator* if there is a pair of vertices $a, b \in V$ such that S is an a-b-separator. In particular, if such a cutset consists of only one vertex, we call it an *articulation vertex*, or *cutvertex*. An a-b-separator is called *minimal* if it does not contain any other a-b-separator. A separator S is called a *minimal separator* if there is a pair of vertices $a, b \in V$ such that S is a minimal a-b-separator.

Similarly, for two vertices $s, t \in V$ of a connected graph G an *s-t-cut* is a set of edges such that after their deletion s and t are in different components of the graph. A cut consisting of just one edge is called a *bridge*. A graph without any cycles is called a *forest*, or an *acyclic graph*; a *tree* is a connected forest.

The *complement* of a graph G is the graph $\overline{G}$ with the same vertex set as G, in which two distinct vertices are adjacent if and only if they are not adjacent in G. The *line graph* $L(G)$ of G is the graph which has vertex set $E(G)$ and two vertices e and f of $L(G)$ are adjacent if they have a vertex in common in G. By $2K_2$ we denote the graph consisting of two disjoint and independent copies of K_2. A graph is *bipartite* if its vertex set can be partitioned into two independent sets, which are then said to be a *bipartition* of the graph. By $K_{m,n}$ we denote the complete bipartite graph with m vertices on one side of the bipartition and n on the other one. A graph is *cobipartite* if its complement is bipartite.

The *distance* between two vertices s and t is the length of a shortest path between these two vertices and will be denoted by $\text{dist}_G(s, t)$. The set of vertices that have distance k to a vertex s is called the *k-th distance layer* from s of G and is denoted by $L^k_G(s)$. For every vertex $v \in V$ we say that $N^k_s(v) := L^k_G(s) \cap N_G(v)$. A vertex x with largest distance from s is called *eccentric* with respect to s and its distance to s is the *eccentricity* $\text{ecc}_G(s)$ of s. The *diameter* of G, denoted $\text{diam}(G)$, is the largest such value among all vertices.

An *ordering of vertices* in G is a bijection $\sigma : V(G) \to \{1, 2, \ldots, n\}$. For an arbitrary ordering σ of vertices in G, we will denote by $\sigma(v)$ the position of vertex $v \in V(G)$. Given two vertices u and v in G we say that u is *to the left* (resp. *to the right*) of v if $\sigma(u) < \sigma(v)$ (resp. $\sigma(u) > \sigma(v)$) and we will denote this by $u \prec_\sigma v$ (resp. $u \succ_\sigma v$). A vertex v is an *end-vertex* of an ordering σ of G if $\sigma(v) = n$. A subset $D \subseteq V$ is called a *dominating set* of G if every vertex in V has a neighbour in D. If the set D forms a path in G it is called a *dominating path*. Two vertices s and t of G form a *dominating pair* if every path between them is dominating.

We consider a *partially ordered set* Π to be a pair (X, π) where X is the *ground set* and π is a reflexive, antisymmetric and transitive binary relation on X, also called a *partial order*. We also denote $(x, y) \in \pi$ by $x \prec_\pi y$. For a partially ordered set (X, π) we call $x \in X$ an *immediate predecessor* of $y \in X$ if $x \prec_\pi y$ and there is no element $z \in X$ such that $x \prec_\pi z$ and $z \prec_\pi y$. For a binary relation π' on X we say that the *reflexive and transitive closure* of π' is the smallest binary relation $\pi' \subseteq \pi$ that is reflexive and transitive.

For further terms related to graphs and graph classes, we refer the reader to [22, 48, 75, 133, 147].

0.2 Some $\mathcal{NP}$-Complete Graph Problems

In the following we present some common $\mathcal{NP}$-complete graph problems that are studied in this work.

Maximum Independent Set and Maximum Clique

A *clique* in a graph G is a set of pairwise adjacent vertices of G. An *independent set*, on the other hand, is a set of pairwise nonadjacent vertices.

Maximum Clique

Instance:	An undirected graph $G = (V, E)$.
Task:	Find a clique C of maximum size.

Maximum Independent Set

Instance:	An undirected graph $G = (V, E)$.
Task:	Find an independent set I of maximum size.

Graph Colouring

A *proper vertex colouring* of $G = (V, E)$ is a function $f : V \to \mathbb{N}$ with $f(v) \neq f(w)$ for all $vw \in E$. The value $f(v)$ for $v \in V$ is called the *colour* of v. A vertex colouring is said to be *optimal* if it uses the least amount of colours.

Graph Colouring

Instance:	An undirected graph $G = (V, E)$.
Task:	Find an optimal colouring of the vertices of G.

Domination

A vertex in a graph is said to be *universal* if it is adjacent to every other vertex, and *non-universal* otherwise. A subset $D \subseteq V$ is called a *dominating set* of G if every vertex in $V \setminus D$ has a neighbour in D. If the set D forms a path in G it is called a *dominating path*. Two vertices s and t of G form a *dominating pair* if every path between them is dominating.

Dominating Set

Instance:	An undirected graph $G = (V, E)$.
Task:	Find a set $S \subseteq V$ of minimum size such that every $v \in V \setminus S$ has a neighbour in S.

Hamiltonian Path and Cycle

Hamiltonian Path and Hamiltonian Cycle are two of the most famous and oldest problems in graph theory. A *hamiltonian cycle* of G is a cycle containing all vertices of G. If G contains a hamiltonian cycle it is called *hamiltonian*.

HAMILTONIAN CYCLE

Instance:	An undirected graph $G = (V, E)$.
Task:	Find a hamiltonian cycle C.

A *hamiltonian path* of G is a cycle containing all vertices of G. If G contains a hamiltonian path, it is called *traceable*.

HAMILTONIAN PATH

Instance:	An undirected Graph $G = (V, E)$.
Task:	Find a hamiltonian path P.

Minimum Fill-In

Following Heggernes [87], graph $H = (V, E \cup F)$ is called a *triangulation* of $G = (V, E)$ if H is chordal and we say that it is a *minimal triangulation* if for every proper subset F' of F, the graph $(V, E \cup F')$ is not chordal. An elimination ordering σ of the vertices of G is a vertex ordering given as input for the *Elimination Game*, as defined in Algorithm 1, to compute a triangulation of G called G^+_σ. If G^+_σ is a minimal triangulation, we call σ a *minimal elimination ordering*. If G^+_σ is equal to G, then σ is a *perfect elimination ordering* and G is chordal by definition. The *deficiency* of a vertex v is defined as the set $D_G(v) = \{uw \notin E \ : \ u, w \in N_G(v)\}$. A cardinality-wise smallest triangulation is called the *minimum fill-in*.

Algorithm 1: Elimination Game algorithm

```
Input: A graph $G = (V, E)$ and an ordering $\sigma = (v_1, \dots, v_n)$.
Output: The filled graph $G^+_\sigma$.
1 $G^0 = G$;
2 for $i = 1$ to $n$ do
3     Let $F^i = D_{G^{i-1}}(v)$;
4     Obtain $G^i$ by adding the edges in $F^i$ to $G^{i-1}$ and removing $v_i$;
5 $G^+_\sigma = (V, E \cup \bigcup_{i=1}^n F^i)$
```

MINIMUM FILL-IN

Instance:	An undirected graph $G = (V, E)$.
Task:	Find a set $F \subset E$ of minimum size such that $G + F$ is chordal.

0.3 Graph Classes

The problems discussed in the previous section have many applications in such diverse areas as computer science, biology, logistics, traffic engineering and scheduling. Many real world (optimisation-) problems can essentially be reduced to such graph problems. Therefore, it is of significant interest to find correct and efficient algorithms to solve them.

Typically, these problems can be separated into two classes: problems for which an efficient algorithm can be found (these problems are said to be in $\mathcal{P}$) and problems for which it is conjectured that no efficient algorithm can be found (these problems are said to be $\mathcal{NP}$-hard).

For the latter class, it is necessary to find different approaches to algorithm design. These can be roughly separated into three types. We could drop our notion of efficiency and accept algorithms that do not run in strictly polynomial time. There are different approaches to this, ranging from the field of *exact algorithms* [60] to *parametrised complexity* [50]. Another possibility arises if we accept solutions that are not necessarily optimal, but approximate an optimal solution. The resulting *approximation algorithms* [145] are polynomial and yield good results for some of the mentioned problems.

However, if our application demands fast computation times and additionally there is no margin of error allowed, neither of these two approaches is viable. In this case, it becomes important to exploit the structure of the given input to give time-efficient and optimal algorithms. One way to achieve this is by restricting the algorithm to a *subclass of graphs*, i.e., a family of graphs with additional structural properties.

This approach has proven to be very effective for such subclasses as interval graphs, permutation graphs, comparability graphs and many others. Many applications mentioned above deliver graph problems on just these graph classes and research in this area has been fruitful. In the following we will give a short overview of the graph classes that will be addressed most frequently in this text.

We will give some of the most important characterisations for each class and state the complexity of their recognition, as well as the complexity of some optimisation problems. As will be made clear later on in this text, we are particularly interested in characterisations through linear vertex orderings.

Many of these classes are introduced more thoroughly by Golumbic [75]. Furthermore, a comprehensive overview of the field is given by Brandstädt et al. [22].

Perfect Graphs

A graph for which every induced subgraph H has the property that the clique number $\omega(H)$ coincides with its colouring number $\chi(H)$ is said to be *perfect*. Perfect graphs were introduced by Claude Berge in the 1960s, after which a host of interesting results on their structure were discovered. The most famous of these is the Strong Perfect Graph Theorem, which was conjectured in 1962 by Berge [11] and finally proven in 2006 by Chudnovsky et al. [33].

Theorem 0.3.1 (Chudnovsky et al. [33]). *(Strong Perfect Graph Theorem) An undirected graph is perfect if and only if it does not contain an induced subgraph isomorphic to C_{2k+1} or $\overline{C_{2k+1}}$.*

Many other important graph classes are perfect. In fact, the concept was motivated by a result from Gallai [66] which proved that the complement of a bipartite graph is perfect. Chudnovsky et al. [32] showed that perfect graphs can be recognized in polynomial time. However, their algorithm has a time complexity of $\mathcal{O}(n^9)$. The weighted independent set, weighted clique and colouring problems can be solved in polynomial time using linear programming [78].

Chordal Graphs

Chordal graphs were introduced by Hajnal and Surányi [82] in 1958 and have been studied extensively ever since. In essence, they form a natural generalisation of trees, sharing many similar properties. Chordal graphs form a kind of archetype among graph classes and the many ways in which they can be characterised will serve as a template for desired properties in other classes.

Definition 0.3.2. *A graph $G = (V, E)$ is* chordal *if each cycle in G of length at least 4 has at least one chord.*

One of the oldest characterisations of chordal graphs can be stated using separators.

Theorem 0.3.3 (Dirac [49]). *A graph $G = (V, E)$ is chordal if and only if every minimal cutset in every induced subgraph of G is a clique.*

As will be the case for many of the graph classes studied here, the most fruitful characterisation, from an algorithmic point of view, is given using a linear order of the vertices.

Definition 0.3.4. *Let $G = (V, E)$ be a graph. A vertex $v \in V$ is said to be* simplicial *in G if $N(v)$ is a clique in G. The ordering $(v_1, \dots, v_n)$ of vertices of V is a* perfect elimination ordering (PEO) *of G if for all $i \in \{1, \dots, n\}$, the vertex v_i is simplicial in $G_i = G[v_1, \dots, v_i]$.*

Note that in many texts perfect elimination orderings are defined as exactly the reverse of these defined here. However, for our purposes this definition of the ordering will be much more convenient.

Theorem 0.3.5 (Dirac [49], Fulkerson and Gross [64], Rose [123]). *A graph is chordal if and only if it has a perfect elimination ordering.*

By computing a perfect elimination ordering Tarjan and Yannakakis [136] showed that chordal graphs can be recognized in linear time.

Many of the classical problems such as maximum clique [70] can be solved in polynomial time on chordal graphs, and some problems such as maximum independent set [69, 126] and graph colouring even admit a linear time algorithm [75]. Most of these algorithms use a perfect elimination ordering and solve the problem in a greedy manner.

If neither G nor its complement contains an induced cycle of length 5 or more, then G is said to be *weakly chordal*. A *two-pair* in a graph is a pair of non-adjacent vertices such that every induced path between the two vertices has exactly two edges. We use the following fact about weakly chordal graphs:

Lemma 0.3.6 (Spinrad and Sritharan [132]). *Let $G = (V, E)$ be a graph with a two-pair $\{x, y\}$. Then G is weakly chordal if and only if $G + xy$ is weakly chordal.*

Similarly, the deletion of some particular vertices does not destroy the property of being weakly chordal.

Lemma 0.3.7. *Let $G = (V, E)$ be a graph and $v \in V$ such that v is simplicial or adjacent to at least $n - 2$ vertices of V. Then G is weakly chordal if and only if $G - v$ is weakly chordal.*

Proof. If v is simplicial, then it cannot be part of an induced cycle of G of size ≥ 4. Suppose that v is part of an induced cycle of size ≥ 5 in $\overline{G}$. Then there is an edge uw in this cycle, such that $vu, vw \notin E(\overline{G})$; a contradiction to v being simplicial.

Suppose that v has at least $n - 2$ neighbours in G. Then v has only one neighbour in $\overline{G}$ and, thus, cannot be part of an induced cycle. Suppose v is part of an induced cycle of size ≥ 5 in G. Then v must be non-adjacent to at least two vertices; a contradiction. □

A *split graph* G is a graph whose vertex set can be divided into sets C and I such that C is a clique in G and I is an independent set in G. It is easy to see, that every split graph is chordal, whereas every chordal graph is also weakly chordal. Furthermore, a graph is split if and only if both the graph itself and its complement are chordal.

Comparability Graphs

The class of comparability graphs comprises those graphs which correspond to some finite partial order

Definition 0.3.8. *Let $P = (V, \preceq)$ be a finite partially ordered set. Then $G_P = (V, E_P)$ with $xy \in E_P$ if $x \prec y$ or $y \prec x$ is the* comparability graph *of the partially ordered set P. A graph $G = (V, E)$ is a* comparability graph *if there is a partially ordered set P such that G_P is isomorphic to G. Furthermore, a graph is a* cocomparability graph *if $\overline{G}$ is a comparability graph.*

Similarly, comparability graphs can also be defined through an orientation of their edges which reveals the partial ordering beneath.

Definition 0.3.9. *An orientation $D = (V, A)$ of a given undirected graph $G = (V, E)$ is said to be* transitive *if for each pair of arcs $xy, yz \in A$ it holds that the arc xz is also contained in A.*

Obviously, a graph is a comparability graph if and only if it has an acyclic transitive orientation, as such an orientation directly conveys the needed partial ordering. However, the following stronger characterisation also holds.

Theorem 0.3.10 (Ghouila-Houri [71]). *A graph is a comparability graph if and only if it has a transitive orientation.*

This can be simplified by only analysing the odd cycles of a graph.

Theorem 0.3.11 (Gilmore and Hoffman [72], Ghouila-Houri [71]). *A graph $G = (V, E)$ is a comparability graph if and only if there is no sequence $(x_1, x_2, x_3, \ldots, x_{2n+1})$ of (not necessarily distinct) vertices from V with $n \geq 2$ such that $x_i x_{i+1} \in E$ and $x_i x_{i+2} \notin E$ (cyclically).*

The complements of comparability graphs have even stronger algorithmic properties. This is mainly due to a useful characterisation through a linear vertex order. This ordering can be immediately obtained through a linear extension of the partial ordering underlying the complement graph.

Definition 0.3.12. *Let $G = (V, E)$ be a graph and let $\sigma = (v_1, \ldots, v_n)$ be a linear ordering of its vertices. The ordering σ is said to be a* cocomparability order *if for any $i, j, k \in \{1, \ldots, n\}$ with $i < j < k$ we have that $v_i v_k \in E$ implies $v_i v_j \in E$ or $v_j v_k \in E$.*

Theorem 0.3.13 (Kratsch and Stewart [104]). *A graph $G = (V, E)$ is a cocomparability graph if and only if it has a cocomparability ordering.*

Comparability and cocomparability graphs can be recognized in polynomial time [74]. Furthermore, many optimisation problems can be solved efficiently, especially on cocomparability graphs. These comprise colouring [74], domination [104] and minimum path cover [45].

Interval Graphs

Many graph classes are defined as so-called *intersection graphs*.

Definition 0.3.14. *For a given family of sets $\mathcal{M}$, the* intersection graph *$G_{\mathcal{M}}$ of these sets has $\mathcal{M}$ as its vertex set, and two sets are adjacent in $G_{\mathcal{M}}$ if the intersection of the corresponding sets is not empty. The family $\mathcal{M}$ is called the* model *of G.*

Most of these classes have a geometric background, i.e., as the intersections of various geometric objects on a line, in the plane or in space. The most well known such class by far is the class of *interval graphs*.

Definition 0.3.15. *Let $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ be a finite collection of closed intervals of the real line and let $G_{\mathcal{I}}$ be its intersection graph. Then G is said to be an* interval graph *if G is the intersection graph $G_{\mathcal{I}}$ of an interval model $\mathcal{I}$.*

Interval graphs were introduced independently by Hajós [83] in a combinatorial setting and by Benzer [10] in the context of genetics. As each interval can be interpreted as a different interval in time, these graphs can be used to check for conflicts in the scheduling of timetables and similar applications.

There are many interesting characterisations of interval graphs and the following theorem shows two examples.

Theorem 0.3.16 (Gilmore and Hoffman [72]). *Let $G = (V, E)$ be a graph. The following conditions are equivalent:*

1. *G is an interval graph.*
2. *G contains no induced C_4 and $\overline{G}$ is transitively orientable;*
3. *The maximal cliques of G can be linearly ordered such that for each vertex $v \in V$, the maximal cliques containing v occur consecutively.*

Interval graphs can also be defined through a set of forbidden induced subgraphs given by Lekkerkerker and Boland [109]. For algorithmic purposes a characterisation using a linear vertex ordering is particularly useful.

Definition 0.3.17. *Let $G = (V, E)$ be a graph and let $\sigma = (v_1, \dots, v_n)$ be a linear ordering of its vertices. The ordering σ is an* interval order *if for any $i, j, k \in \{1, \dots, n\}$ with $i < j < k$ we have that $v_iv_k \in E$ implies $v_iv_j \in E$.*

Theorem 0.3.18 (Olariu [118]). *A graph is an interval graph if and only if its vertex set admits an interval order.*

In the following we will introduce some variants of interval graphs, as there are many different intuitive possibilities to both generalise and strengthen this concept.

Definition 0.3.19. *A graph G is a* proper interval graph *if G is an interval graph with an interval model where no two intervals $I_x, I_y \in \mathcal{I}$ contain each other.*

Definition 0.3.20. *A graph G is a* unit interval graph *if G is an interval graph with an interval model of unit-length intervals.*

Theorem 0.3.21 (Roberts [122]). *The graph $G = (V, E)$ is a proper interval graph if and only if G is a unit-interval graph.*

These graphs can also be characterised using a linear vertex ordering.

Definition 0.3.22. *Let $G = (V, E)$ be a graph and let $\sigma = (v_1, \dots, v_n)$ be a linear ordering of its vertices. The ordering σ is a* unit interval order *if for any $i, j, k \in \{1, \dots, n\}$ with $i < j < k$ we have that $v_iv_k \in E$ implies $v_iv_j \in E$ and $v_jv_k \in E$.*

Theorem 0.3.23 (Looges and Olariu [110]). *A graph is a unit interval graph if and only if its vertex set admits a unit interval order.*

An important generalisation of the interval graph that we will encounter in this text is defined as the intersection graph of arcs on a circle.

Definition 0.3.24. *A graph G is a* circular-arc graph *if there is a finite collection of arcs on a circle such that G is the intersection graph of this model.*

Research on circular-arc graphs goes back as far as the 1960's where Hadwiger et al. [80] and Klee [96] made first observations on these graphs, culminating in a series of publications by Tucker [138, 139, 140, 141, 142].

Circular-arc graphs can be recognised in linear time [113] and many optimisation problems such as domination [29], maximum clique [16] and maximum independent set [92] can be solved in polynomial or even linear time.

AT-free Graphs

In a classical paper of algorithmic graph theory from the early 1960s by Lekkerkerker and Boland [109], the authors used a forbidden substructure called an *asteroidal triple* to characterise interval graphs. An *asteroidal triple* is an independent triple of vertices, such that for any two of them there is a path that avoids the third.

Definition 0.3.25. *An* asteroidal triple *(AT) of a given graph $G = (V, E)$ is a set of three independent vertices such that there is a path between each pair of these vertices that does not contain any vertex of the neighbourhood of the third. A graph is called* asteroidal triple free *or* AT-free *if G does not contain an asteroidal triple. Furthermore, a graph is called* coAT-free *if $\overline{G}$ is AT-free.*

A vertex z such that there is no pair of vertices a and b, for which there is an induced a-z-path that avoids b and an induced b-z-path that avoids a, is said to be *admissible*.

Theorem 0.3.26 (Lekkerkerker and Boland [109])**.** *A graph G is an interval graph if and only if it is chordal and AT-free.*

This characterization gave rise to the introduction of the class of asteroidal triple free graphs (AT-free graphs) and due to the fact that these graphs form a superclass of both the interval and cocomparability graphs, there has been considerable research interest for the last two decades.

We say that two vertices $u, v \in V$ of a given graph $G = (V, E)$ form a *dominating pair* of G if every path connecting u and v is a dominating set of G.

Theorem 0.3.27 (Corneil et al. [39])**.** *Every connected AT-free graph has a dominating pair.*

With the help of the concept of dominating pairs we can derive another interesting characterisation of AT-free graphs. To this end, we say that a connected graph G with a dominating pair satisfies the *spine property* if for every nonadjacent dominating pair (a, b) in G there exists a neighbour a' of a such that (a', b) is a dominating pair of the connected component of $H \setminus \{a\}$ containing b.

Theorem 0.3.28. *A graph $G = (V, E)$ is AT-free if and only if every connected induced subgraph of G satisfies the spine property.*

Kratsch and Spinrad [103] and Köhler [97] showed that AT-free graphs can be recognised in $\mathcal{O}(n^{2.82})$ or $\mathcal{O}(nm)$ by using fast matrix multiplication. In fact, Kratsch and Spinrad [103] also showed that the complexity of recognition is at least as hard as recognizing triangles in a graph (see also [133]).

0.4 Graph Searching

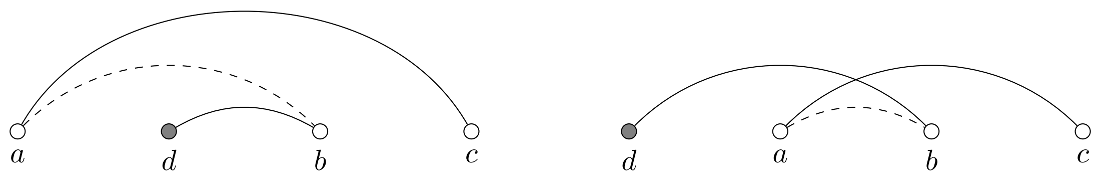
Graph search is one of the oldest and most fundamental algorithmic concepts in both graph theory and computer science. In essence, it is the systematic exploration of the vertices in a graph, beginning at a chosen vertex and visiting every other vertex of the graph such that a vertex is visited only if it is adjacent to a previously visited vertex.

Such a general definition of a graph search leaves much freedom for a selection rule determining which node is chosen next. By restricting this choice with specific rules, various different graph searches can be defined. Arguably the earliest examples are the *Breadth First Search* (BFS) and the *Depth First Search* (DFS), which were referred to in the context of maze traversals in the 19th century [58].

Graph searches such as *Breadth First Search* (BFS) and *Depth First Search* (DFS) are, in the most general sense, mechanisms for systematically visiting all vertices of a graph. Using such simple procedures, a variety of important graph properties can be tested, many optimisation problems can be simplified and they are sub-routines in many more algorithms. For example, BFS is an elementary component of several graph algorithms, such as finding connected components, testing for bipartiteness, computing shortest paths with respect to the number of edges, or the Edmonds-Karp algorithm for computing the maximum flow in a network [57]. Similarly, DFS is the basis of algorithms for finding biconnected components in undirected graphs [90], strongly connected components in directed graphs [135], topological orderings of directed acyclic graphs [137], planarity testing [91], or solving mazes [58].

We focus on connected searches, that is, a graph search or graph traversal that starts at a vertex and explores the graph by visiting a vertex in the neighbourhood of the already visited vertices. If no further restriction is given, we call such a search a *generic search*. The search paradigms of BFS and DFS can be simply characterized by using a queue or a stack as the data structure for the unvisited vertices in the current neighbourhood. However, there are more sophisticated searches such as *Lexicographic Breadth First Search* (LBFS) [126] and *Lexicographic Depth First Search* (LDFS) [37]. Furthermore, we also consider *Maximum Cardinality Search* (MCS) [136] and *Maximal Neighbourhood Search* (MNS) [37].

In the following, we will give an overview of some of the most popular graph searching algorithms. In each case we will give an algorithmic characterisation, as well as a characterisation using the discovery order of the visited vertices. A thorough overview of this topic can be found in [37].



(S): Given an ordering σ of V, if $a \prec_\sigma b \prec_\sigma c$ and $ac \in E$ and $ab \notin E$, then there exists a vertex $d \prec_\sigma b$ such that $db \in E$.

Generic Search

The most general definition of a connected graph search is known as *generic search.* Here, the search algorithm is only restricted at any step to visit some neighbour of the already visited vertices. This search can be used to check whether a graph is connected in linear time.

Algorithm 2: Generic Search

```
  Input: Connected graph $G$ and a distinguished vertex $s \in V$
  Output: A vertex ordering $\sigma$
1 begin
2     $S \leftarrow \{s\}$;
3     for $i \leftarrow 1$ to $n$ do
4         Choose and remove an unnumbered vertex $v$ from $S$;
5         $\sigma(i) \leftarrow v$;
6         foreach unnumbered vertex $w$ adjacent to $v$ do add $w$ to $S$;
```

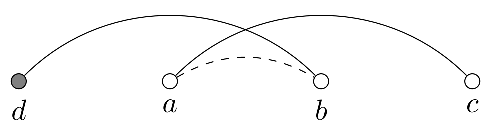
The following condition characterises when a given linear order of the vertices of a graph is the possible output of a generic search.

Theorem 0.4.1 (Corneil and Krueger [37])**.** *For an arbitrary graph $G = (V, E)$, an ordering σ of V is a search ordering of G of a generic search if and only if σ has property (S).*

Breadth First Search

The *Breadth First Search* algorithm, also known as *BFS*, can be derived from generic search by using the data structure of a *queue*. Every time a new vertex is visited, we scan its neighbourhood and consecutively add its neighbours to the end of the queue. We then proceed to visit the vertex at the beginning of the queue.

Due to the nature of this search, it can be used to compute a rooted tree in which any path from the root to some other vertex is shortest, in the sense that it uses the least number of edges. Therefore, this algorithm can be used to find shortest paths in



(B): Given an ordering σ of V, if $a \prec_\sigma b \prec_\sigma c$ and $ac \in E$ and $ab \notin E$, then there exists a vertex $d \prec_\sigma a$ such that $db \in E$.

graphs where all edges have equal (positive) weight and can be generalised to Dijkstra's algorithm. Apart from this, it is used as a subroutine in countless other algorithms.

Algorithm 3: BFS

```
Input: Connected graph $G$ and a distinguished vertex $s \in V$
Output: A vertex ordering $\sigma$
begin
    initialize a queue $Q$ with $Q \leftarrow \{s\}$;
    for $i \leftarrow 1$ to $n$ do
        dequeue $v$ from beginning of $Q$;
        $\sigma(i) \leftarrow v$;
        foreach unnumbered vertex $w$ adjacent to $v$ do
            if $w \notin Q$ then
                enqueue $w$ to end of $Q$;
```

The following condition characterises when a given linear order of the vertices of a graph is the possible output of a BFS.

Theorem 0.4.2 (Corneil and Krueger [37]). *For an arbitrary graph $G = (V, E)$, an ordering σ of V is a BFS ordering of G if and only if σ has property (B).*

Depth First Search

Depth First Search, also known as DFS, is the other classical graph search method. It can be used to test for connectivity, to compute topological orderings and is used in many other algorithms.

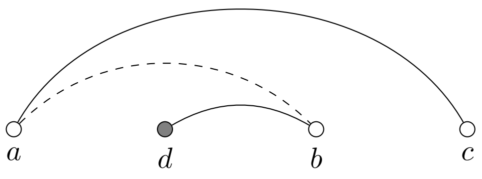
Whereas BFS uses a queue to choose new vertices, the defining data structure for DFS is the *stack*. Every time a new vertex is visited, we scan its neighbourhood and consecutively add its neighbours to the top of the stack and then proceed to visit the vertex at the top of the stack.

In a series of publications, Hopcroft and Tarjan discovered many properties of DFS [90, 135] and found several interesting applications; for example, the testing of planarity [91].

The following condition characterises when a given linear order of the vertices of a graph is the possible output of a DFS.

Algorithm 4: DFS

```
   Input: Connected graph $G$ and a distinguished vertex $s \in V$
   Output: A vertex ordering $\sigma$
1  begin
2      initialize a stack $S$ with $S \leftarrow \{s\}$;
3      for $i \leftarrow 1$ to $n$ do
4          pop $v$ from the top of $S$;
5          $\sigma(i) \leftarrow v$;
6          foreach unnumbered vertex $w$ adjacent to $v$ do
7              if $w \in S$ then
8                  remove $w$ from $S$;
9              push $w$ on top of stack $S$;
```



(D): Given an ordering σ of V, if $a \prec_\sigma b \prec_\sigma c$ and $ac \in E$ and $ab \notin E$, then there exists a vertex d with $a \prec_\sigma d \prec_\sigma b$ such that $db \in E$.

Theorem 0.4.3 (Corneil and Krueger [37]). *For an arbitrary graph $G = (V, E)$, an ordering σ of V is a DFS ordering of G if and only if σ has property (D).*

Lexicographic Breadth First Search

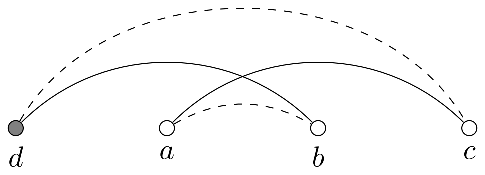
In 1976, Rose, Tarjan and Lueker defined a linear time algorithm (Lex-P) which computes a perfect elimination ordering, if any exists and thus, forms a recognition algorithm for chordal graphs [126]. This algorithm, since named *Lexicographic Breadth First Search*, or LBFS, exhibits many interesting structural properties and has been used as an ingredient in many other recognition and optimisation algorithms. A thorough survey on LBFS can be found in [43].

If two vertices have the same label in step 5, we say that they are *tied*. We call a set of tied vertices S encountered in step 5 of Algorithm 5 a *slice*. Given an LBFS order τ and two vertices u and v with $u \prec_\tau v$, we denote the vertex-minimal slice with respect to τ containing u and v as $\Gamma^\tau_{u,v}$.

Lemma 0.4.4 (Corneil et al. [43]). *Let τ be an arbitrary LBFS of a graph G and let $u, v \in V$ with $u \prec_\tau v$. Let w be the τ-first vertex of the connected component C_u of $\Gamma^\tau_{u,v}$ containing u. There exists a w-u-path in $\Gamma^\tau_{u,v}$ all of whose vertices, with the possible*

Algorithm 5: LBFS

```
   Input: Connected graph $G = (V, E)$ and a distinguished vertex $s \in V$
   Output: A vertex ordering $\sigma$
1  begin
2      $label(s) \leftarrow \{n + 1\}$;
3      foreach vertex $v \in V \setminus \{s\}$ do assign to $v$ the empty label;
4      for $i \leftarrow 1$ to $n$ do
5          pick an unnumbered vertex $v$ with lexicographically largest label;
6          $\sigma(i) \leftarrow v$;
7          foreach unnumbered vertex $w \in N(v)$ do append $(n - i)$ to $label(w)$;
```



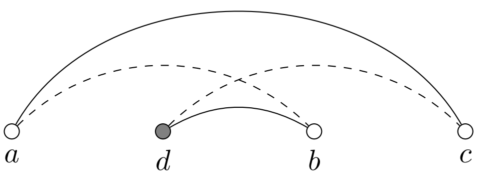
(LB): Given an ordering σ of V, if $a \prec_\sigma b \prec_\sigma c$ and $ac \in E$ and $ab \notin E$, then there exists a vertex $d \prec_\sigma a$ such that $db \in E$ and $dc \notin E$.

exception of u, are not adjacent to v. Moreover, all vertices on this path, other than u, occur before u in τ. Such a path is called a prior path.

The following condition characterises when a given linear order of the vertices of a graph is the possible output of an LBFS.

Theorem 0.4.5 (Brandstädt et al. [21], Corneil and Krueger [37], Golumbic [75])**.** *For an arbitrary graph $G = (V, E)$, an ordering σ of V is an LBFS ordering of G if and only if σ has property (LB).*

There is a simple implementation of LBFS in linear time [79] using the technique of *partition refinement*. This is defined as follows. Given a set S, we call $\mathcal{Q} = (Q_1, Q_2, \dots, Q_k)$ a partition of S if for all Q_i, Q_j with $i \neq j$ we have $Q_i \cap Q_j = \emptyset$ and $\bigcup_{i=1}^{k} Q_i = S$. We say that a set $T \subseteq S$ refines $\mathcal{Q}$ if every partition class $Q_i \in \mathcal{Q}$ is replaced with subpartition classes $A_i = Q_i \cap T$ and $B_i = Q_i \setminus A_i$. The partition refinement scheme for LBFS functions as follows. Beginning with the partition $\mathcal{Q} = (V)$, select a vertex s and refine $\mathcal{Q}$ with $N(s)$ by placing $A = V \cap N(s)$ before $B = V \setminus A$. The vertex whose neighbourhood is used to refine the partition classes is called a *pivot*. Choose the next pivot v among the vertices of the first set in $\mathcal{Q}$ and use $N(v)$ to refine, maintaining the order of the partition classes created so far, i.e., $(A \cap N(v), A \setminus N(v), B \cap N(v), B \setminus N(v))$. This process is repeated until all partition classes have been refined.



(LD): Given an ordering σ of V, if $a \prec_\sigma b \prec_\sigma c$ and $ac \in E$ and $ab \notin E$, then there exists a vertex d with $a \prec_\sigma d \prec_\sigma b$ such that $db \in E$ and $dc \notin E$.

Lexicographic Depth First Search

In [37], Corneil and Krueger defined *Lexicographic Depth First Search* as a lexicographic analogue to DFS. Since then, it has been used for many applications, most notably to solve the minimum path cover problem on cocomparability graphs [45].

Algorithm 6: LDFS

```
Input: Connected graph $G = (V, E)$ and a distinguished vertex $s \in V$
Output: A vertex ordering $\sigma$
1 begin
2     $label(s) \leftarrow \{0\}$;
3     foreach vertex $v \in V \setminus \{s\}$ do assign to $v$ the empty label;
4     for $i \leftarrow 1$ to $n$ do
5         pick an unnumbered vertex $v$ with lexicographically largest label;
6         $\sigma(i) \leftarrow v$;
7         foreach unnumbered vertex $w \in N(v)$ do prepend $i$ to $label(w)$;
```

The following condition characterises when a given linear order of the vertices of a graph is the possible output of an LDFS.

Theorem 0.4.6 (Corneil and Krueger [37]). *For an arbitrary graph $G = (V, E)$, an ordering σ of V is a LDFS ordering of G if and only if σ has property (LD).*

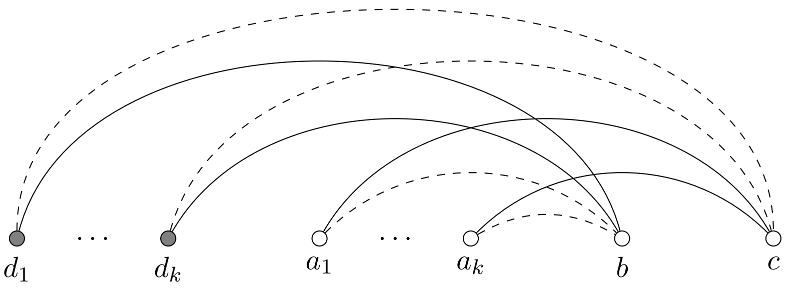
Maximum Cardinality Search

Maximum Cardinality Search (MCS) was introduced in 1984 by Tarjan and Yannakakis [136] as a simple alternative to LBFS for recognising chordal graphs. They noticed that, instead of remembering the *order* in which previous neighbours of a vertex had appeared, it sufficed to just store the *number* of previously visited neighbours for each vertex. This observation resulted in an algorithm which has a linear running time and an easy implementation.

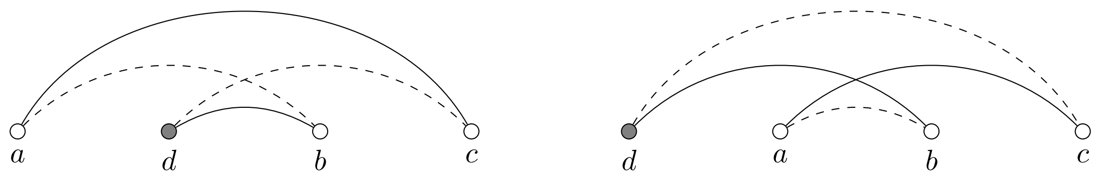
Theorem 0.4.7 (Brandstädt et al. [21]). *For an arbitrary graph $G = (V, E)$, an ordering σ of V is an MCS ordering of G if and only if σ has property (MC).*

Algorithm 7: Maximum Cardinality Search

```
Input: Connected graph $G = (V, E)$ and a distinguished vertex $s \in V$
Output: A vertex ordering $\sigma$
1 begin
2     $\sigma(1) \leftarrow s$;
3     for $i \leftarrow 2$ to $n$ do
4         pick an unnumbered vertex $v$ with largest amount of numbered
            neighbours;
5         $\sigma(i) \leftarrow v$;
```



(MC): Given an ordering σ of V, if $a_1, \dots, a_k \prec_\sigma b$ and $a_i c \in E$ and $a_i b \notin E$ for all $i \in \{1, \dots, k\}$, then there exist vertices $d_1, \dots, d_k$ with $d_1, \dots, d_k \prec_\sigma b$ such that $d_i b \in E$ and $d_i c \notin E$ for all $i \in \{d_1, \dots, d_k\}$.



(MN): Given an ordering σ of V, if $a \prec_\sigma b \prec_\sigma c$ and $ac \in E$ and $ab \notin E$, then there exists a vertex d with $d \prec_\sigma b$ and $db \in E$ and $dc \notin E$.

Maximal Neighbourhood Search

Maximal Neighbourhood Search (MNS) was introduced by Corneil and Krueger [37] in 2008 as a generalisation of LBFS, LDFS and MCS. Instead of using strings (like LBFS and LDFS) or integers (like MCS), the algorithm uses sets of integers as labels and the maximal labels are those sets which are inclusion maximal. Unlike the labels of LBFS, LDFS and MCS, the labels of MNS are not totally ordered and there can be many different maximal labels. Corneil and Krueger showed that every search ordering of LBFS, LDFS and MCS is also an MNS ordering. This result was generalized in 2009 by Berry et al. [12] who showed that the set of MNS orderings is equal to the set of orderings of *Maximum Label Search*.

Given the 3-point conditions for LBFS and LDFS, it is intriguing to ask what happens if we relax the position of d to be simply *before* b. This property, defined as (MN), obviously holds for LBFS and LDFS, but also for Maximum Cardinality Search which will be the subject of the next section. This last fact was first observed by Tarjan and Yannakakis [136] and they managed to prove that any search with this property returns a perfect elimination ordering. However, they did not state a proper search paradigm which is implied by (MN).

In [37] the authors used this property to define a proper search algorithm, stated in Algorithm 8, which they called *Maximal Neighbourhood Search* and proved the following:

Theorem 0.4.8 (Corneil and Krueger [37])**.** *For an arbitrary graph $G = (V, E)$, an ordering σ of V is an MNS ordering of G if and only if σ has property (MN).*

Algorithm 8: Maximal Neighbourhood Search

```
Input: Connected graph $G = (V, E)$ and a distinguished vertex $s \in V$
Output: A vertex ordering $\sigma$
1 begin
2     $label(s) \leftarrow \{n + 1\}$;
3     foreach vertex $v \in V \setminus \{s\}$ do assign to $v$ the empty label;
4     for $i \leftarrow 1$ to $n$ do
5         pick an unnumbered vertex $v$ with maximal label under set inclusion;
6         $\sigma(i) \leftarrow v$;
7         foreach unnumbered vertex $w$ adjacent to $v$ do add $i$ to $label(w)$;
```

Multisweep Searches

A technique that has proven to be very fruitful in recent years is that of “multisweeping”. This describes the multiple application of some graph searches, where each run of the search uses the ordering given by the previous application as a so-called “tie-break” rule, that is, a priority list which decides which vertex can be visited next in those cases where the given search paradigm allows several different options.

Algorithm 9: Multisweep Search

```
Input: A search paradigm $\mathcal{P}$, connected graph $G = (V, E)$ and a linear order $\sigma$
       of the vertices
Output: A vertex ordering $\tau$
1 begin
2     for $i \leftarrow 1$ to $n$ do
3         pick an unnumbered vertex $v$ that is rightmost in $\sigma$ among the feasible
            vertices w.r.t. the search paradigm $\mathcal{P}$;
4         $\tau(i) \leftarrow v$;
```

Many recognition and optimisation algorithms use this form of search either as the main algorithm or as a subroutine. Some examples are the following. For AT-free graphs LBFS provides a linear time algorithm for finding dominating pairs in connected asteroidal triple-free graphs, where a dominating pair is a pair of vertices such that every path connecting them is a dominating set in the graph [40]. The first vertex x is simply the end-vertex of an arbitrary LBFS and the second vertex y is the end-vertex of an LBFS starting in x. Moreover, one can use five LBFS executions followed by a modified LBFS to recognize interval graphs [43]. For unit-interval graphs it is even enough to use three applications of LBFS, as was shown by Corneil [36]. Crescenzi et al. [46] have shown that the diameter of huge real world graphs can usually be found with only a few BFS executions. Furthermore, it was shown that (L)BFS can be used to approximate the diameter of graphs in the classes of chordal graphs [52], HHD-free graphs [51], k-chordal graphs [41] and hyperbolic graphs [31]. In a recent result, Dusart and Habib [55] have even shown that applying LBFS to a cocomparability graph at most n times yields a cocomparability order, giving rise to the question of convergence of this operation. Sometimes it can even be useful to combine different types of searches in this manner, as can be seen in [45], where the authors use LDFS and other searches to compute a minimum path cover on cocomparability graphs.

0.5 Abstract Convexity in Finite Sets

Before we can begin to define a notion of convexity for graphs in Chapter 1, we need to introduce some elementary concepts and notation from *abstract convexity.* The field of abstract or generalised convexity was established in the first half of the 20th century in a variety of different settings and by many different researchers. The common aim of these diverse approaches was to axiomatise the manifold concepts and properties from geometry, which are known as convexity, in a form that resembles the modern (or abstract) algebra developed in the late 19th and early 20th century. This is a vast field and a proper introduction would go far beyond the scope of this text. Therefore, we will simply introduce the fundamental concepts and some of the basic notation, as well as short excursions to the subfields of interval convexity and convex geometry. A thorough overview of abstract convexity and interval convexity is given in the comprehensive (and slightly overwhelming) book by van de Vel [144]. This survey, however, also covers convexities over infinite and uncountable sets, making some discussions overcomplicated for our discrete setting. For the theory of convex geometries Edelman and Jamison [56] give an extensive survey, while the book on greedoids by Korte et al. [101] covers convex geometries under the dual structure of *anti-matroids*.

0.5.1 Abstract Convexity in Finite Sets

We begin by giving a very general definition of a convexity space.

Definition 0.5.1. *Let V be a finite set and let $\mathcal{C}$ be a collection of subsets of X with the properties:*

(i) $\emptyset \in \mathcal{C}$ *and* $V \in \mathcal{C}$

(ii) $A \in \mathcal{C}$ *and* $B \in \mathcal{C}$ *implies* $A \cap B \in \mathcal{C}$.

We call the pair $(V, \mathcal{C})$ a convexity space *on the ground set V. The elements of $\mathcal{C}$ are called* convex sets*. A convex set whose complement is also convex is called a* halfspace*. Given a convexity space $(V, \mathcal{C})$ we define the* convex hull *of $X \subset V$, namely* $\operatorname{conv}(X)$*, to be the smallest convex set containing X, or alternatively the intersection of all convex supersets of X.*

The convex hull of a set has many interesting properties.

Lemma 0.5.2 (Edelman and Jamison [56])**.** *For a convexity space $(V, \mathcal{C})$ the convex hull is a closure operator, i.e., it has the following properties:*

(i) $X \subseteq \operatorname{conv}(X)$,

(ii) $X \subseteq Y$ *implies* $\operatorname{conv}(X) \subseteq \operatorname{conv}(Y)$,

(iii) $\operatorname{conv}(\operatorname{conv}(X)) = \operatorname{conv}(X)$,

for $X, Y \subseteq V$.

One of the most important concepts from convexity in a graph theoretic setting is that of the *extreme point.*

Definition 0.5.3. *Given a convexity space $(V, \mathcal{C})$ we define a* basis *B of $X \subseteq V$ to be minimal with $B \subseteq X$ and* $\operatorname{conv}(B) = \operatorname{conv}(X)$*. We call a point $p \in X$* extreme point *of*

X, if $p \notin \operatorname{conv}(X - p)$ and the set of all extreme points of X is denoted by $\operatorname{ex}(X)$. *A* copoint *C attached at p is a maximal convex set in $X - p$.*

We can also define the intersection of convexities $(V, \mathcal{C}_1), \ldots, (V, \mathcal{C}_k)$ *over the same ground set to be $(V, \bigcap_{i=1}^{k} \mathcal{C}_i)$. Obviously, this construction is again a convexity space.*

The join *of a family of convexities $\mathcal{C}_1, \ldots, \mathcal{C}_k$ on a common ground set V is the convexity generated by $\bigcup_{i=1}^{k} \mathcal{C}_i$. This convexity consists of all sets of type $\bigcap_{i=1}^{k} C_i$, where $C_i \in \mathcal{C}_i$ for all $i \in \{1, \ldots k\}$.*

The following property describing the interaction between the hull- and the join-operation will be very important in the study of convex geometries.

Definition 0.5.4. *A convexity space $(V, \mathcal{C})$ is said to be* join-hull commutative *if for every non-empty convex set $C \in \mathcal{C}$ and every $a \in V$ it holds that* $\operatorname{conv}(C \cup \{a\}) = \bigcup_{c \in C} \operatorname{conv}(\{c, a\})$.

In an attempt to somehow quantify the complexity of a given convexity in a similar way as is done by the concept of dimension in linear algebra and geometry, many different types of measures have been proposed. We will define a variety of these here. However, we will mainly be interested in the *Carathéodory number.*

Definition 0.5.5. *Let $(V, \mathcal{C})$ be a convexity space. We say that $(V, \mathcal{C})$ has* Carathéodory number *d if d is the smallest positive integer, such that for every $X \subseteq V$ and every $p \in \operatorname{conv}(X)$ there is a subset $X' \subseteq X$ with $p \in \operatorname{conv}(X')$ and $|X'| \leq d$.*

We say that $(V, \mathcal{C})$ has Helly number *$h(\mathcal{C})$, if h is the smallest integer such that for any family $\mathcal{F}$ of order h that has pairwise intersecting elements, the intersection of all elements is not empty.*

The Radon number *r of $(V, \mathcal{C})$ is defined as the smallest integer such that every set $A \subseteq V$ with r elements has a* Radon partition*, i.e., a partition $A = A_1 \dot\cup A_2$ with* $\operatorname{conv}(A_1) \cap \operatorname{conv}(A_2) \neq \emptyset$.

All of these concepts are derived from the Carathéodory, Helly and Radon theorems, respectively, which describe properties of convex combinations in the space $\mathbb{R}^d$. For example, the Theorem of Carathéodory states that if a point $x \in \mathbb{R}^d$ lies in the convex hull of a set X, then x can be written as the convex combination of at most $d+1$ points in X [27, 134]. In abstract convexity and, in particular, in the case of graph convexity, we will usually not have a canonic notion of dimension of the ground set underlying a convexity in the way we have in the standard convexity of $\mathbb{R}^d$. In this setting it will sometimes make sense to use the Carathéodory number as an indicator of dimension in that space, in the sense that the structure underlying $(V, \mathcal{C})$ has dimension $c - 1$ if the convexity has Carathéodory number c. For this notion to be reasonable, however, the convexity $\mathcal{C}$ must somehow be *natural* with respect to the structure of the underlying set V.

As we will frequently use the Carathéodory number to analyse different convexities, the following result by Duchet [54] will be a very useful tool for its computation.

Definition 0.5.6. *Let $(V, \mathcal{C})$ be a convexity space and $X \subset V$. We say that X is* irredundant *if there is some element $x \in X$ such that x is not contained in the convex hull of any true subset of X. A set that is not irredundant will be called* redundant.

These irredundant sets are very closely linked to the Carathéodory number of a convexity space.

Lemma 0.5.7 (Duchet [54])**.** *In any convexity space, the Carathéodory number equals the maximum cardinality of an irredundant set.*

Convexity spaces, in the very general sense defined in this section, can be seen more as an umbrella term for more interesting special spaces. Unless furnished with more structural properties, very little can be shown or derived from these structures. To be able to achieve stronger properties that can be useful in a graph theoretic setting, we need to move to more specific structures, either by using more axioms, as we will do with the *anti-exchange property* in Section 0.5.3, or by restricting the ways in which these spaces are generated, as we will see in the following Section 0.5.2, where we study convexity spaces derived from *intervals*.

0.5.2 Interval Convexity

In this text we will mainly be dealing with convexities that will be defined using *intervals*. Such convexities have much stronger properties than the general spaces defined in the previous section. In particular, they will be a very natural concept in the setting of graph theory, where definitions of intervals using paths or walks are used to give some form of geometric intuition of the graphs at hand.

Definition 0.5.8. *Let V be a set and let $I : V \times V \to \mathcal{P}(V)$ be a function with the following properties:*

(i) $a, b \in I[a,b]$.

(ii) $I[a,b] = I[b,a]$.

We call I an interval operator *on V and $I[a,b]$ the* interval between a and b. *The pair (V, I) is called an* interval space. *The* extension *(or* ray*) at a away from b is the set $\{x \;:\; a \in I[b,x]\}$ and is denoted as a/b. Furthermore, we will denote the set $I[a,b] \setminus \{a,b\}$ as $I(a,b)$.*

If it is clear from the context, we will omit the name of the interval operator and, in analogy to intervals defined on the real line, write $I[a,b]$ as $[a,b]$. Throughout this text, we will mainly make use of *closed* intervals, as this sometimes makes notation of properties more compact (for example the statement of the Pasch and Peano axioms given later in this section). However, in Chapter 2 and in Section 1.5.1 we use open intervals, as this simplifies the notation in those cases. With some adjustments all results shown for closed intervals can be easily adapted for open intervals as well and vice versa.

This concept of an interval operator appears in many areas of mathematics under different names and sometimes in slight variations and forms a strong abstraction of conventional metric (or geodesic) intervals. For example in [34], the authors define something called a *strict betweenness*. This is, in essence, the same thing as an interval operator, only that interval ends must be distinct and are not contained in the intervals themselves. Where possible we have adapted these kinds of results to the same terminology.

In a similar manner to metric spaces, we can now use these intervals to define a convexity:

Definition 0.5.9. *Given an interval space (V, I), the* interval convexity *induced by I on V is defined as follows: A subset C of V is* interval convex *if $I[a, b] \subset C$ for all $a, b \in C$. In addition, we can give a different definition of an* extreme point *of such a convexity space by saying that p is an extreme point of X if there are no $a, b \in X$ such that $p \in I(a, b)$.*

Note that for technical reasons the definition of extreme points uses *open* intervals.

Interval convexities have many interesting properties and their analysis is usually more intuitive than the general convexities, as they have a strong relation to geometry. One such advantage is the following technical lemma by Duchet [54] which furnishes us with a strong tool to compute the Carathéodory number for interval convexities.

Lemma 0.5.10 (Duchet [54])**.** *In an interval convexity space, the Carathéodory number is the smallest integer c such that every $c + 1$-element set is redundant.*

While this definition of interval convexity is very broad and subsumes the most commonly known concepts of intervals, the downside is, of course, that it is too general to accurately describe the types of intervals we will study in later chapters. Therefore, we would like to define some additional axioms that will lead to different types of interval spaces, some of which are modelled on classical geometric concepts. In the following, we will give an overview of some of the most well known axioms and present some of their implications.

Geometric Interval Spaces

One particularly strong set of axioms yields the *geometric interval operator* and is an attempt to capture some more of the properties of metric spaces:

Definition 0.5.11. *An interval operator I is said to be* geometric *if it fulfils the following three properties:*

Idempotent Law: $I[a, a] = \{a\}$.
Monotone Law: *If $a, b, c \in V$ and $c \in I[a, b]$, then $I[a, c] \subseteq I[a, b]$.*
Inversion Law: *If $a, b \in V$ and $c, d \in I[a, b]$, then $c \in I[a, d]$ implies $d \in I[c, b]$.*

An interval space (V, I) is called a geometric interval space *if the operator I is geometric.*

Most of the classical interval operators from geometry, such as the geodesic interval, are in fact geometric in this sense. However, while the Idempotent and Monotone Law hold for many of the convexities presented in this text, the Inversion Law is too strong for use in convexities derived from graphs and does not hold in many cases.

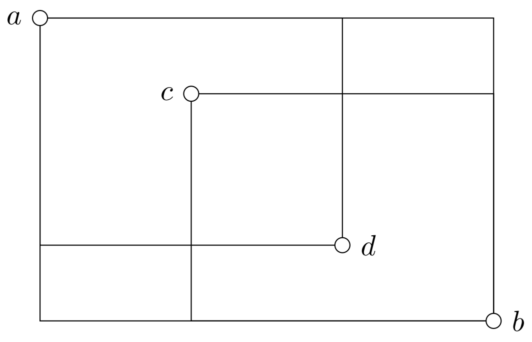


Figure 0.1: Inversion Law

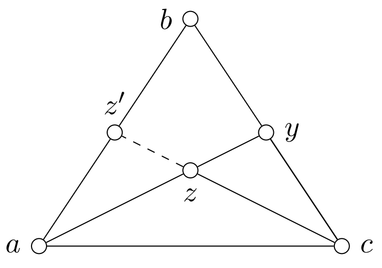


Figure 0.2: Peano Property and Pasch Property

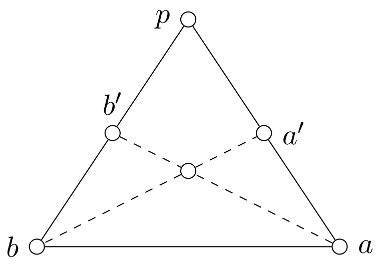


Figure 0.3: Peano Property and Pasch Property

Pasch-Peano Spaces

Moritz Pasch and Giuseppe Peano are two of the originators of modern axiomatic geometry. Using simple notions such as point and betweenness, they devised a rigorous axiomatic foundation for euclidean geometry. The following properties of Peano and Pasch are translations of two of these axioms into the terminology of interval spaces.

Peano Property: For all $a, b, c \in V$ and $y \in I[b, c]$ and $z \in I[a, y]$, there is a point $z' \in I[a, b]$ such that $z \in I[c, z']$.

Pasch Property: For all $p, a, b \in V$ and $a' \in I[p, a]$ and $b' \in I[p, b]$, the intervals $I[a, b']$ and $I[a', b]$ intersect.

The Peano Property, in particular, is very useful in the study of interval convexities, as it implies join-hull commutativity.

Theorem 0.5.12 (van de Vel [144])**.** *An interval convexity $(\mathcal{C}_I, V)$ is join-hull commutative if and only if its interval operator satisfies the Peano Property.*

The Chvátal Property

In [34] Chvátal describes two properties that are stronger versions of the Peano Property.

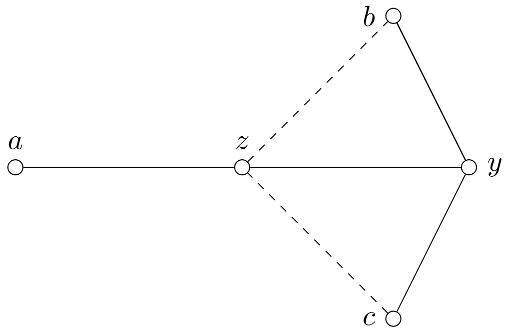


Figure 0.4: The (Strong) Chvátal Property.

Chvátal Property:	For all $a, b, c \in V$ and $y \in I[b,c]$ and $z \in I[a,y]$, it holds that $z \in I[a,b]$ or $z \in I[a,c]$ or $z \in I[b,c]$.
Strong Chvátal Property:	For all $a, b, c \in V$ and $y \in I[b,c]$ and $z \in I[a,y]$, it holds that $z \in I[a,b]$ or $z \in I[a,c]$.

As these two properties will be very useful in the study of graph convexities, we will give a short summary of their consequences on an interval space.

Lemma 0.5.13. *The Strong Chvátal Property implies the Chvátal Property which in turn implies the Peano Property.*

Proof. The first implication is obvious. For the second, suppose that the Chvátal Property holds and let $y \in I[b,c]$ and $z \in I[a,y]$. If $z \in I[a,c]$ we can identify z' with a, as $a \in I[a,b]$ and see that the Peano Property holds. If $z \in I[b,c]$ we can set z' to b as $b \in I[a,b]$, which also implies the Peano Property. If $z \in I[a,b]$ we can set z' to z and see that $z \in I[c,z']$, proving the statement. $\square$

Both Chvátal properties can also be stated differently using extreme points. Note that the use of open intervals in the following lemmas is important, due to the use of extreme points.

Lemma 0.5.14 (Chvátal [34])**.** *An interval space (V, I) fulfils the Chvátal Property if and only if for all subsets X of V and all $a, b, z \in X$ such that $z \in I(a,b)$, there are $\overline{a}, \overline{b} \in \mathrm{ex}_I(X)$ such that $z \in I[\overline{a}, \overline{b}]$.*

Lemma 0.5.15 (Chvátal [34])**.** *An interval space (V, I) fulfils the Strong Chvátal Property if and only if for all subsets X of V and all $a, b, z \in X$ such that $z \in I(a,b)$, there is a $\overline{b} \in \mathrm{ex}_I(X)$ such that $z \in I[a, \overline{b}]$.*

The power of the (Strong) Chvátal property can be seen in the fact that it severely bounds the Carathéodory number of a convexity.

Lemma 0.5.16 (Chvátal [34])**.** *An interval space (V, I) that fulfils the Chvátal Property has Carathéodory number 2.*

0.5.3 Convex Geometries

In the context of optimisation the existence and the structural properties of the extreme points of a convexity are of utmost importance. For example, in linear programming it suffices to search for an optimal solution of a bounded optimisation problem among the set of extreme vertices (or simplices as they are known in that context), leading to the famous *simplex algorithm.*

One of the strongest statements on extreme points in classical (geometric) convexity is the Minkowski-Krein-Milman theorem which roughly states that a convex polygon can be generated by convex combinations of its corners. In the language of abstract convexity this translates to the following definition.

Definition 0.5.17. *A convexity space* $(V, \mathcal{C})$ *fulfils the* Minkowski-Krein-Milman property *if and only if every convex set is the convex hull of its extreme points.*

Another important structural characteristic of classical (geometric) convexities is the *anti-exchange property*

Definition 0.5.18 (Edelman and Jamison [56])**.** *We say that a convexity space* $(V, \mathcal{C})$ *fulfils the* anti-exchange property*, if for any given convex set* X*:*

$$p, q \in V \backslash X \text{ and } q \in \operatorname{conv}(X + p) \text{ implies } p \notin \operatorname{conv}(X + q).$$

A convexity space that fulfils the anti-exchange property is called a convex geometry*.*

For an abstract convexity space the Minkowski-Krein-Milman property and the anti-exchange property are, in fact, equivalent. If either of these hold for convexity space, we call such a space a *convex geometry.* The following theorem gives an overview of the many different equivalent definitions of the concept of convex geometry.

Theorem 0.5.19 (Edelman and Jamison [56])**.** *Let* $(V, \mathcal{C})$ *be a convexity space. Then the following statements are equivalent:*

(i) $(V, \mathcal{C})$ *is a convex geometry.*

(ii) For every convex set X*, there exists a point* $p \in V \setminus X$ *such that* $X + p$ *is convex.*

(iii) Every maximal chain of convex sets $C_1 \subsetneq \ldots \subsetneq V$ *has exactly length* $|V|$*.*

(iv) For every point p *and* C *a copoint attached at* p*, the set* $C + p$ *is convex.*

(v) Every set $X \subseteq V$ *has a unique basis.*

(vi) We have $\operatorname{conv}(\operatorname{ex}(X)) = X$ *for every convex set* X*.*

(vii) We have $p \in \operatorname{ex}(\operatorname{conv}(X + p))$ *for every convex set* X *and* $p \notin X$*.*

Property *(ii)* of the above theorem implies the following useful observation.

Observation 0.5.20. *If $(V, \mathcal{C})$ is a convex geometry the elements of V have a linear ordering $\sigma = (v_1, \dots, v_n)$ such that $\{v_1, \dots, v_i\}$ is convex for every $i \in \{1, \dots, n\}$.*

Such an ordering will be called a *convexity ordering*. A convexity ordering $\sigma = v_1, \dots, v_n$ such that $\{v_i, \dots v_n\}$ is convex for every $i \in \{1, \dots, n\}$ is a *halfspace ordering*. A similar property is stated in the following.

Theorem 0.5.21 (Farber and Jamison [59])**.** *If $(V, \mathcal{C})$ is a convex geometry, then $X \in \mathcal{C}$ if and only if there is an ordering $(x_1, \dots, x_k)$ of $V \setminus X$ such that x_i is an extreme point of $X \cup \{x_i, \dots, x_k\}$ for each $i \in \{1, \dots, k\}$.*

The intersection of two convex geometries need not yield a new convex geometry, as the following example shows: Let $V = \{a, b, c\}$ and $\mathcal{C}_1 = \{\{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$ and $\mathcal{C}_1 = \{\{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, b, c\}\}$. Then $(V, \mathcal{C}_1)$ and $(V, \mathcal{C}_2)$ are both convex geometries. The intersection $(V, \mathcal{C}_1 \cap \mathcal{C})$ is not a convex geometry, as for $\{c\} \in \mathcal{C}_1 \cap \mathcal{C}_2$ there is no element $p \in V$ such that $\{c\} + p$ is convex. However, with another operation, called a *join*, it is possible to generate a new convex geometry out of two others

Definition 0.5.22. *Given two convexity spaces on the same ground set, $(V, \mathcal{C}_1)$ and $(V, \mathcal{C}_2)$ we define the* join *of $\mathcal{C}_1$ and $\mathcal{C}_2$ to be the convexity space:*

$$\mathcal{C}_1 \bigvee \mathcal{C}_2 := \{X \subseteq V \ : \ X = C_1 \cap C_2 \textit{ for some } C_1 \in \mathcal{C}_1 \textit{ and } C_2 \in \mathcal{C}_2\}.$$

Theorem 0.5.23 (Edelman and Jamison [56])**.** *If $(V, \mathcal{C}_1)$ and $(V, \mathcal{C}_2)$ are convex geometries then $(V, \mathcal{C}_1 \bigvee \mathcal{C}_2)$ is also a convex geometry.*

In general, it can be very complicated to show that some given convexity space is a convex geometry. Therefore, we will need some specific sufficient conditions which in many cases are technically much easier to check. One such condition was given by Chvátal [34]

Lemma 0.5.24 (Chvátal [34])**.** *Let $(V, \mathcal{C}_I)$ be a convexity space obtained by an interval operator I that fulfils the (Strong) Chvátal Property. Then $(V, \mathcal{C}_I)$ is a convex geometry.*

Another sufficient criterion for convex geometry is given by the Peano property combined with the following.

Definition 0.5.25. *A convexity space $(V, \mathcal{C})$ is said to be* straight *if the two following conditions are fulfilled:*

1. *For any three distinct points $a, b, c \in C$ such that $a \in \operatorname{conv}(\{b, c\})$ it holds that $b \notin \operatorname{conv}(\{a, c\})$.*
2. *For any four points $a, b, c, d \in V$ such that $b \in \operatorname{conv}(\{a, c\})$ and $c \in \operatorname{conv}(\{b, d\})$ it holds that $b, c \in \operatorname{conv}(\{a, d\})$.*

Theorem 0.5.26. *Let $(V, \mathcal{C})$ be a straight convexity space that is join-hull commutative. Then $(V, \mathcal{C})$ is a convex geometry.*

Proof. Let C be convex and let a, b be elements of $V \setminus C$ such that $a \in \operatorname{conv}(C \cup \{b\})$. Suppose that $b \in \operatorname{conv}(C \cup \{a\})$. As $(V, \mathcal{C})$ is join-hull commutative, there exist $c_1, c_2 \in C$ such that $a \in \operatorname{conv}(\{c_1, b\})$ and $b \in \operatorname{conv}(\{c_2, a\})$. If $c_1 = c_2$, then there is a contradiction to the fact that $(V, \mathcal{C})$ is straight. Suppose, therefore, that $c_1 \neq c_2$. Then the straightness of $(V, \mathcal{C})$ implies that $a, b \in \operatorname{conv}(\{c_1, c_2\})$. This is a contradiction to the convexity of C. □

Due to Theorem 0.5.12, this yields the following corollary for interval spaces.

Corollary 0.5.27. *Let $(V, \mathcal{C}_I)$ be a straight convexity space obtained by an interval operator I that fulfils the Peano property. Then $(V, \mathcal{C}_I)$ is a convex geometry.*

1 Characterising Graph Classes with Convexity

Many of the most important classes of graphs (such as forests, chordal graphs or interval graphs) can be defined through linear vertex orderings. These characterisations are typically given in the following way and we have already seen some examples in Chapter 0.

> A graph $G = (V, E)$ is a member of the graph class $\mathcal{G}$ if and only if there is a linear ordering of its vertices $(v_1, \ldots, v_n)$, such that v_i satisfies a given property $\mathcal{P}$ in the graph $G[v_1, \ldots, v_i]$.

An analogue to such a characterisation is also possible for convex sets in a convex geometry.

Theorem 1.0.28 (Farber and Jamison [59]). *If $(V, \mathcal{C})$ is a convex geometry, then X is convex if and only if there is an ordering of the elements of $V \setminus X$, say $(v_1, \ldots, v_k)$ such that v_i is an extreme point of $X \cup \{v_1, \ldots, v_i\}$ for all $i \in \{1, \ldots, k\}$.*

This similarity in structure indicates that there is a strong relationship between graph classes characterised by linear vertex orderings and convex geometries. In fact, all of the above mentioned classes (forests, chordal graphs and interval graphs) also possess a characterisation through convex geometries. This is usually done by constructing a particular convexity $(V, \mathcal{C}_{\mathcal{G}})$ for a given class $\mathcal{G}$ on the vertices of the graph and proving a statement of the following type:

> The graph $G = (V, E)$ belongs to the class $\mathcal{G}$ if and only if $(V, \mathcal{C}_{\mathcal{G}})$ is a convex geometry.

Describing graph classes in such a way has many benefits. Not only does every convex geometry also imply some form of characterisation using a linear vertex order, as we have seen in Section 0.5.3, it also makes it possible to apply all distinctive features of such a structure, such as the existence of extreme points or the concept of the Carathéodory number.

In the following, we will first present some common methods to construct convexities on graphs, as well as techniques to prove that these are convex geometries. Then we will give a summary of some classes that can be described in such a way. This summary will be comprised of some original results, as well as known results dating back to the 1970s. However, many of these are given rather implicitly in the literature, as their authors were more interested in the convexities themselves than in finding new characterisations for graph classes. This has made it necessary to “reprove” some of these old results in order to present everything in a uniform manner. This uniformity is an important part of the argument that many central definitions and concepts on these graph classes can be unified using the language of convexity.

For each of the studied classes we will in turn prove that it can be characterised by a convex geometry, describe the set of extreme vertices, define a characterising linear vertex order derived from the convex geometry and study the Carathéodory number of the convexity, giving a tight bound for all classes studied apart from AT-free graphs. Furthermore, we will discuss in each case how extreme vertices and vertex order characterisations can be computed. However, we begin by demonstrating some of the techniques on the example of a convexity which is a convex geometry for any graph.

1.1 Convexity in all Graphs

Most convexities studied in this chapter will be interval convexities, where the intervals are usually defined using paths, walks or specific separators. One possibility to define a convexity by a special type of separator is given in the form of *interception convexity* which to the best of our knowledge has not been studied in the literature before.

Definition 1.1.1. *Let $G = (V, E)$ be a graph and $z, a, b \in V$. We say that z is in the* interception interval *of a and b, denoted as $z \in I_{\text{int}}[a, b]$ if and only if there exist vertices $u, v \in N(z)$ such that u and v are in different connected components of $G-(N[z]\setminus\{u, v\})$, where a is in the same component as u and b is in the same component as v. We call u the* witness *of a and v the* witness *of b with regard to $z \in I_{\text{int}}[a, b]$. The interval convexity $(V, \mathcal{C}_{\text{int}})$ induced by this operator is called the* interception convexity.

In the following, we shall show that this convexity forms a convex geometry on any graph. First however, we wish to analyse the extreme vertices of this convexity. We will see that the following simple definition is enough to characterise these completely.

Definition 1.1.2. *A vertex v in a graph G is said to be* avoidable *if between any pair x and y of neighbours of v there exists an x-y-path all the internal vertices of which avoid v and all neighbours of v. Equivalently, a vertex v is avoidable if every induced P_3 with midpoint v closes to an induced cycle.*

As mentioned, the avoidable vertices form the extreme points of interception convexity.

Lemma 1.1.3. *Let $G = (V, E)$ be a graph. A vertex $x \in V$ is an extreme point of the interception interval operator I_{int} if and only if it is avoidable in G.*

Proof. Suppose a vertex $z \in V$ is not avoidable in G. Then there are vertices $a, b \in N(z)$ such that the path $a - z - b$ does not close to an induced cycle. This implies that a and b are in different connected components of $G - (N[z] \setminus \{a, b\})$ and therefore, that $z \in I[a, b]$, showing that z is not an extreme point of I.

Suppose a vertex $z \in V$ is not an extreme point of I, i.e., there exist $a, b \in V$ with $Z \neq a, b$ such that $z \in I[a, b]$. Then by definition there are $u, v \in N(z)$ such that u and v

are in different connected components of $G - (N[z] \setminus \{u, v\})$. This implies that the path $u - z - v$ does not close to an induced cycle and therefore, that z is not avoidable. □

Avoidable vertices appear in many different contexts throughout algorithmic graph theory. In Chapter 3 we will give a thorough study of these vertices, as well as a generalisation of these. The following theorem shows that every graph contains at least one avoidable vertex.

Theorem 1.1.4. *Every graph $G = (V, E)$ contains at least one avoidable vertex.*

We will prove this theorem in various ways in Chapter 3. This result is a strong indicator that interception convexity is a convex geometry and it is possible to extend it in that way. However, we want to show the even stronger statement that the interception interval operator fulfils the Strong Chvátal Property.

We say that the *interior of a path P avoids a vertex v* if none of the interior vertices of P is contained in $N[v]$. Similarly, we say that a vertex v *intercepts the interior of a path P* if some interior vertex of P is contained in $N[v]$.

Lemma 1.1.5. *Let $G = (V, E)$ be a graph and let $(V, \mathcal{C}_{\text{int}})$ be its interception convexity. If $z \in I_{\text{int}}[a, b]$, then z intercepts the interior of any path between a and b.*

Proof. Let $z \in I_{\text{int}}[a, b]$ and suppose that there is a path P between a and b whose interior avoids z. In this case, a and b cannot be in distinct connected components of $G - N[z] \setminus \{u, v\}$, where u and v are the witnesses of a and b, respectively, with regard to $z \in I_{\text{int}}[a, b]$; a contradiction to $z \in I_{\text{int}}[a, b]$. □

Using this lemma we can show that the Strong Chvátal Property holds for interception convexity for any graph.

Theorem 1.1.6. *Let $G = (V, E)$ be an arbitrary graph. The interception convexity $(V, \mathcal{C}_{\text{int}})$ on G fulfils the Strong Chvátal Property.*

Proof. Let $a, b, c_1, c_2, c_3 \in V$ be distinct vertices such that with $b \in I[a, c_2]$ and $c_2 \in I[c_1, c_3]$. We need to show that $b \in I[a, c_1]$ or $b \in I[a, c_3]$. By the definition of $b \in I[a, c_2]$, we know that there are vertices $a', c_2' \in V$ such that in $G - (N[b] \setminus \{a', c_2'\})$ the vertices a' and c_2' are in different connected components, the vertices a and a' are in the same connected component and c_2 and c_2' are in the same connected component. Furthermore, there are vertices $c_1', c_3' \in V$ such that in $G - (N[c_2] \setminus \{c_1', c_3'\})$ the vertices c_1' and c_3' are in different connected components, the vertices c_1 and c_1' are in the same connected component and c_3 and c_3' are in the same connected component.

As $c_2 \in I_{\text{int}}[c_1, c_3]$, the interior of any path between c_1 and c_3 must be intercepted by c_2, due to Lemma 1.1.5. Suppose that both c_1 and c_3 have paths to the connected component C_a of $G - (N[b] \setminus \{a', c_2'\})$ that contains a and the interior vertices of those paths are not contained in $N[c_2] \setminus \{c_1', c_3'\}$. This leads to a c_1-c_3-path whose interior vertices are not contained in $N[c_2] \setminus \{c_1', c_3'\}$; a contradiction to the fact that $c_2 \in I_{\text{int}}[c_1, c_3]$.

Furthermore, suppose that both c_1 and c_3 have paths to the connected component of $G - (N[b] \setminus \{a', c_2'\})$ that contains a such that the interior of those paths avoid b. Then these paths must also avoid c_2 as the contrary would lead to an a-c_2-path whose interior avoids b. Again this is a contradiction to $c_2 \in I_{\text{int}}[c_1, c_3]$ and we assume without loss of generality that c_1 has no path to a vertex in C_a whose interior avoids b and no path whose interior vertices are not contained in $N[c_2] \setminus \{c'_1, c'_3\}$.

Let P be a path between c_1 and c'_1 that lies in $G - (N[c_2] \setminus \{c'_1, c'_3\})$. If this path avoids b, then, as $c'_1 c_2 \in E$, we see that c_1 is in the same component of $G - (N[b] \setminus \{a', c'_2\})$ as c_2 and $b \in I_{\text{int}}[a, c_1]$. Therefore, we assume that x is the last neighbour of b before c_1 – note that x can be equal to c_1. We claim that in $G - (N[b] \setminus \{a', x\})$ the vertices x and a' are in distinct connected components and c_1 is in the same component as x and a' is in the same component as a. The connected component containing a must contain all vertices of C_a, as all paths connecting these vertices are also in $G - (N[b] \setminus \{a', x\})$. Due to the path P, we also see that x and c_1 are in the same component. It remains to be shown that these components are distinct. To this end, suppose that a and c_1 are in the same connected component of $G - (N[b] \setminus \{a', x\})$. However, this implies a path between c_1 and a vertex of C_a whose interior vertices contain no vertices of $N[b]$ apart from x. If such a path does not contain x, then we have a contradiction to the fact that there are no paths from c_1 to C_a whose interior avoids b. If such a path Q does contain x, then the path $c_1 - P - x - Q - v$ with $v \in C_a$ cannot contain any vertices of $N[c_2] \setminus \{c'_1, c'_3\}$, as such a vertex would have to be on $Q - x$, implying a path from c_2 to C_a whose interior avoids b; again a contradiction. Altogether, we see that $b \in I_{\text{int}}[a, c_1]$, which proves the theorem. □

Using Lemma 0.5.24, we can see that interception convexity is a convex geometry.

Corollary 1.1.7. *Let G be an arbitrary graph. The interception convexity on G is a convex geometry.*

As discussed in Section 0.5.3, the interception convexity implies a linear vertex ordering of a graph.

Definition 1.1.8. *Let $G = (V, E)$ be a connected graph and let $\sigma = (v_1, \dots, v_n)$ be a linear order of its vertices. We say that σ is an* interception convexity order *if $\{v_1, \dots v_i\}$ is interception convex for every $i \in \{1, \dots, n\}$.*

In Chapter 3, we will discuss how to compute such an order efficiently. We sum up these results as follows.

Theorem 1.1.9. *For any graph $G = (V, E)$ the following properties are equivalent:*

(i) The interception interval operator I_{int} of G fulfils the Strong Chvátal Property;

(ii) The interception convexity of G is a convex geometry;

(iii) G possesses a interception-convexity order.

Due to Lemma 0.5.16, we can also state the following.

Corollary 1.1.10. *The interception convexity of any graph $G = (V, E)$ has a Carathéodory number of at most 2.*

If we look back at Section 0.5.3 in which we introduced convex geometries, we can sum up two different approaches to prove that a given interval convexity is a convex geometry. The first has been described here, i.e., proving the (Strong) Chvátal Property. This should always be the favoured option, as it immediately yields a strong tight bound on the Carathéodory number, and it can be achieved by either showing the property directly or by proving the equivalent properties in Lemma 0.5.14 and Lemma 0.5.15. If none of the Chvátal properties hold, then a second option is to show that the convexity is straight and the interval operator fulfils the Peano Axiom. In this case, we can use Theorems 0.5.12 and 0.5.26 to prove that the convexity is a convex geometry. If both of these attempts fail, then it can be very hard to give a characterisation with a convex geometry, as can be seen in the example of AT-free graphs [28].

1.2 Convexity in Trees

One of the most natural characterisations of a graph class with a convex geometry is given for the class of trees and was first defined by Farber and Jamison [59] in 1986. In the following, we will elaborate on this definition and adapt it to our notation and setting, as well as proving some of the most important properties.

Definition 1.2.1. *Let $G = (V, E)$ be a graph and let $z, a, b \in V$. We say that z is in the* path interval *of a and b, denoted as $z \in I_{\text{path}}[a, b]$ if and only if z is on an a-b-path. The interval convexity $(V, \mathcal{C}_{\text{path}})$ induced by this operator is called the* path convexity*.*

The extreme points of this convexity can easily be seen to be the leaves of a graph.

Lemma 1.2.2. *Let $G = (V, E)$ be a graph and let $(V(G), \mathcal{C}_{\text{path}})$ be the corresponding path convexity. For any $X \in \mathcal{C}_{\text{path}}$ a vertex $x \in X$ is an extreme point of X if and only if x is a leaf in $G[X]$.*

Proof. Suppose x is a leaf in $G[X]$ and x is contained on a path P between two vertices in X. Then all vertices on P must also be in X, as it is convex. This is a contradiction to x being a leaf in $G[X]$.

Let x be an extreme point of X. Suppose x has two neighbours v_1 and v_2 in $G[X]$. As $v_1 - x - v_2$ is a path in G we know that $x \in I_{\text{path}}[v_1, v_2]$. This is a contradiction to x being extreme in X and x must be a leaf. □

Obviously, these extreme vertices can be computed in linear time, as they are just the vertices of degree one in the graph.

Definition 1.2.3. *Let $G = (V, E)$ be a connected graph and let $\sigma = (v_1, \ldots, v_n)$ be a linear order of its vertices. We say that σ is a* path convexity order *if $\{v_1, \ldots v_i\}$ is path convex for every $i \in \{1, \ldots, n\}$.*

As all vertex sets $X \subset V$ of a forest $G = (V, E)$ for which it holds that $G[X]$ is connected are path-convex, these orderings can be easily computed by any form of connected search.

Lemma 1.2.4. *Let $G = (V, E)$ be a forest and let $\sigma = (v_1, \dots, v_n)$ be the linear vertex order returned by a generic search. Then σ is a path convexity order.*

Corollary 1.2.5. *For any forest $G = (V, E)$ a path convexity order can be computed in linear time.*

Using the previous lemmas, it is straightforward to show that path convexity is in fact a convex geometry on a forest. The following theorem gives a full characterisation of forests in several different forms. Parts of this theorem have been mentioned in passing in the literature (for example, the equivalence of i) and ii) in [59]), however, usually without explicit proofs.

Theorem 1.2.6. *For every graph $G = (V, E)$ the following statements are equivalent:*

i) G is a forest.

ii) The path interval operator I_{path} fulfils the Strong Chvátal Property;

iii) The path convexity $(V(G), \mathcal{C}_{\text{path}})$ is a convex geometry;

iv) G possesses a path convexity order.

Proof. Let $G = (V, E)$ be a forest, $X \subseteq V$ and $x_2 \in I_{\text{path}}[x_1, x_3]$. Note that any extreme point of $\text{conv}(X)$ is contained in X and an extreme point of X. As $\text{conv}(X)$ is path convex, a vertex $x \in \text{conv}(X)$ is extreme in $\text{conv}(X)$ if and only if it is a leaf $G[\text{conv}(X)]$: If it is not a leaf, then it is obviously in an interval between any two neighbours. If it is a leaf in $G[\text{conv}(X)]$, then it cannot be on any path between two vertices in X, as all vertices on this path are contained in $\text{conv}(X)$; a contradiction to the fact that x is a leaf. Let C be the connected component of $G[\text{conv}(X)]$ that contains x_1, x_2 and x_3 (they are in the same component as $\text{conv}(X)$ is convex). Then $G[C]$ is a tree and deleting the last edge on the unique path from x_2 to x_3 splits $G[C]$ into two connected components, one containing x_2 and the other containing x_3, which we call C'. If x_3 is the only vertex in its component, then it was a leaf in $G[C]$ and thus extreme in X. Otherwise, there is a unique path from x_3 to a leaf $\overline{x_3}$ in C', which is also a leaf in $G[C]$, and due to construction we see that $x_2 \in I[x_1, \overline{x_3}]$. Thus the interval operator I fulfils the Chvátal Property.

If the interval operator I_{path} fulfils the Chvátal Property then $(V(G), \mathcal{C}_{\text{path}})$ is a convex geometry by Lemma 0.5.24.

Let $(V, \mathcal{C}_{\text{path}})$ be a convex geometry. Suppose G contains a cycle $C = (x_1, x_2, \dots, x_k, x_1)$ with $k \geq 3$. This implies that $x_2 \in I_{\text{path}}[x_1, x_3]$ and $x_3 \in I_{\text{path}}[x_1, x_2]$ which is a contradiction to $(V, \mathcal{C}_{\text{path}})$ being a convex geometry.

If G is a forest, then its path convexity is a convex geometry. Due to Corollary 0.5.20 it possesses a path convexity order.

Suppose G possesses a path convexity order and is not a forest. Then G must contain a cycle $C = (x_1, x_2 \dots, x_k)$ of size $k \geq 3$. This implies that $x_1 \in I_{\text{path}}[x_k, x_2]$, $x_2 \in I_{\text{path}}[x_1, x_k]$ and $x_k \in I_{\text{path}}[x_1, x_2]$. This is in contradiction to the existence of a path convexity order, as these elements cannot be ordered accordingly. □

As the path convexity fulfils the Strong Chvátal Property, we can state the following corollary due to Lemma 0.5.16.

Corollary 1.2.7. *The path convexity of any tree $T = (V, E)$ has a Carathéodory number of at most 2.*

Another interesting property of path convexity concerns the existence of a halfspace ordering.

Theorem 1.2.8. *Any forest $G = (V, E)$ has a halfspace ordering if and only if it has maximum degree ≤ 2, i.e., is claw-free.*

Proof. One can easily check that the claw graph does not have a halfspace ordering with respect to path convexity. Conversely, if the maximum degree of G is ≤ 2, the forest is a collection of induced paths. The halfspace ordering is achieved by sorting the vertices in the order of their path and concatenating these orders arbitrarily. □

1.3 Convexity in Chordal Graphs

One of the first graph convexities to be studied is the *monophonic convexity*. It was introduced by Jamison-Waldner [94] in 1982 to describe chordal graphs. It was shown very early on by Farber and Jamison [59] that it is a convex geometry as well as having other interesting properties. Dragan et al. [53] then generalized the results on monophonic convexity to larger classes of graphs. Furthermore, Chvátal [34] used monophonic convexity as an example of a convexity that possesses the Strong Chvátal Property. In this section, we give a brief overview of results concerning monophonic convexity. These will be important later on when we analyse other chordal graph classes such as interval graphs.

Definition 1.3.1. *Let $G = (V, E)$ be a graph and let $z, a, b \in V$. We say that z is in the* monophonic interval *of a and b, denoted as $z \in I_{\text{mon}}[a, b]$ if and only if z is on an induced a-b-path. The interval convexity $(V, \mathcal{C}_{\text{mon}})$ induced by this operator is called the* monophonic convexity.

The extreme points of monophonic convexity coincide with the simplicial vertices of a graph. As in the case of trees where the extreme points are the leaves of a graph, this shows up a close connection between this convexity and the already known structural properties of chordal graphs.

Lemma 1.3.2. *Let $G = (V, E)$ be a graph and $X \in \mathcal{C}_{\text{mon}}$. A vertex $x \in X$ is an extreme point of X if and only if x is simplicial in $G[X]$.*

Proof. Suppose x is not simplicial in X. Then x is an inner vertex of a P_3 in $G[X]$. This is a contradiction to x being extreme.

Suppose x is simplicial in $G[X]$ and there are vertices $u, v \in X$, such that x is on an induced path P between u and v. As X is monophonically convex, all vertices on P are in X, especially both neighbours of x on P. This yields a contradiction, as x is simplicial and P is an induced path. □

Simplicial vertices can be found in linear time on chordal graphs, as they are the last vertices visited by an LBFS [126]. Again, we can define a convexity order which will lead to a characterising linear vertex order.

Definition 1.3.3. *Let $G = (V, E)$ be a connected graph and let $\sigma = (v_1, \dots, v_n)$ be a linear order of its vertices. We say that σ is a* monophonic convexity order *if $\{v_1, \dots v_i\}$ is monophonically convex for every $i \in \{1, \dots, n\}$.*

It is easy to see that every monophonic convexity order is a perfect elimination order. The following lemma shows that the converse also holds.

Lemma 1.3.4. *For any graph $G = (V, E)$ a linear vertex order $\sigma = (v_1, \dots, v_n)$ is a monophonic convexity order if and only if it is a perfect elimination order.*

Proof. Suppose that σ is a monophonic convexity order. Then $\{v_1, \dots v_i\}$ is monophonically convex for every $i \in \{1, \dots, n\}$. In particular, v_i cannot be in an interval between two elements that are to the left in σ. This implies that v_i is simplicial in $G[v_1, \dots, v_i]$ and thus, that σ is a perfect elimination order.

Suppose, on the other hand, that σ is a perfect elimination order and that for some $i \in \{1, \dots n\}$ the set $\{v_1, \dots, v_i\}$ is not convex. In this case, there exist indices $l, k \leq i$ and $j > i$ such that v_j is on an induced path P between v_k and v_l. As both of v_l and v_k are to the left of v_j in σ, there must be a vertex x on P, such that both neighbours of x on P are to the left of x in σ. As P is an induced path, these neighbours are not adjacent. This is a contradiction to the fact that σ is a perfect elimination ordering, as all neighbours to the left of a vertex must be adjacent. □

As Rose et al. [126] showed that LBFS performed on a chordal graph returns a perfect elimination order, we can state the following corollary.

Corollary 1.3.5. *For any chordal graph a monophonic convexity order can be found in linear time.*

While it was already shown by Farber and Jamison [59] that monophonic convexity forms a convex geometry on chordal graphs, in [34] Chvátal shows that monophonic convexity also fulfils the Strong Chvátal Property. Again, we can summarise all the studied characterising properties of chordal graphs in the following theorem.

Theorem 1.3.6 (Chvátal [34], Farber and Jamison [59])**.** *For any graph $G = (V, E)$ the following properties are equivalent:*

i) G is chordal;

ii) The monophonic interval operator of G fulfils the Strong Chvátal property;

iii) The monophonic convexity of G is a convex geometry;

iv) G possesses a monophonic convexity (perfect elimination) order.

As the monophonic convexity fulfils the Strong Chvátal Property, we can state the following corollary due to Lemma 0.5.16.

Corollary 1.3.7. *The monophonic convexity of a chordal graph $G = (V, E)$ has a Carathéodory number of at most 2.*

This result has already been shown by Farber and Jamison [59] in a different way. Duchet [54] gives an even stronger statement.

Theorem 1.3.8 (Duchet [54])**.** *Let $G = (V, E)$ be a connected graph which is not a clique. Then the monophonic convexity $(V, \mathcal{C}_{\text{mon}})$ has Carathéodory number at most 2.*

As in the case of path convexity, we can completely characterise the existence of a halfspace ordering with regard to monophonic convexity.

Theorem 1.3.9. *Any chordal graph $G = (V, E)$ has a halfspace ordering if and only if it is claw-free, net-free and 3-sun-free, i.e., it is a unit interval graph.*

Proof. It is easy to ascertain that neither the claw graph nor the net nor the 3-sun possess a halfspace ordering with regard to monophonic convexity. Conversely, any unit interval order of G is a halfspace ordering, as both a unit interval order and its reverse form a perfect elimination order. □

1.4 Convexity in Interval Graphs

In order to describe interval graphs with a convexity, we recall a classical result by Lekkerkerker and Boland [109] that links interval, chordal and AT-free graphs:

Theorem 1.4.1 (Lekkerkerker and Boland [109])**.** *A graph is an interval graph if and only if it is chordal and AT-free.*

Therefore, an interval operator that captures the properties of an interval graph must ensure chordality and forbid asteroidal triples. This can be achieved by forming the union of the monophonic interval and an interval operator which we will later use to characterise asteroidal triple free graphs.

Definition 1.4.2. *Let $G = (V, E)$ be a connected graph and let $a, b \in V$. For $z, a, b \in V$, we say that z is in the* domination interval *of a and b, denoted as $z \in I_{\text{dom}}[a, b]$ if and only if there is an induced a-z-path that avoids b and an induced b-z-path that avoids a. We define the* line interval *between a and b as $I_{\text{line}}[a, b] = I_{\text{dom}}[a, b] \cup I_{\text{mon}}[a, b]$. The interval convexity $(V, \mathcal{C}_{\text{line}})$ induced by this operator is called the* line convexity*.*

As in the case of chordal graphs, the extreme vertices of the line convexity in an interval graph have already been described in the study of these graphs: They form the set of vertices that are rightmost for some interval representation (or in other words rightmost for some interval order). Gimbel [73] proved that these vertices can be characterised as the simplicial and admissible vertices of an interval graph. The following lemma shows that these two properties are enough to characterise extreme vertices of the line convexity in general graphs.

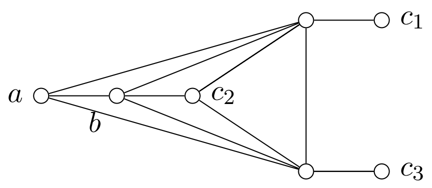


Figure 1.1: An interval graph whose line convexity does not fulfil the Strong Chvátal Property: While $b \in I_{\text{line}}[a, c_2]$ and $c_2 \in I_{\text{line}}[c_1, c_3]$ we see that b is not contained in $I_{\text{line}}[a, c_1]$ or $I_{\text{line}}[a, c_3]$.

Lemma 1.4.3. *Let $G = (V, E)$ be a connected graph. A vertex $x \in X$ is an extreme vertex of the line-convex set $X \subseteq G$ if and only if it is simplicial and admissible in $G[X]$.*

Proof. Let $x \in X$ be simplicial and admissible in $G[X]$. Suppose, x is not extreme in X, i.e., there are vertices $u, v \in X$ distinct from x such that $x \in I_{\text{line}}[u, v]$. If $x \in I_{\text{mon}}[u, v]$, then, just as in Lemma 1.3.2, the vertex x is on an induced path between u and v. Obviously all other vertices on that path must also be part of $I_{\text{line}}[u, v]$ and are, thus, in X, as it is convex. Therefore, we see that $x \in I_{\text{line}}[u, v]$ in $G[X]$ which shows that it is not simplicial; a contradiction

On the other hand, if $x \in I_{\text{dom}}[u, v]$, then there must be an induced u-avoiding x-v-path P and an induced v-avoiding x-u-path P. As a result, all of the vertices on P are in $I_{\text{mon}}[a, v]$ and all of the vertices on Q are in $I_{\text{mon}}[a, u]$. Thus, they are in X, as it is convex and $x \in I_{\text{dom}}[u, v]$ in $G[X]$. This again implies a contradiction, as x was assumed to be admissible in $G[X]$.

For the other direction, let x be an extreme vertex in X. If x is not simplicial in $G[X]$, then it must be the inner vertex of a P_3, say (u, x, v) in $G[X]$ and thus, $x \in I_{\text{mon}}[u, v]$ in $G[X]$. If x is not admissible in $G[X]$, then there must be $u, v \in X$ such that $x \in I_{\text{dom}}[u, v]$ in $G[X]$. Both of these cases form a contradiction to the assumption that x is extreme in $G[X]$. □

Note that the end-vertices of LBFS on interval graphs are simplicial and admissible [44], making it possible to compute an extreme vertex of the line convexity in linear time.

Having defined the most important concepts for this convexity, we wish to formulate a characterising theorem of the type already given for trees and chordal graphs. However, this will need to take a slightly different form. The example given in Figure 1.1 shows that the Strong Chvátal Property does not hold for the line convexity in every interval graph. Therefore, we will at least aim to prove the weaker Chvátal Property which still yields that the convexity is a convex geometry and gives a bound of 2 on the Carathéodory number.

To this end we will again define a linear vertex order based on the convexity.

Definition 1.4.4. *Let $G = (V, E)$ be a connected graph and let $\sigma = (v_1, \dots, v_n)$ be a linear order of its vertices. We say that σ is a* line convexity order *if $\{v_1, \dots v_i\}$ is line-convex for every $i \in \{1, \dots, n\}$.*

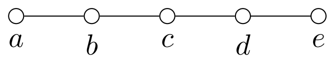


Figure 1.2: The linear vertex order $\sigma = (c, b, d, a, e)$ forms a line convexity order that is not an interval order.

In the following, it will be convenient to have a characterisation of a line convexity order given by the monophonic and domination intervals. The following concept has already been introduced by Corneil and Stacho [38] and will be analysed thoroughly in Section 1.5.

Definition 1.4.5. *Let $G = (V, E)$ be a connected graph and let $\sigma = (v_1, \ldots, v_n)$ be a linear order of its vertices. We say that σ is a* domination convexity order *(or an* AT-free order*) if for every triple $a, b, z \in V$ with $z \in I_{\text{dom}}[a, b]$ it holds that $z \prec_\sigma a$ or $z \prec_\sigma b$.*

Lemma 1.4.6. *Let $G = (V, E)$ be a connected graph and let $\sigma = (v_1, \ldots, v_n)$ be a linear order of its vertices. Then σ is a line convexity order if and only if it is a monophonic convexity order and a domination convexity order.*

Proof. If σ is a monophonic convexity order, then Theorem 1.3.6 implies that G is chordal. Suppose that G contains an asteroidal triple $\{x, y, z\}$. Then we know that $x \in I_{\text{dom}}[y, z]$, $y \in I_{\text{dom}}[x, z]$ and $z \in I_{\text{dom}}[x, y]$. This is a contradiction to the fact that σ is a domination convexity order, as there is no possible order of these three vertices that is allowed. Consequently, the graph is chordal and AT-free and, according to Theorem 1.4.1, it is an interval graph. $\square$

Note that while any interval order of an interval graph is, in fact, a line convexity order, the reverse must not be the case (see Figure 1.2). The following theorem shows that line convexity orders can be found by a simple application of LBFS. Thus, as in the case of monophonic and tree convexity, these orders can be found in linear time by using this search. This is not the case for interval orders, where it has been shown by Ma [111] that for every constant c there is an interval graph for which c sweeps of LBFS do not yield an interval order.

Lemma 1.4.7 (Corneil et al. [40])**.** *Let $G = (V, E)$ be an asteroidal triple free graph and let $\sigma = (v_1, \ldots, v_n)$ be a linear order of its vertices computed by LBFS. Then the vertex v_i is admissible in $G[v_1, \ldots, v_i]$.*

Theorem 1.4.8. *Let $\sigma = (v_1, \ldots, v_n)$ be an arbitrary LBFS order of an interval graph G. Then σ is a line convexity order.*

Proof. As every LBFS of a chordal graph is a perfect elimination order and every perfect elimination order is a monophonic convexity order, it remains to show that σ is a domination convexity order.

Suppose that $z \in I_{\text{dom}}[a, b]$ and without loss of generality $a \prec_\sigma b \prec_\sigma z$. As σ is an LBFS order, due to Lemma 1.4.7, z is both simplicial and admissible in $G[\{v_1, \ldots, v_i\}]$, where $v_i = z$.

Let P be an induced a-avoiding z-b-path. As σ is a monophonic convexity order we know that $X = \{v_1, \dots, v_i = z\}$ is monophonically convex. As P is an induced path and both $b, z \in X$, by definition all vertices of P must be elements of X. Analogously, the same holds for some induced b-avoiding z-a-path. This is a contradiction to the fact that z is admissible in $G[X]$, proving that σ is also a domination convexity order. □

By showing that LBFS always maintains a convex set, wherever it is started or what choices are made at ties, this theorem goes some way to explaining why this search is such a successful tool when applied to interval graphs. In fact, if it were possible to decide whether a given vertex order is a line convexity order in linear time, this would yield another linear time algorithm for recognising interval graphs.

Theorem 1.4.9. *Any interval graph $G = (V, E)$ has a halfspace ordering with regard to its line convexity if and only if it is claw-free, i.e., if it is a unit interval graph.*

Proof. It is a straightforward exercise to show that the claw graph does not have a halfspace ordering with regard to its line convexity. Conversely, any unit interval graph has an example of a halfspace order in its unit interval order. □

In the proof of Theorem 1.4.13 we will make frequent use of the following lemmas.

Lemma 1.4.10. *For any graph $G = (V, E)$ it holds that $z \in I_{\text{line}}[a, b]$ if and only if a and b are not adjacent and there is an induced path from z to b whose inner vertices avoid a and an induced path from z to a whose inner vertices avoid b.*

Proof. For one direction, let there be an induced path P from z to b whose inner vertices avoid a and an induced path Q from z to a whose inner vertices avoid b.

If either of a or b is adjacent to z, without loss of generality let this be a, then it is easy to see that $a - z - P$ forms an induced path from a to b that contains z. This implies that $z \in I_{\text{mon}}[a, b] \subseteq I_{\text{line}}[a, b]$. Otherwise, a, b and z form an independent triple such that there is an induced a-z-path that avoids b and an induced b-z-path that avoids a and thus, $z \in I_{\text{dom}}[a, b] \subseteq I_{\text{line}}[a, b]$.

Now, let $z \in I_{\text{line}}[a, b]$. Then z is contained in $I_{\text{mon}}[a, b]$ or $I_{\text{dom}}[a, b]$. Either way, we see that there is an induced path from z to b whose inner vertices avoid a and an induced path from z to a whose inner vertices avoid b, proving the statement. □

Lemma 1.4.11. *Let $G = (V, E)$ be an interval graph and let $z \in I_{\text{line}}[a, b]$. Then there is no path between a and b whose inner vertices avoid z.*

Proof. Suppose that $z \in I_{\text{dom}}[a, b]$. If there is a z-avoiding path between a and b, then a, b and z form an asteroidal triple; a contradiction to the fact that G is an interval graph.

On the other hand, if $z \in I_{\text{mon}}[a, b] \setminus I_{\text{dom}}[a, b]$, then we can assume without loss of generality that $az \in E$, as otherwise there must be an a-avoiding path from b to z and a b-avoiding path from a to z which implies $z \in I_{\text{dom}}[a, b]$. Let P be an induced a-b-path whose inner vertices avoid z. Furthermore, let Q be an induced a-avoiding b-z-path. Let b' be the vertex closest to z in Q that is adjacent to a vertex of P (note that this could be b itself) and let a' be a vertex that is closest to a in P such that a' and b' are adjacent. Note that a' and a cannot be equal, as Q is a path whose inner vertices avoid a and the vertices a and b are not adjacent.

We claim that the cycle $a - z - Q - b' - a' - P - a$ is an induced cycle of length larger or equal 4. It is easy to see that the cycle has more than four vertices. To see that it is induced, it is enough to note that both P and Q were chosen to be induced and a' and b' were chosen to not be adjacent to any vertices on $Q_{[z,b')}$ and $P_{[a,a')}$ respectively. Furthermore, a is not adjacent to any vertices on $Q_{(z,b']}$ and z is not adjacent to any vertices on $P_{(a,a']}$. Therefore, the cycle cannot contain any chords and is induced; a contradiction to the fact that the interval graph G is chordal. □

As an easy implication, we show that every line interval in an interval graph is in fact a line convex set.

Lemma 1.4.12. *Let $G = (V, E)$ be an interval graph. Then every line interval $I_{\text{line}}[a, b]$ is line convex, i.e., for $x, y \in I_{\text{line}}[a, b]$ and $z \in I_{\text{line}}[x, y]$ it holds that $z \in I_{\text{line}}[a, b]$.*

Proof. Note that if both a and b are adjacent to z, then $z \in I_{\text{mon}}[a, b] \subseteq I_{\text{line}}[a, b]$. Therefore, we can assume that without loss of generality a is not adjacent to z. Let P be an a-x-path whose inner vertices avoid b and let Q be an a-y-path whose inner vertices avoid b. Then $R := x - P - Q - y$ is an x-y-path whose inner vertices avoid b. Due to Lemma 1.4.11, the inner vertices of R cannot avoid z, as $z \in I_{\text{line}}[x, y]$. Thus, z intercepts R which implies that $z - R - a$ is an a-z-path whose inner vertices avoid b. Analogously, it can be shown that there is a b-z-path whose inner vertices avoid a. By Lemma 1.4.10, we see that $z \in I_{\text{line}}[a, b]$, proving the statement. □

Using these results, we can prove the desired characterising theorem.

Theorem 1.4.13. *For any graph $G = (V, E)$ the following properties are equivalent:*

i) G is an interval graph;

ii) The line interval operator I_{line} of G fulfils the Chvátal Property;

iii) The line convexity of G is a convex geometry;

iv) G possesses a line-convexity order.

Proof. We prove the theorem by showing that **i)** $\Rightarrow$ **ii)** $\Rightarrow$ **iii)** $\Rightarrow$ **iv)** $\Rightarrow$ **i)**.

i) $\Rightarrow$ **ii)** : Let $b \in I_{\text{line}}[a, c_2]$ and $c_2 \in I_{\text{line}}[c_1, c_3]$. To prove that I_{line} fulfils the Chvátal Property we need to show that $b \in I_{\text{line}}[a, c_1]$, $b \in I_{\text{line}}[a, c_3]$ or $b \in I_{\text{line}}[c_1, c_3]$.

If both c_1 and c_3 are adjacent to b, then we see that $b \in I_{\text{mon}}[c_1, c_3] \subseteq I_{\text{dom}}[c_1, c_3]$ and we are done. Therefore, we assume without loss of generality that c_1 and b are not adjacent.

Let P be a path between a and b whose internal vertices avoid c_2. Suppose both c_1 and c_3 intercept $P_{[a,b)}$. This would imply a c_1-c_3-path whose internal vertices avoid c_2; a contradiction to Lemma 1.5.16. Hence, we can assume that c_1 does not intercept P, implying that P is a c_1-avoiding a-z-path. Let Q be a c_1-c_2-path whose internal vertices avoid c_3 and let R be a z-c_2 path whose internal vertices avoid a. If Q avoids a, then there is a z-c_1 path in form of $z - R - c_2 - Q - c_1$ whose internal vertices avoid a, implying that $b \in I_{\text{line}}[a, c_1]$ and the statement. Thus we assume that a intercepts Q which yields a c_3-avoiding path from a to c_1.

If c_3 intercepts P, then there is a c_1-avoiding a-c_3 path in form of $c_3 - P - a$ which implies that $a \in I_{\text{line}}[c_1, c_3]$. Using Lemma 1.4.12, we see that $b \in I_{\text{line}}[c_1, c_3]$, as both $a, c_2 \in I_{\text{line}}[c_1, c_3]$ and $b \in I_{\text{line}}[a, c_2]$.

Therefore, we assume that P avoids c_3. In this case, we can construct an a-c_3-path whose inner vertices avoid c_1 in the same way that we constructed a c_3-avoiding path from a to c_1. Again this implies that $a \in I_{\text{line}}[c_1, c_3]$ and using Lemma 1.4.12 we see that $b \in I_{\text{line}}[c_1, c_3]$, as both $a, c_2 \in I_{\text{line}}[c_1, c_3]$ and $b \in I_{\text{line}}[a, c_2]$. Thus, the Chvátal Property holds.

ii) $\Rightarrow$ **iii)** : This follows from Lemma 0.5.24.

iii) $\Rightarrow$ **iv)** : If the line convexity of G is a convex geometry, then, due to Corollary 0.5.20, it possesses a line convexity order.

iv) $\Rightarrow$ **i)** : Suppose G possesses a line-convexity order and is not an interval graph. Then G must contain an induced cycle $C = (x_1, x_2, x_3 \ldots, x_k)$ of size $k \geq 4$ or an asteroidal triple $\{x, y, z\}$. In the first instance we have $x_1 \in I_{\text{line}}[x_k, x_2]$, $x_2 \in I_{\text{line}}[x_1, x_3]$ and $x_3 \in I_{\text{line}}[x_2, x_k]$, in the second $y \in I_{\text{line}}[x, z]$, $z \in I_{\text{line}}[x, y]$ and $x \in I_{\text{line}}[y, z]$. Both cases are in contradiction to the existence of a line-convexity order, as these elements cannot be ordered accordingly. □

As the line convexity fulfils the Chvátal Property, we can state the following corollary due to Lemma 0.5.16.

Corollary 1.4.14. *The line convexity of an interval graph $G = (V, E)$ has a Carathéodory number of at most 2.*

Note that Alcón et al. [3] also constructed a convex geometry characterising interval graphs, where the authors use so-called *tolled-walks* to define the intervals used for the convexity.

Definition 1.4.15 (Alcón et al. [3]). *Let $G = (V, E)$ be a connected graph. A* tolled walk *between u and v is a walk $T = (u, w_1, \ldots, w_k, v)$ with $k \geq 1$ such that $uw_i \in E$ if and only if $i = 1$ and $vw_i \in E$ if and only if $i = k$.*

Let the toll interval *$I_{\text{toll}}[a, b]$ contain all vertices z such that z is on a tolled walk between a and b. The interval convexity $(V, \mathcal{C}_{\text{toll}})$ induced by this operator is called the* toll convexity

Their convexity is equivalent to the one defined here. However, the use of tolled walks is not very intuitive and does not show up the importance of asteroidal triples in interval graphs. Furthermore, as they did not prove the Chvátal Property for their convexity, they were not able to give a bound for the Carathéodory number in interval graphs.

1.5 AT-Free Convexity

AT-free graphs are widely believed to exhibit a "linear structure" [98] akin to the interval graphs and two results in particular corroborate this claim: In [39] it was shown that every AT-free graph contains a *dominating pair*, i.e., a pair of vertices such that every path between them forms a dominating set for the whole graph. This result was strengthened in the same paper [39] which characterised AT-free graphs with the so-called *spine property*: A graph H has the spine property if for every non-adjacent dominating pair s and t there exists a neighbour of t, say t', such that s and t' are a dominating pair in the connected component of $H - t$ that contains s. As shown in [39], a graph G is an asteroidal triple free graph if and only if every connected induced subgraph of G has the spine property. This can be seen as a generalisation of the fact that the maximal cliques of interval graphs form a chain. In this section, we will analyse how this "linear structure" is related to a convex geometry on AT-free graphs.

In Section 1.4, we have already seen that the structure of AT-free graphs can be captured through an interval operator which we called the *domination interval.* In this section, we will expand this operator to a proper convexity and show that this is, in fact, a convex geometry. The domination interval was introduced by Broersma et al. [26] as a tool to solve the independent set problem on AT-free graphs. Many years later, Corneil and Stacho [38] used these same intervals to define a characterising linear vertex order for AT-free graphs (called AT-free orders), which in turn motivated the construction of a convexity for AT-free graphs by Chang et al. [28]. In this paper, the authors show that the constructed convexity is, in fact, a convex geometry and used this result to generate all the AT-free orders.

In this section, we will give an overview of the results for convexity in AT-free graphs and use these to define a new vertex order characterisation of AT-free graphs. Furthermore, we study the Carathéodory number of this convexity and attempt to bound it with a constant number.

In the following, we repeat and expand on Definitions 1.4.2 and 1.4.5.

Definition 1.5.1. *Let $G = (V, E)$ be a graph and $z, a, b \in V$. We say that z is in the* domination interval *of a and b, denoted as $z \in I_{\text{dom}}[a, b]$ if and only if there is an induced a-z-path that avoids b and an induced b-z-path that avoids a. The interval convexity $(V, \mathcal{C}_{\text{dom}})$ induced by this operator is called the* domination convexity*.*

One of the first results shown for domination intervals was the following characterisation of AT-free graphs.

Theorem 1.5.2 (Broersma et al. [26], Köhler [99])**.** *A graph $G = (V, E)$ is AT-free if and only if for any $a, b, z \in V$ such that $z \in I_{\text{dom}}[a, b]$ it holds that $I_{\text{dom}}[a, z] \subseteq I_{\text{dom}}[a, b]$ and $I_{\text{dom}}[z, b] \subseteq I_{\text{dom}}[a, b]$.*

The extreme vertices of domination convexity, i.e., those not included in any domination interval of which it is not an endpoint, are known as *admissible vertices*. Corneil et al. [39] have shown that the last vertex visited by an arbitrary LBFS on an AT-free graph is admissible, making it possible to compute such vertices in linear time.

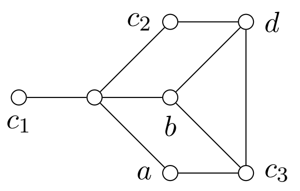


Figure 1.3: An AT-free graph that fulfils neither the strong nor the regular Chvátal Property: One can see that $b \in I_{\text{dom}}[a, c_2]$, while $c_2 \in I_{\text{dom}}[c_1, c_3]$. However, the vertex b is contained neither in $I_{\text{dom}}[c_1, c_3]$ nor in $I_{\text{dom}}[a, c_1]$, nor in $I_{\text{dom}}[a, c_3]$. As the intervals $I_{\text{dom}}[a, c_1]$ and $I_{\text{dom}}[a, c_3]$ only contain their respective endpoints, this example does not even fulfil the Peano-Property. Furthermore, given the convex set $C = \{d, c_3\}$ and vertex c_1, one can see that this convexity is not join-hull commutative.

In Figure 1.3 one can see that none of the techniques presented here can be used to show that $(V, \mathcal{C}_{\text{dom}})$ is a convex geometry. The domination convexity of the graph given there does not fulfil the Peano Property and thus, it also does not fulfil the (Strong) Chvátal Property. Furthermore, that convexity is not join-hull commutative. Therefore, it is necessary to use a non-standard technique to show that the domination convexity is a convex geometry in an AT-free graph. Recently, this was achieved by Chang et al. [28] who use a complicated construction to verify the anti-exchange property.

Theorem 1.5.3 (Chang et al. [28])**.** *Let $G = (V, E)$ be a graph and let $(V, \mathcal{C}_{\text{dom}})$ be its domination convexity. Then G is AT-free if and only if $(V, \mathcal{C}_{\text{dom}})$ is a convex geometry.*

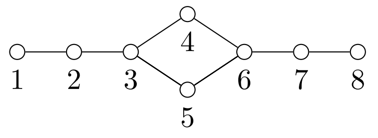
Now that it has been established that we are dealing with a convex geometry, we will take a closer look at the corresponding linear vertex characterisation which is implied by the convex geometry.

1.5.1 A New Linear Vertex Order Characterisation

An important algorithmic tool in the theory of algorithmic graph theory has been the use of linear vertex orderings which characterise particular graph classes. It was long conjectured that such a characterising linear vertex ordering must also exist for AT-free graphs. In a recent result, Corneil and Stacho [38] answered this conjecture in the positive by showing that AT-free graphs can be characterised by the following linear vertex order. Note that in this section we will mainly use *open* domination intervals to make notation simpler.

Definition 1.5.4. *Let $G = (V, E)$ be a connected graph and let $\sigma = (v_1, \dots, v_n)$ be a linear order of its vertices. We say that σ is a* domination convexity order *(or an* AT-free order*) if for every triple $a, b, z \in V$ with $z \in I_{\text{dom}}(a, b)$ it holds that $z \prec_\sigma a$ or $z \prec_\sigma b$.*

Theorem 1.5.5 (Corneil and Stacho [38], Chang et al. [28])**.** *A graph is AT-free if and only if it has an AT-free order.*



Arbitrary AT-free:	$(4, 5, 2, 7, 3, 6, 1, 8)$
LexComp:	$(4, 5, 3, 6, 2, 7, 1, 8)$
$\text{BFS}^{\text{conv}}(G, 1)$:	$(1, 2, 3, 4, 5, 6, 7, 8)$

Figure 1.4: Graph with its various AT-free orders

However, the notion of these orderings leaves quite a bit of freedom. Ideally, such an ordering would somehow capture the structure given in the spine property in [39] (as it is in the case of interval orderings which immediately gives us the chain of maximal cliques). However, the so-called *LexComp* ordering that is constructed in [38] has one significant drawback: For some graphs the resulting ordering is "folded" in a way that seems to contradict our notion of linear behaviour. For example, given the path graph with $2n+1$ vertices, the P_{2n+1}, where the vertices are numbered from left to right along the path, we would expect any viable linear vertex ordering to be $(1, 2, \ldots, 2n+1)$ or its inversion. The algorithm in [38], on the other hand, might output $(n+1, n, n+2, n-1, \ldots, 1, 2n+1)$. In addition, this construction can even yield vertex orders $\sigma := (v_1, \ldots, v_n)$ such that there are $i \in \{1, \ldots, n\}$ for which $G[v_1, \ldots, v_i]$ is not connected, for example the chordless cycle in five vertices. More examples can be found in Figure 1.4.

In an attempt to remedy this issue, the authors of [38] investigate whether it is possible to find AT-free orderings that coincide with search orders. After proving that there are graphs G such that no LBFS ordering of G is an AT-free order, they conjecture that every AT-free graph has an AT-free order that is a BFS order.

Conjecture 1.5.6 (Corneil and Stacho [38])**.** *Let $G = (V, E)$ be an AT-free graph. Then there exists a BFS ordering $\sigma = (v_1, \ldots, v_n)$ that is an AT-free order.*

Using the concepts of convex geometry presented here, we will prove an even stronger version of this conjecture and show how such an order can be used to wed the notion of an AT-free ordering to the spine property. We will also give a polynomial time algorithm to compute such an order that takes at most the time needed to compute all domination intervals, which is possible in time $\mathcal{O}(n^3)$ by [26]. This can be done by showing that BFS can be executed such that at every step the search chooses a vertex, such that the set of visited vertices forms a domination convex set.

Theorem 1.5.7. *Let G be a connected AT-free graph. Then for any vertex $s \in V$ there is a linear vertex order $\sigma := (s = v_1, \ldots, v_n)$ that is an AT-free order and a BFS order.*

Proof. Let σ be a BFS order starting in an arbitrary vertex s of G with the following tie-break rule: At each step i, choose the vertex v_i such that $\operatorname{conv}(\{s = v_1, \ldots, v_i\})$ has smallest cardinality among all allowed choices at step i. We will show, that $\{s = v_1, \ldots, v_i\}$ is convex for $i \in \{1, \ldots, n\}$, which implies that σ is an AT-free order. The proof will be by induction on the BFS steps.

For $k = 1$ the claim is true, as every one element set is convex in $\mathcal{C}$.

We show the claim for step k, assuming it is true for $k-1$. Suppose v_k is chosen. Then $\{v_1, \ldots, v_{k-1}\}$ is convex and v_k is such that $\operatorname{conv}(\{v_1, \ldots, v_{k-1}, v_k\})$ is smallest among all vertices that can be chosen by the search in step k. As we are conducting a BFS, there is a vertex $y \in \{v_1, \ldots v_{k-1}\}$ that is adjacent to all possible choices, but no others. Assume that $\{v_1, \ldots, v_k\}$ is not convex. Then there is a vertex $p \in V \setminus \{v_1, \ldots, v_k\}$, such that $p \in I_{\mathrm{dom}}(v, v_k)$ for some vertex $v \in \{v_1, \ldots v_{k-1}\}$. As $(V, \mathcal{C}_{\mathrm{dom}})$ is a convex geometry, we can deduce that $\mathrm{conv}(\{v_1, \ldots, v_{k-1}, p\}) \subsetneq \mathrm{conv}(\{v_1, \ldots v_{k-1}, v_k\})$. This implies that $yp \notin E$, due to the choice of v_k. Let w be the vertex that forced v into the BFS ordering (it may be that $y = w$). Due to the definition of BFS, we see that $\mathrm{dist}_G(s, w) \leq \mathrm{dist}_G(s, y) < \mathrm{dist}_G(s, p)$. We can assume that $wp \notin E$, as otherwise p would have been chosen before v_k. Therefore, the vertices v, v_k and p form an asteroidal triple, due to the p-avoiding walk from v to v_k along w, s and y; a contradiction to the fact that G is AT-free. □

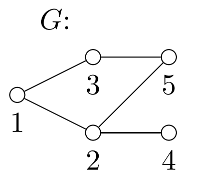
This theorem implies an algorithm for computing an AT-free BFS order which will be denoted by BFS$^{\mathrm{conv}}$.

Algorithm 10: BFS$^{\mathrm{conv}}$

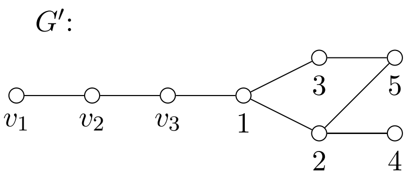
```
   Input: Connected graph $G$ and a distinguished vertex $s \in V$
   Output: A vertex ordering $\sigma$
1  begin
2      Compute $I_{\mathrm{dom}}(v, w)$ for every pair of vertices $v, w \in V$;
3      $Q \leftarrow \{s\}$;
4      $S \leftarrow \emptyset$;
5      for $i \leftarrow 1$ to $n$ do
6          Dequeue the first vertex $v$ from beginning of $Q$ such that there are no
            $u \in S$ and $z \in V - S$ with $z \in I_{\mathrm{dom}}(u, v)$;
7          $\sigma(i) \leftarrow v$;
8          $S \leftarrow S \cup \{v\}$;
9          foreach unnumbered vertex $w$ adjacent to $v$ do
10             if $w \notin Q$ then Enqueue $w$ to end of $Q$;
```

Any such ordering $\sigma := (v_1, \ldots, v_n)$ obviously has the property that for every $i \in \{1, \ldots, n\}$ the induced subgraph $G[\{v_1, \ldots, v_i\}]$ is connected. This is already an improvement on the orders produced by the algorithm given in [38] and in Figure 1.4 we compare orders computed by the different algorithms. On the other hand, returning to the example given in the introduction, the P_{2k+1} path graph, we can see that starting the BFS$^{\mathrm{conv}}$ in vertex $k + 1$ still yields an undesirable order.

Starting in an admissible vertex, which in the case of P_{2k+1} will be one of the endpoints or one of their neighbours, is an easy remedy of this problem. However, with a little modification to our search routine we can not only solve this issue, but make an intriguing link with the AT-free graphs characterisation through the spine property. We shall call a vertex ordering $\sigma = (v_1, \ldots, v_n)$ a *monotone dominating pair order* if for every $i \in \{1, \ldots, n\}$ the vertices v_1 and v_i form a dominating pair in the induced subgraph $G[v_1, \ldots, v_i]$.



BFS$^{\text{conv}}(G, 1)$: $(1, 2, 3, 4, 5)$



BFS$^{\text{conv}}(G', v_1)$: $(v_1, v_2, v_3, 1, 2, 3, 5, 4)$

Figure 1.5: Graph for which BFS$^{\text{conv}}$ does not necessarily output a monotone dominating pair ordering and the graph G' constructed from G as in Theorem 1.5.10.

Theorem 1.5.8 (Corneil et al. [40]). *Let $G = (V, E)$ be a connected AT-free graph and suppose that s is an admissible vertex. Let $\sigma = (v_1, \ldots, v_n)$ be a vertex order produced by LBFS (G, s). Then for any $i \in \{1, \ldots, n\}$ the vertices v_1 and v_i form a dominating pair of $G[v_1, \ldots, v_i]$, i.e., σ is a monotone dominating pair order.*

In the following, we will prove an analogous result for BFS$^{\text{conv}}$.

Lemma 1.5.9. *Let $G = (V, E)$ be an AT-free graph and let s be an admissible vertex of eccentricity $k > 2$. If $\sigma := (s = v_1, \ldots, v_n = t)$ is the output of BFS$^{\text{conv}}(G, s)$, then s and t form a dominating pair.*

Proof. Suppose s and t are not a dominating pair. Then there is an s-t-path P and a vertex $w \in V$ such that P avoids w. W.l.o.g. we can assume that P is induced. As s is admissible and $sw, st \notin E$ we must assume that t intercepts every w-s-path. Therefore, w must be in the distance layer $L_G^k(s)$ and $N_s^{k-1}(w) \subseteq N_s^{k-1}(t)$. As $k > 2$, we can deduce that $t \in I_{\text{dom}}(w, s)$ which is a contradiction to σ being an AT-free order. $\square$

However, applying a BFS$^{\text{conv}}$ with an admissible start vertex must not always result in a monotone dominating pair order, as can be seen in Figure 1.5.

Corneil et al. [39] showed that for an AT-free graph G and an admissible vertex s the graph G' obtained by adding a pendant vertex v to s is also AT-free and v is admissible in G'. With this operation we can artificially raise the eccentricity of our starting vertex and generalise Lemma 1.5.9 to all AT-free graphs.

Theorem 1.5.10. *Let G be a connected AT-free graph. For every admissible vertex s there is a vertex ordering σ beginning in s that is both AT-free and a monotone dominating pair ordering.*

Proof. We construct an auxiliary graph by adding a three vertex path to s in the following way: $G' = (V \cup \{v_1, v_2, v_3\}, E \cup \{v_1v_2, v_2v_3, v_3s\})$. As s is admissible, the graph G' is again AT-free and v_1 is admissible in G' with $\text{ecc}_{G'}(v_1) > 2$. The order $\sigma' = (v_1, v_2, v_3, w_1, \ldots, w_n)$ that is generated by BFS$^{\text{conv}}(G', v_1)$ is an AT-free order and with Lemma 1.5.9 it is easy to see that $\sigma = (w_1, \ldots, w_n)$ is a monotone dominating pair order for G. $\square$

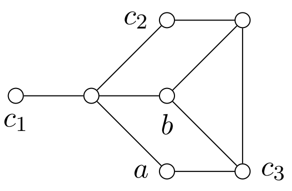


Figure 1.6: An AT-free graph that fulfils neither the strong nor the regular Chvátal Property: One can see that $b \in I_{\text{dom}}[a, c_2]$, while $c_2 \in I_{\text{dom}}[c_1, c_3]$. However, the vertex b is contained neither in $I_{\text{dom}}[c_1, c_3]$ nor in $I_{\text{dom}}[a, c_1]$, nor in $I_{\text{dom}}[a, c_3]$. Furthermore, the domination convexity on this graph has Carathéodory number at least 3, as $\{a, c_1, c_3\}$ form an irredundant set of size 3.

All the characterising properties of AT-free graphs presented here can be summed up in a structure theorem of the type given for the other graph classes studied in this chapter.

Theorem 1.5.11. *For any graph $G = (V, E)$ the following properties are equivalent:*

(i) The graph G is AT-free;

(ii) For any $a, b, z \in V$ such that $z \in I_{\text{dom}}[a, b]$ it holds that $I_{\text{dom}}[a, z] \subseteq I_{\text{dom}}[a, b]$ and $I_{\text{dom}}[z, b] \subseteq I_{\text{dom}}[a, b]$;

(iii) The domination convexity of G is a convex geometry;

(iv) G possesses a domination-convexity (AT-free) order;

(v) G possesses a domination-convexity (AT-free) order that is a BFS order and a monotone dominating pair order.

1.5.2 The Carathéodory Number of Domination Convexity

While Chang et al. [28] show that domination convexity is in fact a convex geometry for AT-free graphs, they also give an example showing that both the strong and the regular Chvátal Property, which we have used previously to bound the Carathéodory number, do not hold (see Figure 1.6). Furthermore, they also give an example of an AT-free graph whose domination convexity has Carathéodory number 3 (see Figure 1.6). While it is disappointing that AT-free graphs do not possess the type of 1-dimensional structure which a Carathéodory number of 2 implies, we were not able to construct examples of graphs with higher Carathéodory number than 3 and in fact, we conjecture that this is the upper bound for AT-free graphs.

Conjecture 1.5.12. *Let $G = (V, E)$ be an AT-free graph. Then $(V, \mathcal{C})$ has a Carathéodory number of at most 3. This bound is tight.*

In the following, we will discuss several properties of domination intervals on AT-free graphs that will hopefully help to serve as a first step in the proof of Conjecture 1.5.12. The main drift of the following arguments can be summarised as follows: As we wish to show that the Carathéodory number does not exceed 3, we can use Lemma 0.5.10 which states that a convexity has Carathéodory number c if c is the smallest number such that every set of size $c+1$ is redundant. Therefore, our goal is to prove that every 4-element subset of the vertex set of an AT-free graph is redundant with regard to domination convexity.

To this end, we will proceed to analyse the convex hulls of all sets of size ≤ 4. An important tool in this analysis is the concept of the *index of a vertex* with regard to the convex hull of a subset of V.

Definition 1.5.13. *Let $(\mathcal{C}, V)$ be an interval convexity with interval operator I and let X be a subset of V. The* index of an element $z \in \operatorname{conv}(X)$ with regard to X *is defined inductively in the following way: For all $x \in X$ we have $\operatorname{ind}_X(x) = 0$ and for $z \in \operatorname{conv}(X)$ we say that $\operatorname{ind}_X(z)$ is the smallest number k such that there exist $a, b \in \operatorname{conv}(X)$ with $\max(\operatorname{ind}_X(a), \operatorname{ind}_X(b)) \leq k-1$ and $z \in I[a,b]$.*

The index describes, in a way, how deep a vertex is nested in the convex hull of a set and is closely linked to redundant sets.

Observation 1.5.14. *Let $(\mathcal{C}, V)$ be an interval convexity with interval operator I and let X be a subset of V. If the largest index of an element of $\operatorname{conv}(X)$ is k and $|X| > 2^k$, then X is redundant.*

We will use the analysis of the highest index of some vertex in a convex hull to decide whether a given set is redundant. The following two lemmas show properties of the domination intervals that will be important tools in the following.

Lemma 1.5.15. *Let $G = (V,E)$ be an AT-free graph, with $y \in I_{\text{dom}}[x,z]$ and $z \in I_{\text{dom}}[w,y]$ for $w,x,y,z \in V$. Then $y \in I_{\text{dom}}[w,x]$ and $z \in I_{\text{dom}}[w,x]$.*

Proof. We can assume that $wx \notin E$ as otherwise $\{y,x,z\}$ form an asteroidal triple. Let P be a z-avoiding x-y-path. If w is adjacent to a vertex on P, then $\{w,y,z\}$ form an asteroidal triple. Joining P to the w-avoiding y-z-path yields a w-avoiding x-z-path.

Let Q be a y-avoiding w-z-path. If x is adjacent to a vertex on Q, then $\{x,y,z\}$ form an asteroidal triple. Joining Q to the x-avoiding y-z-path yields an x-avoiding w-y path. Therefore, $y \in I_{\text{dom}}[w,x]$ and $z \in I_{\text{dom}}[w,x]$. □

As we have already shown for some other graph convexities, the domination intervals of AT-free graphs are, in fact, domination convex sets.

Lemma 1.5.16. *Let $G = (V,E)$ be an AT-free graph and $X = \{x_1, x_2\} \subseteq V$ of cardinality 2. Then every interval is convex, i.e., any $v \in \operatorname{conv}(X)$ has index at most 1 with regard to X.*

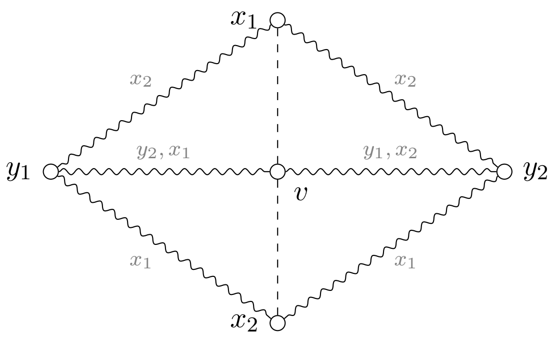


Figure 1.7: Explanatory figure for Lemma 1.5.16. The dashed lines are non-edges and the waved lines are paths. A grey edge-labelling symbolizes that this vertex misses the path.

Proof. Suppose, there is a vertex $v \in V$ with index 2. Then there are $y_1, y_2 \in \operatorname{conv}(X)$ with $v \in I_{\text{dom}}[y_1, y_2]$ such that without loss of generality y_1 has index 1. Suppose, y_2 has index 0. Then again without loss of generality $y_2 = x_1$. As $y_1 \in I_{\text{dom}}[x_1, x_2]$ Theorem 1.5.2 implies $v \in I_{\text{dom}}[x_1, x_2]$ which is a contradiction to the assumption that v has index 2. Therefore, y_2 also has index 1, i.e., $y_1 \in I_{\text{dom}}[x_1, x_2]$ and $y_2 \in I_{\text{dom}}[x_1, x_2]$.

If v is adjacent to x_1, then $\{y_1, y_2, x_1\}$ form an asteroidal triple, as there are a y_1-v-path avoiding y_2 and a y_2-v-path avoiding y_1 and there are x_1-avoiding paths from both y_1 and y_2 to x_2. The same argument holds for x_2 and we can assume that $vx_1 \notin E$ and $vx_2 \notin E$. Let P be a y_2 avoiding v-y_1-path. Without loss of generality x_1 is not adjacent to any vertex on P, as otherwise there is a y_2-avoiding x_1-x_1-path and as $y_2 \in I_{\text{dom}}[x_1, x_2]$ the vertices $\{x_1, x_2, y_2\}$ form an asteroidal triple.

Suppose x_2 intercepts a y_1-avoiding y_2-v-path. Then for the same reason as above x_1 does not intercept this path and, as there are x_2-avoiding paths from both y_1 and y_2 to x_1, we see that $y_2 \in I_{\text{dom}}[y_1, x_2]$. As $v \in I_{\text{dom}}[y_1, y_2]$ Theorem 1.5.2 implies $v \in I_{\text{dom}}[y_1, x_2]$ which is a contradiction to v being of index 2, as shown above.

Therefore, there are both x_1-avoiding x_2-v and x_2-avoiding x_1-v-paths implying $v \in I_{\text{dom}}[x_1, x_2]$. This is a contradiction to the assumption.

Hence, there are no vertices in $\operatorname{conv}(X)$ with index 2 with regard to X. Due to the definition of the index, this implies that every vertex has index less or equal to 1. □

The following lemma forms a restricted version of the Chvátal Property which only holds when all vertices are independent.

Lemma 1.5.17. *Let $G = (V, E)$ be an AT-free graph, and let $a, c_1, c_3 \in V$ be independent. Then for $c_2 \in I_{\text{dom}}[c_1, c_3]$ and $b \in I_{\text{dom}}[a, c_2]$, either $b \in I_{\text{dom}}[a, c_1]$ or $b \in I_{\text{dom}}[a, c_3]$ or $b \in I_{\text{dom}}[c_1, c_3]$.*

Proof. Without loss of generality, we can assume that c_1 is not adjacent to the c_2-avoiding a-b-path, as otherwise $\{c_1, c_2, c_3\}$ would be an asteroidal triple. If there is an a-avoiding c_1-c_2-path, then $b \in I_{\text{dom}}[a, c_1]$ and we are done. Suppose there is an a-avoiding c_1-c_2-path. If c_3 is adjacent to any vertex on the c_2-avoiding a-b-path, then

$c_3 \in I_{\text{dom}}[a, c_2]$. As a is adjacent to the c_3-avoiding c_1-c_2-path, the set $\{a, c_2, c_3\}$ forms an asteroidal triple. If c_3 is not adjacent to any vertex on the c_2 avoiding a-b-path, then $z \in I_{\text{dom}}[a, c_3]$ and we are done. Therefore, we can assume that a is adjacent to the c_1-avoiding c_2-c_3 path. Therefore, a is in the interval between c_1 and c_3 and by Lemma 1.5.16 we have $b \in I_{\text{dom}}[c_1, c_3]$. □

This property can be used to analyse the index of vertices generated in the convex hull of independent sets. First we shall study the case of sets of size 3.

Lemma 1.5.18. *Let $G = (V, E)$ be an AT-free graph and let $x_1, x_2, x_3 \in V$ be independent. Then any vertex in $\operatorname{conv}(\{x_1, x_2, x_3\})$ has index at most 1.*

Proof. Suppose there is a vertex z in $\operatorname{conv}(\{x_1, x_2, x_3\})$ with index 2. Then there are vertices y_1 and y_2 such that without loss of generality $y_1 \in I_{\text{dom}}[x_1, x_2]$ and y_2 is of index less or equal 1.

If $y_2 \in I_{\text{dom}}[x_1, x_2]$, then $z \in I_{\text{dom}}[x_1, x_2]$ by Lemma 1.5.16; this is a contradiction to the assumption. If y_2 is, in fact, x_3, then by Lemma 1.5.17 z must have index 1. Therefore, we can assume without loss of generality that $y_2 \in I_{\text{dom}}[x_1, x_3]$.

Case 1 ($x_2y_2 \notin E$): By Lemma 1.5.17 we see that $z \in I_{\text{dom}}[x_1, x_2]$ or $z \in I_{\text{dom}}[x_1, y_2]$ or $z \in I_{\text{dom}}[x_2, y_2]$. The first two cases imply by Theorem 1.5.2 that the index of z is 1; a contradiction. Therefore, $z \in I_{\text{dom}}[x_2, y_2]$ and $y_2 \in I_{\text{dom}}[x_1, x_3]$ and Lemma 1.5.17 again implies that the index of z is 1.

Case 2 ($x_3y_1 \in E$): By symmetry to the above case this also yields a contradiction.

Case 3 ($x_2y_2 \in E$ and $x_3y_1 \in E$): The vertex x_1 cannot be adjacent to z, as otherwise y_1, x_1, x_2 form an asteroidal triple. If $zx_2 \notin E$, then z must be in the interval between x_2 and x_1. By symmetry the same must hold for x_3. Hence, we can assume that both x_2 and x_3 are adjacent to z. Altogether, this implies that $\{x_1, x_2, x_3\}$ form an asteroidal triple.

As all cases lead to a contradiction, there cannot be a vertex in $\operatorname{conv}(\{x_1, x_2, x_3\})$ with index 2, and thus every vertex has index less or equal 1. □

This result can be generalised to independent sets of arbitrary size.

Theorem 1.5.19. *Let $G = (V, E)$ be an AT-free graph and let $\{x_1, x_2, \ldots, x_k\} \in V$ be independent. Then any vertex in $\operatorname{conv}(\{x_1, x_2, \ldots, x_k\})$ has index at most 1.*

Proof. We will prove this theorem by induction over k. The induction basis is given for $k = 2$ and $k = 3$ in Lemmas 1.5.16 and 1.5.18, respectively. Now let us assume that the statement holds for all $l < k$.

Let $X = \{x_1, x_2, \ldots, x_k\} \in V$ be an independent set. Let $y \in \operatorname{conv}(X)$ be a vertex of index 2. Hence, $y \in I_{\text{dom}}[a, b]$ for some $a, b \in \operatorname{conv}(X)$ with $\mathit{index}(a), \mathit{index}(b) \leq 1$. By the induction hypothesis, we know that every vertex in $\operatorname{conv}(X')$ for $X' \subsetneq X$ has index at most 1. Therefore, y is not in the convex hull of a true subset of X. Suppose that without loss of generality $a \in X$. Note that not both can be elements of X, as $\mathit{index}(y) = 2$.

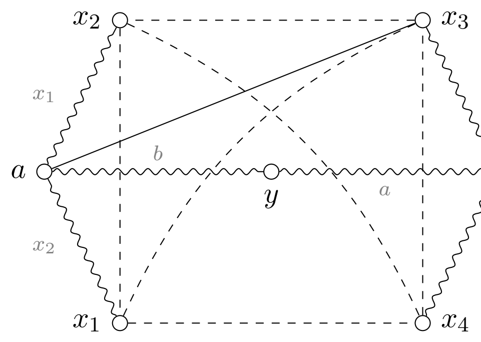


Figure 1.8: Explanatory figure to proof of Theorem 1.5.19. The waved lines form paths. A grey edge-labelling symbolizes that the pass avoids this vertex.

However, then we see that $y \in \operatorname{conv}(x_i, x_j, x_r)$ with $x_r = a$ and $b \in I_{\text{dom}}[x_i, x_j]$; a contradiction, as $k \geq 4$.

This implies that $index(a) = index(b) = 1$ and without loss of generality, we can assume that $a \in I_{\text{dom}}[x_1, x_2]$ and $b \in I_{\text{dom}}[x_3, x_4]$, due to the fact that y is not contained in the convex hull of a true subset of X.

Suppose that both x_3 and x_4 are not adjacent to a. Then a, y, b, x_3 and x_4 together fulfil the prerequisites of Lemma 1.5.17, as a, x_3 and x_4 are independent. This implies that y is contained in either $I_{\text{dom}}[x_3, x_4]$, $I_{\text{dom}}[a, x_3]$ or $I_{\text{dom}}[a, x_4]$. Either way, the vertex y is contained in the convex hull of a true subset of X, which is a contradiction.

On the other hand, if without loss of generality x_3 is adjacent to a, then there is an x_1-avoiding x_2-x_3-path and an x_2 avoiding x_1-x_3 path. As a result, x_3 is contained in $I_{\text{dom}}[x_1, x_2]$. Therefore, we see that $\operatorname{conv}(X) \subseteq \operatorname{conv}(X - x_3)$; this is a contradiction to our assumptions, proving the theorem. □

As every vertex of index 1 is generated by at most two vertices, Theorem 1.5.19 implies that any independent set of size ≥ 3 is redundant.

Corollary 1.5.20. *If G is an AT-free graph, then any independent set of vertices of G of size ≥ 3 is redundant with regard to domination convexity.*

This corollary implies that were we to restrict the definition of the Carathéodory number to only independent sets X, this modified Carathéodory would be bounded by 2, just as is the case in the graph classes studied before. However, we have seen in Figure 1.6 that problems arise when a set X contains at least one edge.

For sets of size 3 we can even show a bit more.

Lemma 1.5.21. *Let $G = (V, E)$ be an AT-free graph and let $x_1, x_2, x_3 \in X$ be such that $G[X]$ is an induced P_3. Then any vertex in $\operatorname{conv}(\{x_1, x_2, x_3\})$ has index at most 1.*

Proof. Suppose without loss of generality that $x_1x_3 \notin E$ and let $y \in I_{\text{dom}}[x_1, x_3]$. If y and x_2 are not adjacent, then $x_1{-}x_2{-}x_3$ forms a y-avoiding x_1-x_3-path, as $G[X]$ is an induced P_3. This implies that $\{x_1, y, x_3\}$ form an asteroidal triple; a contradiction. Therefore, any vertex in $I_{\mathrm{dom}}[x_1, x_3]$ is adjacent to x_2 and $\mathrm{conv}(x_1, x_2, x_3) = \mathrm{conv}(\{x_1, x_3\})$. Due to Lemma 1.5.16, every vertex in $\mathrm{conv}(x_1, x_2, x_3)$ has index at most 1. □

This implies that a set of size 3 is redundant both if it is independent and if it contains ≥ 2 edges. With the example given in Figure 1.6 we see that the following holds.

Corollary 1.5.22. *Let G be an AT-free graph. If a set $X \subseteq V$ of cardinality 3 is irredundant, then it contains exactly one edge.*

This fact makes it difficult to characterise redundant and irredundant sets of cardinality 4. It has already been shown that independent sets are redundant. The following lemma shows that sets containing ≥ 5 edges are also redundant.

Lemma 1.5.23. *Let $G = (V, E)$ be an AT-free graph and let $\{x_1, x_2, x_3, x_4\} \subseteq X$ be such that $G[X]$ contains ≥ 5 edges. Then any vertex in $\mathrm{conv}(\{x_1, x_2, x_3, x_4\})$ has index at most 1.*

Proof. If $G[X]$ contains 6 edges, then it must be the complete graph and therefore, all intervals between the vertices of X contain only their endpoints. This implies that $\mathrm{conv}(X) = X$ and we are done. Hence, we assume that $G[X]$ contains exactly 5 edges. Without loss of generality let x_1 and x_2 be the unique pair of nonadjacent vertices of X. Just as in the proof of Lemma 1.5.21, any vertex in the interval between x_1 and x_2 must be adjacent to both x_3 and x_4, as these are adjacent to both x_1 and x_2.

Therefore, all vertices of index 1 are in the interval between $x_1$and x_2 and there can be no vertices of index 2, due to Lemma 1.5.16. □

There are essentially two different graphs with four edges that can be induced by four vertices: This can either be a triangle with a pendant or an induced C_4. For the second case we get the following result.

Lemma 1.5.24. *Let $G = (V, E)$ be an AT-free graph and let $\{x_1, x_2, x_3, x_4\} \subseteq X$ be such that $G[X]$ is an induced C_4. Then any vertex in $\mathrm{conv}(\{x_1, x_2, x_3, x_4\})$ has index at most 1.*

Proof. Let $y \in \mathrm{conv}(X)$ be of index 2. Then there exist vertices $a, b \in \mathrm{conv}(X)$ with $\mathrm{ind}_X(a) \leq 1$ and $\mathrm{ind}_X\, b \leq 1$. Suppose that without loss of generality $\mathrm{ind}_X(b) = 0$ and $b = x_1$. Furthermore, we can assume without loss of generality that $a \in I_{\mathrm{dom}}[x_2, x_3]$, as otherwise y is in the convex hull of two vertices of X and cannot have index 2. As $G[X]$ is an induced C_4 and x_2 and x_3 are not adjacent, both x_2 and x_3 are adjacent to x_1. Similarly to the previous lemmas, this implies that a is adjacent to x_1, as otherwise x_2, x_3 and b form an asteroidal triple. This is a contradiction to the fact that $y \in I_{\mathrm{dom}}[x_1, b]$, proving the statement. □

For a set $X \subseteq V$ we call a vertex $v \in \mathrm{conv}(X)$ *critical with regard to X* if v is not contained in any convex hull of a true subset of X. Obviously, any irredundant set must contain at least one critical vertex. Therefore, we need to examine what properties must hold for a 4-element set to contain a critical vertex.

Lemma 1.5.25. *If $G = (V, E)$ is an AT-free graph and $X = \{x_1, x_2, x_3, x_4\} \subseteq V$ is a set of cardinality 4, such that* $\operatorname{conv}(X)$ *contains a critical vertex of index 2, then $G[X]$ contains exactly one edge.*

Proof. Let $y \in \operatorname{conv}(X)$ have index 2. Hence, $y \in I_{\mathrm{dom}}[a, b]$ for some $a, b \in \operatorname{conv}(X)$ and as y is critical and thus, not in the convex hull of a true subset of X we see that $index(a) = index(b) = 1$. Without loss of generality, we can assume that $a \in I_{\mathrm{dom}}[x_1, x_2]$ and $b \in I_{\mathrm{dom}}[x_3, x_4]$, again due to the fact that y is critical.

Suppose that $P_{a,y}(b)$ is a b-avoiding a-y-path and that $P_{b,y}(a)$ is an a-avoiding b-y-path. If both x_3 and x_4 see $P_{a,y}(b)$, then there is a b-avoiding x_3-x_4-path and G contains an asteroidal triple in $\{b, x_3, x_4\}$; a contradiction. Without loss of generality x_3 misses $P_{a,y}(b)$. With the same argument we can assume that without loss of generality x_1 misses $P_{b,y}(a)$. Suppose a misses an x_3-b path. Then $y \in I_{\mathrm{dom}}[a, x_3]$; a contradiction to $y \notin X^*$. Therefore, a intercepts every x_3-b-path. By symmetry b intercepts every x_1-a path. By a similar argument, y must see every x_3-b-path and every x_1-a-path.

Case 1 ($x_4a \notin E$ and $x_4y \in E$): This implies $a \in I_{\mathrm{dom}}[x_4, x_3]$; a contradiction to $y \notin X^*$.

Case 2 ($x_4a \in E$ and $x_4y \notin E$): This implies $y \in I_{\mathrm{dom}}[x_3, x_4]$; a contradiction to $y \notin X^*$.

Case 3 ($x_4a \notin E$ and $x_4y \notin E$): Suppose there is a y-avoiding x_4-b-path. Then $b \in I_{\mathrm{dom}}[x_4, y]$ and as $y \in I_{\mathrm{dom}}[a, b]$ by Lemma 1.5.15 we have $y \in I_{\mathrm{dom}}[a, x_4]$; again a contradiction to $y \notin X^*$. Therefore, y intercepts every x_4-b-path and $y \in I_{\mathrm{dom}}[x_3, x_4]$; another contradiction.

As a result $x_4a \in E$ and $x_4y \in E$ and by symmetry $x_2y \in E$ and $x_2b \in E$ must hold.

Only one of x_1 and x_2 can be adjacent to x_3, as otherwise $\{x_1, a, x_2\}$ form an asteroidal triple.

Case 1 ($x_2x_3 \in E$ and $x_3x_1 \notin E$): Then $y \in I_{\mathrm{dom}}[b, x_1]$; a contradiction to $y \notin X^*$.

Case 2 ($x_1x_4 \in E$ and $x_3x_1 \notin E$): Then $y \in I_{\mathrm{dom}}[a, x_3]$; a contradiction to $y \notin X^*$.

Case 3 ($x_3x_1 \in E$ and $x_2x_3 \notin E$): Then $a \in I_{\mathrm{dom}}[x_3, x_2]$; a contradiction to $y \notin X^*$.

As a result, we can assume that $x_1x_3 \notin E$ and $x_2x_3 \notin E$ and therefore, $\{x_1, x_2, x_3\}$ forms an independent set. Furthermore, we see that x_4 is adjacent to neither x_1 nor x_3. As we have already shown in Theorem 1.5.19 that for independent sets X there are no vertices of index 2 with regard to X, we see that $x2$ and x_4 must be adjacent. Altogether, this proves the statement of the lemma. □

In order to prove Conjecture 1.5.12 in the way attempted here, a few more steps are needed. One would need to show that in any AT-free graph and any set X, a vertex v cannot have index larger than 2 with regard to X. This fact in combination with Lemma 1.5.25 could then be used to show that a set of cardinality 4 cannot contain any critical vertices and must therefore be redundant.

Should it turn out that there are vertices of index larger than 2, then these partial results could be used in a complete case analysis of all possible combinations of 4 element

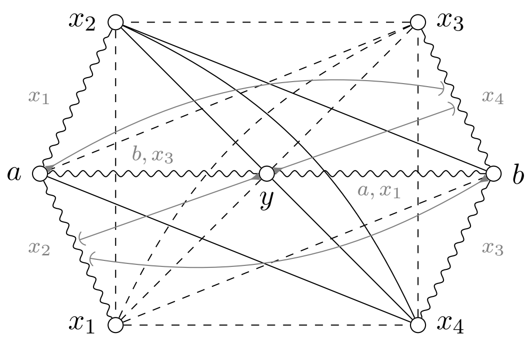


Figure 1.9: Explanatory figure to Theorem 1.5.25. The dashed lines are non-edges and the waved lines are paths. A grey edge-labelling symbolizes that this vertex misses the path. A grey edge from a vertex to a path symbolizes that this vertex intercepts every such path.

sets. However, we have already made some attempts to this end and it seems unlikely that such an approach will be successful without some new techniques.

1.6 Conclusion

We have given an overview of some of the most important graph classes that can be characterised with a convex geometry. Comparing these characterisations, we see that many concepts such as leaves, simplicial vertices and admissible vertices can be defined in a standard way as extreme points of these convexities. Furthermore, all of these convex geometries yield interesting new vertex order characterisations that should be compared in more detail with their known counterparts.

With this overview we have attempted to show that the language of convexity can be very useful to compare analogous structures among different graph classes. We have seen that many results, such as the characterisation of interval graphs as chordal AT-free graphs, can be transferred to results on convexities. This indicates that convexity might be used as a unifying structure in order to better classify the many diverse concepts of algorithmic graph theory.

In an attempt to find a most general convexity that is a convex geometry for all graphs, we have introduced the notion of interception convexity. This convexity has as its extreme points the set of avoidable vertices in a graph, a concept that we will study in greater detail in Chapter 3 and which we will see to be very useful.

Using a result by Chvátal [34], we have given a characterisation of interval graphs with a convex geometry. Using this approach, it was possible to show that the Carathéodory number of this convexity is bounded by 2, settling a question stated in [3]. Furthermore, this convex geometry yields a new vertex order characterisation of interval graphs. It remains to be shown that this vertex order characterisation can be checked in linear time. As such an order can be computed in linear time, this would imply a new linear recognition algorithm for interval graphs.

While studying the convexity of AT-free graphs, we resolved an open question from [38] by proving that any given AT-free graph has an AT-free order that coincides with a BFS order. The proof implies a polynomial time algorithm for the computation of such an order that is at least as fast as the computation of the domination intervals of that graph. As a result, we were able to show that there is a close link between the vertex order characterisation of AT-free graphs, and their characterisation through the spine property. As checking whether a vertex order is an AT-free order is, in fact, as difficult as recognising AT-free graphs, it should still be possible to find AT-free orders in linear time. This could be done by giving a linear time implementation of BFS^{conv}, or by constructing another search scheme with similar structural properties.

Linear vertex orderings of other graph classes, such as interval orderings or cocomparability orderings, have found many applications in optimisation algorithms on these classes. To the best of our knowledge, no such results are known with respect to AT-free orderings. By using AT-free BFS orderings, such results might be easier to attain. Two of the most likely candidates are the independent set problem and the vertex colouring problem. Should it be possible to compute AT-free orders in linear time, it might even be possible to develop robust optimisation algorithms (see [133]) on AT-free graphs, similar to the maximum clique algorithm on comparability graphs. It is still an open question whether every AT-free graph admits a DFS order whose reversal is AT-free [38].

Finally, we analysed the the Carathéodory number of the convexity on AT-free graphs. While we were not able to give a bound on this number, our results suggest that this bound is 3. We have made some progress in proving this conjecture. However, there still remains much to be shown. Due to the nature of the Carathéodory number, this would imply that AT-free graphs are in some way "two-dimensional", in the same sense that interval graphs are "one-dimensional" as intersection graphs of intervals on a line. This raises the question whether AT-free graphs can be characterised using some form of intersection model which mirrors this "two-dimensionality".

2 Bilateral AT-free Graphs and Orders

We have seen in the previous chapter that for nearly all of the presented graph convexities the class of graphs having a corresponding halfspace ordering is easily characterised. For trees it was shown that these graphs are exactly the claw-free trees, i.e., the paths. For chordal and interval graphs this defines exactly the class of unit interval graphs. In the case of AT-free graphs, however, this question is not as easy to resolve.

In this chapter, we will show that the class of graphs having an AT-free halfspace ordering, which from now on will be called *bilateral AT-free*, is difficult to characterise and is, in fact, $\mathcal{NP}$-hard to recognise. Note that in this chapter we will use mainly *open* domination intervals, as it makes notation easier here.

Definition 2.0.1. *A graph $G = (V, E)$ is* bilateral asteroidal triple free *if and only if there exists an ordering σ of V such that for any triple a, b, c where $a \in I_{\text{dom}}(b, c)$, we have $b \prec_\sigma a \prec_\sigma c$.*

We will show how this class relates to other known subclasses of AT-free graphs. Furthermore, using results from the previous chapter, we present some subclasses of AT-free graphs which we show to be bilateral AT-free. For these classes, we present linear time algorithms to compute a bilateral AT-free ordering.

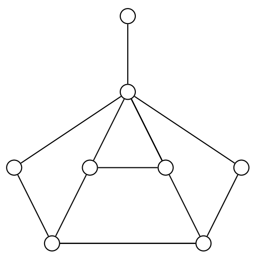
2.1 Subfamilies of AT-free Graphs

Before we turn to the problem of recognising bilateral AT-free graphs, we will first compare this class to other known subfamilies of AT-free graphs. Figure 2.1a shows a graph which is AT-free and not bilateral AT-free. This shows that the family of bilateral AT-free graphs is a true subset of AT-free graphs. However, this class still contains important subclasses of AT-free graphs.

Lemma 2.1.1 (Mouatadid [116])**.** *Every cocomparability ordering is a bilateral AT-free ordering, i.e., every cocomparability graph is also a bilateral AT-free graph.*

Proof. Suppose there exists a graph G which has a cocomparability ordering σ that is not a bilateral AT-free ordering. In particular, there exist $z, a, b \in V(G)$ with $z \in I_{\text{dom}}(a, b)$ such that $z \prec_\sigma a \prec_\sigma b$. Let Q be the $z - a$ path avoiding b and let P be the $z - b$ path avoiding a. As P is a b-z path and $z \prec_\sigma a \prec_\sigma b$, there exists an edge $uv \in E$ such that $u \prec_\sigma a \prec_\sigma v$ and $ua, va \notin E$. This contradicts the fact that σ is a cocomparability ordering. □

As, for example, the C_5 is bilateral AT-free but not a cocomparability graph, we can state the following corollary.


(a) Graph that is AT-free but not bilateral AT-free.

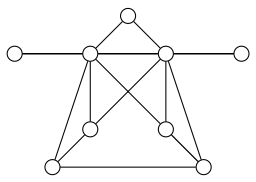

(b) Graph that is bilateral AT-free but neither path-orderable nor strong asteroid free.

Figure 2.1: Examples of AT-free graphs that differentiate between the subclasses.

Corollary 2.1.2. *The class of cocomparability graphs is strictly contained in the class of bilateral AT-free graphs.*

In an attempt to characterise AT-free graphs using linear vertex orderings, Corneil et al. [42] introduced the following subfamily of AT-free graphs.

Definition 2.1.3. *A graph $G = (V, E)$ is* path-orderable *if and only if there exists an ordering σ of V, such that for any three vertices $a, b, c \in V$, where $a \prec_\sigma b \prec_\sigma c$ and $ac \notin E$, any a-c-path contains at least one neighbour of b. Such an ordering is called a* path order*.*

This family can also be shown to be a subfamily of bilateral AT-free graphs.

Lemma 2.1.4 (Mouatadid [116])**.** *Every path-ordering is a bilateral AT-free ordering, i.e., every path-orderable graph is bilateral AT-free*

Proof. Suppose there exists a graph G which has a path-ordering σ that is not a bilateral AT-free ordering. In particular, there exist $z, a, b \in V(G)$ with $z \in I_{\text{dom}}(a, b)$ such that $z \prec_\sigma a \prec_\sigma b$. Since σ is a path-ordering, any z-b-path must contain a neighbour of a; this is a contradiction to the fact that $z \in I_{\text{dom}}(a, b)$. □

The graph shown in Figure 2.1b is not a path orderable graph, but it does have a bilateral AT-free ordering, implying the following.

Corollary 2.1.5. *The class of path orderable graphs is strictly contained in the class of bilateral AT-free graphs.*

Furthermore, we wish to compare bilateral AT-free graphs with the class of *strong asteroid free graphs*, a class that was defined by Corneil et al. [42] as a polynomially recognisable superclass of path orderable graphs. The definition of these graphs is quite involved and probably best understood in context. Therefore, we will defer it to Section 2.2.1. Summing up the relations between the analysed graph classes, we get:

- cocomparability $\subsetneq$ path orderable $\subsetneq$ strong asteroid free $\subsetneq$ AT-free
- cocomparability $\subsetneq$ path orderable $\subsetneq$ bilateral AT-free $\subsetneq$ AT-free

- strong asteroid free $\overset{?}{\subsetneq}$ bilateral AT-free

This leaves us with the following question.

Question 2.1.6. *Does the class of bilateral AT-free graphs contain strong asteroid free graphs or are the two classes incomparable?*

To answer this question we first present a characterisation of bilateral AT-free graphs which will also lead to a proof of the $\mathcal{NP}$-completeness of their recognition.

2.2 The Recognition of Bilateral AT-free Graphs

In order to characterise bilateral AT-free graphs, we closely follow two sources: Gallai's paper on transitively orientable (or comparability) graphs [67] and a paper by Corneil et al. [42] which proves that the recognition of path-orderable graphs is NP-complete. In both of these papers, the properties of the graph class is captured by a respective form of orientation of the complement of the graph, i.e., a claim of the following type is made: a graph G belongs to the graph class $\mathcal{G}$ if and only if its complement $\overline{G}$ has an orientation $\mathcal{O}$ that fulfils the properties $\mathcal{P}$. If the structure of the graph class is defined through a particular linear vertex ordering, in this example a bilateral AT-free ordering, such an orientation is given in a very natural way. Given a characterising linear vertex ordering, an edge uv in the complement is directed from u to v if and only if u appears before v in that ordering. From the properties of such an orientation one can derive the concept of a *forcing relation* which describes in what way the orientation of one edge influences the orientations of other edges. Here we will use the same approach, i.e., we will use a forcing-relation to describe an orientation on the complement graph whose existence is equivalent to G being bilateral AT-free.

To be pertinent, the desired orientation will need to reflect the bilateral AT-free order, i.e., if an edge uv is directed from u to v we want u to be before v in the linear order. This immediately implies that the orientation needs to be acyclic.

In fact, locally directing the edges in the order that they would appear in the bilateral AT-free order yields a simple rule that is already strong enough to characterise bilateral AT-free graphs. Let $G = (V, E)$ be a graph and let uv and vw be edges of $\overline{G}$. We define a relation $\mathcal{F} \subseteq E(\overline{G})$ in the following way: If $u \in I_{\text{dom}}(v, w)$ or $w \in I_{\text{dom}}(u, v)$, then $uv\mathcal{F}'vw$. As this relation is symmetric, the reflexive transitive closure of $\mathcal{F}'$, say $\mathcal{F}$, is an equivalence relation. We call $\mathcal{F}$ the *forcing relation* and say that $e, f \in E(\overline{G})$ *force* each other if they are in the same equivalence class of $\mathcal{F}$. This terminology is motivated by the fact that the orientation of one of the edges in an equivalence class of $\mathcal{F}$ determines the orientation of all other edges of that class. These equivalence classes will be called *forcing classes of* G. We say that an orientation $\mathcal{O}$ of $\overline{G}$ *agrees with the forcing* if for any vertex $v \in V$ such that $uv, vw \in E(\overline{G})$ and $u \in I_{\text{dom}}(v, w)$ or $w \in I_{\text{dom}}(u, v)$ both vw and vu are oriented in the same direction with regard to v (an example can be seen in Figure 2.2).

The existence of an acyclic orientation that agrees with the forcing is already characterising for bilateral AT-free graphs.

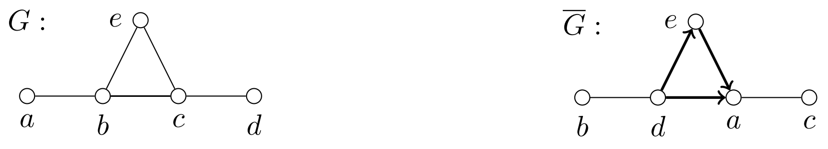


Figure 2.2: A graph G together with its complement which is partly oriented in accordance to the forcing rule. In G one can see that $e \in I_{\text{dom}}(a, d)$. Assuming without loss of generality that in the complement ad is oriented towards a, this implies that the edge de is oriented towards e and that ea is oriented towards a.

Lemma 2.2.1. *A graph $G = (V, E)$ has a bilateral AT-free ordering if and only if there is an acyclic orientation of $\overline{G}$ that agrees with the forcing.*

Proof. Suppose G has a bilateral AT-free ordering σ. We orient the edges of the complement according to this ordering. Suppose $uv, vw \in E(\overline{G})$ such that $u \in I_{\text{dom}}(v, w)$. Then, without loss of generality, $w \prec_\sigma v$ and therefore, $w \prec_\sigma u \prec_\sigma v$. This implies that both uv and wv are directed towards v and the orientation agrees with the forcing. As σ is a linear order, the orientation is acyclic.

Suppose there is an acyclic orientation that agrees with the forcing. Then a topological sort yields a linear ordering σ of the vertices. For $u \in I_{\text{dom}}(v, w)$ we know that in the complement uv and wv are oriented in the same direction with respect to v; also uw and vw are oriented in the same direction with respect to w. This implies that $v \prec_\sigma u \prec_\sigma w$ or $w \prec_\sigma u \prec_\sigma v$, i.e., that σ is a bilateral AT-free order. $\square$

To further understand the forcing relation, we will first need to study some properties of bilateral AT-free orders.

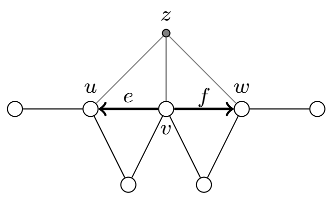
Lemma 2.2.2. *Suppose G has a bilateral AT-free order σ and $u, v, w \in V$, such that $I_{\text{dom}}(u, v) \cap I_{\text{dom}}(u, w) \neq \emptyset$. Then, either $u \prec_\sigma v \wedge w \prec_\sigma v$ or $v \prec_\sigma u \wedge v \prec_\sigma w$. In particular, this implies that $uv\mathcal{F}uw$.*

Proof. Let $z \in I_{\text{dom}}(u, v) \cap I_{\text{dom}}(u, w)$. Suppose $u \prec_\sigma v$. Then $u \prec_\sigma z \prec_\sigma v$ which in turn implies $w \prec_\sigma z \prec_\sigma v$ due to σ being bilateral AT-free. The case where $v \prec_\sigma u$ follows analogously. Furthermore, let $z \in I_{\text{dom}}(u, v) \cap I_{\text{dom}}(u, w)$. Then $uz\mathcal{F}uv$ and $uz\mathcal{F}uw$. By transitivity of $\mathcal{F}$ we can imply that $uv\mathcal{F}uw$. $\square$

This implies a more general statement.

Corollary 2.2.3. *Let $I_{\text{dom}}(a_1, b), \dots, I_{\text{dom}}(a_k, b)$ be a set of intervals such that for $i \in \{1, \dots, k-1\}$ we have $I_{\text{dom}}(a_i, b) \cap I_{\text{dom}}(a_{i+1}, b) \neq \emptyset$. Then either $a_1, \dots, a_k \prec_\sigma b$ or $b \prec_\sigma a_1, \dots, a_k$.*

Using Lemma 2.2.2, it is possible to *force* orientations of edges along induced paths. As can be seen in Figure 2.3, the orientation of the edge vw is forced by the orientation of vu



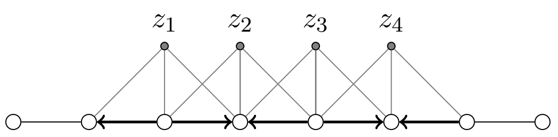


Figure 2.3: On the left we see an example where $z \in I_{\text{dom}}(u, v) \cap I_{\text{dom}}(u, w)$ in the complement. Due to Lemma 2.2.2 the orientation of e forces the orientation of f. This idea can be used to force orientations along induced paths, as can be seen in the graph on the right.

Lemma 2.2.4. *Let $G = (V, E)$ be a graph such that there are $a_1, \dots, a_{2k+1} \in V(G)$ with $I_{\text{dom}}(a_1, a_2) \cap I_{\text{dom}}(a_2, a_3) \neq \emptyset, \dots, I_{\text{dom}}(a_{2k+1}, a_1) \cap I_{\text{dom}}(a_1, a_2) \neq \emptyset$. Then G does not have a bilateral AT-free order.*

Proof. As any reverse of a bilateral AT-free order σ is also bilateral AT-free, we can assume without loss of generality that $a_1 \prec_\sigma a_2$. Using the previous claim, this implies that $a_3 \prec_\sigma a_2$. By repeatedly applying Lemma 2.2.2, we see that $a_{2k+1} \prec_\sigma a_1$. Due to the fact that $I_{\text{dom}}(a_{2k+1}, a_1) \cap I_{\text{dom}}(a_1, a_2) \neq \emptyset$, this implies $a_2 \prec_\sigma a_1$ by Lemma 2.2.2; this is a contradiction to the assumption that $a_1 \prec_\sigma a_2$. □

Using all the gathered information about forcings, we can turn our attention to the recognition of bilateral AT-free graphs. If a given graph has only one forcing class, it is easy to see that one can decide in polynomial time whether a bilateral AT-free order exists. We orient one of the edges arbitrarily and use the forcing rules to derive the orientations of all the other edges. These rules can be attained by computing all domination intervals of the graph, which can be done in polynomial time. Similarly, if the graph has a constant number k of forcing classes, we compute both possible orientations for each class in polynomial time and for each of the 2^k different combinations of orientations of these classes we check whether there is an oriented cycle (also in polynomial time).

Problems arise when the number of forcing classes is dependent on the size of the graph. A related approach to the one taken here was used in [67] to recognise comparability graphs. In that case, it was possible to break the interdependence of the different forcing classes using modular decomposition and, as a result, the number of forcing classes was not relevant to the complexity of recognition. For bilateral AT-free graphs this, unfortunately, is not the case. In fact, we will show that these graphs are NP-complete to recognise. This is shown by giving a reduction to the well known NP-complete decision problem NOT-ALL-EQUAL 3-SAT [128]. We use the problem description given in [68].

NOT-ALL-EQUAL 3-SAT

Instance:	An instance $\mathcal{I}$ consisting of a set X of variables and a collection C of clauses over X such that each clause $c \in C$ has $\lvert c \rvert = 3$.
Task:	Find a truth assignment for U such that each clause in C has at least one true literal and at least one false literal.

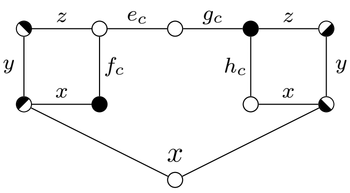


Figure 2.4: Gadget for the clause $c = x \vee y \vee z$

Given an instance $\mathcal{I}$ of NOT-ALL-EQUAL 3-SAT, we construct a corresponding graph $G(\mathcal{I})$ whose complement is bilateral AT-free if and only if $\mathcal{I}$ has a truth assignment such that each clause in C has at least one true literal. In principal, the construction works as follows: For every variable there is a vertex labelled by that variable. For every clause $x \vee y \vee z \in C$ we introduce a gadget as shown in Figure 2.4. Each of these gadgets consists of two C_4s and for each of these, three of the edges correspond to the three literals in the clause. For the variable x the example in Figure 2.4 shows how the gadget is attached to the variable-vertices. The vertex of the edge labelled x that is adjacent to the vertex labelled x is called the *base* of that edge.

Depending on whether all incident edges to this vertex are outgoing (ingoing), the variable is set to true (false). Due to construction, any orientation adhering to the forcing will fall into one of these cases. Given an orientation, a positive literal will be deemed to be true if it is directed towards its base. If it is directed away from its base, it will be false. A negative literal behaves in the opposite way.

Due to construction, the edges that are not labelled will always be directed in opposite directions, in the sense that one will be directed away and one towards the vertices connecting the two C_4s. Thus, a truth-assignment in which all three literals are either all true or all false will always yield a directed cycle in exactly one of the two C_4s. Vice versa, an orientation that agrees with the forcing always yields a truth assignment that has at least one true and at least one false literal.

Coming to the details of our construction, we first have to clarify how the variable vertices are connected to the clause-gadgets. To this end, we say that each edge has one positive and one negative vertex (where it must be added that a vertex can be positive for one edge and negative for another). In the example in Figure 2.4, the positive vertices are white and the negative vertices are white. If a variable v occurs in a clause we will add an edge between that variable and the positive (negative) vertex of the literal-edge if v if that literal is positive (negative).

We have claimed that, in an orientation agreeing with the forcing, all variable vertices have either only ingoing or only outgoing edges. To this end, for every variable vertex x we add an auxiliary vertex v_x that is adjacent to all of $N[x]$ in G. To make sure that every literal l_x^i is assigned the appropriate truth value with regard to the variable x, we add an auxiliary vertex that is adjacent to x and both vertices of the edge for l_x^i. These auxiliary vertices ensure that the attached edges force each other, thus transferring the truth assignments along the edges. The full construction for a sample clause with the attachments to the variable vertices can be seen in Figure 2.5a.

As all incident edges of a variable-vertex are oriented in the same direction, i.e., ingoing or outgoing, we also assure that a variable-vertex transfers the same truth assignment to all clauses in which it is contained as a literal; Figure 2.5b illustrates this fact by showing two gadgets joined at a variable-vertex.

We call the set of all auxiliary vertices defined as above A. These are denoted as the smaller grey vertices in all figures and play an important role in forcing an orientation of one edge to another. In the following, we wish to show that the construction described above is the complement of an asteroidal triple free graph. In order to do this, we need to describe all triangles in the complement, as these form all possible asteroidal triples. Then it is possible to show that all vertices contained in domination intervals of the graph are, in fact, auxiliary vertices.

Lemma 2.2.5. *Let $z, a, b \in V(G(\mathcal{I}))$ such that $z \in I_{\mathrm{dom}}^{\overline{G}}(a, b)$. Then z is an element of A and $a, b \notin A$.*

Proof. First, we will show that any triangle of $G(\mathcal{I})$ contains exactly one vertex of A. It is easy to see that at most one of the vertices of A is contained in every triangle, as A forms an independent set in $G(\mathcal{I})$. To show that at least one of these is used, let v be an arbitrary vertex in $V(G(\mathcal{I})) \setminus A$. As none of the clause gadgets contain a triangle without the vertices of A, assume that $v \in X$. However, any two neighbours of x cannot be adjacent, as x only has exactly one neighbour for each edge belonging to a literal using the variable represented by x. Thus, any triangle in $G(\mathcal{I})$ contains exactly one edge of A.

Now, let $z, a, b \in V(G(\mathcal{I}))$ such that $z \in I_{\overline{G}}(a, b)$. Obviously, $\{z, a, b\}$ form a triangle in $G(\mathcal{I})$. If z is an element of A, we are done. Therefore, assume without loss of generality that $a \in A$. As there is an a-avoiding path from z to b in $\overline{G(\mathcal{I})}$, there must be a neighbour of z and a neighbour of b in $\overline{G(\mathcal{I})}$ that are not adjacent to a (these need not be distinct). However, the vertex a has only three non-neighbours in $\overline{G(\mathcal{I})}$ that form an induced P_3 in $G(\mathcal{I})$; this yields a contradiction, proving the statement. □

This also shows that the complement of our construction is AT-free, as in any asteroidal triple each vertex is both an endpoint of an interval, as well as contained in on itself.

Corollary 2.2.6. *the graph $\overline{G(\mathcal{I})}$ is asteroidal triple free.*

The following observation can be easily checked and shows that *all* auxiliary vertices are contained in some interval.

Observation 2.2.7. *If z is an element of A and u, v and w are its three neighbours in $G(\mathcal{I})$, where u-v-w forms an induced P_3 in $G(\mathcal{I})$, then $z \in I_{\mathrm{dom}}^{\overline{G}}(u, v)$ and $z \in I_{\mathrm{dom}}^{\overline{G}}(v, w)$. In particular, if u-v-w forms an induced P_3 in $G(\mathcal{I})$ all of whose vertices are adjacent to a vertex $z \in A$, then $uv\mathcal{F}vw$ in $\overline{G}$.*

Now we are in a position to show that any fulfilling assignment of a given instance of the NOT-ALL-EQUAL 3-SAT problem yields an acyclic orientation of the constructed graph that agrees with the forcing and vice versa.

Lemma 2.2.8. *The graph $G(\mathcal{I})$ has an acyclic orientation that agrees with the forcing if and only if $\mathcal{I}$ has a truth assignment that solves the NOT-ALL-EQUAL 3-SAT problem.*

Proof. Suppose we have a solution to the NOT-ALL-EQUAL 3-SAT problem for a given instance $\mathcal{I}$ with variables $\{x_1, \dots, x_n\}$ and clauses $c_1, \dots, c_m$. For all $i \in \{1, \dots, n\}$, if x_i is set to false, we will orient one of the edges between x_i and one of the literal gadgets inwards, i.e., towards x_i and outwards otherwise. Of the two edges f_{c_j} and h_{c_j} in the clause-gadget for the clause c_j, we orient one in an arbitrary direction for all $j \in \{1, \dots, m\}$. Using Lemma 2.2.2 and the forcing relation, this yields an orientation that respects the forcing for the whole graph. Suppose this orientation creates a directed circuit. Due to the construction of $G(\mathcal{I})$, this circuit can only be in one of the C_4s in the clause-gadgets. Such a directed circuit would imply, that all literals of this clause are set to the same truth value, which is a contradiction to our using a solution of the NOT-ALL-EQUAL 3-SAT problem. Therefore, our orientation agrees with the forcing and is acyclic and thus, yields a bilateral AT-free ordering, due to Lemma 2.2.1.

Suppose we are given a bilateral AT-free ordering of this graph. Then this ordering yields an acyclic orientation of our graph that agrees with the forcing. This orientation in turn yields a solution for the NOT-ALL-EQUAL 3-SAT problem by setting a variable x_i to true, if it has only ingoing edges, and to false otherwise. □

This lemma implies the desired result.

Theorem 2.2.9. *The problem of deciding whether a graph is bilateral AT-free is NP-complete.*

Proof. This problem is in NP, as checking whether a given linear vertex ordering is bilateral with regard to a given graph $G = (V, E)$ can be checked in polynomial time by computing all the domination intervals.

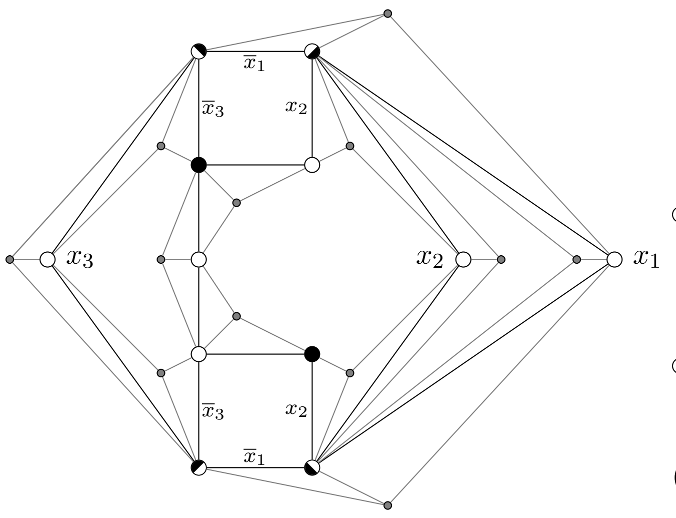
As the construction used in Lemma 2.2.8 is polynomial in the size of the input and NOT-ALL-EQUAL 3-SAT is NP-complete, the problem of deciding whether a graph is bilateral AT-free is also NP-complete. □

2.2.1 Strong Asteroid Free Graphs

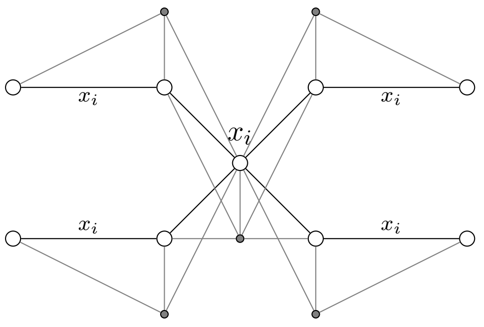
Using the results from the previous section, we are finally equipped to answer Question 2.1.6 which asked whether the the families of bilateral AT-free graphs and Strong Asteroid free graphs are comparable. To this end, we introduce some notions due to Corneil et al. [42].

Definition 2.2.10. *For a graph $G = (V, E)$ and a vertex $v \in V$ let $C_1, \dots, C_k$ be the connected components of $G - N[v]$ and let $B_i^1, \dots, B_i^l$ be the connected components of the graph induced by the vertices of C_i in $\overline{G}$ for $1 \leq i \leq k$; the B_i^j are called the* blobs *of v in G.*

For a graph $G = (V, E)$ the altered knotting graph *is given by $K^*[G] = (V_{K^*}, E_{K^*})$, where V_{K^*} and E_{K^*} are defined as follows: For each vertex $v \in V$ there are copies*



(a) A complete construction of the gadget for $\overline{x_1} \vee x_2 \vee \overline{x_3}$. The auxiliary interval vertices and their edges are grey.



(b) A sketch of two gadgets joined at the vertex representing x_1. The auxiliary interval vertices and their edges are grey.

Figure 2.5

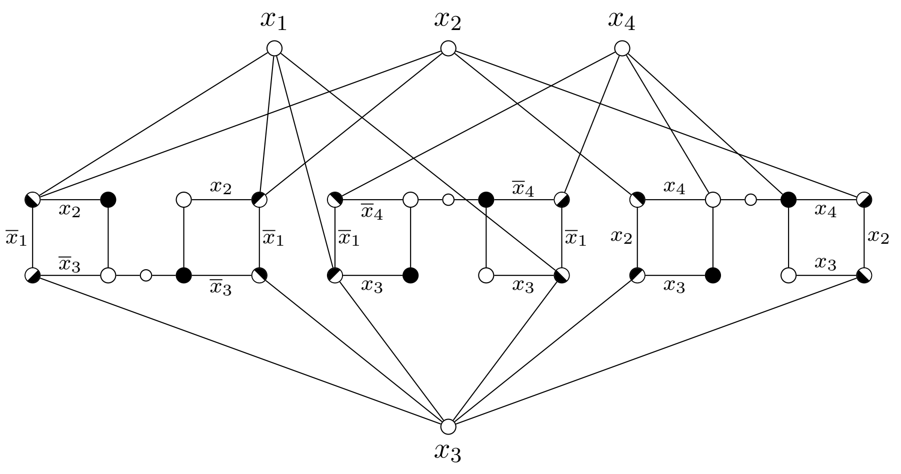


Figure 2.6: A sketch of the construction for the instance $\mathcal{I} = (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\overline{x}_1 \vee x_3 \vee \overline{x}_4)$, where the auxiliary interval vertices are omitted.

$v_1, \dots, v_{i_v}$ in V_K, where i_v is the number of blobs of v in $\overline{G}$. For each edge vw of E there is an edge v_iw_j in E_{K^*}, where w is contained in the i-th blob of v in $\overline{G}$ and v is contained in the j-th blob of w in $\overline{G}$.

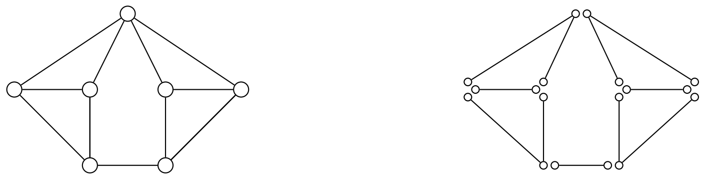


Figure 2.7: A graph G together with its altered knotting graph $K^*[G]$.

An example of a graph together with its altered knotting graph can be found in Figure 2.7. We can use domination intervals to give a sufficient criterion for when two edges are knotted in $K^*[\overline{G}]$: If two edges in $\overline{G}$ both represent domination intervals in the original graph G and share a vertex, then this implies that these two edges are knotted.

Lemma 2.2.11. *If $I_{\text{dom}}(a,b)$ and $I_{\text{dom}}(b,c)$ share a vertex $z \in V$, then $ab \in E(\overline{G})$ and $bc \in E(\overline{G})$ are knotted at b in $K^*[\overline{G}]$.*

Proof. Suppose $ac \in E(G)$. Then a and c are in the same connected component of $G - N[b]$. As they are both adjacent to z in $\overline{G}$, which is also in the same connected component, they are even in the same blob and thus the edges are knotted at b. If they are not adjacent, they both have a b-avoiding path to z and are therefore in the same connected component of $G - N[b]$. As they are not adjacent in G, we can assume they are in the same blob and the edges are again knotted at b. □

Corneil et al. [42] showed that the odd cycles in the altered knotting graph $K^*[\overline{G}]$ correspond to a particular configuration in the original graph G which we call an *odd strong asteroid*.

Definition 2.2.12. *An* odd strong asteroid *of size $2k+1$ in a graph G is a sequence of vertices $v_0, v_1, \dots, v_{2k+1}$, where $v_1, \dots, v_{2k+1}$ are distinct, $v_0 = v_{2k+1}$, and v_i and v_{i+1} are in the same blob of $v_{(i+k+1)}$ in G for all $0 \le i \le k$, where addition is taken modulo k.*

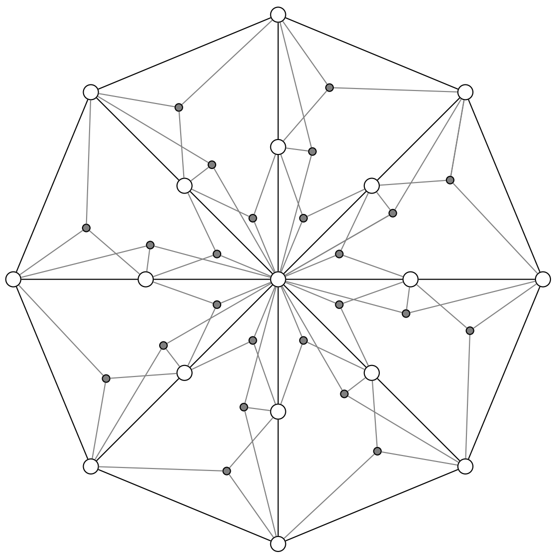
A graph is strong asteroid free *if it does not contain an odd strong asteroid.*

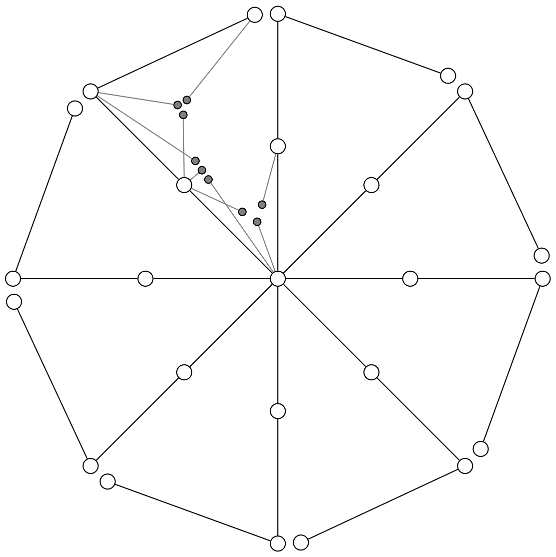
Using Lemmas 2.2.11 and 2.2.1, we can construct a graph which is asteroid free and not bilateral AT-free.

Proposition 2.2.13. *There is a graph that is strong asteroid free, but which does not have a bilateral AT-free order.*

Proof. We claim that the complement of the graph shown in Figure 2.8a is as desired. Let z be a grey vertex of the graph and let a and b be two black vertices, such that z, a, b form a triangle. It is easy to see that this implies $z \in I_{\text{dom}}(a,b)$. Using the forcing, we create an oriented cycle along the outer edges and due to Lemma 2.2.1 the complement of this graph does not have a bilateral AT-free ordering.

On the other hand, one can use Lemma 2.2.11 to construct the altered knotting graph $K^*[\overline{G}]$ which is given in Figure 2.8b. This graph does not contain an odd cycle, implying that G is strong asteroid free. □


(a) Complement of a graph which is strong asteroid free, but not bilateral AT-free.


(b) The according altered knotting graph where the grey vertices are only depicted in one of the slices of the cycle.

This resolves Question 2.1.6, by showing that there are both strong asteroid free graphs that are not bilateral AT-free (Figure 2.8a) and bilateral AT-free graphs are not strong asteroid free (Figure 2.1b).

Corollary 2.2.14. *The classes of bilateral AT-free graphs and strong asteroid free graphs are not comparable.*

2.3 AT-free Orders in Subclasses of AT-free graphs

After having established the existence of AT-free BFS orders and a polynomial-time algorithm for their computation in Chapter 1, we are, of course, interested in finding a simple linear time algorithm for this problem.

2.3.1 Claw-free AT-free Graphs

In many graph classes, forbidding induced claw-graphs yields strong structural properties for BFS searches. For example, in [36] and in [115] the authors use these structural properties to generate unit interval respectively minimal triangulation orderings. In the following, we will show that by forbidding induced claws in AT-free graphs it is possible to use three sweeps of BFS to compute a bilateral AT-free order, also proving that claw-free AT-free graphs are bilateral AT-free.

As an overview of the proof, we will show that the first application of BFS ends in an admissible vertex. Using this fact, we can then proceed to show that a BFS starting in this admissible vertex yields an AT-free order. Finally, using this order as a tie-breaker in a third run of BFS outputs a bilateral AT-free order.

Lemma 2.3.1. *Let G be claw-free and AT-free. Then the last vertex of a BFS is admissible.*

Proof. Let s be the first and z the last vertex of the BFS and let $k := \operatorname{dist}_G(s,z)$. Suppose there are $a, b \in V$ such that $z \in I[a,b]$. As G is AT-free, at least one of a or b must be in the last layer $L^k_G(s)$ of the BFS, without loss of generality this is a. If $\operatorname{dist}_G(s,b) < \operatorname{dist}_G(s,z)$, then $N^{k-1}_s(a) \subseteq N^{k-1}_s(z)$, as otherwise there is a z-avoiding a-b-path. If $\operatorname{dist}_G(s,b) = \operatorname{dist}_G(s,a) = \operatorname{dist}_G(s,z)$, then either $N^{k-1}_s(a) \subseteq N^{k-1}_s(z)$ or $N^{k-1}_s(b) \subseteq N^{k-1}_s(z)$, as G is AT-free, and without loss of generality we can assume this to be true for a. Therefore, a and z have a common neighbour c in $L^{k-1}_G(s)$. If c is not the start vertex of the BFS, then c has a neighbour d in L^{k-2}_G and a, z, c, d form a claw. If c is the start vertex, then b must also be adjacent to c and a, b, c, d form a claw. $\square$

For the second step in this proof, we need a technical lemma which characterises the distance layers given by a BFS starting in an admissible vertex.

Lemma 2.3.2. *Let G be a claw-free, AT-free graph and let $s \in V$ be admissible in G and t eccentric with respect to s. Then all but the first distance layers of s, i.e., $L^0_G(s), L^2_G(s), \dots, L^k_G(s)$, with $k = \operatorname{ecc}_G(s)$, are cliques and s and t form a dominating pair.*

Proof. For $L^0_G(s)$ this is obvious. Let $i \geq 2$ and suppose there are $a, b \in L^i_G(s)$ with $ab \notin E$. As s is admissible, without loss of generality $N^{i-1}_s(a) \subseteq N^{i-1}_s(b)$. Therefore, a and b have a common neighbour $c \in L^{i-1}_G$. This c in turn has a neighbour $d \in L^{i-2}_G$ and a, b, c, d form a claw, which is a contradiction to the assumption.

As any path P between s and t has one vertex from each distance layer $L^i_G(s)$ and s is adjacent to all vertices in $L^1_G(s)$, they must form a dominating pair. $\square$

The following statement is a simple application of Lemma 2.3.2.

Lemma 2.3.3. *Let G be an AT-free, claw-free graph. Then a BFS starting in an admissible vertex yields an AT-free order that is a monotone dominating pair order.*

Proof. Let σ be such a BFS on G starting in an admissible vertex s. Suppose $z \in I_{\text{dom}}(a,b)$ and $a, b \prec_\sigma z$. We can assume that a, b and z do not have the same distance to s (otherwise we can construct a claw as above). As G is AT-free, on the other hand, at least one of a or b must be in the same layer as z. Without loss of generality we can assume that b and z are in the same layer $L^i_G(s)$ and a is in layer $L^j_G(s)$ with $j < i$. As b and z are independent of each other, they must be in the first layer of the BFS. As a cannot be the start vertex (it is not adjacent to the other two); a contradiction. Lemma 2.3.2 states that σ must be a monotone dominating pair order. $\square$

Given a dominating pair, we will need to compare the distances of any two no-adjacent vertices to this dominating pair.

Lemma 2.3.4. *Let $G = (V, E)$ be a connected graph with a dominating pair s and t. Let u and v be two vertices with $uv \notin E$ and $\mathrm{dist}_G(s, u) < \mathrm{dist}_G(s, v)$. Then $\mathrm{dist}_G(t, u) \geq \mathrm{dist}_G(t, v)$.*

Proof. As s and t form a dominating pair, we see that for every vertex $x \in V$ it holds that $\mathrm{dist}_G(s, x) + \mathrm{dist}_G(x, t) \leq \mathrm{dist}_G(s, t) + 2$. Suppose $\mathrm{dist}_G(u, t) < \mathrm{dist}_G(v, t)$. This implies $\mathrm{dist}_G(s, t) + 2 \leq \mathrm{dist}_G(s, u) + \mathrm{dist}_G(u, t) + 2 \leq \mathrm{dist}_G(s, v) + \mathrm{dist}_G(v, t) \leq \mathrm{dist}_G(s, t) + 2$. Therefore, $\mathrm{dist}_G(s, u) + \mathrm{dist}_G(u, t) = \mathrm{dist}_G(s, t)$ and there must be a shortest s-t-path P that contains u. As P is dominating and $uv \notin E$; this is a contradiction to $\mathrm{dist}_G(s, u) < \mathrm{dist}_G(s, v)$ and $\mathrm{dist}_G(t, u) < \mathrm{dist}_G(t, v)$. □

Using all of the above lemmas, we can finally show the main theorem of this section.

Theorem 2.3.5. *Let G be a claw-free AT-free graph. Then G has a bilateral AT-free ordering that is a monotone dominating pair order and this order can be found in linear time.*

Proof. Let y be an admissible vertex of G and $\sigma_1 := BFS(G, y)$ as well as $\sigma_2 = BFS^+(G, \sigma_1)$. We have already shown that $s := \sigma_1(n) = \sigma_2(1)$ is admissible and that both σ_1 and σ_2 are AT-free orders that are monotone dominating pair orders. Thus, suppose there is a $z \in I_{\mathrm{dom}}(a, b)$ such that $z \prec_{\sigma_2} a, b$. Again we can assume that $1 = \mathrm{dist}_G(s, a) = \mathrm{dist}_G(s, z) < \mathrm{dist}_G(s, b)$.

Case 1 ($\mathrm{dist}_G(y, b) > \mathrm{dist}_G(y, z)$): We also know that $\mathrm{dist}_G(s, b) > \mathrm{dist}_G(s, z)$; this is a contradiction to the statement of Lemma 2.3.4.

Case 2 ($\mathrm{dist}_G(y, b) < \mathrm{dist}_G(y, z)$): Then $b \prec_{\sigma_1} z$ which in turn implies $z \prec_{\sigma_1} a$; this is a contradiction, as it would imply that $a \prec_{\sigma_2} z$, because $a, z \in L^1_G(s)$.

Case 3 ($\mathrm{dist}_G(y, b) = \mathrm{dist}_G(y, z)$): As before, we can assume that $1 = \mathrm{dist}_G(y, b) = \mathrm{dist}_G(y, z) < \mathrm{dist}_G(y, a)$ and thus $z \prec_{\sigma_1} a$; again a contradiction.

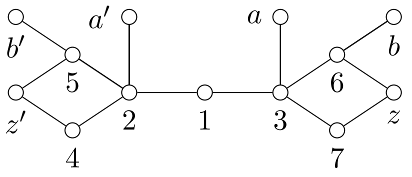
□

Corollary 2.3.6. *Any claw-free AT-free graph is bilateral AT-free.*

2.3.2 AT-free Graphs without Bad Claws

In the proof of Lemma 2.3.3 we can see that the main obstacles are triples of vertices $a, b, z \in V(G)$ with $z \in I_{\mathrm{dom}}(a, b)$ that form the prongs of a claw. Therefore, it is not necessary to forbid all induced claws in the graph to avoid these difficulties. Rather, it suffices to forbid just those claws which have this property with regard to the intervals. This justifies the following definition:

Definition 2.3.7. *Let G be a graph and let $a, b, z, c \in V$ induce a claw with base c. We will call such a claw a* bad claw*, if $z \in I_{\mathrm{dom}}(a, b)$.*



BFS: σ_1: $(1, 2, 3, a', 4, 5, a, 6, 7, z', b', b, z)$
BFS(σ_1): σ_2: $(z, 7, 6, 3, b, a, 1, 2, 5, 4, a', b', z')$
BFS(σ_2): σ_3: $(z', 4, 5, 2, b', a', 1, 3, a, 6, 7, b, z)$

Figure 2.9: A bad-claw-free graph for which BFS does not yield an AT-free order

It seems reasonable to expect that by forbidding such bad claws we will be able to get similar results to the ones above. On the other hand, there are examples of AT-free bad-claw-free graphs for which the above procedure does not yield either an AT-free order nor a bilateral AT-free ordering (see Figure 2.9). In particular, Lemma 2.3.1 does not hold in general for these graphs. Therefore, we will use LBFS which guarantees us an admissible vertex as its end-vertex.

Lemma 2.3.8 (Corneil et al. [40])**.** *Let $G = (V, E)$ be an AT-free graph and let σ be an ordering of V produced by an LBFS. Then the vertex $t := \sigma(n)$ is admissible in G.*

In fact, the properties of LBFS even make up for the absence of the strong structural property of Lemma 2.3.2 and we can prove analogues to both Lemma 2.3.3 and Theorem 2.3.5.

Lemma 2.3.9. *Let G be AT-free and bad-claw-free. Then an LBFS starting in an admissible vertex yields an AT-free order that is a monotone dominating pair order.*

Proof. Let σ be an LBFS order starting in an admissible vertex s. Suppose $z \in I_{\text{dom}}(a, b)$ and $a, b \prec_\sigma z$. Without loss of generality, we see that $i := \text{dist}_G(s, b) = \text{dist}_G(s, z)$, as G is AT-free. For that same reason either $N_s^{i-1}(b) \subseteq N_s^{i-1}(z)$ or $N_s^{i-1}(a) \subseteq N_s^{i-1}(z)$ or both.

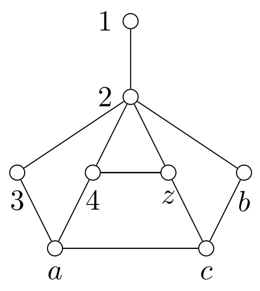
Now suppose $\text{dist}_G(s, a) = i$. As s is admissible, and a, b and z are independent, they must have a common neighbour c with $\text{dist}_G(s, c) = i - 1$ and therefore a, b and z and c form a bad claw; a contradiction.

Therefore, we can assume that $j := \text{dist}_G(s, a) < i$. With the above we see that $N_s^{i-1}(b) \subseteq N_s^{i-1}(z)$ and there is a b-avoiding a-z-path P. Let x be the σ-last vertex of P. As $b \prec_\sigma z \preceq_\sigma x$, due to Theorem 1.5.8 the vertex b must see every s-x-path and thus also every x-a-path; a contradiction. Thus, every LBFS starting in an admissible vertex yields an AT-free order.

Finally, Theorem 1.5.8 states that every LBFS order of an AT-free graph starting in an admissible vertex is a monotone dominating pair order. □

Analogously to claw-free AT-free graphs, a final third sweep of LBFS yields the desired bilateral AT-ordering.

Theorem 2.3.10. *Let G be an AT-free graph that does not have a bad claw as an induced subgraph. Then G has a bilateral AT-free ordering that is a monotone dominating pair order and such an order can be found in linear time.*



LBFS: σ_1: $(1, 2, 4, z, 3, b, a, c)$
LBFS(σ_1): σ_2: $(c, a, b, z, 4, 3, 2, 1)$
LBFS(σ_2): σ_3: $(1, 2, 3, 4, z, b, a, c)$

Figure 2.10: Example of a graph with a bad claw. On the right, one can see that the second σ_2 is not an AT-free order and σ_3 is not a bilateral AT-free order. In fact, this is an example of an AT-free graph that does not possess a bilateral AT-free ordering.

Proof. Let σ_1 be an order given by LBFS starting in an arbitrary vertex of G. Let $\sigma_2 := LBFS^+(G, \sigma_1)$ with $y := \sigma_1(n) = \sigma_2(1)$ and let $\sigma_3 := LBFS^+(G, \sigma_2)$.

We have already shown that y and $s := \sigma_2(n) = \sigma_3(1)$ are admissible and that both σ_2 and σ_3 are AT-free orders.

Suppose σ_3 is not a bilateral AT-free ordering. Then there is a $z \in I_{\text{dom}}(a, b)$ with $z \prec_{\sigma_3} a, b$, as σ_3 is AT-free. Let a, z and b be such that z is σ_2-leftmost with this property. Without loss of generality we can assume that a and z have the same distance to s, due to s being admissible. Also, a, z and b cannot all have the same distance to s, as this would imply a bad claw. Thus, $\text{dist}_G(s, z) = \text{dist}_G(s, a) < \text{dist}_G(s, b)$.

Case 1 ($\text{dist}_G(z, y) < \text{dist}_G(b, y)$): We also know that $\text{dist}_G(s, b) > \text{dist}_G(s, z)$ and that y and s form a dominating pair; this is a contradiction to the statement of Lemma 2.3.4.

Case 2 ($\text{dist}(b, y) < \text{dist}_G(z, y)$): As σ_2 is AT-free, we can deduce that $b \prec_{\sigma_2} z \prec_{\sigma_2} a$. Let z' be the σ_3-first vertex in the connected component C of $\Gamma^{\sigma_3}_{z,a}$. Due to the Prior Path Theorem there is a z'-z path P in C such that P avoids a. As $z \prec_{\sigma_3} a$ we can assume that $a \prec_{\sigma_2} z'$. If $z'b \notin E$ we see that $z' \prec_{\sigma_3} a \prec_{\sigma_3} b$ and $z' \in I_{\text{dom}}(a, b)$; a contradiction to the choice of z. Therefore, let $z'b \in E$. With $\text{dist}_G(b, y) < \text{dist}_G(a, y)$ this implies that $l := \text{dist}_G(z', y) \leq \text{dist}_G(a, y)$. Because $z'b \in E$ and y is admissible $N^{l-1}_y(a) \subset N^{l-1}_y(z')$; a contradiction to $a \prec_{\sigma_2} z'$.

Case 3 ($\text{dist}_G(z, y) = \text{dist}_G(b, y)$): As G is bad-claw-free, $\text{dist}_G(a, y) \neq \text{dist}_G(b, y)$. Suppose $\text{dist}_G(a, y) < \text{dist}_G(b, y)$. As $\text{dist}_G(s, a) < \text{dist}_G(s, b)$ and y and s are a dominating pair; a contradiction to Lemma 2.3.4. Suppose $\text{dist}_G(a, y) > \text{dist}_G(b, y)$. Then $z \prec_{\sigma_2} a$ and with the argument from Case 2, which uses the Prior Path Theorem, we receive a contradiction.

□

Corollary 2.3.11. *Let $G = (V, E)$ be an AT-free graph that does not contain a bad claw. Then G is bilateral AT-free.*

These results indicate that a linear time algorithm to construct AT-free orders could also exist for the general case of AT-free graphs. However, none of the techniques used for the (bad-)claw-free graphs can be transferred. Corneil and Stacho [38] already showed that there are AT-free graphs which do not possess AT-free orders that are also LBFS orders. In addition, Figure 2.10 shows a graph which does not possess a bilateral AT-free ordering. Therefore, it will be necessary to use a different search algorithm, possibly a BFS-derivative based on $\mathrm{BFS}^{\mathrm{conv}}$. We summarise these suppositions in the following conjecture.

Conjecture 2.3.12. *Let $G = (V, E)$ be an AT-free graph. There is a linear time algorithm that computes an AT-free (BFS) order.*

2.4 Conclusion

We have introduced the class of bilateral AT-free graphs which was motivated by the characterising convex geometry for AT-free graphs. These are defined to be the graphs that possess a halfspace ordering with respect to domination convexity. This class is shown to contain some important subclasses of AT-free graphs, such as cocomparability graphs and path orderable graphs. Furthermore, bilateral AT-free graphs are proven to be incomparable to the strong asteroid free graphs.

We have shown that the recognition of bilateral AT-free graphs is $\mathcal{NP}$-complete by giving a reduction from NOT-ALL-EQUAL 3-SAT. For the special case of claw-free AT-free graphs, we have shown that multiple applications of BFS yield AT-free orders with additional structural properties. In fact, if we exchange generic BFS with LexMinBFS, a derivative of BFS defined in [115], we can construct an AT-free, monotone dominating pair order that is also a minimal interval completion order.

This is a surprising result, as it was shown in [88] that the recognition of claw-free AT-free graphs is at least as hard as triangle recognition. This dichotomy is of striking resemblance to the case of comparability graphs, where a characterising linear ordering in the form of a transitive orientation can be found in linear time, while there is no known recognition algorithm that is faster than matrix multiplication [133]. Furthermore, we conjecture that it is possible to generate an AT-free order for any AT-free graph in linear time using some form of BFS (Conjecture 2.3.12). As is the case for comparability graphs, such a linear ordering might then be used for linear time optimisation algorithms that are robust for AT-free graphs, i.e., which can be applied without solving recognition first (for further information on robust algorithms see [133]). In this context, the results on bad-claw-free graphs can be seen as a first step towards a resolution of this conjecture, and give us a strong notion where the algorithmic difficulties lie.

3 Avoidable Vertices and Edges

Chordal graphs are well-known to possess many good structural and algorithmic properties [17, 49, 64, 75]. The main goal of this chapter is to study certain concepts related to chordal graphs in the framework of more general graph classes. The starting point for our research is a result due to Dirac [49] stating that every minimal cutset in a chordal graph is a clique, which implies that every chordal graph with at least one vertex has a *simplicial vertex*, that is, a vertex whose neighbourhood is a clique [64]. Denoting by P_k the k-vertex path, this result can be formulated as follows.

Theorem 3.0.1. *Every chordal graph with at least one vertex has a vertex v such that v is not the midpoint of any induced P_3.*

This theorem was generalised in the literature in various ways, see, e.g., [35, 93, 106, 117, 133]. Two particular ways of generalising Theorem 3.0.1 include:

(i) proving a property of general graphs that, when specialized to chordal graphs, results in the existence of a simplicial vertex, and

(ii) generalising the 'simpliciality' property from vertices, which are paths of length 0, to longer induced paths, and proving the existence of such paths for graphs excluding suitably longer cycles.

Let us explain in more detail the corresponding results.

First generalisation – from chordal graphs to all graphs

A generalisation of the first kind is given by the following theorem, which follows from [117, Theorem 3] as well as from [13, Main Theorem 4.1] and [1, Lemma 2.3].

Theorem 3.0.2. *Every graph G with at least one vertex has a vertex v such that every induced P_3 having v as its midpoint is contained in an induced cycle in G.*

The above property of vertices will be one of the central concepts for this chapter and we formalize it as follows.

Definition 3.0.3. *A vertex v in a graph G is said to be* avoidable *if between any pair x and y of neighbours of v there exists an x,y-path, all the internal vertices of which avoid v and all neighbours of v. Equivalently, a vertex v is* avoidable *if every induced P_3 with midpoint v closes to an induced cycle.*

This terminology is motivated by considering a setting where G represents a symmetric acquaintance relation on a group of people. In this setting, the property of person (equivalently, vertex) a being avoidable can be interpreted as follows: whenever two acquaintances of a need to share some information that they would not like to share with a, they can do so by passing the information along a path completely avoiding both a and all her other acquaintances. Thus, a is in a sense avoidable, as information can be passed around in her immediate proximity without her knowledge.

Note that every simplicial vertex in a graph is avoidable. If we analyse avoidable vertices in graph classes, rather than in general graphs, we see that this definition is a generalisation of many well known concepts. For example, in a tree a vertex is avoidable if and only if it is a leaf, while in a chordal graph a vertex is avoidable if and only if it is simplicial. With this terminology, Theorem 3.0.2 can be equivalently stated as follows.

Theorem 3.0.4. *Every graph with at least one vertex has an avoidable vertex.*

The notion of avoidable vertices has appeared in the literature (with different terminology) in a variety of settings. To our knowledge, the earliest appearance was in the paper from 1976 by Ohtsuki et al. [117], where avoidable vertices were characterized as exactly the vertices from which a minimal elimination ordering can start. Here, a *minimal elimination ordering* of a graph $G = (V, E)$ is a procedure of eliminating vertices one at a time so that before each vertex is removed, its neighbourhood is turned into a clique, and the resulting set F of edges added throughout the procedure is an inclusion-minimal set of non-edges of G such that $(V, E \cup F)$ is a chordal graph (in other words, $(V, E \cup F)$ is a *minimal triangulation* of G). Given a graph G, an avoidable vertex in G can be found in linear time using graph search algorithms such as Lexicographic Breadth First Search (LBFS) [126] (see also [79]) or Maximum Cardinality Search (MCS) [14]. The presentation closest to our setting is the one used by Ohtsuki et al. [117]. In fact, Berry et al. [14, 15] named avoidable vertices *OCF-vertices*, after the initials of the three authors of [117].

Second generalisation – from vertices to longer paths

In order to generalize the notion of simplicial vertices to longer paths, the next definition, partially following Chvátal et al. [35], will be useful.

Definition 3.0.5. *Given an induced path P in a graph G, a* two-sided extension *of P is any induced path in G obtained by adding to P one edge at each end. An induced path is said to be* simplicial *if it has no two-sided extension.*

In this terminology, Theorem 3.0.1 can be stated as follows: every graph with at least one vertex and without induced cycles of length more than 3 has a simplicial induced P_1. Chvátal et al. [35] generalized this result as follows.

Theorem 3.0.6 (Chvátal et al. [35])**.** *For each $k \geq 1$, every $\{C_{k+3}, C_{k+4}, \ldots\}$-free graph that contains an induced P_k also contains a simplicial induced P_k.*

A common generalisation?

Theorems 3.0.4 and 3.0.6 suggest that a further common generalisation might be possible, based on the following generalisation of Definition 3.0.3 (definition of avoidable vertices) to longer paths.

Definition 3.0.7. *An induced path P in a graph G is said to be* avoidable *if every two-sided extension of P is contained in an induced cycle.*

Thus, in particular, a vertex v in a graph G is avoidable if and only if the corresponding one-vertex path is avoidable. Moreover, every simplicial induced path is (vacuously) avoidable.

We conjecture that the following common generalisation of Theorems 3.0.4 and 3.0.6 holds.

Conjecture 3.0.8. *For every $k \geq 1$, every graph that contains an induced P_k also contains an avoidable induced P_k.*

Theorem 3.0.4 implies the conjecture for $k = 1$, while Theorem 3.0.6 implies it for every positive integer k, provided we restrict ourselves to the class of graphs without induced cycles of length more than $k + 2$. Indeed, if G is a $\{C_{k+3}, C_{k+4}, \ldots\}$-free graph that contains an induced P_k, then by Theorem 3.0.6 graph G contains a simplicial induced P_k, and every simplicial induced path is avoidable.

The results given in this chapter are joint work with Maria Chudnovsky, Vladimir Gurvich, Martin Milanič and Mary Servatius. A published extended abstract of this work can be found in [7].

3.1 Characterisation and Existence of Avoidable Vertices

The proof of Theorem 3 in the paper [117] by Ohtsuki, Cheung, and Fujisawa (which itself relied on earlier works of Rose [123, 124, 125]) leads to the characterisation of avoidable vertices given by the following theorem. Since we are not aware of any explicit statement of this result in the literature, we state it here and give a short self-contained proof that does not rely on the concept of minimal elimination orderings.

Theorem 3.1.1. *Let $G = (V, E)$ be a graph and let $v \in V$. Then v is avoidable in G if and only if v is a simplicial vertex in some minimal triangulation of G.*

Proof. Let $G' = (V, E \cup F)$ be a minimal triangulation of G and let $v \in V$ be a simplicial vertex in G'. Suppose for a contradiction that v is not avoidable in G. Then, v contains two neighbours, say x and y, such that x and y belong to different connected components of the graph $G - S$, where $S = N_G[v] \setminus \{x, y\}$. Since v is simplicial in G', set S is a clique in G'. Let F^* be the set of all pairs $\{u, w\} \in F$ such that u and w are in different connected components of the graph $G - S$ and let G^* be the graph $(V, E \cup (F \setminus F^*))$. Since S is a clique in G^*, no induced cycle of G^* contains vertices from two different components of $G^* - S$. It follows that every induced cycle in G^* is also an induced cycle in G', and the fact that G' is chordal implies that G^* is chordal. However, since the set F^* is non-empty (note that it contains $\{x, y\}$), the fact that G^* is chordal contradicts the assumption that $G' = (V, E \cup F)$ is a minimal triangulation of G. This shows that v is avoidable in G.

For the converse direction, let $v \in V$ be an avoidable vertex in G and let $S = N_G(v)$. Let F_0 denote the set of non-adjacent vertex pairs in S and let $G' = (V', E')$ be the graph obtained from $G - v$ by turning S into a clique (that is, $V' = V \setminus \{v\}$ and $E' = E(G - v) \cup F_0$). Moreover, let $G'_1 = (V', E' \cup F')$ be a minimal triangulation of G' and let $G_1 = (V, E \cup (F' \cup F_0))$. Note that S is a clique in G' and therefore also in G'_1. Since G_1 can be obtained from the chordal graph G'_1 by adding to it vertex v and making it adjacent to all vertices of clique S, graph G_1 is chordal. Furthermore, since v is a simplicial vertex in G_1, to complete the proof it suffices to show that G_1 is a *minimal* triangulation of G, or, equivalently, that for every edge $f \in F' \cup F_0$ the graph $G_1 - f$ is not chordal. Suppose first that $f \in F'$. Since $G'_1 = (V', E' \cup F')$ is a minimal triangulation of G', the graph $G'_1 - f$ is not chordal, and thus it contains an induced cycle C of length at least 4. As $G'_1 = G_1 - v$, we see that C is also an induced cycle in $G_1 - f$. Finally, suppose that $f \in F_0$. Then $f = \{x, y\}$ where x and y are two non-adjacent neighbours of v in G. Let $S' = N_G[v] \setminus \{x, y\}$. Since v is an avoidable vertex in G, vertices x and y are in the same component of the graph $G - S'$, and consequently, since G is a spanning subgraph of $G_1 - f$, also in the same component of the graph $(G_1 - f) - S'$. Let P be a shortest x, y-path $(G_1 - f) - S'$. Then $V(P) \cup \{v\}$ induces a cycle of length at least four in the graph $G_1 - f$, which implies that $G_1 - f$ is not chordal, as claimed. This completes the proof. □

Remark 3.1.2. *Sets of vertices of a graph G that are maximal cliques in some minimal triangulation of G were studied in the literature under the name* potential maximal cliques*. This concept was introduced by Bouchitté and Todinca in [19] and has already found many applications in algorithmic graph theory (see, e.g., [20, 61, 62]). In this terminology, Theorem 3.1.1 states that given a vertex $v \in V(G)$, its closed neighbourhood $N_G[v]$ is a potential maximal clique in G if and only if v is avoidable.*

Since every graph has a minimal triangulation, Theorems 3.0.1 and 3.1.1 imply Theorem 3.0.4. An application of Theorem 3.0.4 to vertex-transitive graphs will be given in Section 3.5.

We now discuss the consequence of Theorem 3.0.4 when the theorem is applied to the line graph of a given graph. An edge e in a graph G is said to be *pseudo-avoidable* if the corresponding vertex in the line graph $L(G)$ is avoidable. It is not difficult to see that an edge e in a graph G is pseudo-avoidable if and only if any (not necessarily induced) 3-edge path having e as the middle edge closes to a (not necessarily induced) cycle in G. Note that the concepts of avoidable edges (considered as induced P_2s, in the sense of Definition 3.0.7) and of pseudo-avoidable edges are incomparable, see Fig. 3.1. Assuming notation from Fig. 3.1, one can see that e is not an avoidable edge in G. However, it is pseudo-avoidable, as e is an avoidable vertex in $L(G)$. On the other hand, f is avoidable (even simplicial) in G. However, it is not pseudo-avoidable.

Applying Theorem 3.0.4 to the line graph of the given graph yields the following.

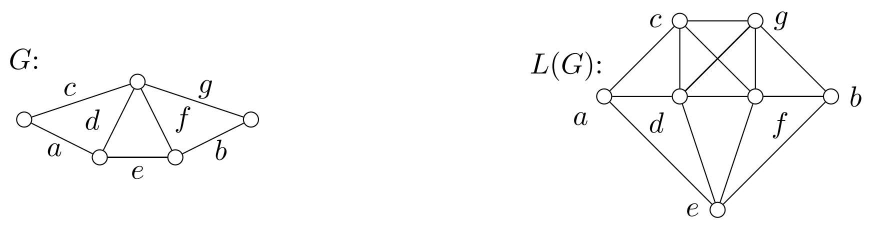


Figure 3.1: A graph G and its line graph $L(G)$.

Corollary 3.1.3. *Every graph with an edge has a pseudo-avoidable edge.*

An application of Corollary 3.1.3 to edge-transitive graphs will be given in Section 3.5.

Recall that Theorem 3.0.4 coincides with the statement of Conjecture 3.0.8 for the case $k = 1$. We now reprove this statement in a slightly stronger form, using an approach that we will adapt in Section 3.4 for the proof of the case $k = 2$ of the conjecture. A vertex in a graph is said to be *universal* if it is adjacent to every other vertex, and *non-universal* otherwise.

Theorem 3.1.4. *For every graph G and every non-universal vertex $v \in V(G)$ there exists an avoidable vertex $a \in V(G) \setminus N[v]$.*

Proof. Suppose that the theorem is false and take a counterexample G with the smallest possible number of vertices and, subject to that, with the largest possible number of edges. The minimality of $|V(G)|$ implies that G is connected. Since G is a counterexample, it has a non-universal vertex $v \in V(G)$ such that no vertex not adjacent to v is avoidable.

Let $b \in V(G) \setminus N[v]$. Since b is not avoidable in G, it is the midpoint of an induced P_3, say $x - b - y$, that is not contained in any induced cycle. Observe that vertices x and y are in different components of the graph $G - (N[b] \setminus \{x, y\})$, since any induced path P from x to y in $G - (N[b] \setminus \{x, y\})$ would imply the existence of an induced cycle $x - b - y - P - x$ closing the 3-vertex path $x - b - y$. It follows that the graph $G - (N[b] \setminus \{x, y\})$ is disconnected. In particular, there exists an inclusion-minimal subset S of $N(b)$ such that $G - (S \cup \{b\})$ is disconnected. Note that $v \notin N[b]$, and, hence, v is a vertex of $G - (S \cup \{b\})$. Let C be the component of $G - (S \cup \{b\})$ containing v. It follows from the minimality of S that every vertex in S has a neighbour in C.

Suppose first that b is not universal in $G - C$. The minimality of $|V(G)|$ implies that $G - C$ is not a counterexample to the theorem. Therefore, there exists an avoidable vertex a in the graph $G - C$ that is neither equal nor adjacent to b. Since a is a vertex of $G - (S \cup \{b\})$ not contained in C, it is not adjacent to v. We claim that a is also avoidable in G, which will contradict the assumption that no vertex not adjacent to v in G is avoidable. Let P be an induced P_3 in G with midpoint a. Since a belongs to a component of $G - (S \cup \{b\})$ different from C, no vertex of C is adjacent to a. Therefore, P is an induced P_3 in the graph $G - C$. Since a is avoidable in $G - C$, there exists an induced cycle in $G - C$ containing P. Since $G - C$ is an induced subgraph of G, we conclude that there exists an induced cycle in G containing P. Since P was arbitrary, it follows that a is avoidable in G; a contradiction.

Therefore, we may assume that b is universal in $G - C$. Let G' be the graph obtained from $G - C$ by adding a new vertex d and making d adjacent to every vertex in $S \cup \{b\}$. Note that if $C \neq \{v\}$, then G' has strictly fewer vertices than G. If, on the other hand, $C = \{v\}$, then G' has the same number of vertices as G but strictly more edges, since $N_G(C) \subseteq S$, while $N_{G'}(d) = S \cup \{b\}$. Denoting by C' any component of $G - (S \cup \{b\})$ other than C, we see that every vertex of C' is not adjacent to d in G'. Hence, d is not universal in G' and the choice of G implies that G' has an avoidable vertex a that is not d and not adjacent to d. This shows that $a \in V(G) \setminus (C \cup S \cup \{b\})$.

We claim that a is avoidable in G. Suppose that $x, y \in N_G(v)$ are not adjacent. We need to show that the path $x - a - y$ closes to an induced cycle. We have $x, y \in N_{G'}(a) \setminus \{b\}$. Since a is avoidable in G', there exists an induced path P from x to y in G' such that a has no neighbours in $V(P) \setminus \{x, y\}$. If $d \notin V(P)$, then $a - x - P - y - a$ is the required cycle in G. Therefore, we may assume that $d \in V(P)$. Observe that $d \neq x, y$. Let p and q be the two neighbours of d in $V(P)$. Then $p, q \in V(G)$. Since b is universal in G' and $b \neq a$, it follows that $b \notin V(P)$. Since p and q are adjacent to d and $b \notin V(P)$, it follows that $p, q \in S$. Moreover, notice that $V(P) \setminus \{p, q\}$ is disjoint from $S \cup \{b\}$. By the minimality of S and, since C is connected, there is an induced path Q in G from p to q such that $V(Q) \setminus \{p, q\} \subseteq C$. However, in this case $a - x - P - p - Q - q - P - y - a$ is the required cycle in G. This shows that a is an avoidable vertex in G not adjacent to v. This contradicts the assumption on v and completes the proof of the theorem. □

3.2 Computing Avoidable Vertices

Knowing that every graph with at least one vertex has an avoidable vertex, the next question is how to compute one efficiently. The obvious polynomial-time method would be to decide for each vertex v of the graph G whether it is avoidable. For this we have to check for each pair of nonadjacent neighbours x and y of v, whether they are in the same connected component of $(G - N[v]) \cup \{x, y\}$. If we use a breadth first search or depth first search to compute the connected components, this gives a running time of $\mathcal{O}(|V(G)||E(\overline{G})|(|V(G)| + |E(G)|))$. The same method can be used to compute the set of all avoidable vertices. However, if we are only interested in computing one or two avoidable vertices, we show next that this can be done in linear time.

We have already seen that in chordal graphs the avoidable vertices are exactly the same as the simplicial vertices. Therefore, any graph search algorithm that can compute simplicial vertices in a chordal graph is a good candidate for computing avoidable vertices. We have already seen that Rose et al. [126] defined a linear-time algorithm (Lex-P), which computes a perfect elimination ordering if there is one, and is thus a recognition algorithm for chordal graphs. This algorithm, since named *Lexicographic Breadth First Search* (LBFS), exhibits many interesting structural properties and has been used as an ingredient in many other recognition and optimisation algorithms on graphs. Any vertex ordering of G that can be produced by LBFS is called an *LBFS* *ordering* (of G).

The pseudocode of Lexicographic Breadth First Search given in Algorithm 5 is presented for connected graphs. However, the method can be generalized to work for arbitrary graphs by executing the search component after component (in an arbitrary order) and concatenating the resulting vertex orderings.

In this context, we will be mainly interested in the properties of the vertices of a given graph G visited *last* by some execution of LBFS, also called *end-vertices*. The essential claim of the following lemma can be found in many papers, for example in [13].

Lemma 3.2.1 (Aboulker et al. [1])**.** *Let $G = (V, E)$ be a graph and let $\sigma = (v_1, \dots, v_n)$ be an LBFS ordering of G. Then for all triples of vertices $a, b, c \in V$ such that $a \prec_\sigma b \prec_\sigma c$ and $ac \in E$, there exists a path from a to b whose internal vertices are disjoint from $N[v_n]$.*

Corollary 3.2.2. *Let $G = (V, E)$ be a graph with at least one vertex and let $\sigma = (v_1, \dots, v_n)$ be an LBFS ordering of G. Then v_n is avoidable in G. In fact, for any $i \in \{1, \dots, n\}$ the vertex v_i is avoidable in $G[v_1, \dots, v_i]$.*

Note that Lexicographic Breadth First Search is a breadth-first search, that is, when LBFS is executed beginning in a vertex s, it orders the vertices of G according to their distance from the starting vertex s. In particular, this implies the following strengthening of Theorem 3.1.4.

Corollary 3.2.3. *For every graph $G = (V, E)$ and every vertex $v \in V$ there is an avoidable vertex $a \in V$ that is eccentric to v.*

This corollary generalizes the fact that for every vertex v in a chordal graph G, there is a simplicial vertex in G that is eccentric to v [146, 59]. Moreover, with Corollary 3.2.3 at hand we can strengthen Theorem 3.0.4 to the following generalisation of Dirac's theorem on chordal graphs (Theorem 3.0.1).

Theorem 3.2.4. *Every graph $G = (V, E)$ with at least two vertices contains two avoidable vertices whose distance to each other is the diameter of G.*

Proof. Let $s \in V$ be a vertex of maximum eccentricity in G and let $\sigma = (s = v_1, \dots, v_n = a)$ be the ordering given by an LBFS starting in s. By Corollary 3.2.2, vertex a is avoidable. On the other hand, if $\tau = (a = w_1, \dots, w_n = b)$ is an LBFS of G starting in a, then b is avoidable due to Corollary 3.2.2. Moreover, $a \neq b$ and $\mathrm{dist}_G(a, b) = \mathrm{ecc}_G(a) = \mathrm{ecc}_G(s) = \mathrm{diam}(G)$. □

Since LBFS can be implemented to run in linear time (see, e.g., [75]), employing the same approach as in the proof of Theorem 3.2.4, except that vertex s is chosen arbitrarily, we obtain the announced consequence for the computation of avoidable vertices.

Theorem 3.2.5. *Given a graph G with at least two vertices, two distinct avoidable vertices in G can be computed in linear time.*

It is important to note that while every LBFS end vertex is avoidable, the converse is not necessarily true. In the graph depicted in Figure 3.2(a), vertex a is avoidable, as it is not contained in any P_3. On the other hand, a cannot be the end vertex of an LBFS, as it is not of farthest distance from any of the other vertices.

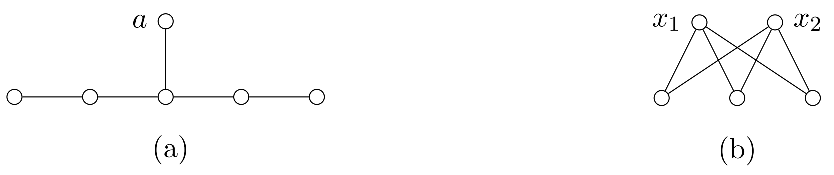


Figure 3.2: Two graphs having avoidable vertices that cannot be the end vertices of an LBFS, resp. an MCS.

We can therefore not use LBFS to find all avoidable vertices of a given graph. In fact, while it is possible to check whether a vertex is avoidable in polynomial time (as explained above), it is NP-complete to decide whether an arbitrary vertex of a given graph can be the end vertex of an LBFS [44]. This holds true even for restricted graph classes such as bipartite graphs [77] and weakly chordal graphs [44].

Berry et al. [14] give another possibility to compute avoidable vertices in linear time using *maximum cardinality search* (MCS) as defined in Algorithm 7. They show that, analogously to LBFS, the last vertex visited by an MCS is always avoidable (or in their notation an OCF-vertex).

It is easy to see that in the graph given in Figure 3.2(a) the set of avoidable vertices is equal to the set of MCS end vertices. However, this is not always the case. In the graph depicted in Figure 3.2(b), every vertex is avoidable, but neither x_1 nor x_2 can be the end vertex of an MCS. Furthermore, just as in the case of LBFS, deciding whether a vertex is an end vertex of MCS is NP-complete [9].

Further analysis on graph searches and avoidable vertices can be found in [15].

3.3 Implications for the Maximum Weight Clique Problem

In this section, we present an application of the concept of avoidable vertices to the maximum weight clique problem: given a graph $G = (V, E)$ with a vertex weight function $w : V \to \mathbb{R}_+$, find a clique in G of maximum total weight, where the weight of a set $S \subseteq V$ is defined as $w(S) := \sum_{x \in S} w(x)$. We will show that this problem, which is NP-hard in general, is solvable in polynomial time in the class of 1-perfectly orientable graphs, and even more generally in the class of hole-cyclically orientable graphs. The importance of these two graph classes, the definitions of which will be given shortly, is due to the fact that they form a common generalisation of two well studied graph classes, the chordal graphs and the circular-arc graphs.

The link between avoidable vertices and the classes of 1-perfectly orientable or hole-cyclically orientable graphs will be given by considering particular orientations of the input graph. Many important graph classes, like chordal graphs, comparability graphs, or proper circular-arc graphs (see, e.g., [131]), can be defined with the existence of a particular kind of orientation on the edges. One such class is the class of *cyclically orientable graphs*, first introduced by Barot et al. [5]. This is the class of graphs that admit an orientation such that every chordless cycle is oriented cyclically. If we allow triangles to be oriented arbitrarily, while all other chordless cycles must be oriented cyclically, we obtain the class of hole-cyclically orientable graphs. More formally, we say that a *hole* in a graph is a chordless cycle of length at least four, that an orientation D of a graph G is *hole-cyclic* if all holes of G are oriented cyclically in D, and that a graph is *hole-cyclically orientable* if it admits a hole-cyclic orientation.

While the class of hole-cyclically orientable graphs does not seem to have been studied in the literature, it generalizes the previously studied class of 1-perfectly orientable graphs, defined as follows. We say that an orientation of a graph is an *out-tournament*, or 1-*perfect* [95], if the out-neighbourhood of every vertex induces a tournament. Similarly, we call a digraph an *in-tournament* [4], or *fraternal* [143], if the in-neighbourhood of every vertex induces a tournament. A graph is said to be 1-*perfectly orientable* if it admits a 1-perfect orientation. Using a simple arc reversal argument, it is easy to see that the existence of a 1-perfect orientation implies a fraternal orientation and vice versa.

The class of 1-perfectly orientable graphs forms a common generalisation of the classes of chordal graphs and of circular-arc graphs [131, 143]. While 1-perfectly orientable graphs can be recognized in polynomial time via a reduction to a 2-SAT [4], their structure is not understood (except in some special cases, see [4, 85, 24, 84]) and the complexity of many classical optimisation problems such as maximum clique, maximum independent set, or k-colouring is still open for this class of graphs.

In this section, we show that the maximum weight clique problem is solvable in polynomial time in the class of 1-perfectly orientable graphs. We do so in the more general context of hole-cyclically orientable graphs. The fact that every 1-perfectly orientable graph is hole-cyclically orientable is a consequence of the following simple lemma (see, e.g., [85]).

Lemma 3.3.1. *Every* 1*-perfect orientation of a graph G is hole-cyclic.*

On the other hand, not every hole-cyclically orientable graph is 1-perfectly orientable, as can be seen in Figure 3.3.

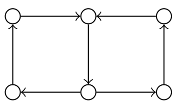


Figure 3.3: A graph with a hole-cyclic orientation that is not 1-perfectly orientable.

Our algorithm for the maximum weight clique problem in the class of hole-cyclically orientable graphs will be based on the fact that the classes of 1-perfectly orientable and hole-cyclically orientable graphs coincide within the class of cobipartite graphs, where they also coincide with circular-arc graphs. The equivalence between properties 1, 3 and 4 in the lemma below was already observed in [85]. Due to Lemma 3.3.1, the list can be trivially extended with the hole-cyclically orientable property.

Lemma 3.3.2. *For every cobipartite graph G, the following properties are equivalent:*

1. *G is 1-perfectly orientable.*
2. *G is hole-cyclically orientable.*
3. *G has an orientation in which every induced 4-cycle is oriented cyclically.*
4. *G is a circular-arc graph.*

Another important notion for the algorithm is that of bisimplicial elimination orderings. A vertex v in a graph G is *bisimplicial* if its neighbourhood is the union of two cliques in G (or, equivalently, if the graph $G[N(v)]$ is cobipartite). Let $G = (V, E)$ be a graph and let $\sigma = (v_1, \ldots, v_n)$ be a vertex ordering of G. We say that σ is a *bisimplicial elimination ordering* of G if v_i is bisimplicial in the graph $G[\{v_1, \ldots, v_i\}]$ for every $i \in \{1, \ldots, n\}$.

Theorem 3.3.3 (Ye and Borodin [148])**.** *The maximum weight clique problem is solvable in time $\mathcal{O}(|V(G)|^4)$ in the class of graphs having a bisimplicial elimination ordering.*

The algorithm can be summarized as follows.

Algorithm 11: Solving the maximum weight clique problem in graphs with a bisimplicial elimination ordering

```
Input: A graph $G = (V, E)$, a weight function $w : V \to \mathbb{R}_+$, and a bisimplicial
       elimination ordering $\sigma = (v_1, \ldots, v_n)$
Output: A maximum weight clique $C^*$ of $G$

1 $C^* := \emptyset$;
2 for $i = 0$ to $n - 1$ do
3     $v := \sigma(n - i)$;
4     Compute a maximum weight clique $C_v$ of $G[N(v)]$;
5     if $w(C_v) > w(C^*)$ then $C^* := C_v$;
6     $G := G - v$;
```

The polynomial involved in the running time of the algorithm given in [148] was not estimated; it was based on polynomial-time solvability of the maximum weight clique problem in the class of perfect graphs. However, the algorithm can be implemented to run in time $\mathcal{O}(|V(G)|^4)$, as follows. Suppose first that the input graph G is equipped with a bisimplicial elimination ordering $(v_1, \ldots, v_n)$. Letting $G_i = G[\{v_1, \ldots, v_i\}]$ for every $i \in \{1, \ldots, n\}$, at each step of the algorithm, the maximum weight of a clique in G_i is computed by comparing the (recursively computed) maximum weight of a clique in G_{i-1} with the maximum weight of a clique containing the current vertex v_i. This latter value is computed by solving the maximum weight clique problem in the graph $G_i[N[v_i]]$, which is a cobipartite graph. The maximum weight clique problem in a cobipartite graph

G is equivalent to the maximum weight independent set problem in its complement $\overline{G}$, which is bipartite. The maximum weight independent set problem in a bipartite graph with n' vertices and m' edges can by reduced to solving an instance of the maximum flow problem [89] and can thus be solved in time $\mathcal{O}(n'(n'+m'))$, see, e.g., [119]. It follows that the maximum weight clique problem is solvable in time $\mathcal{O}(n^3)$ in an n-vertex cobipartite graph. Consequently, the maximum weight clique problem is solvable in time $\mathcal{O}(n^4)$ in the class of n-vertex graphs equipped with a bisimplicial elimination ordering. If a bisimplicial elimination ordering of the input graph is not known, at each step of the algorithm a bisimplicial vertex v_i is first computed in G_i. Testing if a vertex v in G_i is bisimplicial can be done in time $\mathcal{O}(|V(G_i)|^2)$ by testing whether $\overline{G_i[N(v)]}$ is bipartite; hence, in time $\mathcal{O}(|V(G_i)|^3)$ a bisimplicial vertex v_i in G_i can be found, and a bisimplicial elimination ordering of G can be computed in time $\mathcal{O}(n^4)$.

As shown by Addario-Berry et al. [2], every graph with at least one vertex and without even holes has a bisimplicial vertex. We show next that this property also holds for hole-cyclically orientable graphs. This fact, instrumental to the polynomial-time solvability of the maximum-weight clique problem in this class of graphs, is based on a simple argument involving avoidable vertices.

Lemma 3.3.4. *Every avoidable vertex in a hole-cyclically orientable graph is bisimplicial.*

Proof. Let $G = (V, E)$ be a hole-cyclically orientable graph, let D be a hole-cyclic orientation of G, and let $a \in V$ be an avoidable vertex in G. Suppose that $v, w \in N_D^+(a)$ and v and w are not adjacent in G. As a is avoidable, the path $v - a - w$ can be closed to an induced cycle C in G. Since D is a hole-cyclic orientation of G and C is a hole in G, we infer that C is oriented cyclically in D. This is a contradiction to the fact that $v, w \in N_+(a)$. It follows that $N_D^+(a)$ is a clique in G. The same argument also holds for $N_D^-(a)$. Since $N_G(a) = N_D^+(a) \cup N_D^-(a)$, this implies that a is bisimplicial in G. □

Lemma 3.3.4 and Corollary 3.2.2 lead to the following.

Theorem 3.3.5. *Every hole-cyclically orientable graph with at least one vertex has a bisimplicial vertex. Moreover, a bisimplicial elimination ordering of a given hole-cyclically orientable graph can be computed in linear time.*

Proof. By Lemma 3.3.4 and Corollary 3.2.2, every LBFS ordering of a hole-cyclically orientable graph is a bisimplicial elimination ordering. Given any graph G, an LBFS ordering of G can be computed in linear time (see, e.g., [75]). □

Theorems 3.3.3 and 3.3.5 imply that the maximum weight clique problem is solvable in time $\mathcal{O}(|V(G)|^4)$ in the class of hole-cyclically orientable graphs. This running time can be improved further using the structure of cobipartite graphs in this class given by Lemma 3.3.2.[1]

[1]In the unweighted case, the same approach as that used in the proof of Theorem 3.3.6 results in $\mathcal{O}(|V(G)|(|V(G)|\log|V(G)| + |E(G)|))$ algorithm for the maximum clique problem in the class of hole-cyclically orientable graphs.

Theorem 3.3.6. *The maximum weight clique problem is solvable in time* $\mathcal{O}(|V(G)|(|V(G)|\log|V(G)| + |E(G)|\log\log|V(G)|))$ *in the class of hole-cyclically orientable graphs.*

Proof. A bisimplicial elimination ordering of a given hole-cyclically orientable graph can be computed in linear time by Theorem 3.3.5. Thus, we can apply Algorithm 11.

Due to Lemma 3.3.2, a cobipartite hole-cyclically orientable graph is always a circular-arc graph. Solving the maximum weight clique problem in a given circular-arc graph G equipped with a given circular-arc model can be done in time n $\mathcal{O}(|V(G)|\log|V(G)| + |E(G)|\log\log|V(G)|)$ [130, 16]. A circular-arc model of a given circular-arc graph G can be found in time $\mathcal{O}(|V(G)| + |E(G)|)$ [113]. Using these algorithms as subroutines in Algorithm 11 we obtain the stated running time. □

Since every 1-perfectly orientable graph is hole-cyclically orientable, Theorem 3.3.6 has the following consequence.

Corollary 3.3.7. *The maximum weight clique problem is solvable in time* $\mathcal{O}(|V(G)|(|V(G)|\log|V(G)|+|E(G)|\log\log|V(G)|))$ *in the class of* 1*-perfectly orientable graphs.*

A small digression to digraphs

We conclude this section by showing that the existence of avoidable vertices of graphs with at least one vertex, along with the approach used in the proof of Lemma 3.3.4, can be applied not only to orientations of simple graphs but also to digraphs in which pairs of oppositely oriented edges are allowed. This leads to results that can be interpreted in the context of information passing in communication networks.

A simple digraph $D = (V, E)$ is *semi-complete* if for every two distinct vertices $u, v \in V$ we have $(u, v) \in E$ or $(v, u) \in E$, and *out-semi-complete* (resp., *in-semi-complete*) if for every vertex $v \in V$, the subdigraph of D induced by its out-neighbourhood $N_D^+(v)$ (resp., in-neighbourhood $N_D^-(v)$) is semi-complete. We say that a set of vertices in a digraph is *semi-complete* if it induces a semi-complete digraph. The *underlying graph* of a digraph D is the graph G with vertex set $V(D)$ and $uv \in E(G)$ if and only if $(u, v) \in E(D)$ or $(v, u) \in E(D)$.

Lemma 3.3.8. *Let D be an out-semi-complete digraph, let G be its underlying graph, and let v be an avoidable vertex in G. Then, the in-neighbourhood of v in D is semi-complete.*

Proof. Suppose for a contradiction that $N_D^-(v)$ is not semi-complete. Then, there is a pair $x, y \in N_D^-(v)$ of non-adjacent vertices in G. Since v is avoidable in G, the induced path $x - v - y$ extends to an induced cycle C in G. Let $(v = v_1, y = v_2, v_3, \ldots, v_k = x, v_1)$ be a cyclic order of vertices on C. Then $k \geq 4$.

We claim that for every $i \in \{2, \ldots, k\}$ we have $(v_i, v_{i-1}) \in E(D)$. We show this by induction on i. The base case, $i = 2$, holds since $v_2 = y$ is an out-neighbour of $v_1 = v$. Suppose that $(v_i, v_{i-1}) \in E(D)$ for some $i \geq 2$. If $(v_{i+1}, v_i) \notin E(D)$, then, since G is the underlying graph of D, we must have $(v_i, v_{i+1}) \in E(D)$. But then v_{i-1} and v_{i+1} are two non-adjacent vertices in the out-neighbourhood of v_i, contradicting the assumption that D is out-semi-complete. This establishes the inductive step and the claim.

But now, since $(v_k, v_1) \in E(D)$ and $(v_k, v_{k-1}) \in E(D)$, vertices v_1 and v_{k-1} are two non-adjacent vertices in the out-neighbourhood of v_k, contradicting the assumption that D is out-semi-complete. □

Lemma 3.3.8 and Theorem 3.0.4 imply the following.

Theorem 3.3.9. *Every out-semi-complete digraph D with at least one vertex contains a vertex whose in-neighbourhood is semi-complete. In addition, if $|V(D)| \geq 2$, then D contains at least two vertices whose in-neighbourhoods are semi-complete.*

The contrapositive statement is the following.

Corollary 3.3.10. *Every digraph with at least two vertices in which at most one in-neighbourhood is semi-complete has an out-neighbourhood that is not semi-complete.*

A digraph can be viewed as a communication network, where each vertex is both an information source (passing information to its out-neighbours) and an information recipient (receiving information from its in-neighbours). We say that two information sources / receivers are *independent* if the corresponding vertices are non-adjacent. Thus, Theorem 3.3.9 and Corollary 3.3.10 can be interpreted as follows:

- If no participant passes information to at least two independent recipients, then at least two participants each get information from sources that are not pairwise independent.
- If each participant (but possibly one) gets information from at least two independent sources, then there exists a participant who sends information to at least two independent recipients.

3.4 Avoidable Edges in Graphs

We will call an edge e in a graph G *avoidable* (resp., *simplicial*) if the path P_2 induced by its endpoints is avoidable (resp., simplicial). In particular, if an edge e in a graph G is not the middle edge of any induced P_4, then e is simplicial and thus avoidable. A sufficient condition for an edge $e = uv$ in a graph G to be simplicial is that it is *bisimplicial*, i.e., that $N(u) \cup N(v)$ induces a complete bipartite graph. Bisimplicial edges are relevant for *perfect elimination bipartite* graphs, defined as bipartite graphs whose edges can be eliminated by successively removing both endpoints of a bisimplicial edge, and for their subclass *chordal bipartite graphs*, defined as bipartite graphs without induced cycles of length more than 4, see [76, 75]. Note that an edge in a bipartite graph is simplicial if and only if it is bisimplicial. Thus, the fact that every chordal bipartite graph is perfect elimination bipartite can be equivalently stated as follows: every chordal bipartite graph with an edge has a simplicial edge.

The case $k = 2$ of Conjecture 3.0.8 states that every graph with an edge has an avoidable edge. Theorem 3.0.6 settles this case of the conjecture for $\{C_5, C_6, \ldots\}$-free graphs; in fact, it asserts that every $\{C_5, C_6, \ldots\}$-free graph with an edge has a simplicial edge. Since every chordal bipartite graph is $\{C_5, C_6, \ldots\}$-free, this generalizes the above result for chordal bipartite graphs. A related result is that of Hayward [86] stating that a graph is weakly chordal (that is, both the graph and its complement are $\{C_5, C_6, \ldots\}$-free) if and only if its edges can be eliminated one at a time, where each eliminated edge is simplicial in the subgraph consisting of the remaining edges.

In this section, we prove the case $k = 2$ of Conjecture 3.0.8 for all graphs. Given a graph G, two edges will be called *independent* in G if their endpoints form an induced $2K_2$ in G. We first consider the case when the graph contains no two independent edges.

Lemma 3.4.1. *Let G be a graph with at least two edges but with no two independent edges. Then G contains at least two avoidable edges.*

Proof. Suppose that the lemma is false and let G be a counterexample minimising the number of vertices. We may assume that G is not complete, since otherwise any two edges in G are simplicial and thus avoidable. Let S be a minimal cutset of G.

Case 1: *Graph $G - S$ consists of isolated vertices.*

As S is a minimal cutset, every one of these (at least two) isolated vertices must be adjacent to every vertex in S. Let c be such a vertex and let s be an arbitrary vertex in S. Suppose there is an induced P_4, say $P = x - c - s - y$ with cs as a middle edge. Clearly, vertex x must be in S and thus y also has to be in S, as x is adjacent to every vertex outside of S. This is a contradiction as c is adjacent to every vertex in S. Therefore, cs is avoidable and as there are at least two isolated vertices in $G - S$ we can find two such avoidable edges.

Case 2: *Some connected component of $G - S$ contains an edge.*

As there are no independent edges in G there can be only one such connected component, which we will denote with C. Also, in $G - S$ there must be at least one isolated vertex c'. Note that c' is adjacent to every vertex in S. We analyze two further subcases.

Case 2a: *Component C has exactly two vertices.*

Let c_1 and c_2 be the two vertices of C. Any P_4 with c_1c_2 as its middle edge must be of the form $x - c_1 - c_2 - y$ with x and y in S. Since c' is adjacent to every vertex in S but to none in C, we can close this path to an induced cycle using c'. Thus, c_1c_2 is avoidable. Let s_1 be a an arbitrary vertex in S. Then either $c's_1$ is avoidable or it is the middle vertex of an induced P_4 that does not close to an induced cycle. Without loss of generality such a path is of the form $P = s_2 - c' - s_1 - c_1$ where s_2 is an element of S, as every isolated vertex of $G - S$ is adjacent to every vertex of S. Since edges c_1c_2 and $c's_2$ are not independent in G, we infer that s_2 is adjacent to c_2. As P does not close to an induced cycle, c_2 must be adjacent to s_1. We claim that in this case edge s_2c' is avoidable. Suppose s_2c' is the middle edge of an induced P_4. This path must be of the form $P' = c_2 - s_2 - c' - s_3$ with $s_3 \in S \setminus \{s_1\}$. Since S is a minimal cutset in G, vertex s_3 is adjacent to a vertex in C. Thus, s_3 is adjacent to c_1 and path P' closes to an induced cycle. As a result G has two avoidable edges.

Case 2b: *Component C has more than two vertices.*

Then C has at least two edges, but no two independent edges. Since G is a minimal counterexample, C has at least two avoidable edges $e = vw$ and $f = xy$ which are not necessarily disjoint. We claim that e and f are also avoidable in G. By symmetry, it suffices to show the claim for e. Since e is avoidable in C, every P_4 with e as a middle edge that is completely contained in C can be closed to an induced cycle in C and thus in G as well. Suppose that e is the middle edge of an induced P_4, say $P = t_1 - v - w - t_2$, that is not completely contained in C. First we assume that $t_1 \in S$ and $t_2 \notin S$. Then c' is adjacent to t_1, which implies that edges $c't_1$ and wt_2 are independent in G; a contradiction. Thus, both t_1 and t_2 are in S, and we can close P to an induced cycle using the isolated vertex c', which is adjacent to t_1 and t_2 but not to any endpoint of e. It follows that G contains two avoidable edges. □

Clearly, if a graph has a single edge, this edge is avoidable. Thus, Lemma 3.4.1 has the following consequence.

Corollary 3.4.2. *Every graph with at least one edge but with no two independent edges contains an avoidable edge.*

To consider the case not settled by Lemma 3.4.1, we first introduce some more terminology. Two distinct edges will be called *weakly adjacent* if they are not independent. The set $N_G^E(e)$ will denote the set of edges of G that are weakly adjacent to e. The members of $N_G^E(e)$ are the *edge-neighbours* of e. An edge $e \in E(G)$ will be called *universal* in G if every edge of G other than e is weakly adjacent to e. We say that an edge $e = uv$ is *adjacent to* a vertex w in G (and vice versa) if $w \notin \{u, v\}$ and uw or vw is an element of $E(G)$. The set of vertices that are adjacent to an edge e is called the *vertex-neighbourhood of* e and is denoted by $N_G^V(e)$, while the set of edges adjacent to a vertex v is called the *edge-neighbourhood of* v and is denoted by $N_G^E(v)$. Note that the edge-neighbourhood of any edge $e = uv$ is exactly the set of all edges having at least one endpoint in the vertex-neighbourhood of e.

The case not considered by Lemma 3.4.1 is settled in the next lemma.

Lemma 3.4.3. *For every graph G and every non-universal edge $e \in E(G)$ there is an edge $f \in E(G)$ independent of e which is avoidable.*

Proof. Suppose that the lemma is false and take a counterexample G with the smallest possible number of vertices. Since G is a counterexample, it has a non-universal edge $e \in E(G)$ such that no edge independent from e is avoidable.

We first prove a sequence of claims.

Claim 3.4.4. *Every edge that is independent of e is adjacent to all vertex-neighbours of e in G.*

Proof: Suppose that the claim is false and let $f \in E(G)$ be an edge that is independent of e and non-adjacent to at least one vertex-neighbour p of e in G. Let G' be the graph resulting from contracting the edge e in G. If e' denotes the vertex obtained from e in G', then $pe' \in E(G')$ is independent of f in G'. Since G is a minimal counterexample and as f is independent of pe' in G', there must be an edge $g \in E(G')$ that is independent of pe' and avoidable in G'. It is easy to see that g is also an edge in G that is independent of e and avoidable in G; a contradiction. ■

Let $f = vw$ be an edge that is independent of e and which has the fewest edge-neighbours among all such edges. By Claim 3.4.4, f is adjacent to every neighbour of e and is not avoidable, i.e., it is the middle edge of an induced P_4, say $P = x - v - w - y$, that cannot be closed to an induced cycle. This implies in particular that x and y cannot both be adjacent to e.

Suppose that x is adjacent to e. Then y is not adjacent to e. It follows that edge wy is independent of e and thus adjacent to all neighbours of e, including x; a contradiction. Therefore, x is not adjacent to e and, by symmetry, neither is y.

Note that $N_G^V(e) \subseteq N_G^V(f) \setminus \{x, y\}$. Moreover, the set $N_G^V(e) \cup \{v, w\}$ separates e from x, that is, e and x belong to different components of the graph $G - (N_G^V(e) \cup \{v, w\})$. It follows that we can find a minimal set $S \subseteq N_G^V(e)$ (possibly $S = \emptyset$) such that the set $S \cup \{v, w\}$ separates e from x. Let C be the connected component of $G - (S \cup \{v, w\})$ containing e.

Claim 3.4.5. *Edge f is universal in $G - C$.*

Proof: Suppose not. Then, due to the minimality assumption made on G, there must be an edge $h \in E(G - C)$ that is independent of f and avoidable in $G - C$. As shown above, every vertex in S is adjacent to v or w. Therefore, edge h must be fully contained in a connected component C' of $G - (S \cup \{v, w\})$. As there are no edges between C' and C in G, any induced P_4 that has h as a middle edge is contained in $G - C$ and can be closed to an induced cycle. Thus, h is also avoidable in G as well as being independent of e. ■

Claim 3.4.5 has the following consequence.

Claim 3.4.6. *Every edge $g \in E(G - C)$ that does not have vertex-neighbours in C is universal in $G - C$ and adjacent to every vertex in S. In particular, this holds for any edge that is completely contained in a connected component of $G - (S \cup \{v, w\})$ not equal to C.*

Proof: Let g be an edge in $G - C$ that does not have vertex-neighbours (in G) outside of $G - C$. Then g is independent of e. Recall that edge f was chosen to be independent of e with the smallest number of edge-neighbours. By Claim 3.4.5, f is universal in $G - C$. Note that each edge-neighbour of g is either contained in $G - C$ or contains a vertex from $S \cup \{v, w\}$. This means that each edge-neighbour of g is either equal to f or is an edge-neighbour of f. The choice of f implies $N_G^E(g) = (N_G^E(f) \setminus \{g\}) \cup \{f\}$, showing that g is universal in $G - C$.

Let s be an arbitrary vertex in S. Due to the choice of S, vertex s must have a neighbour in e, say t, and is adjacent to f. Thus, edge st is an edge-neighbour of f. As this edge is also an edge-neighbour of g and g cannot be adjacent to t, it follows that g is adjacent to s. ■

The next claim restricts the structure of components of the graph $G - (S \cup \{v, w\})$.

Claim 3.4.7. *Exactly one component of $G - (S \cup \{v, w\})$ other than C contains an edge.*

Proof: Suppose first that at least two connected components of $G - (S \cup \{v, w\})$ other than C contain edges. Consider two such components and one edge in each. By Claim 3.4.6, each of these edges is universal in $G - C$. This contradicts the fact that they are independent. It follows that at most one component of $G - (S \cup \{v, w\})$ other than C contains an edge.

Suppose for a contradiction that no component of $G - (S \cup \{v, w\})$ other than C contains an edge, that is, all such components are trivial. In particular, the component of $G - (S \cup \{v, w\})$ containing vertex x is trivial. Consider the edge $g = xv$. We already know that g is independent of e. This implies that g is not avoidable in G. Thus, there exists an induced P_4 in G having g as the middle edge, say $Q = t_1 - x - v - t_2$, that does not close to an induced cycle. Clearly, t_1 must be in S and thus adjacent to e. It follows that t_2 cannot be adjacent to e, since otherwise we could close Q to an induced cycle. But now, edge t_2v is independent of e and hence, by Claim 3.4.4, adjacent to every neighbour of e, including t_1; a contradiction. ■

By Claim 3.4.7, exactly one component of $G - (S \cup \{v, w\})$ other than C contains an edge. Let C' be this component. We complete the proof by considering three exhaustive cases.

Case 1: *Both v and w have neighbours outside of $S \cup V(C') \cup \{v, w\}$.*

It follows from Claim 3.4.6 that every edge in C' is universal in C'. Therefore, Corollary 3.4.2 implies that C' contains an avoidable edge g. Since g is independent of e, it is not avoidable in G. This means that there exists an induced P_4 in G having g as the middle edge, say $Q = t_1 - g_1 - g_2 - t_2$, that does not close to an induced cycle. Since g is avoidable in C', path Q cannot be fully contained in C'. Without loss of generality, we may assume that $t_1 \notin V(C')$. It follows that $t_1 \in S \cup \{v, w\}$.

Suppose that $t_2 \in V(C')$. If $t_1 \in S$, then t_2 or g_2 must be adjacent to t_1, due to Claim 3.4.6, as g_2t_2 is an edge contained in C'; this is a contradiction. Therefore, $t_1 \in \{v, w\}$. As both v and w have neighbours outside of $S \cup V(C') \cup \{v, w\}$, similar arguments as those used in the proof of Claim 3.4.6 can be used to show that either t_2 or g_2 must be adjacent to t_1; a contradiction. This shows that $t_2 \notin V(C')$ and therefore $t_2 \in S \cup \{v, w\}$.

If both t_1 and t_2 are in S, then Q can be closed to an induced cycle through e, as all vertices in S are adjacent to e. Therefore, we may assume without loss of generality that $t_1 \in \{v, w\}$ and $t_2 \in S$. By the assumption of Case 1, vertex t_1 has a neighbour z outside of $S \cup V(C') \cup \{v, w\}$. If z is in C, then there is a path from t_1 to t_2 through C, and if z is not in C, then z is an isolated vertex of $G - (S \cup \{v, w\})$ and $t_1 - z - t_2$ is a path in G. In either case, using such a path, Q can be closed to an induced cycle in G; a contradiction.

Case 2: *Exactly one of v and w has a neighbour outside of $S \cup V(C') \cup \{v, w\}$.*

We may assume without loss of generality that v does not have any neighbours outside of $S \cup V(C') \cup \{v, w\}$, but w does. Claim 3.4.6 implies that every edge in $G[C' \cup \{v\}]$ is universal. Hence, we infer, using Corollary 3.4.2, that $G[C' \cup \{v\}]$ contains an avoidable edge g. Since v does not have any neighbours outside of $S \cup V(C') \cup \{v, w\}$, edge g (which might contain v as an endpoint) is independent of e. This implies that g is not avoidable in G, i.e., there exists an induced P_4 in G having g as the middle edge, say $Q = t_1 - g_1 - g_2 - t_2$, that does not close to an induced cycle. Since g is avoidable in $G[C' \cup \{v\}]$, path Q cannot be fully contained in $G[C' \cup \{v\}]$. Without loss of generality, we may assume that $t_1 \notin V(C') \cup \{v\}$. It follows that $t_1 \in S \cup \{w\}$.

Suppose that $t_2 \in V(C') \cup \{v\}$. If $t_1 \in S$, then t_2 or g_2 must be adjacent to t_1, due to Claim 3.4.6; this is a contradiction. Therefore, $t_1 = w$. As w has a neighbour outside of $S \cup V(C') \cup \{v, w\}$, similar arguments as those used in the proof of Claim 3.4.6 can be used to show that either t_2 or g_2 must be adjacent to t_1, leading again to a contradiction. This shows that $t_2 \notin V(C') \cup \{v\}$ and therefore $t_2 \in S \cup \{w\}$.

If both t_1 and t_2 are in S, then Q can be closed to an induced cycle through e, as all vertices in S are adjacent to e. Therefore, we may assume without loss of generality that $t_1 = w$ and $t_2 \in S$. By the assumption of Case 2, vertex w has a neighbour z outside of $S \cup V(C') \cup \{v, w\}$. If z is in C, then there is a path from t_1 to t_2 through C. On the other hand, if z is not in C, then z is an isolated vertex of $G - (S \cup \{v, w\})$ and $t_1 - z - t_2$ is a path in G. In either case, using such a path Q can be closed to an induced cycle in G; a contradiction.

Case 3: *Neither v nor w has a neighbour outside of $S \cup V(C') \cup \{v, w\}$.*

Claim 3.4.6 implies that every edge in $G[C' \cup \{v, w\}]$ is universal, hence, by Corollary 3.4.2, there is an avoidable edge g in $G[C' \cup \{v, w\}]$. The assumption of Case 3 implies that edge g is independent of e. This implies the existence of an induced P_4 in G having g as the middle edge, say $Q = t_1 - g_1 - g_2 - t_2$, that does not close to an induced cycle. Since g is avoidable in $G[C' \cup \{v, w\}]$, path Q cannot be fully contained in $G[C' \cup \{v, w\}]$. Without loss of generality, we may assume that $t_1 \notin V(C') \cup \{v, w\}$. It follows that $t_1 \in S$. Moreover, $t_2 \in S$, since otherwise t_2 would belong to $V(C') \cup \{v, w\}$ and Claim 3.4.6 would imply that one of t_2 or g_2 is adjacent to t_1; a contradiction. Since both t_1 and t_2 are in S, path Q can be closed to an induced cycle through e, as all vertices in S are adjacent to e; a contradiction.

This completes the proof of the lemma. □

Lemmas 3.4.1 and 3.4.3 imply the following.

Theorem 3.4.8. *Every graph with an edge has an avoidable edge. Every graph with at least two edges has two avoidable edges.*

Proof. Clearly, if G has a single edge, that edge is avoidable. So let G be a graph with at least two edges. If G does not have two independent edges, then Lemma 3.4.1 applies and the desired conclusion follows. If, on the other hand, G does have a pair of independent edges, say e and e', then e is not universal in G and hence by Lemma 3.4.3 there is an edge $f \in E(G)$ independent of e which is avoidable. Since f is independent of e, it is not universal. Applying Lemma 3.4.3 again, we find that G contains an avoidable edge f' that is independent of f. Clearly, the edges f and f' are distinct. □

An application of this theorem to edge-transitive graphs is given in the next section.

3.5 Consequences for Highly Symmetric Graphs

In this section we summarize the consequences of existence results for avoidable vertices and edges for vertex- and edge-transitive graphs. An *automorphism* of a graph G is a bijection from the vertex set of G to itself that maps edges to edges and non-edges to non-edges. A graph G is said to be *vertex-transitive* if for any two vertices $u, v \in V(G)$ there exists an automorphism of G mapping u to v. Similarly, G is said to be *edge-transitive* if for any two edges $e, f \in E(G)$ there exists an automorphism of G mapping e to f.

By Theorem 3.0.4, the midpoint of any induced P_3 in a vertex-transitive graph is avoidable. This implies the following consequence for vertex-transitive graphs.

Corollary 3.5.1. *Every induced P_3 in a vertex-transitive graph closes to an induced cycle.*

As shown by the example in Figure 3.4, a statement analogous to that of Corollary 3.5.1 fails for longer paths.

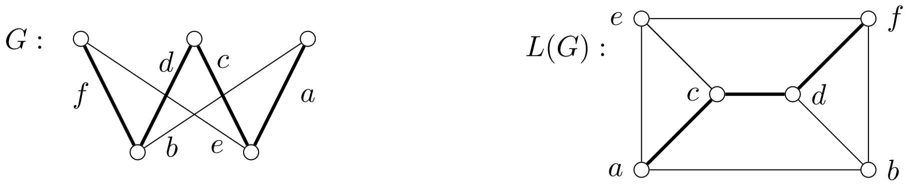


Figure 3.4: The graph $L(G)$ is vertex-transitive and contains an induced four-vertex path (a, c, d, f) that does not close to an induced cycle.

A similar consequence can be derived for edge-transitive graphs, by considering avoidable vertices in the line graph of a graph. By Corollary 3.1.3, the middle edge of any 3-edge path in an edge-transitive graph is pseudo-avoidable.

Corollary 3.5.2. *Every 3-edge path in an edge-transitive graph closes to a cycle.*

A statement analogous to that of Corollary 3.5.2 fails for longer paths. The graph shown in Figure 3.4 is the line graph of the complete bipartite graph $K_{2,3}$, which is edge-transitive. The 4-vertex induced path depicted bold in the figure corresponds to a 4-edge path in $K_{2,3}$ that does not close to a cycle.

By Theorem 3.4.8, the middle edge of any induced P_4 in an edge-transitive graph is avoidable. This implies the following consequence for edge-transitive graphs.

Corollary 3.5.3. *Every induced P_4 in an edge-transitive graph closes to an induced cycle.*

As shown by the example in Figure 3.5, a statement analogous to that of Corollary 3.5.3 fails for longer paths. This 13-vertex example was found using a computer-assisted search based on a catalogue of vertex-transitive graphs due to McKay and Royle [114] (which has been extended in the years since then) and performed using SageMath [127]. Once the graph was found, a drawing of it given in the figure was identified with the help of House of Graphs [25].

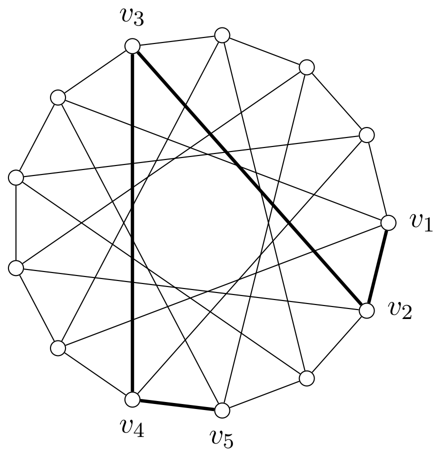


Figure 3.5: An edge-transitive graph with an induced five-vertex path $(v_1, v_2, v_3, v_4, v_5)$ that does not close to an induced cycle.

3.6 Conclusion

We have introduced the notion of avoidability in graphs, a concept that has been implicitly used in a variety of contexts in algorithmic graph theory. We discussed both structural and algorithmic aspects of avoidable vertices, including a characterisation of avoidable vertices as simplicial vertices in some minimal triangulation of the graph, a new proof of the existence of avoidable vertices in graphs with at least one vertex, and the fact that one or two avoidable vertices in a graph can be found in linear time using a simple application of lexicographic breadth first search. This approach was then used to construct a polynomial-time algorithm for the maximum weight clique problem in the class of 1-perfectly orientable graphs and a superclass of these, the graphs admitting an orientation in which every hole is oriented cyclically. We suggested a generalisation of the concept of avoidability from vertices to nontrivial induced paths and proposed a conjecture about their existence (Conjecture 3.0.8). In this respect we showed the validity of the conjectures for edges, that is, two-vertex paths. Many interesting questions remain.

The main open question related to this work is to resolve the status of Conjecture 3.0.8. Theorems 3.0.4 and 3.4.8 imply that the conjecture is true for $k \in \{1, 2\}$. In turn, this fact and Theorem 3.0.6 imply that the conjecture is true for the class of $\{C_6, C_7, \ldots\}$-free graphs, which includes several well studied graph classes such as weakly chordal graphs, cocomparability graphs, and AT-free graphs.

While we have given a linear-time algorithm to compute two distinct avoidable vertices in any nontrivial graph (Theorem 3.2.5), it would also be of interest to devise an algorithm to compute *all* avoidable vertices that is more efficient than the naïve approach.

Having introduced the class of hole-cyclically orientable graphs as a generalisation of 1-perfectly orientable graphs, we can ask for structural properties of these graphs. In particular, it is not known whether they can be recognized in polynomial time. The complexity of the maximum independent set and k-colouring problems (for fixed $k \geq 3$) is also open both for 1-perfectly orientable and for hole-cyclically orientable graphs.

4 Recognising End-Vertices of Graph Searches

Usually, the outcome of a graph search is a *search order*, i.e., a sequence of the vertices in order of visits. There are many results using such orders. For instance, by reversing an LBFS order of a chordal graph, one finds a perfect elimination order of this graph [126]. A perfect elimination order in $G = (V, E)$ is an ordering of the vertices such that for every vertex $v \in V$ the neighbours of v that occur after v in the order form a clique. This not only yields a linear recognition algorithm for chordal graphs, but also a greedy colouring algorithm for finding a minimum colouring for this graph class [75]. As most graph searching paradigms can be implemented in linear time, these algorithms are typically as efficient as possible.

Interestingly, the end-vertices of graph searches, i.e., the last vertices visited in the search, are crucial for several algorithms. Their properties are the key for many multi-sweep algorithms on graphs. For instance, one can use six LBFS runs to construct the interval model of an interval graph. Here, the end-vertices correspond to the end vertices of the interval model and the next search starts in an end-vertex of the previous one. This also yields a linear time recognition algorithm for this graph class [43].

Additionally, end-vertices may have strong structural properties. For example, as a direct consequence of the results mentioned above, the end-vertex of an LBFS on a chordal graph is always simplicial. Moreover, if a cocomparability graph is hamiltonian, then the end-vertex of an LDFS is the start vertex of a hamiltonian path in this graph [45].

LBFS also provides a linear time algorithm for finding dominating pairs in connected asteroidal triple-free graphs [40]. Here, a dominating pair is a pair of vertices such that every path connecting them is a dominating set in the graph. The first vertex x is simply the end-vertex of an arbitrary LBFS and the second vertex y is the end-vertex of an LBFS starting in x. Furthermore, the end-vertex of an LBFS in a cocomparability graph is always a source/sink in some transitive orientation of its complement [79].

The end-vertices of BFS are also helpful for fast diameter computation. Crescenzi et al. [46] have shown that the diameter of large real world graphs can usually be found with only a few applications of BFS. Furthermore, it was shown that the end-vertices

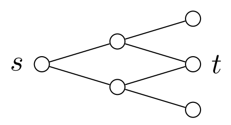


Figure 4.1: Vertex t cannot be the end-vertex of any BFS although it is in the last layer if the BFS starts in s.

of (L)BFS can be used to approximate the diameter of graphs in the classes of chordal graphs [52], HHD-free graphs [51], k-chordal graphs [41] and hyperbolic graphs [31]. Intuitively, the potential end-vertices can be seen as peripheral or extremal vertices of the graph whereas all other vertices appear more centrally and are of greater importance to the connectivity. For example, no search can end on a cut vertex [30]. However, even for BFS finding such vertices is not as trivial as it might seem. Figure 4.1 shows a graph in which the vertex t appears in the last layer of a BFS starting in s. Nevertheless, it cannot be an end-vertex.

In this context, the decision problem arises, whether a vertex can be the *end-vertex* of a graph search, i.e., the last vertex visited by this search.

End-vertex Problem

Instance:	A connected graph $G = (V, E)$ and a vertex $t \in V$.
Task:	Decide whether there is a graph search such that t is the end-vertex of this search on G.

Obviously, this problem is in $\mathcal{NP}$ for any of the searches considered here, since a full search order provides a certificate which can be checked in polynomial time.

Although the problem of checking whether a given tree is a BFS or DFS tree can be decided efficiently (see [81, 100, 112]), surprisingly this does not hold for the end-vertex problem. Corneil, Köhler, and Lanlignel [44] have shown that it is $\mathcal{NP}$-hard to decide whether a vertex can be the end-vertex of an LBFS. Charbit, Habib, and Mamcarz [30] have shown that the end-vertex problems for BFS and DFS are also $\mathcal{NP}$-complete. Furthermore, they extended these results to several graph classes. In this chapter, we address the end-vertex problem for MCS and MNS; for some other related results see Table 4.1.

In the following, we give an overview of graph searching algorithms, especially Maximum Cardinality Search and Maximal Neighbourhood Search. We present $\mathcal{NP}$-completeness results for the end-vertex problem of MNS on weakly chordal graphs and of MCS on general graphs. For some chordal graph classes we give linear time algorithms for several different graph searches. An overview of our results can be found in Table 4.1. We conclude the chapter with some open problems related to our results.

The results given in this chapter have been achieved in joint work with Carolin Denkert, Ekkehard Köhler, Matjaž Krnc, Nevena Pivač, Robert Scheffler and Martin Strehler and can be found in [9].

4.1 $\mathcal{NP}$-Completeness for Maximal Neighbourhood Search

The complexity of the end-vertex problem of MNS was studied by Berry et al. [15] in 2010 resulting in the following characterisation.

Lemma 4.1.1 (Berry et al. [15]). *Let $G = (V, E)$ be a chordal graph and let $t \in V$. Then t can be an end-vertex of MNS if and only if t is simplicial and the minimal separators included in $N(t)$ are totally ordered by inclusion.*

	BFS	LBFS	DFS	LDFS	MCS	MNS
All Graphs	NPC	NPC	NPC	NPC	**NPC**	**NPC**
Weakly Chordal	NPC [30]	NPC [44]	NPC	NPC [30]	?	**NPC**
Chordal	?	?	NPC	?	?	P [15] → **L**
Interval	?	L [44]	**L**	?	?	P → **L**
Unit Interval	?	L	**L**	**L**	**L**	P → **L**
Split	L [30]	P [30] → **L**	NPC [30]	P [30] → **L**	**L**	P → **L**

Table 4.1: Complexity of the end-vertex problem. Bolded results are made in this text. The big L stands for linear time algorithm. The term $P \to L$ describes the improvement from a polynomial algorithm to a linear time algorithm. Non-bolded results without references are direct consequences of other results, e.g., $\mathcal{NPC}$ of DFS on split graphs implies $\mathcal{NPC}$ on weakly chordal, chordal and general graphs. On the other hand, the linear time algorithm for interval graphs is also a linear time algorithm for the subclass of unit interval graphs.

Since this property can be checked efficiently, they conclude that the end-vertex problem of MNS on chordal graphs is solvable in polynomial time. In Section 4.3.1 we provide an approach which solves this problem in linear time (see Corollary 4.3.4).

Charbit et al. [30] conjectured that the problem can be solved efficiently on general graphs. However, in this section we will present an $\mathcal{NP}$-completeness proof for the end-vertex-problem of MNS on weakly chordal graphs.

Theorem 4.1.2. *The end-vertex-problem of MNS is $\mathcal{NP}$-complete for weakly chordal graphs.*

To prove this we use a reduction from 3-SAT. Let $\mathcal{I}$ be an instance of 3-SAT. We construct the corresponding graph $G(\mathcal{I})$ as follows (see Figure 4.2 for an example). Let $X = \{x_1, \ldots, x_k, \overline{x}_1, \ldots, \overline{x}_k\}$ be the set of vertices representing the literals of $\mathcal{I}$. The edge-set $E(X)$ forms the complement of the matching in which x_i is matched to $\overline{x}_i$ for every $i \in \{1, \ldots, k\}$. Let $C = \{c_1, \ldots, c_l\}$ be the set of clause vertices representing the clauses of $\mathcal{I}$. The set C is independent in $G(\mathcal{I})$ and every c_i is adjacent to every vertex of X, apart from those representing the literals of the clause associated with c_i for every $i \in \{1, \ldots, l\}$. Additionally, we add the vertices s, b and t. The vertex b is adjacent to all literal vertices. The vertices s and t are adjacent to all literal and all clause vertices. Finally, we add the edge bt.

The following two lemmas provide some properties that an MNS ordering must fulfil if t is its end-vertex. Let $\mathcal{I}$ be an arbitrary instance of 3-SAT and $G = G(\mathcal{I})$ with $n = |V(G)|$.

Lemma 4.1.3. *Let σ be an MNS ordering of vertices in G, ending with t. Then:*

1. *The vertex b is on the left of any clause vertex in ordering σ;*
2. *The vertex s is on the left of b in ordering σ.*

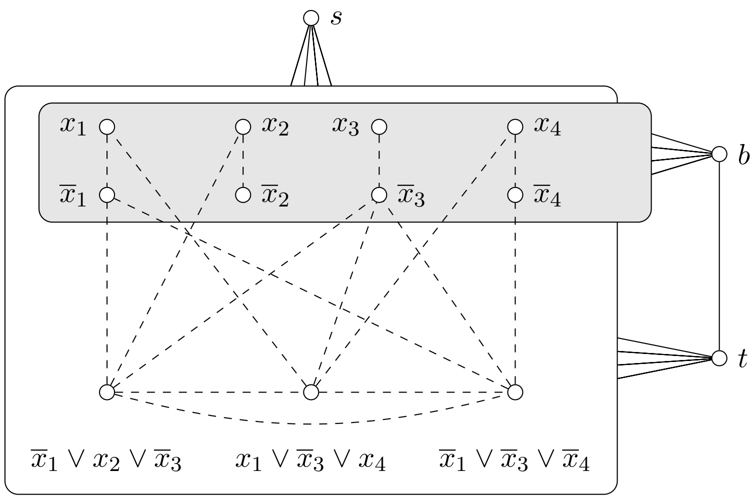

s
x_1
x_2
x_3
x_4
$\overline{x}_1$
$\overline{x}_2$
$\overline{x}_3$
$\overline{x}_4$
b
t
$\overline{x}_1 \vee x_2 \vee \overline{x}_3$
$x_1 \vee \overline{x}_3 \vee x_4$
$\overline{x}_1 \vee \overline{x}_3 \vee \overline{x}_4$

Figure 4.2: The $\mathcal{NP}$-completeness reduction for the end-vertex problem of MNS on weakly chordal graphs. The depicted graph is $G(\mathcal{I})$ for $\mathcal{I} = (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_3 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee \overline{x}_4)$. In both boxes only non-edges are displayed by dashed lines. The connection of a vertex with a box means that the vertex is adjacent to all vertices in the box.

Proof. We prove both claims separately. Assume that there is a clause vertex c_i that is visited before the vertex b, i.e., $c_i \prec_\sigma b$. By the construction we know that $N(b) \subset N(t)$ and $C \subseteq N(t)$ while $N(b) \cap C = \emptyset$. If vertex c_i is visited in σ before the vertex b, then the label of vertex t will contain the label c_i, while for b this is not the case. This implies that at any step of the search process in G, the label set of b will be a proper subset of the label set of t, and the algorithm will take vertex t before b.

Assume now that the vertex b is on the left of s in the ordering σ. Since $N(s) \subset N(t)$ and b is in the neighbourhood of t and s is not, the label set of s will be a proper subset of the label set of t. Thus, the search algorithm will visit t before s. □

Lemma 4.1.4. *Let σ be an MNS ordering of vertices in G, ending in t. Then the first $k+1$ vertices in σ are s, as well as an arbitrary assignment of $\mathcal{I}$ (not necessarily satisfying).*

Proof. It follows from Lemma 4.1.3 that the first vertex in σ is s or one of the literal vertices. Assume that we are at some step of an MNS search and until now we have only chosen at most one literal vertex per variable, possibly including the vertex s. Further assume that there is at least one variable whose two literal vertices have not been chosen so far. These vertices are adjacent to all the vertices which have been chosen before. Lemma 4.1.3 implies that a clause vertex cannot be next in the MNS ordering. For the same reason, if s has not been visited so far, we cannot take b next. If s has already been chosen, then the labels of literal vertices of the non-chosen variables are proper supersets of the label of b; again we cannot choose the vertex b. The same holds for literal vertices of variables, for which the other literal vertex has already been visited. Thus, we know that we have to take s or a literal vertex of an unvisited variable. Notice that each of these vertices is a possible choice in an MNS, since they all have maximum label. This proves the statement. $\square$

So far we have proven some necessary conditions that have to be satisfied in order for t to be an MNS end-vertex in G. In what follows, we will prove that t can be the end-vertex of MNS in G whenever the corresponding 3-SAT instance $\mathcal{I}$ has a satisfying assignment.

Lemma 4.1.5. *If the 3-SAT instance $\mathcal{I}$ has a satisfying assignment $\mathcal{A}$, then t is an end-vertex of MNS on G.*

Proof. Let $\mathcal{I}$ be an instance of 3-SAT and $\mathcal{A}$ a satisfying assignment of $\mathcal{I}$. We now construct the MNS ordering which ends in t.

From Lemma 4.1.4 it follows that we can start an MNS search on G in the vertex s and then take all the literal vertices that belong to $\mathcal{A}$. Since each clause vertex is not adjacent to its corresponding literals, it follows that for each clause vertex there is at least one label missing among the labels produced by the assignment vertices. The same holds for the unvisited literal vertices, since their negated literal label is missing. Furthermore, since b is adjacent to all literal vertices, labels of unvisited literal vertices as well as of clause vertices do not contain the label set of vertex b. Therefore, we can take b as the next vertex. Since all remaining literal vertices and clause vertices are adjacent to s, while t is not, these remaining vertices can be visited before t. $\square$

We now show that t cannot be the end-vertex of an MNS if there is no satisfying assignment of $\mathcal{I}$.

Lemma 4.1.6. *If the 3-SAT instance $\mathcal{I}$ has no satisfying assignment, then t cannot be the end-vertex of MNS on G.*

Proof. Let $\mathcal{I}$ be an instance of 3-SAT which does not have a satisfying assignment. Suppose that there exists an MNS ordering σ of G which ends in t. It follows from Lemma 4.1.4 that MNS has to take the vertex s and an arbitrary assignment of $\mathcal{I}$ at the beginning. Since the assignment that was chosen is not satisfying, there is at least one clause vertex that has been labelled by all vertices chosen so far. Hence, the labels of these clause vertices properly contain the labels of all remaining literal vertices, as well as the labels of b and t. As a result, MNS has to take one of these clause vertices next. Lemma 4.1.3 implies that then t cannot be an end-vertex, since one clause vertex was chosen before b. $\square$

To complete the proof of Theorem 4.1.2 we have to show, that $G(\mathcal{I})$ is weakly chordal for any choice of $\mathcal{I}$.

Lemma 4.1.7. *The graph $G(\mathcal{I})$ is weakly chordal for any instance $\mathcal{I}$ of 3-SAT.*

Proof. By contradiction assume that there exists an induced cycle of length ≥ 5 in $G(\mathcal{I})$ or its complement. We start with $G(\mathcal{I})$. Vertex t cannot be part of such a cycle, since there is only one vertex which is not adjacent to t. Thus, s also cannot be part of such a cycle, since it is adjacent to all vertices but t and b. Note, if the cycle contains four or more vertices that are not clause vertices there must be a chord in the cycle. Therefore, the cycle contains at least two clause vertices c_i and c_j. The neighbours of c_i and c_j in the cycle are literal vertices. If the neighbours are four different vertices, then there exists at least four edges between them. Therefore, there must be a chord. If both share one neighbour, then there is also a chord between these shared neighbours and one of the other two. The clause vertices cannot share both neighbours, since the cycle has more than four vertices. Thus, there is no induced cycle with more than four vertices in $G(\mathcal{I})$.

Consider now $\overline{G(\mathcal{I})}$. Since t is only adjacent to s, it is not part of a cycle. The same holds for s, since it is only adjacent to t and b. As the clause vertices build a clique, there are at most two of them in a cycle which must be consecutive. Between literal vertices of different variables and between b and a literal vertex must lie at least one clause vertex in the cycle. This leads to a contradiction, since we need at least two non-consecutive clause vertices. $\square$

Theorem 4.1.2 follows from the Lemmas 4.1.5, 4.1.6 and 4.1.7.

4.2 $\mathcal{NP}$-Completeness for Maximum Cardinality Search

To the best of our knowledge, the end-vertex problem for Maximum Cardinality Search has not been studied in the literature. It is easy to see that for trees the end-vertices correspond exactly to the leaves.

However, in this section, we will prove that the end-vertex problem is $\mathcal{NP}$-complete on general graphs, by giving a reduction from 3-SAT.

Theorem 4.2.1. *The MCS end-vertex problem is $\mathcal{NP}$-complete.*

For each instance $\mathcal{I}$ of 3-SAT we construct a corresponding graph $G(\mathcal{I})$ as follows (see Figures 4.3, 4.4 and 4.5 for an example): Each literal is represented by an edge, and each clause by a triangle. The triangles representing the clauses together form a clique C of size $3l$. We also define start vertices s and s', where s is adjacent to all vertices of x_1 and $\overline{x}_1$ and s' is just adjacent to s. Two consecutive literals x_{j-1} and x_j are connected using two auxiliary vertices in the way described in Figure 4.4. For each literal one of its two vertices is adjacent to all three vertices of each clause which contains the negation of that literal, as depicted in Figure 4.5.

Additionally, the graph contains a clique K with $3(4k+8(k-1))+4$ vertices, i.e., three vertices for each literal and auxiliary vertex, as well as 4 connector vertices. Every vertex among the literal vertices and the auxiliary vertices is adjacent to exactly 3 vertices in K such that every vertex in K apart from the 4 connector vertices are adjacent to exactly one vertex outside of K. Two of the connector vertices are then completely connected to all vertices of x_k and the other two to all vertices of $\overline{x}_k$.

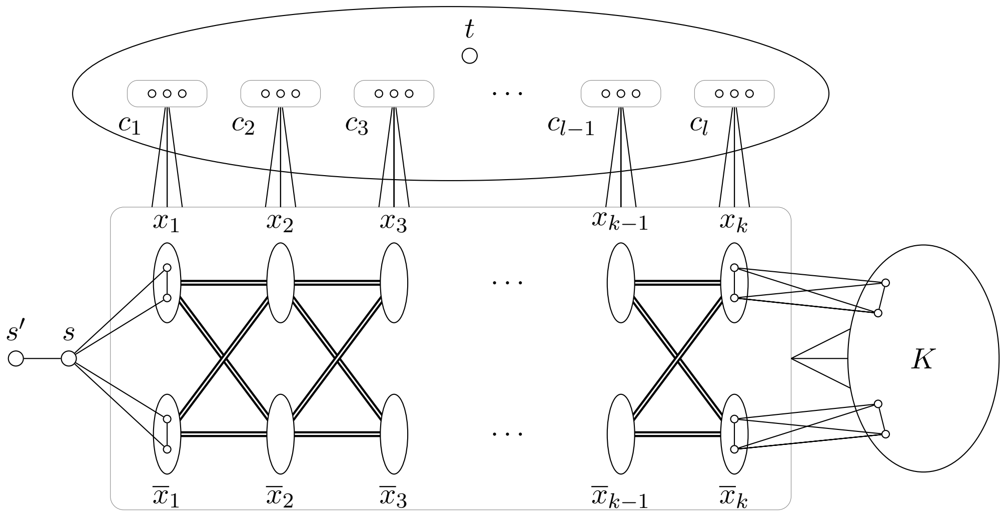


Figure 4.3: This represents a general construction of $G(\mathcal{I})$ for an arbitrary instance $\mathcal{I}$. The double edges denote a construction linking the various literals and are explained in Figure 4.4

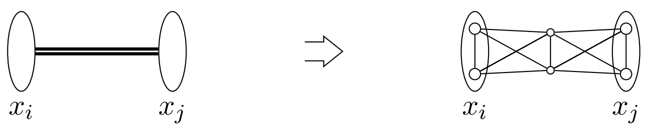


Figure 4.4: Each double edge in the construction shown in Figure 4.3 is to be replaced by the construction on the right-hand side. Furthermore, each node, in particular the auxiliary vertices in the middle, is assigned to three exclusive neighbours in the clique K.

Finally, we add a vertex t, for which we wish to decide whether it is an end-vertex, and this is adjacent to all clause vertices.

Lemma 4.2.2. *If $\mathcal{I}$ has a satisfying assignment $\mathcal{A}$, then there is a maximum cardinality search on $G(\mathcal{I})$ that ends in t.*

Proof. Let $\mathcal{A} = (b_1, b_2, \dots, b_l)$ be a satisfying assignment of $\mathcal{I}$. We can construct a corresponding MCS order σ ending in t as follows: As $\sigma(1)$ we use the root s' and $\sigma(2) = s$. If $b_1 = 1$ we choose the vertices of x_1 next; if $b_1 = 0$ we choose the vertices of $\overline{x}_1$. In the next step we use the construction described in Figure 4.4 to choose either x_2 or $\overline{x}_2$, depending on the value of b_2. We proceed in this way, until we have visited every literal given by $\mathcal{A}$, that is until we reach x_k or $\overline{x}_k$. This is possible, because we have chosen a satisfying assignment and the label of every clause vertex is at most 2, while there is always an auxiliary vertex or a literal vertex with label that is at least 2.

At the point where we have visited both vertices in the k-th literal, one pair of connector vertices of the clique K will have label 2. As we have chosen a fulfilling assignment in the manner described above, those connector vertices have maximal label at this point.

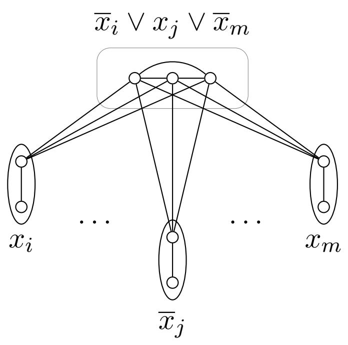


Figure 4.5: Each clause block consists of three nodes and each of these nodes is connected to one specific node of the corresponding literals.

Hence, it is possible to visit the whole clique K next. Note that after every vertex in K has been chosen, each vertex that has not been visited and that is neither a clause vertex nor t has a label larger or equal to 3, as each has three neighbours in the clique. On the other hand, every clause vertex can only have label at most 3, unless t is visited, as it has only three neighbours in $G - (C \cup \{t\})$. Thus, we can visit every vertex apart from the clause vertices and t first, then choose all the clause vertices in any order possible and finally visit t last. □

Lemma 4.2.3. *If $\mathcal{I}$ does not admit a satisfying assignment, then t cannot be an MCS end-vertex of $G(\mathcal{I})$.*

Proof. Suppose that $\mathcal{I}$ does not admit a satisfying assignment and there is an MCS ending in t. First observe that any maximum cardinality search that ends in t must begin in s or s'. If we start in an arbitrary vertex that is neither s nor s', then at any point of the search there will always be a vertex with label larger or equal to 2 until s' is the only vertex left. Without loss of generality, it begins in s', as otherwise we can choose s' as the second vertex. It is enough to show that it is not possible to reach K before visiting vertex t. First, note that K can only be entered through the connector vertices, which in turn are only adjacent to x_k or $\overline{x}_k$, as at any point of the MCS there is a vertex with label ≥ 2, while a vertex in K has label ≤ 1 until the connector vertices are chosen.

Suppose that at any point in the search a clause vertex is chosen for the first time before we have reached K. Then either all of the vertices of that clause are visited consecutively, or the remaining vertices of that clause will be labelled with 3 before we reach K (they must have had label 2 before the clause was entered and the choice of one clause vertex increased the label to 3). In this case, therefore, t cannot be chosen last in the search, as at least three clause vertices, and as a consequence also t, must be chosen before we can enter K.

It remains to be shown, that a clause vertex has to be visited before entering K. We have already argued that one of the literals of x_k must be visited before we can enter K, as we can only enter through the connector vertices. If we disregard the clause vertices and K, then the vertex set corresponding to a given variable x_i with $i < k$ forms a separator between s and the vertices corresponding to the variable x_k. Therefore, any search must traverse at least one vertex of each variable. Furthermore, as soon as the first vertex belonging to a literal is chosen, the other vertex in that literal has the largest label and must be chosen next. Hence, at least one literal for every variable must be visited, before the search reaches x_k (note that it is possible to visit both literals of a variable). As a result, we need to traverse *at least* a whole assignment $\mathcal{A}$ of the literals before entering K. However, as $\mathcal{I}$ does not admit a satisfying assignment, this implies that by the time we have visited one literal of x_k, at least one of the clause vertices must have label 3, as all its literals have been assigned in the negative, and we must visit this vertex before entering the clique K.

Therefore, it is impossible to reach K before we have chosen a clause vertex and t cannot be an end-vertex of an MCS. □

The following corollary concludes the proof of Theorem 4.2.1.

Corollary 4.2.4. *Let $\mathcal{I}$ be an instance of 3-SAT. Then $\mathcal{I}$ has a satisfying assignment if and only if t is a possible MCS end-vertex of $G(\mathcal{I})$.*

4.3 Linear Time Algorithms for some Chordal Graph Classes

While the end-vertex problem of MCS is $\mathcal{NP}$-complete on general graphs, we will now present linear time algorithms for the end-vertex problem on split graphs and unit interval graphs. With the used approaches we were also able to improve some polynomial results for the other searches to linear time algorithms. We begin with the following lemma which we will use repeatedly throughout.

Lemma 4.3.1. *Given a graph $G = (V, E)$ and a vertex $t \in V$. There is a linear time algorithm which decides whether t is simplicial.*

Proof. At first, we mark each neighbour of t with a special bit. Now for every neighbour v of t we count, how many neighbours of v are also neighbours of t. Because of the special bits this can be done in $\mathcal{O}(|N(v)|)$. If for every neighbour of t this number is equal to the degree of t, t is simplicial. Otherwise it is not. The overall running time is linear in the size of G, since we visit each edge only a constant number of times. □

4.3.1 Split Graphs

As split graphs are chordal, we know that Lemma 4.1.1 yields a necessary condition for being an MCS end-vertex. In Figure 4.6 we present an example which shows that this condition is not sufficient. However, in the following, we show that a slight strengthening of this condition is enough for a complete characterisation.

Theorem 4.3.2. *Let $G = (V, E)$ be a split graph. Then $t \in V$ is the last vertex of some MCS-ordering σ of G if and only if*

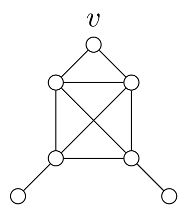


Figure 4.6: Vertex v is an MNS end-vertex, since it fulfils the conditions of Lemma 4.1.1. However, by enumerating all possible searches one can see that v is not an MCS end-vertex.

1. *The vertex t is simplicial;*
2. *The neighbourhoods of the vertices with a smaller degree than t are totally ordered by inclusion.*

Proof. As G is a split graph, we can assume that its vertex set can be partitioned into a clique C and an independent set I such that C is a maximal clique in G, i.e., the neighbourhood of each vertex in I is a proper subset of the neighbourhood of each vertex in C.

Let t be an end-vertex of MCS on G. As G is chordal, we see that t must be simplicial. Now, assume the second condition does not hold. Then there are two vertices v and w with smaller degrees than t whose neighbourhoods are incomparable with regard to inclusion. Without loss of generality assume that v is taken before w in σ in the MCS where t is an end-vertex.

We first show that v and w have to be elements of I. If t is an element of I, then this is easy to see, as every vertex in C has a higher degree than the vertices in I. If t is an element of C, then t does not have any neighbour in I, as it is simplicial. Therefore, v and w cannot be elements of C, since otherwise their degree would not be smaller than the degree of t.

Now observe that all neighbours of v are visited before w. Indeed, after we have taken the vertex v in σ, the remainder of $N(v)$ has a larger label than the other vertices in $C \setminus N(v)$. Since there is at least one neighbour of v which is not a neighbour of w, the remainder of C will always have larger label than w after we have taken all neighbours of v. Hence, all vertices of C have to be visited before w in σ. However, this is a contradiction, since from the moment where all the vertices in C have been visited, the label of t is always greater than the label of w and, thus, t has to be chosen before w in σ.

Let us now assume that both conditions hold for t. We claim that the following ordering is a valid MCS search that has t as an end-vertex. We choose the neighbourhoods of all vertices with lower degree than t in the order of the inclusion ordering. Every time the complete neighbourhood of such a vertex u has been visited, we choose u next. If t is an element of C, there are no remaining vertices of I and we take the remaining vertices of C in an arbitrary ordering, where t is the last vertex. If t is an element of

I we take the remaining vertices of C first. Since the neighbourhood of t is not greater than the neighbourhood of any remaining vertex, we can visit t last. □

Proposition 4.3.3. *The MCS end-vertex problem can be decided in linear time if the given input is a split graph.*

Proof. By Lemma 4.3.1 the simpliciality of t can be checked in linear time. To check the second condition, we first sort the vertices with degrees smaller than t by their degree. This can be done in linear time using counting sort. Let $v_1, \ldots, v_k$ be this order, where $deg(v_1) \geq \ldots \geq deg(v_k)$. Then we create an array A of size n, whose elements correspond to the vertices of G. At first we mark each element of A which corresponds to a neighbour of v_1 with one. For each v_i with $1 < i \leq k$ we check, whether all elements corresponding to a neighbour of v_i are marked with $i - 1$. If this is not the case, the neighbourhoods of v_{i-1} and v_i are incomparable with regard to inclusion. Otherwise, we mark each neighbour of v_i with i. The overall running time of this algorithm is $\mathcal{O}(n+m)$, since we visit each edge a constant number of times. □

Note, that the same approach can be used to improve the results of Charbit et al. [30] for the end-vertex problems of LBFS and LDFS on split graphs to linear running time. Furthermore, we can use it to improve the complexity of the end-vertex problem of MNS on chordal graphs. To decide this problem, we can check the conditions of Lemma 4.1.1. As the minimal separators of a chordal graph can be determined in $\mathcal{O}(n + m)$ using MCS [107], the technique of Proposition 4.3.3 leads to a linear time algorithm.

Corollary 4.3.4. *The end-vertex problem of LBFS and of LDFS on split graphs can be solved in linear time. Furthermore, the end-vertex problem of MNS on chordal graphs can be solved in linear time.*

4.3.2 Unit Interval Graphs

As any MCS (and also every LDFS) is an MNS, we know that in a chordal graph $G = (V, E)$ a necessary condition for a vertex $t \in V$ being an MCS (or LDFS) end-vertex is that t is simplicial and that the minimal separators in its neighbourhood can be ordered by inclusion (as seen in Lemma 4.1.1). In the following we will proceed to show that in unit interval graphs this is also a sufficient condition.

In [64] Fulkerson and Gross showed that a graph G is an interval graph if and only if the maximal cliques of G can be linearly ordered such that, for each vertex v, the maximal cliques containing v occur consecutively. We will call such an ordering a *linear order of the maximal cliques*. It is possible to find such a linear order in linear time, as can be seen, for example, in [43] or [102].

This property of the maximal cliques can also be expressed through a linear ordering of the vertices. It has been shown by numerous authors (for example by Olariu [118]) that a graph $G = (V, E)$ is an interval graph if and only if it has an *interval order*, i.e., an ordering σ of its vertices such that for $u \prec_\sigma v \prec_\sigma w$, the existence of $uw \in E$ implies the existence of the edge $uv \in E$.

For unit interval graphs, Looges and Olariu [110] proved a similar result:

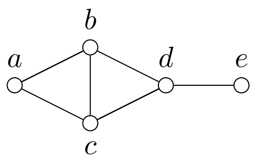


Figure 4.7: The graph depicted here is a unit interval graph for which the orderings of MNS, LDFS and MCS differ. The order (b, c, d, a, e) is an MCS order that is not an LDFS order, while (b, c, d, e, a) is an LDFS order that is not an MCS order. Furthermore, the order (d, b, c, e, a) is an MNS order that is neither an MCS nor an LDFS order.

Lemma 4.3.5 (Looges and Olariu [110]). *A graph $G = (V, E)$ is a unit interval graph if and only if it has an ordering $\sigma = (v_1, \ldots, v_n)$ of its vertices such that for $u \prec_\sigma v \prec_\sigma w$, the existence of $uw \in E$ implies the existence of the edges $uv \in E$ and $vw \in E$. Consequently, for two indices $i < j$ such that $v_iv_j \in E$ the set of vertices $\{v_i, v_{i+1}, \ldots v_{j-1}, v_j\}$ forms a clique.*

We call such a linear vertex order a *unit interval order.* Both an interval order as well as a unit interval order can be computed in linear time [36, 43, 102, 110]. Note that even in unit interval graphs MNS, MCS and LDFS can output different search orders. In Figure 4.7 we give an example of a unit interval graph for which there is an MCS order that is not an LDFS order and vice versa. Furthermore, this example gives an MNS order that is neither an MCS nor an LDFS order. However, the following theorem shows that the end-vertices of these searches are the same.

Theorem 4.3.6. *Let $G = (V, E)$ be a unit interval graph and let t be a vertex of G. Then the following statements are equivalent:*

(i) Vertex t is simplicial and $G - N[t]$ is connected.

(ii) Vertex t is the last vertex of some unit interval order.

(iii) Vertex t is an end-vertex of MNS (MCS, LDFS).

Proof. To prove that (i) implies (ii), let $\sigma = (v_1, \ldots, v_n)$ be a unit interval order of G and let $t = v_l$. As t is simplicial, the closed neighbourhood of t forms a maximal clique of G. Thus, all vertices of $N[t]$ must appear consecutively in σ, i.e., $N[t] = \{v_i, v_{i+1}, \ldots v_{j-1}, v_j\}$ for some $i \leq l \leq j$, due to Lemma 4.3.5. Suppose that $i \neq 1$ and $j \neq n$. Then $G - N[t]$ is disconnected, as there can be no edge between a vertex left of v_i and a vertex to the right of v_j, again due to Lemma 4.3.5. This is a contradiction to the choice of t. Therefore, we can suppose without loss of generality that $j = n$. Because $\{v_i, v_{i+1}, \ldots v_{j-1}, v_j\}$ is a clique, it is easy to see that $\sigma' = (v_1, \ldots, v_{l-1}, v_{l+1}, \ldots v_n, v_l = t)$ is also a unit interval order.

To prove that (ii) implies (iii), let $\sigma = (v_1, \ldots, v_n)$ be a unit interval order of G and let $t = v_n$. Let $\sigma' = (w_1, \ldots, w_n)$ be an MNS (MCS, LDFS) order such that the first index k at which $v_k \neq w_k$ is rightmost among all such search orders. This implies that at moment $k-1$ in the search the vertex w_k must have a larger label than v_k. In other words, there is a vertex v_i with $i < k$ such that v_i is adjacent to w_k but not v_k. Since $v_k \prec_\sigma w_k$ this is a contradiction to the fact that σ is a unit interval order.

To show that (iii) implies (i) assume that $G - N[t]$ is not connected. Let C_1 and C_2 be distinct connected components of $G - N[t]$. Suppose there is a vertex w in $N(t)$ that is adjacent to a vertex $c_1 \in C_1$ and a vertex $c_2 \in C_2$. Then t, w, c_1 and c_2 form an induced claw in G which is a contradiction to the fact that G is a unit interval graph [23]. Therefore, the neighbourhood of C_1 in $N(t)$, say N_1, and the neighbourhood of C_2 in $N(t)$, say N_2 are disjoint. Both N_1 and N_2 form separators of G and, thus, each of these must contain a minimal separator. Therefore, the minimal separators in $N(t)$ are not totally ordered by inclusion. As t is an MNS end-vertex, this is a contradiction to Lemma 4.1.1. □

Since Condition i) in Theorem 4.3.6 can be decided in linear time, we can state the following corollary.

Corollary 4.3.7. *The MCS and LDFS end-vertex problem can be decided in linear time on unit interval graphs.*

For DFS there is a simple characterisation of the end-vertices of (claw, net)-free graphs using hamiltonian paths. This also holds for unit interval graphs, as they form a subclass of (claw, net)-free graphs [23].

Theorem 4.3.8. *Let $G = (V, E)$ be a (claw, net)-free graph. Then $t \in V$ is the end-vertex of some DFS if and only if t is not a cut vertex.*

Proof. It is clear, that a cut vertex cannot be an end-vertex of a DFS, since it cannot be an end-vertex of the generic search [30]. It remains to show, that every other vertex t can be an end-vertex. Let $G' = G - t$. G' is still a (claw, net)-free graph. Furthermore, it contains a hamiltonian path P since it is connected [23]. Thus, we can start the DFS in G with P and then take t as the last vertex. □

Corollary 4.3.9. *The end-vertex problem of DFS can be decided in linear time on (claw, net)-free graphs, and, in particular, on unit interval graphs.*

4.3.3 Interval graphs

In the same vein as for unit interval graphs, we can use a result by Kratsch et al. [105] to characterise DFS end-vertices on interval graphs using hamiltonian paths.

Lemma 4.3.10 (Kratsch et al. [105])**.** *Let $G = (V, E)$ be a connected graph, and let t be a vertex of G. Then t is an end-vertex of DFS if and only if there is a set $X \subseteq V$ such that $N[t] \subseteq X$ and $G[X]$ has a hamiltonian path with endpoint t.*

On interval graphs this characterisation can be simplified to the following.

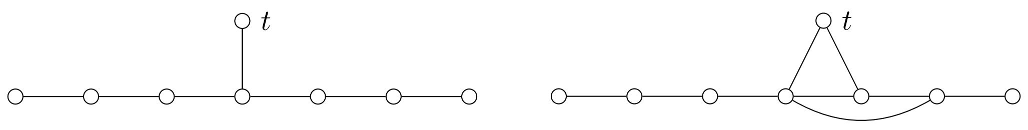


Figure 4.8: Both graphs are interval graphs, but not unit interval graphs, since both contain an induced claw. On the left hand side the MCS end-vertex t is such that $G - N[t]$ is disconnected. On the right hand side vertex t is an MNS end-vertex but not an MCS end-vertex.

Theorem 4.3.11. *Let G be an interval graph. Then $t \in V$ is the end-vertex of a DFS if and only if $G[N(t)]$ contains a hamiltonian path.*

Proof. Suppose $t \in V$ such that $G[N(t)]$ contains a hamiltonian path. Then $G[N[t]]$ must contain a hamiltonian path ending in t and by Lemma 4.3.10 t is a DFS end-vertex of G.

Now assume that t is a DFS end-vertex. By Lemma 4.3.10 there exists a set $X \subseteq V$ such that $N[t] \subseteq X$ and $G[X]$ has a hamiltonian path with endpoint t. Let X be the set of smallest cardinality that fulfils these properties. We claim that $X = G[N[t]]$.

Let $P = (v_1, \dots, v_l)$ be the hamiltonian path of $G[X]$ with endpoint t, i.e., $v_l = t$. Suppose there is a vertex $v_i \in X \setminus N[t]$ and let v_i be the leftmost such vertex in P. If $i = 1$, then $P' = (v_2, \dots v_l)$ is a hamiltonian path in $X \setminus v_i$ with endpoint t, in contradiction to the minimality of X.

Therefore, let $j < i < k$ such that v_j is the rightmost vertex of $N[t]$ to the left of v_i in P and v_k is the leftmost vertex of $N[t]$ to the right of v_i, i.e., $v_{j+1}, \dots, v_{k-1} \in X \setminus N[t]$. If t is equal to v_k, then $v_j v_k \in E$ and $P' = (v_1, \dots, v_j, v_k)$ is a hamiltonian path of $X \setminus \{v_{j+1}, \dots, v_{k-1}\}$; this is a contradiction to the minimality of X. Otherwise, $(t, v_j, \dots, v_k, t)$ forms a cycle of length ≥ 4 in G. As $v_{j+1}, \dots, v_{k-1}$ are not adjacent to t, there must be an edge between some non-consecutive vertices in $(v_j, \dots, v_k)$. As above, this is a contradiction to the minimality of X. □

As the hamiltonian path problem can be solved on interval graphs in linear time, as was shown by Rao Arikati and Pandu Rangan [121], we can state the following corollary.

Corollary 4.3.12. *The end-vertex problem of DFS can be decided in linear time on interval graphs.*

The end-vertex problem for MCS on the class of interval graphs, however, appears to be more complicated than in the case of unit interval graphs, as implication *iii)* $\Rightarrow$ *i)* from Theorem 4.3.6 does not necessarily hold here. As a result, not only simplicial vertices that are end-vertices of an interval order are candidates for being an MCS end-vertex. Furthermore, in interval graphs MCS is able to "jump" between non-consecutive cliques, making an analysis much harder. Figure 4.8 contains an example where an end-vertex t is such that $G - N[t]$ is disconnected.

The lemma below gives a relaxed necessary condition which covers the cases similar to the one given in Figure 4.8.

Lemma 4.3.13. *Let $G = (V, E)$ be an interval graph and let $C_1, C_2, \dots, C_k$ be a linear order of the maximal cliques of G. Suppose $t \in V$ satisfies the following conditions:*

1. *t is simplicial.*
2. *If C_i is the unique clique containing t, then $i = 1$ or $i = k$ or*

$$C_{i-1} \cap C_i \subseteq C_i \cap C_{i+1}$$

and

$$|C_i \cap C_{i+1}| \leq |C_j \cap C_{j+1}| \text{ for all } j > i,$$

or the same holds for the reverse order $C_k, C_{k-1}, \dots, C_1$.

Then t is the end-vertex for some MCS on G.

Proof. As G is an interval graph, we can assume that there is a linear order $C_1, C_2, \dots, C_k$ of the maximal cliques of G, in the sense that for any vertex $v \in V$ all cliques containing v are consecutive. Also, it is easy to see that any vertex is simplicial if and only if it is contained in exactly one of these maximal cliques.

We assume that t fulfils both properties. We now construct an MCS search ordering which starts in a simplicial vertex of C_1 and ends in the vertex t. Without loss of generality, we assume that the second property holds for $C_1, C_2, \dots, C_k$, as otherwise we can just begin our search at C_k and use the same arguments.

We proceed by visiting the maximal cliques of G consecutively – choosing the vertices of each clique in an arbitrary order – until we have completely visited C_{i-1}. As $C_{i-1} \cap C_i \subseteq C_i \cap C_{i+1}$, we can visit a vertex of $C_{i+1} \setminus C_i$ next, ignoring the simplicial vertices of C_i. Due to the fact that $|C_i \cap C_{i+1}| \leq |C_j \cap C_{j+1}|$ for all $j > i$ we can then visit all vertices of C_{i+1} and the vertices of all other maximal cliques apart from C_i. Finally, we visit the remaining simplicial vertices of C_i, choosing t last. □

Note that the sufficient condition given by Lemma 4.3.13 may not be necessary in general. For example, Figure 4.9 shows an example of a graph where the second condition from Lemma 4.3.13 is not true for vertex t, however it is still an MCS end-vertex. Nonetheless, we believe that it is still possible to give a characterisation of MCS end-vertices for the family of interval graphs which may be checked efficiently.

Conjecture 4.3.14. *The MCS end-vertex problem can be decided in polynomial time if the given input is an interval graph.*

4.4 Conclusion

We have shown that the end-vertex problem is $\mathcal{NP}$-complete for MNS on weakly chordal graphs and for MCS on general graphs. Moreover, we have given linear time algorithms to compute end-vertices for LDFS and MCS on unit interval graphs as well as for DFS on interval graphs and for MCS on split graphs. Using the same techniques, we were able to improve the analyses of running times of various previous results from polynomial

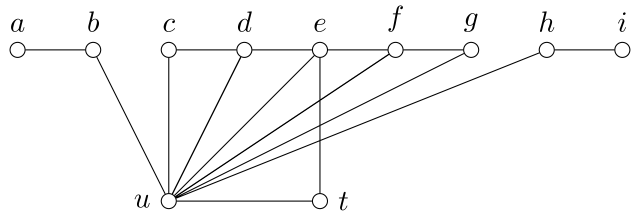


Figure 4.9: An example of an interval graph G and a vertex $t \in V(G)$, where t is an MCS end-vertex of the search $(a, b, u, h, i, c, d, e, f, g, t)$. However, it is easy to see that for the linear order of the maximal cliques of the form $C_1 = \{a, b\}$, $C_2 = \{b, u\}$, $C_3 = \{c, d, u\}$, $C_5 = \{d, e, u\}$, $C_6 = \{e, t, u\}$, $C_7 = \{e, f, u\}$, $C_8 = \{f, g, u\}$, $C_9 = \{u, h\}$ and $C_{10} = \{h, i\}$ the second condition of Lemma 4.3.13 is not fulfilled. In fact, this condition is false for any linear order of the maximal cliques.

time to linear time. A complete list of the achieved results can be found in Table 4.1. However, many open questions still remain.

For all the searches investigated here, apart from DFS, the complexity of the end-vertex problem on chordal graphs is still open. This is especially interesting, as nearly all of these problems are already hard on weakly chordal graphs. Even on interval graphs the complexity for some end-vertex problems is open.

Besides the complexity results for the end-vertex problem on graphs with bounded chordality, there are further results on bipartite graphs. Charbit et al. [30] showed, that it is $\mathcal{NP}$-complete for BFS. Gorzny and Huang [77] showed the same result for LBFS. Furthermore, they present a polynomial time algorithm for the LBFS end-vertex problem on AT-free bipartite graphs. It is an open question, whether these results can be extended to the end-vertex problems of MCS and MNS.

As mentioned in the introduction, the recognition of search trees of BFS and DFS is easy [81, 100, 112], although the corresponding end-vertex problems are hard. In the following chapter, we will show that LDFS-trees can be recognized in polynomial time, while the recognition of LBFS-trees is $\mathcal{NP}$-complete on the class of weakly chordal graphs. Furthermore, we will show that recognition of both MNS and MCS-trees is $\mathcal{NP}$-complete. However, it remains an interesting question, whether there exists a search and a graph class, such that recognition of the search trees on that class is hard, whereas the end-vertex problem is polynomial.

5 The Recognition of Graph Search Trees

A structure that is closely related to a graph search is the corresponding search tree. Such trees can be of particular interest, as for instance the tree obtained by a BFS contains the shortest paths from the root r to all other vertices in the graph. The trees generated by DFS can be used for fast planarity testing of graphs [91]. Moreover, if a cocomparability graph has a hamiltonian path, then such a path can be found by a combination of various graph searches [45]. First, one can use at most n LBFS runs, where n is the number of vertices, to find a cocomparability ordering [55]. Afterwards, the last visited vertex of an LDFS on this cocomparability ordering is the first vertex of a hamiltonian path. Finally, the search tree of a rightmost neighbour search on the LDFS ordering is a hamiltonian path.

So far, there is no satisfactory answer as to why graph searching works so well. Interesting examples are multi-sweep algorithms, such as finding dominating pairs in connected asteroidal triple-free graphs [40]. One can prove that these algorithms are correct. However, it is not clear why multiple runs of a simple algorithm could give such a strong insight into graph structure. Indeed, there seem to be some hidden structural properties of graph searches, which are waiting for discovery and algorithmic exploitation.

As a step in this direction, we study the problem of whether a given tree can be a search tree of a particular search. For BFS-like searches, one usually connects each vertex $v \in V$ to its neighbour which appeared first in the BFS order. Furthermore, for DFS-like searches, one connects each vertex $v \in V$ to the last neighbour visited before v. However, there is no such obvious definition of a tree for MCS or MNS. Thus, we define $\mathcal{F}$- and $\mathcal{L}$-trees: given an ordering, in an $\mathcal{F}$-tree each vertex v is connected to its neighbour which appeared first in the ordering before v, whereas in an $\mathcal{L}$-tree each vertex is connected to its neighbour which appeared last before v. A proper definition will be given in Section 5.0.1. This motivates the decision problem, where we are given a connected graph $G = (V, E)$ and a spanning tree T, and we need to decide whether there is a graph search of the given type such that T is the $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G. We call this problem the $\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem.

Already in 1972, Tarjan [135] gave a complete characterisation of DFS trees as so-called palm trees. However, no algorithm that determines if a given spanning tree of a graph G is a DFS tree of G was specified in that work. Using the concept of palm trees, Hopcroft and Tarjan developed a linear time algorithm for testing planarity of a graph [91]. Exploiting properties of DFS and BFS trees, the problem of checking whether a given spanning tree of G can be obtained by a DFS on G was formulated by Hagerup and Novak [81]. A few years later, Korach and Ostfeld gave a linear time algorithm for the proposed problem of recognition of DFS-trees [100]. A similar result for the recognition of BFS-trees was given by Manber in 1990 [112].

A problem that is closely related to the search tree recognition problem is the so-called end-vertex problem, i.e., the problem of determining whether a given vertex v in a graph G can be visited last by some graph search method. As a result of numerous new applications in algorithms, the end-vertex problem has received some attention in recent literature. In particular, the end-vertex of an LBFS on a chordal graph is always simplicial [126]. Furthermore, the end-vertex of an LBFS on a cocomparability graph is always a source/sink in some transitive orientation of its complement [79]. End-vertices are of particular interest for multi-sweep algorithms, as every consecutive search starts at the end vertex of the previous search. Here, LBFS provides a linear time algorithm for finding dominating pairs in connected asteroidal triple-free graphs, where a dominating pair is a pair of vertices such that every path connecting them is a dominating set in the graph [40]. The first vertex x is simply the end-vertex of an arbitrary LBFS and the second vertex y is the end-vertex of an LBFS starting in x. Moreover, one can use five LBFS executions followed by a modified LBFS to recognize interval graphs [43]. Crescenzi et al. [46] have shown that the diameter of large real world graphs can usually be found with only a few BFS executions.

Surprisingly, the problem of deciding whether a vertex can be an end-vertex of a graph search is hard. In 2010, Corneil, Köhler, and Lanlignel [44] have shown that it is $\mathcal{NP}$-hard to decide whether a vertex can be the end vertex of an LBFS. Later, Charbit, Habib, and Mamcarz generalized this result to BFS, DFS, and LDFS. Furthermore, they extended these results to several graph classes. Recently, Beisegel et al. [9] proved $\mathcal{NP}$-hardness results for MCS and MNS, and they also provided linear time algorithms for this problem on split graphs and unit interval graphs. Some of the results given here can be found in [8] as an extended abstract.

Although research initially began with the recognition of search trees, the results on the end-vertex problem are currently more extensive. In the light of the new results on the end-vertex problem, we fill in the gaps in the analysis of the complexity of the search tree recognition problem. In this chapter, we show that the tree recognition problem of LBFS, LDFS, MCS and MNS for $\mathcal{F}$-trees is $\mathcal{NP}$-complete on weakly chordal graphs. By showing that $\mathcal{F}$-trees for these searches are equivalent on chordal graphs we give a polynomial time algorithm for the recognition problem on this graph class. In the special case of split graphs these trees even coincide with $\mathcal{F}$-trees of BFS, yielding a linear time recognition algorithm. We give a summary of our results and the previous work on $\mathcal{F}$-tree recognition in Table 5.1.

Tree results	BFS	LBFS	DFS	LDFS	MCS	MNS
All Graphs	L [112]	**NPC**	?	**NPC**	**NPC**	**NPC**
Weakly Chordal	L	**NPC**	?	**NPC**	**NPC**	**NPC**
Chordal	L	**P**	?	**P**	**P**	**P**
Split	L	**L**	?	**L**	**L**	**L**

Table 5.1: Complexity of the $\mathcal{F}$-tree recognition problem. Our results are denoted by bold letters and L denotes linear time algorithms.

For $\mathcal{L}$-trees we present a polynomial time recognition algorithm for LDFS on general graphs and a linear time algorithm for the $\mathcal{L}$-tree problem on chordal graphs for LBFS, LDFS, MCS and MNS by proving that all these searches share the same set of $\mathcal{L}$-trees. Table 5.2 summarizes the known and the new results on $\mathcal{L}$-tree recognition.

Tree results	BFS	LBFS	DFS	LDFS	MCS	MNS
All Graphs	?	?	L [81, 100]	**P**	?	?
Weakly Chordal	?	?	L	**P**	?	?
Chordal	?	**L**	L	**L**	**L**	**L**

Table 5.2: Complexity of the $\mathcal{L}$-tree recognition problem. Our results are denoted by bold letters and L denotes linear time algorithms.

The results given in this chapter have been achieved in joint work with Carolin Denkert, Ekkehard Köhler, Matjaž Krnc, Nevena Pivač, Robert Scheffler and Martin Strehler. Some of these results can be found in a published extended abstract [8].

5.0.1 The Search Tree Recognition Problem

The definition of the term *search tree* varies between different paradigms. However, typically, it consists of the vertices of the graph and, given the search order $(v_1, \dots, v_n)$, for each vertex v_i exactly one edge to a $v_j \in N(v_i)$ with $j < i$. By specifying to which of the previously visited neighbours a new vertex is adjacent in the tree, we can define different types of graph search trees. For example, in a BFS a vertex is typically adjacent to the leftmost neighbour in the search order, while in DFS a vertex v is adjacent to the rightmost neighbour to the left of v. This motivates the following definition.

Definition 5.0.1. *Given a search discovery order $\sigma := (v_1, \dots, v_n)$ of a given search on a connected graph $G = (V, E)$, we define the* first-in tree *(or $\mathcal{F}$-tree) to be the tree consisting of the vertex set V and an edge from each vertex to its leftmost neighbour in σ.*

The last-in tree *(or $\mathcal{L}$-tree) is the tree consisting of the vertex set V and an edge from each vertex v_i to its rightmost neighbour v_j in σ with $j < i$.*

As explained above, if σ and T are the output of a classical BFS, then T is an $\mathcal{F}$-tree with respect to σ, while for a classical DFS the tree T is an $\mathcal{L}$-tree with respect to σ. Given this definition, we can state the following decision problem.

$\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem

Instance:	A connected graph $G = (V, E)$ and a spanning tree T.
Task:	Decide whether there is a graph search of the given type such that T is its $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G.

Note that we have defined the $\mathcal{F}$-tree ($\mathcal{L}$-tree) Recognition Problem without a given start vertex for the search. It is also possible to define this problem with a fixed start vertex and we call this the *rooted $\mathcal{F}$-tree ($\mathcal{L}$-tree) Recognition Problem*. It is easy to see that a polynomial-time algorithm for the rooted Tree Recognition Problem yields a polynomial-time algorithm for the general problem by simply repeating the procedure for all vertices. The other direction, however, is not necessarily true. For all results given here we show that either there is a polynomial algorithm for the rooted version, or that the problem is $\mathcal{NP}$-complete for the general case, making a distinction insignificant for the examined cases.

One can also consider the unlabelled case of the Tree Recognition Problem where we have to decide whether there is a search tree of G which is isomorphic to the tree of the input. We call this problem the *unlabelled $\mathcal{F}$-tree ($\mathcal{L}$-tree) Recognition Problem*. Obviously, this problem is $\mathcal{NP}$-hard for $\mathcal{L}$-trees of DFS, since it includes the hamiltonian path problem. The following theorem shows that for many graph classes this problem is at least as hard as the labelled version.

Theorem 5.0.2. *Let $\mathscr{C}$ be a graph class which is closed under insertion of leaves. Then the unlabelled $\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem on $\mathscr{C}$ is at least as hard as the labelled $\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem on $\mathscr{C}$.*

Proof. Let G and T be an input instance of the $\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem of graph class $\mathscr{C}$ and let $V(G) = \{v_1, \dots v_n\}$. We create a graph G' and a tree T' as follows. For every $v_i \in V(G)$ we add $i \cdot n$ vertices to G and to T. These vertices are adjacent to v_i both in G' and in T'. Therefore, for every $1 \leq i \leq n$ the vertex v_i is the only vertex of G' and T' with a degree $i \cdot n \leq d_{T'}(v_i) \leq d_{G'}(v_i) < (i+1) \cdot n$. Let L be the set of these inserted leaves of G'.

We will show that T is a search tree of G of a given type of search if and only if T' is isomorphic to a search tree of G'. This will then imply that if we can solve the unlabelled $\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem on G' with T' in polynomial time, then we can solve the labelled $\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem on G with T in polynomial time as well.

Assume T is an $\mathcal{F}$-tree ($\mathcal{L}$-tree) of the given type on G. Note that a leaf in a graph cannot influence the order of the search and must be adjacent to its neighbour in the search tree. Therefore, T' is isomorphic to an $\mathcal{F}$-tree ($\mathcal{L}$-tree) of the given type on G'.

On the other hand, assume that T' is isomorphic to a search tree of G' of the given search. If this search tree is rooted in a leaf of G', then the same tree rooted in the unique neighbour of that leaf is also a valid search tree that is isomorphic to G'. Due to the degree condition above we know that in every isomorphism of T' vertex v_i must be mapped to itself. Furthermore, we can assume that T' is not rooted in a leaf, i.e., it is rooted in one of the vertices of G. Hence, the tree resulting from deleting all leaves from T' is a search-tree of G which is equal to T. □

When comparing the different searches, one can see that graph search trees behave very similarly to the searches themselves, in the sense that, for example, an LBFS tree is also a BFS tree, but not vice versa. Some examples of graph search trees illustrating these relationships can be found in Figure 5.1.

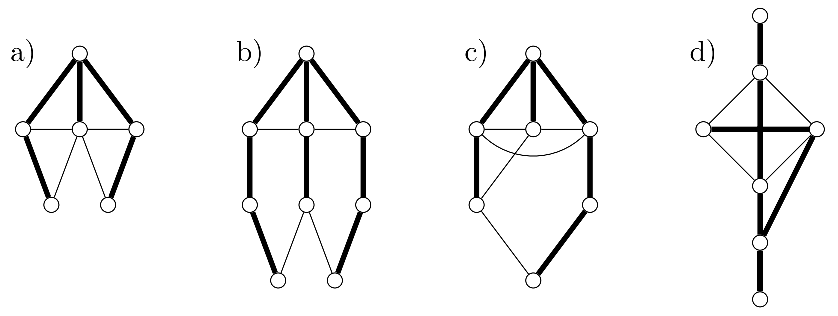


Figure 5.1: Four examples of graphs with their search trees denoted by the thick edges. The graph in a) depicts a search tree of BFS that is not an $\mathcal{F}$-tree for LBFS or MNS. The graph in b) depicts an $\mathcal{F}$-tree of MNS and BFS that is not an $\mathcal{F}$-tree for LBFS. The graph in c) shows a search tree that is an $\mathcal{F}$-tree of MNS, BFS and LBFS that is not an $\mathcal{F}$-tree of MCS. Finally, the graph in d) gives an example of a search tree that is an $\mathcal{L}$-tree for DFS, but not for LDFS.

5.1 A Polynomial Algorithm for $\mathcal{L}$-trees of LDFS

As Lexicographic Depth First Search is a special case of DFS, the most natural search tree to be considered here is the $\mathcal{L}$-tree. We give a polynomial-time algorithm (Algorithm 12) which, given a graph G and a spanning tree T, decides whether T is an $\mathcal{L}$-tree of LDFS on G. This is an interesting contrast to the fact that it is $\mathcal{NP}$-complete to decide whether a given vertex is an end-vertex of LDFS, as shown by Charbit et al. [30].

In essence, Algorithm 12 runs an LDFS and at every step checks whether there is still a possible choice of vertex which does not contradict the search tree.

To prove that Algorithm 12 works correctly, we first state a few lemmas about $\mathcal{L}$-trees of DFS.

Lemma 5.1.1 (Tarjan [135])**.** *Let $G = (V, E)$ be a graph and let T be a spanning tree of G. Then T is an $\mathcal{L}$-tree of G generated by DFS if and only if for each edge $uv \in E$ it holds that either $e \in E(T)$ or u is an ancestor of v in T or v is an ancestor of u in T.*

Lemma 5.1.2 (Korach and Ostfeld [100])**.** *Let $G = (V, E)$ be a graph with spanning tree T. Let G_i be a connected subgraph of G with a spanning tree T_i which is the restriction of T to G_i. If T is an $\mathcal{L}$-tree of DFS on G, then T_i is an $\mathcal{L}$-tree of DFS on G_i.*

We can give an analogous result for LDFS, which just considers induced subgraphs of G.

Lemma 5.1.3. *Let $G = (V, E)$ be a graph with spanning tree T. Let G_i be a connected induced subgraph of G with a spanning tree T_i which is the restriction of T to G_i. If T is an $\mathcal{L}$-tree of LDFS on G, then T_i is an $\mathcal{L}$-tree of LDFS on G_i. In particular, if T is rooted in $r \in V$ and $r \in V(T_i)$, then T_i is also rooted in r.*

Algorithm 12: Algorithm which decides whether T is an $\mathcal{L}$-tree of LDFS on G rooted in r.

```
Input: Graph $G = (V, E)$, spanning tree $T$ of $G$, and a vertex $r \in V$.
Output: $T$ is an $\mathcal{L}$-tree of LDFS on $G$ or not.
begin
    $S \leftarrow \{r\}$;
    foreach vertex $v \in V - r$ do label($v$) $\leftarrow \emptyset$;
    foreach vertex $v \in N(r)$ do
        prepend 0 to label($v$);
        pred($v$) $\leftarrow r$;
    while $S \neq V$ do
        choose a node $v \in V - S$ with lexicographic largest label, such that
          $\{\text{pred}(v), v\} \in E(T)$ ;
        if no such $v$ exists then return $T$ is not an $\mathcal{L}$-tree of LDFS on $G$;
        $S \leftarrow S \cup \{v\}$;
        for $w \in N(v) \setminus S$ do
            prepend $i$ to label($w$);
            pred($w$) $\leftarrow v$;
    return $T$ is an $\mathcal{L}$-tree of LDFS on $G$.
```

Proof. Let G_i be a connected induced subgraph of G and let T_i be the restriction of T to G_i. Suppose that T is an $\mathcal{L}$-tree of LDFS on G. We will show that in this case T_i is an $\mathcal{L}$-tree of LDFS on G_i.

Let σ be an LDFS search order of G that results in the search tree T and let $\sigma(1) := r$. We run an LDFS on G_i by always choosing the vertex with largest label which is leftmost in σ and call the new search order τ. Suppose that the resulting search tree R does not coincide with T_i. Let v be the leftmost vertex in τ that does not have the same parent in R as it does in T. Let u be the parent of v in R.

Because v was chosen to be leftmost in τ such that it has a different parent in R than in T, the unique path P from u to r in R is identical to that in T. Therefore, we can see that u must be an ancestor of v in T, due to Lemma 5.1.1. Let w be the unique child of u in T that is an ancestor of v; in particular $v \neq w$ and $w \prec_\sigma v$.

Suppose that $w \prec_\tau v$. This implies that w is adjacent to u in R, as v was chosen to be the first vertex of τ with different parents in T and R. As u and w are ancestors of v in T, the path from u to v in T must be completely contained in G_i (otherwise T_i would not be connected). Because w and v are both children of u, somewhere on this path there must be a pair of vertices that are adjacent in G_i none of which is an ancestor of the other in R. Due to Lemma 5.1.1, this is a contradiction to the fact that R is a DFS tree of G_i.

Therefore, we can assume that $v \prec_\tau w$. This implies that at the point where v was chosen, the label of v was strictly larger than that of w; this is a contradiction, as all vertices that have labelled v are on P, due to Lemma 5.1.1. Therefore, it is identical to the label v had at the point when w was chosen over v in σ. □

Theorem 5.1.4. *The (rooted) $\mathcal{L}$-tree recognition problem for LDFS can be solved in polynomial time.*

Proof. Algorithm 12 tests for a fixed $r \in V$ whether T can be an $\mathcal{L}$-tree for LDFS on G that is rooted in r. Therefore, assuming the Algorithm 12 works correctly and in polynomial time, it is enough to apply it to all vertices in G to decide whether T is, in fact, an $\mathcal{L}$-tree of LDFS. As we begin the search in r we from now on assume that T is rooted in a fixed vertex r.

First suppose that the algorithm returns "T is an $\mathcal{L}$-tree of LDFS on G". In this case, the algorithm has successfully executed an LDFS and it remains to show that the resulting search order has T as its $\mathcal{L}$-tree. This, however, is safeguarded by the fact that at every point at which we have added a vertex v to our search order, the predecessor of v, i.e., its parent in the resulting search tree, is also adjacent to v in T.

Now assume that the algorithm returns "T is not an $\mathcal{L}$-tree of LDFS on G". This implies that at some point of Algorithm 12 there is no vertex x of lexicographically largest label, such that the predecessor of x is adjacent to x in T. Let v be such a vertex of lexicographically largest label, whose predecessor is not its parent in T. Note that the tree R constructed thus far by Algorithm 12 is a subtree of T.

Assume that T is, in fact, an $\mathcal{L}$-tree of G generated by LDFS. Let u be the predecessor assigned to v by the algorithm. Thus, due to Lemma 5.1.1, u must be an ancestor of v in T. Let w be the unique child of u in T that is also an ancestor of v and let P be the unique path from v to r in T; in particular, $u, w \in V(P)$. As a result of Lemma 5.1.3, P is an $\mathcal{L}$-tree of LDFS on $G[V(P)]$ since T is an $\mathcal{L}$-tree of LDFS on G.

However, Algorithm 12 and Lemma 5.1.1 imply that P cannot be an $\mathcal{L}$-tree of LDFS on $G[V(P)]$. As we start in r and as P is a path, we must choose all vertices up to u in the order of the path. Due to Lemma 5.1.1, the vertices have the same labels as they did when Algorithm 12 halted. Therefore, v has a lexicographically larger label than w. As a result, P and, thus, T cannot be $\mathcal{L}$-trees of LDFS. □

5.2 $\mathcal{NP}$-completeness for $\mathcal{F}$-trees of LBFS

It was shown in [44] that the LBFS end-vertex problem is $\mathcal{NP}$-complete. In the following we show that the same holds for the tree-recognition problem.

Theorem 5.2.1. *The $\mathcal{F}$-tree-recognition problem of LBFS is $\mathcal{NP}$-complete on weakly chordal graphs.*

We prove Theorem 5.2.1 by giving a reduction from 3-SAT. Let $\mathcal{I}$ be an instance of 3-SAT. We construct the corresponding graph $G(\mathcal{I})$ and the spanning tree $T(\mathcal{I})$ as follows (for an example see Figure 5.2). Let $X = \{x_1, \ldots, x_k, \overline{x_1}, \ldots, \overline{x_k}\}$ be the set of vertices representing the literals of $\mathcal{I}$. The edge set $E(X)$ forms the complement of the matching in which x_i is matched to $\overline{x_i}$ for every $i \in \{1, \ldots, k\}$. For each clause C_i of $\mathcal{I}$

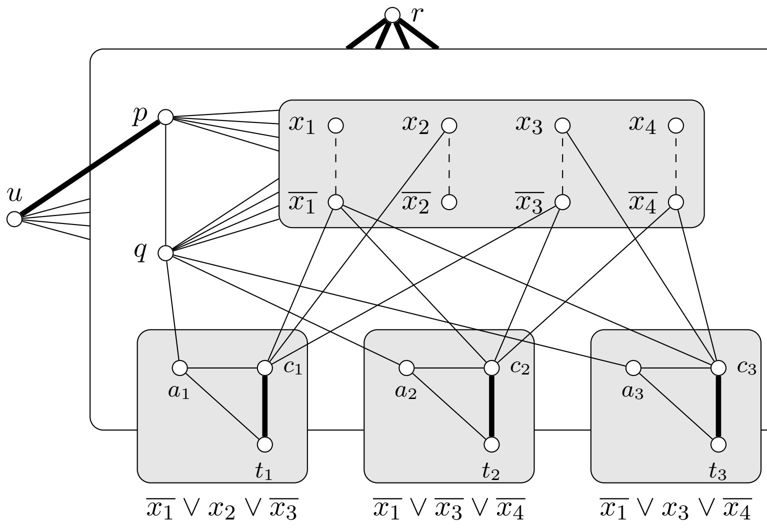

r
p
u
q
x_1
x_2
x_3
x_4
$\overline{x_1}$
$\overline{x_2}$
$\overline{x_3}$
$\overline{x_4}$
a_1
c_1
t_1
a_2
c_2
t_2
a_3
c_3
t_3
$\overline{x_1} \vee x_2 \vee \overline{x_3}$
$\overline{x_1} \vee \overline{x_3} \vee \overline{x_4}$
$\overline{x_1} \vee x_3 \vee \overline{x_4}$

Figure 5.2: The $\mathcal{NP}$-completeness construction for the tree-recognition problem of LBFS. The depicted graph is $G(\mathcal{I})$ for $\mathcal{I} = (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4})$. In the box containing the literal vertices, only non-edges are displayed by dashed lines. The connection of a vertex with a box implies, that the vertex is connected to all vertices in this box. Tree edges are depicted by thick edges.

we have a triangle consisting of vertices a_i, c_i and t_i. For every triangle representing a clause C_i, the vertex c_i is adjacent to each literal of the clause C_i.

In addition, we have vertices r, p, q and u. Vertex r is adjacent to every vertex apart from the t_i and u, while u is adjacent to all vertices apart from the t_i and r. Vertex p has additional edges to each vertex in X and to q, while q is also adjacent to all vertices in X and each of the a_i. Altogether, $G(\mathcal{I})$ consists of the vertex set $V(G(\mathcal{I})) := X \cup \{r, p, q, u\} \cup C_1 \cup \ldots \cup C_l$, where C_i represents the vertices of the clause-gadget of C_i and the edge set is defined as above.

The corresponding spanning tree $T(\mathcal{I})$ consists of the edges incident to r, an edge between u and p and the edges $c_i t_i$ for all $i \in \{1, \ldots l\}$; they are denoted as thick lines in Figure 5.2.

We proceed to prove Theorem 5.2.1 by showing that $T(\mathcal{I})$ is an $\mathcal{F}$-tree of LBFS of $G(\mathcal{I})$ if and only if $\mathcal{I}$ has a satisfying assignment $\mathcal{A}$.

Lemma 5.2.2. *If $\mathcal{I}$ admits a satisfying assignment $\mathcal{A}$, then $T(\mathcal{I})$ is a possible $\mathcal{F}$-tree of LBFS on $G(\mathcal{I})$.*

Proof. Let $\mathcal{A}$ be a satisfying assignment of $\mathcal{I}$. The following valid search order produces $T(\mathcal{I})$ as its search tree. We begin in r and then choose p. Next, we can choose vertices from X according to the assignment $\mathcal{A}$ in an arbitrary order, i.e., we choose x_i or $\overline{x_i}$ corresponding to whether the variable x_i is set to 1 or 0 in $\mathcal{A}$. We are then forced to visit the vertex q, as each remaining vertex of X is not adjacent to one of the visited vertices of X. After choosing the remaining vertices of X we proceed to the vertices of the clause gadgets. As a fulfilling assignment sets at least one literal to 1 in each clause, every c_i has a neighbour that appears earlier in the search order than q which is the leftmost neighbour of a_i in the search order. Hence, for each clause gadget C_i we must choose c_i before a_i. Therefore, we can choose all vertices c_i and then all vertices a_i. Finally, we can choose u and then all the t_i.

It is easy to see that all edges incident to r belong to the search tree of the constructed order, as well as pu. On the other hand, c_it_i must be in the search tree for every $i \in \{1, \dots, l\}$, as c_i was always chosen before a_i. Therefore, the search tree of the constructed order coincides with $T(\mathcal{I})$. □

We now show the other direction of the proof.

Lemma 5.2.3. *If $\mathcal{I}$ does not admit a satisfying assignment, then $T(\mathcal{I})$ cannot be an $\mathcal{F}$-tree of LBFS on $G(\mathcal{I})$.*

Proof. We show that for at least one clause gadget C_i the vertex a_i is visited before c_i, thus making $T(\mathcal{I})$ an infeasible search tree.

To prove this, we analyze the order in which the vertices of X are visited in any feasible LBFS search. It is easy to see that any LBFS must begin in r, as r is the only vertex whose incident edges are all tree edges. Next, we are forced to choose p, as otherwise pu cannot be a tree edge. If q is chosen next, then, as a result, a_i must be visited before c_i for every $i \in \{1, \dots, l\}$ and $T(\mathcal{I})$ cannot be the resulting search tree. Therefore, a subset of the vertices of X must be chosen before the vertex q.

If a vertex x_i is visited, then q receives a larger label than $\overline{x_i}$, as they otherwise share the same set of neighbours among the visited vertices up to that point (and analogously if $\overline{x_i}$ is visited before q). Thus, q must be chosen between any literal vertex and its negation. The largest subset of X that can be visited before q must, therefore, be an assignment of $\mathcal{I}$. As $\mathcal{I}$ is not satisfiable, any such assignment must leave at least one clause unfulfilled. If C_i is such a clause, then at the point at which q is chosen, c_i does not contain any neighbours among the visited literal vertices. As a result, a_i receives a larger label than c_i and is visited earlier.

Consequently, in any LBFS there must be a clause C_i such that a_i is visited before c_i and c_it_i cannot be in the search tree. This shows that $T(\mathcal{I})$ cannot be a $\mathcal{F}$-tree of an LBFS. □

Corollary 5.2.4. *Let $\mathcal{I}$ be an instance of 3-SAT. Then $\mathcal{I}$ has a satisfying assignment if and only if $T(\mathcal{I})$ is a possible $\mathcal{F}$-tree of LBFS on $G(\mathcal{I})$.*

To conclude the proof of Theorem 5.2.1, it remains to show that $G(\mathcal{I})$ is weakly chordal for every 3-SAT instance $\mathcal{I}$.

Lemma 5.2.5. *For each instance $\mathcal{I}$ of 3-SAT, the graph $G(\mathcal{I})$ is weakly chordal.*

Proof. We need to show that both $G(\mathcal{I})$ and $\overline{G(\mathcal{I})}$ do not contain a cycle of length ≥ 5. As all the t_i are simplicial, we can disregard them, due to Lemma 0.3.7. In the remaining graph, both r and u are adjacent to all vertices apart from each other and can, thus, be deleted, due to Lemma 0.3.7.

Let H' be the graph resulting from deleting r, u and all the t_i; it suffices to show that H' is weakly chordal. In addition, it is easy to see that every non-edge $x_i\overline{x_i}$ forms a two-pair in H', i.e., the longest induced path between these two vertices is of length 2. Using Lemma 0.3.6, we see that H' is weakly chordal if and only if $H' + x_i\overline{x_i}$ is weakly chordal. Furthermore, if we add the edges $x_i\overline{x_i}$ for all $i \in \{1, \ldots, k\}$ to H', the vertex p becomes simplicial. Therefore, it remains to show that the graph H which is constructed from H' by adding the edges $x_i\overline{x_i}$ for all $i \in \{1, \ldots, k\}$ and then deleting p is weakly chordal.

It is sufficient to show that $\overline{H}$ is weakly chordal. To this end, we apply Lemma 0.3.7. We can delete q from $\overline{H}$ as it is simplicial. In the remaining graph, all the a_i are adjacent to all but one vertex and can, thus, also be deleted. The remaining graph is a split graph, as the c_i form a clique and the literal vertices form an independent set, and, as a result it is weakly chordal. □

Since $T(\mathcal{I})$ can only be an $\mathcal{F}$-tree of LBFS on $G(\mathcal{I})$ if it is rooted in r, the above also proves the $\mathcal{NP}$-completeness of the rooted $\mathcal{F}$-tree recognition problem of LBFS. It is easy to see that the construction used can be adapted to trees $T(\mathcal{I})$ of arbitrary height ≥ 2 by simply adding a path of desired length to r. This yields the following corollary.

Corollary 5.2.6. *For every integer $h \geq 2$ the rooted $\mathcal{F}$-tree-recognition problem of LBFS is $\mathcal{NP}$-complete on weakly chordal graphs for spanning trees of height h.*

Note that for spanning trees of height 1 the $\mathcal{F}$-tree recognition problem is trivial for any search. As the class of weakly chordal graphs is closed under the insertion of leaves, Theorem 5.0.2 implies the following corollary.

Corollary 5.2.7. *The unlabelled $\mathcal{F}$-tree-recognition problem of LBFS is $\mathcal{NP}$-complete on weakly chordal graphs.*

5.3 $\mathcal{NP}$-Completeness for $\mathcal{F}$-trees of MNS, MCS and LDFS

As we have done for LBFS, we will show that the $\mathcal{F}$-tree problems for MNS, MCS and LDFS are $\mathcal{NP}$-complete.

Theorem 5.3.1. *The $\mathcal{F}$-tree-recognition problem of MNS, MCS and LDFS is $\mathcal{NP}$-complete on weakly chordal graphs.*

For the proof we construct a polynomial reduction from 3-SAT. Let $\mathcal{I}$ be an instance of 3-SAT. We construct the corresponding graph $G(\mathcal{I})$ as follows (see Figure 5.3 for an example): Let $X = \{x_1, \ldots, x_k, \overline{x}_1, \ldots, \overline{x}_k\}$ be the set of vertices representing the literals of $\mathcal{I}$. The edge-set $E(X)$ forms the complement of the matching in which x_i is matched to $\overline{x}_i$ for every $i \in \{1, \ldots, k\}$. Let $C = \{c_1, \ldots, c_l\}$ be the set of vertices representing the clauses of $\mathcal{I}$. The set C is independent in $G(\mathcal{I})$ and every c_i is adjacent to each vertex of X, except those representing the literals of the clause associated with c_i for

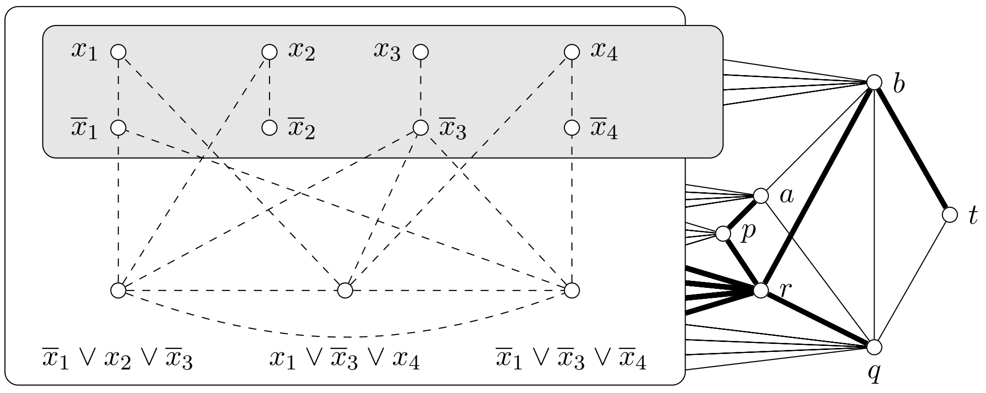

x_1
x_2
x_3
x_4
$\overline{x}_1$
$\overline{x}_2$
$\overline{x}_3$
$\overline{x}_4$
$\overline{x}_1 \vee x_2 \vee \overline{x}_3$
$x_1 \vee \overline{x}_3 \vee x_4$
$\overline{x}_1 \vee \overline{x}_3 \vee \overline{x}_4$
a
b
p
r
q
t

Figure 5.3: The $\mathcal{NP}$-completeness construction for the tree-recognition problem of MNS. The depicted graph is $G(\mathcal{I})$ for $\mathcal{I} = (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_3 \vee x_4) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3)$. In both boxes only non-edges are displayed by dashed lines. The connection of a vertex with a box means that the vertex is connected to all vertices in this box. Tree edges are depicted by thick edges.

every $i \in \{1, \dots, l\}$. Additionally, we add the vertices r, p, q, a, b and t. The vertices r, p, q and a are adjacent to all literal vertices and all clause vertices and b is adjacent to all literal vertices. Finally, we add the edges ab, ap, aq, bq, br, bt, pr, qr and qt. The spanning tree $T(\mathcal{I})$ of $G(\mathcal{I})$ consists of all edges incident to r and the edges pa and bt.

We first state two lemmas which specify properties that must hold for any search order of MNS (MCS) that produces $T(\mathcal{I})$ as an $\mathcal{F}$-tree of $G(\mathcal{I})$.

Lemma 5.3.2. *If MNS (MCS, LDFS) generates the $\mathcal{F}$-tree $T(\mathcal{I})$ on $G(\mathcal{I})$, it chooses b before every clause vertex c_i.*

Proof. If we take the vertex q before b, we will insert the edge qt to the search tree, which is not an element of $T(\mathcal{I})$. Thus, this is not allowed in a search that generates the $\mathcal{F}$-tree $T(\mathcal{I})$. The neighbourhood of b is properly contained in the neighbourhood of q. Furthermore, q is adjacent to each clause vertex, while b is adjacent to none of them. Hence, if vertex c_i is taken before b, then the label of q will always be greater than the label of b in both MNS and MCS and both searches will take q before b. □

Lemma 5.3.3. *Let σ be an MNS (MCS, LDFS) ordering of $G(\mathcal{I})$ that generates the $\mathcal{F}$-tree $T(\mathcal{I})$. Then $\sigma(1) = r$, $\sigma(2) = p$ and $\sigma(i)$ for $3 \leq i \leq k + 2$ forms an arbitrary assignment of the variables (not necessarily satisfying).*

Proof. Any MNS resulting in the search tree $T(\mathcal{I})$ must start in r, since every other vertex is incident to an edge in $G(\mathcal{I})$ which is not an element of $T(\mathcal{I})$. Since a is adjacent to every neighbour of r in $G(\mathcal{I})$ but only to p in $T(\mathcal{I})$, the search has to choose p as the next vertex. Now the literal vertices and the clause vertices have the unique maximal label, since they were labelled both by r and p and every other vertex was labelled by at most one of these two vertices. Because of Lemma 5.3.2, we cannot take a clause vertex. Thus, we have to take a literal vertex. With the same argumentation it follows that we have to take a whole assignment, since the literal vertices of variables whose two literal vertices have not yet been chosen always have the unique maximal label. □

Now we present both directions of the proof of Theorem 5.3.1 in two separate lemmas.

Lemma 5.3.4. *If $\mathcal{I}$ has a satisfying assignment $\mathcal{A}$, then $T(\mathcal{I})$ is an $\mathcal{F}$-tree of MCS on $G(\mathcal{I})$ and therefore, also an $\mathcal{F}$-tree of MNS.*

Proof. In the following, we give a search order which results in the desired search tree $T(\mathcal{I})$. We start with r and then we take p. By doing this, we insert every edge of $T(\mathcal{I})$ apart from bt to the search tree. Next, we take the literal vertices which correspond to the assignment $\mathcal{A}$ in an arbitrary order. As a result, the labels of all literal vertices and of the vertices a, b and q are equal to $k+1$. Since $\mathcal{A}$ is satisfying, each clause vertex was not labelled by at least one of the chosen literal vertices. Hence, it has a label $\leq k+1$ and we can take b as the next vertex and insert the last missing edge of $T(\mathcal{I})$. The remaining vertices can be chosen in any possible order, as they do not influence the search tree. □

Lemma 5.3.5. *If $\mathcal{I}$ has a satisfying assignment $\mathcal{A}$, then $T(\mathcal{I})$ is an $\mathcal{F}$-tree of LDFS on $G(\mathcal{I})$.*

Proof. As in proof of Lemma 5.3.4, we present a search order of LDFS which results in the search tree $T(\mathcal{I})$. We start with r and then we take p. By doing this, we insert every edge of $T(\mathcal{I})$ apart from bt to the search tree. Next, we take the literal vertices which correspond to the assignment $\mathcal{A}$ in an arbitrary order. Now the vertex a is the only vertex that was labelled by the last $k+1$ chosen vertices. Thus, we have to take a as the next vertex. Afterwards b and q are the only vertices that were labelled by the last $k+1$ chosen vertices. Furthermore, the labels of b and q are identical at this point. Thus, we can take b as the next vertex inserting the last missing edge of $T(\mathcal{I})$. The order of the remaining vertices has no influence on the search tree. □

Lemma 5.3.6. *If $\mathcal{I}$ does not have a satisfying assignment, then $T(\mathcal{I})$ is not an MNS $\mathcal{F}$-tree of $G(\mathcal{I})$ and therefore, also not an MCS or LDFS $\mathcal{F}$-tree.*

Proof. Assume that $T(\mathcal{I})$ is an MNS $\mathcal{F}$-tree of $G(\mathcal{I})$. By Lemma 5.3.3 we have to start with r, then p and next, the literal vertices that correspond to an arbitrary assignment. Since this assignment cannot be satisfying, there is at least one clause vertex which was labelled by every vertex chosen up till now. In the label of every non-clause vertex at least one chosen vertex is missing. Thus, we have to visit a clause vertex next. This contradicts Lemma 5.3.2. □

To conclude the proof of Theorem 5.3.1, it remains to show that $G(\mathcal{I})$ is weakly chordal for every 3-SAT instance $\mathcal{I}$.

Lemma 5.3.7. *For every instance $\mathcal{I}$ of 3-SAT the graph $G(\mathcal{I})$ is weakly chordal.*

Proof. To begin with, we will use Lemma 0.3.7 to delete some vertices which cannot be part of a cycle of length ≥ 5 in $G(\mathcal{I})$ or its complement. We can delete t, since it is simplicial. Now the vertices r and a are adjacent to every other vertex and therefore, we can delete these as well. In the resulting graph we can use the same argumentation to delete q and p. The remaining graph only contains the literal vertices, the clause vertices and b. Since x_i and $\overline{x}_i$ form a two-pair for every $1 \leq i \leq k$, we can add the edges $x_i\overline{x}_i$, due to Lemma 0.3.6. The resulting graph is a split graph, where $X \cup \{b\}$ forms the clique and C forms the independent set. Thus, it is weakly chordal. $\square$

Theorem 5.3.1 follows immediately from Lemma 5.3.4, Lemma 5.3.5, Lemma 5.3.6 and Lemma 5.3.7. Using the same arguments as in the case of the $\mathcal{F}$-tree recognition problem of LBFS, this also yields the $\mathcal{NP}$-completeness of the rooted problem.

Corollary 5.3.8. *For every integer $h \geq 2$ the rooted $\mathcal{F}$-tree-recognition problem of MNS, MCS and LDFS is $\mathcal{NP}$-complete on weakly chordal graphs for spanning trees of height h.*

As the class of weakly chordal graphs is closed under the insertion leaves, we can also give the following corollary.

Corollary 5.3.9. *The unlabelled $\mathcal{F}$-tree-recognition problem of MNS, MCS and LDFS is $\mathcal{NP}$-complete on weakly chordal graphs.*

5.4 Search Trees on Chordal Graphs

In the previous sections, we have seen that the recognition of $\mathcal{F}$-trees is hard for almost all studied searches apart from BFS, even if the input is restricted to weakly chordal graphs. This raises the question whether the same is true if we restrict the input of the problem to chordal graphs. In fact, we will show that for chordal graphs it is possible to state a polynomial time algorithm for the $\mathcal{F}$-tree recognition problem for most searches. This yields a nice boundary of complexity between the classes of weakly chordal and chordal graphs.

Furthermore, when restricting to chordal graphs, it is also possible to give some positive algorithmic results regarding the recognition of $\mathcal{L}$-trees for searches other than (L)DFS and BFS. We will see that in chordal graphs the search trees turn out to be DFS-trees with additional properties. All of these results are based on the fact that in chordal graphs all MNS-type searches compute elimination orders, as was shown by Corneil and Krueger [37]. As this fact will be the defining feature of the search trees computed by these searches on chordal graphs, we will gather some information on perfect elimination orders in the following.

In [129], Shier defines a graph search (MEC) which is able to compute all perfect elimination orderings of a chordal graph. The MEC search forms a generalisation of MNS that chooses an element with maximal label in some connected component of the subgraph induced by the remaining vertices. This not only shows that on a chordal graph a perfect elimination ordering behaves like a graph search, in the sense that if $(v_1, \ldots, v_n)$ is a perfect elimination order of G, then for any $i \in \{1, \ldots, n\}$ the graph

$G[v_1, \dots, v_i]$ is connected, the analysis of this algorithm also yields the following useful lemma which is taken from the proof of Theorem 2 in [129].

Lemma 5.4.1 (Shier [129])**.** *Let $G = (V, E)$ be a chordal graph and let $\sigma = (v_1, \dots, v_n)$ be a PEO of G. Let C be the connected component of $G - \{v_1, \dots, v_{i-1}\}$ which contains v_i. Then for every $w \in C$ it holds that $N(w) \cap \{v_1, \dots, v_{i-1}\} \subseteq N(v_i)$.*

The next lemma shows that in any perfect elimination order of a chordal graph the vertices of an induced path occur in exactly the order of the path.

Lemma 5.4.2. *Let $G = (V, E)$ be a chordal graph and let $\sigma = (v_1, \dots, v_n)$ be a perfect elimination order of G. Let $v_{i_1}, \dots, v_{i_k}$ be an induced path of G with $i_1 = \min\{i_1, \dots, i_k\}$. Then it holds that $i_1 < i_2 < \dots < i_k$.*

Proof. By definition we see that $i_1 < i_2$. Assume i_j is the first index for which $i_j > i_{j+1}$. Then v_{i_j} is to the right of $v_{i_{j-1}}$ and $v_{i_{j+1}}$ which are not adjacent. This is a contradiction, as it implies that v_{i_j} is not simplicial in the graph that is induced by v_{i_j} and the vertices to the left of it in σ. □

We will see that most searches which compute a perfect elimination ordering have a strong property concerning the connected components of unvisited vertices.

Definition 5.4.3. *A connected graph search $\mathcal{A}$ is said to have the* component-neighbour property *if the following holds: Let $\sigma^* = (v_1, \dots, v_k)$ be the prefix of a search order σ of $\mathcal{A}$ on the chordal graph $G = (V, E)$ and let x and y be two vertices of G which lie in the same component of $G - \{v_1, \dots, v_k\}$. If $(v_1, \dots, v_k, y)$ is the prefix of a search order of $\mathcal{A}$ on G and $(v_1, \dots, v_k, x)$ is not, then there is a vertex v_i with $1 \leq i \leq k$ such that $v_i y \in E$ but $v_i x \notin E$.*

Lemma 5.4.1 implies the following observation.

Observation 5.4.4. *The search defined by perfect elimination orders, as well as LBFS, LDFS, MCS and MNS fulfil the component-neighbour property on chordal graphs.*

The results in the next sections hold not only for PEOs and MNS-type searches, but for all searches which compute a PEO and which fulfil the component-neighbour property. This motivates the following definition: we call a graph searching scheme $\mathcal{A}$ whose output is a perfect elimination order for any chordal graph and which fulfils the component-neighbour property an *edge-forced PEO-finder (EFPF)*.

5.4.1 $\mathcal{F}$-Trees on Chordal Graphs

As the a perfect elimination order is a connected search order on a chordal graph, we can define an $\mathcal{F}$-tree for these orders in the same way we have done for other graph searches. We will proceed to show that these trees are equivalent to the $\mathcal{F}$-trees of LBFS, LDFS, MCS and MNS on chordal graphs, as well as those of any search which is an EFPF. This is surprising, due to the fact that even on the class of split graphs (a subclass of

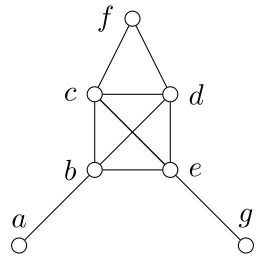


Figure 5.4: This is an example of a split and thus chordal graph in which all examined searches differ in the same way as they do on general graphs. Note that (b, e, c, d, g, f, a) is an MNS order which is neither a BFS, MCS nor LDFS order. On the other hand, one can see that (b, c, d, e, f, g, a) is an MCS order that is neither a BFS nor an LDFS order, while (b, c, d, e, g, f, a) is an LDFS order which is neither an MCS nor a BFS order, and (b, c, d, e, a, f, g) is an LBFS order that is neither an MCS nor an LDFS order. Furthermore, it holds that (b, c, a, d, e, f, g) is a BFS order that is not an MNS order, while (b, e, c, d, f, g, a) is both an MCS and an LDFS order that is not a BFS order.

chordal graphs) the search orders of all of these search schemes differ, as can be seen in Figure 5.4.

The following characterisation of these trees is an important ingredient for the recognition algorithm.

Lemma 5.4.5. *Let $G = (V, E)$ be a connected chordal graph and let T be a spanning tree of G with root r. Let $\sigma = (r = v_1, \ldots, v_n)$ be the a perfect elimination order of G. Then T is the $\mathcal{F}$-tree of σ if and only if every edge incident to r in G is part of T and σ is the linear extension of a partial order π that is defined as the reflexive and transitive closure π of the binary relation π' constructed as follows:*

1. *Let $w \neq r$ be a vertex of G and v its parent in T. Then $v \prec_{\pi'} w$.*
2. *If xy is an edge in $G - T$ and v is the parent of x, then $v \prec_{\pi'} y$. Furthermore, if v and y are not adjacent, then $x \prec_{\pi'} y$.*

Proof. Assume σ is the linear extension of π and all edges incident to r are part of T. Assume for contradiction that $T' \neq T$ is the $\mathcal{F}$-tree of σ. Then there must be an edge xy in $G - T$ which is part of T' and this is edge is not incident to r. Therefore, x has a parent v and y has a parent w. By definition of π it follows that both v and w are left to x and y in σ. Thus, the first neighbour of x in σ cannot be y and the first neighbour of y in σ cannot be x. This is a contradiction to xy being part of the $\mathcal{F}$-tree of σ.

Now assume that T is the $\mathcal{F}$-tree of σ. It is clear that every edge incident to r must be in T, since $\sigma(1) = r$. Furthermore, it is also clear that the parent v of x must be to the left of x in σ per definition. Let xy be an edge in $G - T$ and let v be the parent of x in T. If v is to the right of y in σ, then v cannot be the parent of x. Furthermore, if $vy \notin E$ it is not possible that v and y are to the left of x since otherwise σ would not be a PEO. Therefore, x must be to the left of y. □

Now we are in a position to fully characterize $\mathcal{F}$-trees for any search scheme that is an EFPF.

Theorem 5.4.6. *Let $G = (V, E)$ be a connected chordal graph, let T be a spanning tree of G and let r be the root of T. If the search scheme $\mathcal{A}$ is an EFPF, then T is an $\mathcal{F}$-tree of $\mathcal{A}$ of G rooted in r if and only if:*

1. *Every edge incident to r in G is part of T.*
2. *There is a partial order π of the vertices of G which fulfils the conditions of Lemma 5.4.5.*
3. *The partial order π can be extended to a partial order π^* such that the following holds: If x is the immediate predecessor of y in π^* and there is a vertex $z \in V$ with $yz \in E$ but $xz \notin E$ then $y \prec_{\pi^*} z$.*

Proof. Assume that there is a partial order π^* of the vertices and a spanning tree T of G which fulfil Conditions 1, 2 and 3 of the theorem. We will show that there is a search order σ of $\mathcal{A}$ on G which is a linear extension of π^*. Due to Lemma 5.4.5, the $\mathcal{F}$-tree of σ must then be T. Due to construction, r is a minimal element of π^*, and we can start our $\mathcal{A}$-search in r. In the following, we modify $\mathcal{A}$ such that under all vertices it is allowed to take next it takes a minimal element of π^*. If there always is such an element, it is clear that the constructed search order is a linear extension of π^*. Assume for contradiction that there is a step in $\mathcal{A}$ where there is no such vertex. Consider the first step in which this occurs and let y be a vertex that can be chosen by $\mathcal{A}$ in this step which is as small as possible in π^*. Since y is not minimal in π^*, there is a vertex x which is an immediate predecessor of y in π^* and which cannot be taken by $\mathcal{A}$. Note that in the construction of π and π^* a relation between two vertices is only added to the partial order if these two vertices are adjacent, or if there is at least one vertex between them in the order already. Furthermore, by taking the reflexive and transitive closure we only add a relation between two vertices that are not immediate predecessors of each other. This implies that any immediate predecessor in π^* is always adjacent to its successor and thus x must be adjacent to y. Therefore, by the component-neighbour condition there must be a vertex z which was already taken by $\mathcal{A}$ and which is adjacent to y but not to x. However, this is a contradiction, as in this case Condition 3 would imply that the relation $y \prec_{\pi*} z$ is in the partial order, and z would not have been minimal in π^* when it was taken by $\mathcal{A}$. Therefore, $\mathcal{A}$ can always take a minimal element of π^*, creating T as its $\mathcal{F}$-tree.

For the other direction of the proof, we assume that T is the $\mathcal{F}$-tree of the search order σ of $\mathcal{A}$ on G rooted in r. We need to check the three conditions of the theorem. Due to Lemma 5.4.5, T fulfils Condition 1 and σ is a linear extension of the partial order π of Condition 2. We will show that σ must also be a linear extension of π^*. Assume there are vertices x and y such that x is an immediate predecessor of y in π and there is a vertex z to the left of y in σ such that $yz \in E$ but $xz \notin E$. As explained in the previous case, by the construction of π, the vertex y must be a neighbour of its immediate predecessor x. Furthermore, since σ is a linear extension of π we know that x is also to the left of y

in σ. This is a contradiction to the fact that σ is a PEO of G, since y is not simplicial in the graph induced by the vertices to the left of y in σ. Therefore, y must be to the left of z in σ. Since there is an edge between the two vertices y and z, after the insertion of $y \prec_{\pi^*} z$ it still holds that every vertex is adjacent to its immediate predecessors. Hence, this argument can be repeated recursively. It follows that σ must be the linear extension of a partial order π^* that fulfils Condition 3. □

Using this characterisation we can decide the $\mathcal{F}$-tree problem for any EFPF in polynomial time.

Theorem 5.4.7. *On chordal graphs the rooted $\mathcal{F}$-tree recognition problem of every EFPF, in particular of LBFS, LDFS, MCS and MNS, can be solved in time $\mathcal{O}(nm)$.*

Proof. We have to check the conditions of Theorem 5.4.6. It is obvious that Condition 1 can be checked in linear time. We find the partial order π of Condition 2 by creating a directed graph G^π. At first for every tree edge vw, where v is the parent of w we insert $v \to w$ to G^π. For every non-tree edge we can check Condition 2 of Lemma 5.4.5 in time $\mathcal{O}(n)$ and insert the corresponding edges to G^π. Thus, we can construct G^π in $\mathcal{O}(nm)$. Note, that G^π has only $\mathcal{O}(m)$ edges. Now we can check whether G^π is acyclic in linear time. If it is not, there is no partial order π that fulfils the conditions of Lemma 5.4.5 and T is not an $\mathcal{F}$-tree of $\mathcal{A}$. Otherwise we check whether Condition 3 of Theorem 5.4.6 holds. For this we have to check for every edge $x \to y$ in G^π whether there is a vertex z that is adjacent to y but not to x in G. If so we insert the edge $y \to z$ to G^π. This costs us $\mathcal{O}(n)$ many steps. This we repeat as long as there is still such an edge. At the end we still know that G^π has only $\mathcal{O}(m)$ edges and we can again check whether it is acyclic. If so T is an $\mathcal{F}$-tree of $\mathcal{A}$ of G, otherwise it is not. By the explanation above the running time of the algorithm is in $\mathcal{O}(nm)$. □

Note that the described algorithm does not use the search $\mathcal{A}$. Thus, it can be possible to solve the $\mathcal{F}$-tree recognition problem of $\mathcal{A}$ faster than we can compute an arbitrary search order of $\mathcal{A}$. Furthermore, it holds that all EFPFs compute the same $\mathcal{F}$-trees.

Corollary 5.4.8. *For all EFPFs the set of rooted $\mathcal{F}$-trees of a chordal graph is the same.*

One can also consider connected searches that compute PEOs on chordal graphs but are not EFPFs. An example would be a version of MNS with a special tie breaking rule, as can be seen in Figure 5.5a). However, as the search defined by PEOs is an EFPF, every search algorithm that computes PEOs on chordal graphs, always has an $\mathcal{F}$-tree that can also be computed by any EFPF.

We conclude this section by showing that the unrooted $\mathcal{F}$-tree recognition problem of EFPFs on chordal graphs can be solved with the same time complexity as the rooted problem. In contrast to MNS-type searches, BFS does not necessarily compute a PEO on a chordal graph and is thus not an EFPF. In fact, Figure 5.1a) shows an example of a chordal graph with a spanning tree that is an $\mathcal{F}$-tree for BFS and that is not an $\mathcal{F}$-tree for LBFS or MNS. However, if it is known that a given spanning tree of a chordal graph is an $\mathcal{F}$-tree for LBFS, one can show that this tree is an $\mathcal{F}$-tree for LBFS with a fixed root r if and only if this is also the case for BFS.

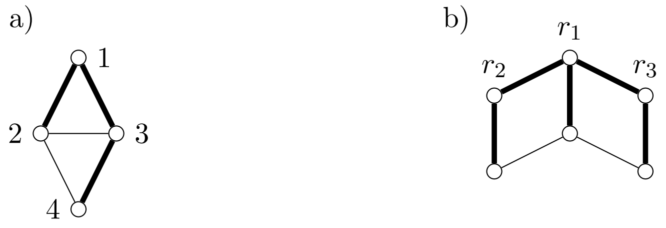


Figure 5.5: The graph in a) is an example of a chordal graph with a valid $\mathcal{F}$-tree for any EFPF. However, using MNS, with the added tie-break that vertices with lower numbering are prioritized, it is impossible to achieve this search tree, as the vertex labelled with 2 is always chosen before 3. The graph in b) is not chordal and the denoted spanning tree is an $\mathcal{F}$-tree of BFS for the roots r_1, r_2 and r_3, while for LBFS the only possible $\mathcal{F}$-tree is rooted in r_1.

Proposition 5.4.9. *Let $G = (V, E)$ be a chordal graph. Under the condition that T is an $\mathcal{F}$-tree of LBFS of G, it is an $\mathcal{F}$-tree of LBFS rooted in r if and only if it is an $\mathcal{F}$-tree of BFS rooted in r.*

Proof. In the proof of Theorem 4) in [112] Manber showed that the potential roots of an $\mathcal{F}$-tree of BFS tree T induce a subtree of T in G. We call this subtree the *core subtree* T_C of T. For every vertex v in T_C the subtree T_v of T which is induced by the descendants of v of T rooted in v which are not part of T_C is called the *side subtree* of v. We claim that all non-tree edges of G lie within some side subtree. Assume there is an edge xy between the two side subtrees T_v and T_w. Manber [112] showed that v and w must be adjacent and x and y must have the same distance to the root in T_v and T_w, respectively. Assume that x and y are the adjacent vertices of T_v and T_w with the smallest distance to the root. Then the edges xy and vw together with the paths between v and x in T_v and w and y in T_w build an induced cycle of length at least four. This is a contradiction to G being chordal.

Since every non-tree edge is part of a side subtree, it holds for every $r \in T_C$ that T is an $\mathcal{F}$-tree of LBFS of G rooted in r. □

Note that this proposition does not hold for all graphs in general. In Figure 5.5b) we see a graph that is not chordal with a spanning tree whose possible roots for being a BFS $\mathcal{F}$-tree are different to those for LBFS.

Since the roots of an $\mathcal{F}$-tree of BFS can be found in linear time [112], we can solve the unrooted tree recognition problem of LBFS and, due to Corollary 5.4.8, also of every other EFPF, with the same time complexity as the rooted problem.

Corollary 5.4.10. *On chordal graphs the $\mathcal{F}$-tree recognition problem of every EFPF, in particular of LBFS, LDFS, MCS and MNS, can be solved in time $\mathcal{O}(nm)$. Furthermore, it is possible to compute all roots of such a tree in linear time.*

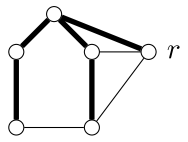


Figure 5.6: The tree denoted by the thick edges is an $\mathcal{L}$-tree of LBFS, MCS and MNS rooted in r. However, it is not an $\mathcal{L}$-tree of DFS.

5.4.2 $\mathcal{L}$-Trees on Chordal Graphs

In this section, we will show that the $\mathcal{L}$-tree recognition problems of LBFS, LDFS, MCS and MNS can be solved in linear time on chordal graphs. Furthermore, as in the previous section, we will prove that all these searches compute the same $\mathcal{L}$-trees. We begin by showing that $\mathcal{L}$-trees of perfect elimination orderings and, therefore, $\mathcal{L}$-trees constructed by the four searches fulfil a strong structural property.

Lemma 5.4.11. *Let T be an $\mathcal{L}$-tree of the chordal graph $G = (V, E)$ constructed by the search order $\sigma = (v_1, \ldots, v_n)$ which is a perfect elimination order. Let v_i be arbitrary and C be the component of $G - \{v_1, \ldots, v_{i-1}\}$ which contains v_i. Then every component of $C - v_i$ contains exactly one child of v_i in T.*

Proof. We first show that there is at most one child of v_i in every component of $C - v_i$. Assume there are two children u and w in the same component C' of $C - v_i$. Without loss of generality, we assume that u is the σ-leftmost vertex in C'. Since C' is a connected component, there is an induced path between u and w in C'. Due to Lemma 5.4.2, the neighbour of w in this path is to the left of w but to the right of v_i in σ. Therefore, w cannot be connected to v_i in the $\mathcal{L}$-tree T of σ and there is at most one child of v_i in C'.

Due to the fact that C is a connected component, we see that v_i has a neighbour in every component C' of $C - v_i$. □

This lemma implies that every edge not in the $\mathcal{L}$-tree must connect a vertex to one of its ancestors. Due to Lemma 5.1.1, the tree is an $\mathcal{L}$-tree of DFS.

Corollary 5.4.12. *Let σ be a perfect elimination ordering of a chordal graph G and let T be the $\mathcal{L}$-tree of σ. Then for every edge $xy \in G - T$ it holds that either x is an ancestor of y or y is an ancestor of x, i.e., T is an $\mathcal{L}$-tree of DFS on G.*

Since LBFS, MCS and MNS compute perfect elimination orderings on chordal graphs, we know that their $\mathcal{L}$-trees on chordal graphs are $\mathcal{L}$-trees of DFS. Note that the converse is not true in general. Furthermore, this does not hold for graphs in general, as the example in Figure 5.6 shows.

As in the result for $\mathcal{F}$-trees of chordal graphs, we will show that all EFPFs compute the same $\mathcal{L}$-trees on chordal graphs.

Theorem 5.4.13. *For all EFPFs the set of rooted $\mathcal{L}$-trees of a chordal graph is the same.*

Proof. Let $G = (V, E)$ be a chordal graph, let $\sigma = (v_1, \ldots, v_n)$ be a perfect elimination ordering and let T be the $\mathcal{L}$-tree of σ. Furthermore, let $\mathcal{A}$ be an arbitrary EFPF. Assume $\mathcal{A}$ has chosen vertex v_i and the path from v_1 to v_i in the $\mathcal{L}$-tree of $\mathcal{A}$ is the same as in T. Let x be a child of v_i in T and let C be the component of $G - \{v_1, \ldots, v_i\}$ which contains x. Note that, due to Corollary 5.4.12, all neighbours of C lie on the path between v_1 and v_i and are elements of the set $\{v_1, \ldots, v_i\}$. Assume for contradiction that x cannot be the first vertex of C which is taken by $\mathcal{A}$ and let y be a vertex which can be the first vertex of C. Since $\mathcal{A}$ fulfils the component-neighbour property, there must be a vertex $v_j \in \{v_1, \ldots, v_i\}$ which is adjacent to y but not to x. This is a contradiction to Lemma 5.4.1. Therefore, there is a search order of $\mathcal{A}$ which has T as its $\mathcal{L}$-tree. □

We will now show that there is a simple algorithm that solves the $\mathcal{L}$-tree recognition problem of all EFPF searches.

Theorem 5.4.14. *On chordal graphs the rooted $\mathcal{L}$-tree recognition problem of all EFPFs, in particular of LBFS, LDFS, MCS and MNS, can be solved in linear time.*

Proof. Due to Theorem 5.4.13, it is sufficient to present an algorithm for the $\mathcal{L}$-tree problem of one EFPF. We will use a special version of LBFS to solve the problem: under all vertices which have maximal label in LBFS choose a vertex whose last visited neighbour is its parent in T. If there is no such vertex, we will show that T is not an $\mathcal{L}$-tree of LBFS rooted in r. To this end, consider the first step of LBFS where this happens and let v be a vertex with maximal label whose distance to the root of T is minimal.

Case 1: *The parent of v in T, say w, has already been visited by LBFS.*

Let G_w be the graph which is constructed by deleting w and all vertices that were taken before w and let C be the component of G_w which contains v. Since the neighbour of v visited last is not w, there must be a vertex in C which is visited before v. Let x be the vertex visited first in C. Due to Lemma 5.4.1, this implies that $N(v) \cap S \subseteq N(x)$, where S is the set of vertices visited before x. Hence, x is adjacent to w in G. Furthermore, by assumption the search has always chosen a vertex whose last visited neighbour is its parent in T, up until the choice of v. This implies that x is a child of w in T. Therefore, w has two children in T which lie in the same component of G_w. Due to Lemma 5.4.11, T cannot be an $\mathcal{L}$-tree of LBFS on G.

Case 2: *The parent of v in T, say w, has not yet been visited by LBFS.*

Then, due to the minimality of the distance of v to the root, the label of w is not maximal. Therefore, there is a vertex x that has labelled v and not w. Due to Corollary 5.4.12, all neighbours of v that have been taken so far must be an element of the path between the root and v in T. Therefore, they must be taken by every search of LBFS starting in r which constructs T as its $\mathcal{L}$-tree. This means v will always be taken before w and, therefore, T cannot be constructed by LBFS.

Thus, we have shown that the modified LBFS decides the $\mathcal{L}$-tree recognition problem of LBFS on chordal graphs. It remains to show that its running time is linear in the size of the graph. We use the standard implementation of LBFS with partition refinement. For every vertex we save its parent in T and its last visited neighbour in the run of

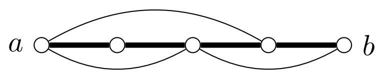


Figure 5.7: The path denoted by the thick edges is an $\mathcal{L}$-tree of DFS on this biconnected chordal graph. Both a and b can be the root of the tree. The path is also an $\mathcal{L}$-tree of a perfect elimination ordering of the graph. However, in this case only a can be the root of the tree, since a is not simplicial.

LBFS. After the neighbourhood of the last chosen vertex v refined the partition we iterate through the neighbours of v a second time. If v is the parent of the neighbour w in the tree T, w is pulled to the front of its partition class. Therefore, if there is a vertex with maximal label whose last visited neighbour is its parent in T, then the first vertex of our partition must be such a vertex. This can be checked in constant time. Since the neighbourhood of every vertex is visited only a constant number of times, the algorithm has an overall running time which is linear in the size of the graph. □

It remains to show that also the unrooted problem can be solved in linear time. For this we adapt the idea which Korach and Ostfeld [100] have used to recognize $\mathcal{L}$-trees of DFS. They showed that an $\mathcal{L}$-tree of DFS on a biconnected graph has either exactly one vertex which can be the root of the tree or it is a path. In that case, both end points of the path can be the root of the tree. Due to Corollary 5.4.12, the $\mathcal{L}$-trees of EFPFs are also $\mathcal{L}$-trees of DFS. Therefore, an $\mathcal{L}$-tree of an EFPF on a biconnected graph can have at most two roots. Note that for $\mathcal{L}$-trees of EFPFs which are paths it does not hold that both endpoints can be the root. We present a counterexample in Figure 5.7.

The idea of the algorithm is as follows. We first identify the maximal biconnected components (blocks) of the graph G. For every block G_i we consider the subtree T_i of T which is induced by G_i and check in time linear in the size of the block whether T_i is an $\mathcal{L}$-tree of DFS. If this is not the case, then, due to Corollary 5.4.12, T_i cannot be an $\mathcal{L}$-tree of an EFPF and, therefore, T cannot be an $\mathcal{L}$-tree of an EFPF. If T_i is an $\mathcal{L}$-tree of DFS then we can find all possible roots of T_i in linear time [100]. Since there can be at most two roots, we only have to solve the rooted recognition problem of $\mathcal{L}$-trees of EFPFs on T_i a constant number of times. If this fails for all roots, we know that T_i is not an $\mathcal{L}$-tree for any EFPF and, therefore, T is not an $\mathcal{L}$-tree of any EFPF. If for all G_i it holds that T_i is an $\mathcal{L}$-tree of EFPFs, we construct a directed graph $B = (V_B, E_B)$ as follows. For every block G_i of G there is vertex $v_i \in V_B$. Furthermore, for every cut vertex v of G there is a vertex v in V_B. For an arbitrary block G_i of G and a cut vertex v of G which is an element of G_i we add the directed edge $v_i \to v$ to E_B. If v can be the root of T_i we additionally insert the direct edge $v \to v_i$ to E_B. We can show the following lemma.

The following lemma is essentially the same as Lemma 3.3 in [100] and has just been adapted to our current setting.

Lemma 5.4.15 (based on Lemma 3.3 in [100])**.** *Let $G = (V, E)$ be a graph, T be a spanning tree of G and $G_1, \ldots, G_k$ the blocks of G. T is an $\mathcal{L}$-tree of some EFPF rooted* *in the vertex $r \in G_i$ if and only if there is directed path from v_i to every vertex v_j in B.*

We can find all vertices of B which fulfil this property in linear time [100]. Therefore, the following holds.

Corollary 5.4.16. *On chordal graphs the $\mathcal{L}$-tree recognition problem of every EFPF, in particular of LBFS, LDFS, MCS and MNS, can be solved in linear time. Furthermore, it is possible to compute all roots of such a tree in linear time.*

5.5 Recognising $\mathcal{F}$-trees on Split Graphs in Linear Time

We have seen in the previous section that on chordal graphs the set of $\mathcal{F}$-trees is the same for any EFPF, in particular for LBFS, LDFS, MCS and MNS, even though this does not hold for the respective search orders (see Figure 5.4). For split graphs, however, this also holds for BFS and we exploit this fact to derive a linear time algorithm for split graphs from the recognition algorithm for $\mathcal{F}$-trees of BFS [112].

Theorem 5.5.1. *A tree T is an $\mathcal{F}$-tree of BFS on a split graph G if and only if it is an $\mathcal{F}$-tree of an EFPF, in particular of LBFS, LDFS, MCS and MNS.*

Proof. Let $G = (V, E)$ be a split graph and let T be an $\mathcal{F}$-tree for BFS on G, generated by the order τ. Let $I = \{i_1, \dots, i_\ell\}$ be the independent set and $C = \{c_1, \dots, c_k\}$ be the clique of G. We show that there is an MNS ordering σ that generates a search tree that coincides with T which resolves the question for all EFPFs, due to Corollary 5.4.8.

Suppose τ starts with a clique vertex, without loss of generality c_1, that is, c_1 is the root of the search tree. Then, all other clique vertices c_2 to c_k are in the first layer of the $\mathcal{F}$-tree, and additionally, all independent set vertices which are adjacent to c_1 are in the first layer as well. Without loss of generality, i_1 to i_q are adjacent to c_1. Then i_{q+1} to i_ℓ are in the second layer of the tree T. Furthermore, suppose c_2 to c_k are indexed in the order of occurrence in the BFS order. Note that BFS may choose i_1 to i_q in arbitrary order before the last clique vertex is chosen.

Now, we construct an MNS order σ, such that the $\mathcal{F}$-tree of σ is T. We simply pick c_1 to c_k in ascending order, that is, we start with the same root c_1, followed by the clique vertices in unchanged order. Since all vertices in the clique have the same neighbourhood of visited vertices at every step and none of the i_x has a larger neighbourhood, this does not contradict the MNS search paradigm. Finally, we add the independent set vertices to σ. Here, we have to choose the independent vertices with larger neighbourhoods first. As the whole neighbourhood of each of these vertices is already chosen, this does not change the edges of the tree, i.e., the first visited neighbour. Since the neighbours of the independent set vertices are visited in the same order as in the BFS, the same $\mathcal{F}$-tree T is generated.

Now suppose that τ starts with an independent vertex and, without loss of generality, we label the root of the search tree T by i_1. Then the neighbours of i_1, say c_1 to c_q are in the first layer of the search tree. All other clique vertices and all independent set vertices which are neighbours of c_1 to c_q are in the second layer of the $\mathcal{F}$-tree T. Finally, all remaining independent set vertices are the in third layer. Again, note that c_1 to c_k are assumed to be indexed in the order of occurrence in the BFS order.

Again, a similar order σ, now starting with i_1, followed by c_1 to c_k in order of the indices, and afterwards followed by i_2 to i_ℓ, respecting neighbourhood inclusions, yields the same tree T and it is an MNS order analogous to the above argumentation.

Due to Corollary 5.4.8, every $\mathcal{F}$-tree of MNS is also an $\mathcal{F}$-tree of LBFS, and thus also of BFS. As the $\mathcal{F}$-trees for all EFPFs are the same on any chordal and thus split graph, due to Corollary 5.4.8, this concludes the proof of the theorem. □

Note that the $\mathcal{F}$-trees of DFS differ from the $\mathcal{F}$-trees of BFS on split graphs. Both the rooted and the unrooted $\mathcal{F}$-tree problem can be solved in linear time for BFS [112], and, therefore, this also holds for the other searches.

Corollary 5.5.2. *The (rooted) $\mathcal{F}$-tree problem of any EFPF, in particular for LBFS, LDFS, MCS and MNS, can be solved in linear time on split graphs.*

5.6 Conclusion

We have shown that the $\mathcal{F}$-tree problem is $\mathcal{NP}$-complete for LBFS, LDFS, MCS and MNS. Furthermore, we have given polynomial time algorithms for the $\mathcal{L}$-tree problem of LDFS and for both the $\mathcal{F}$-tree and the $\mathcal{L}$-tree problems of LBFS, LDFS, MCS and MNS on chordal graphs. To the best of our knowledge, no hardness results for the $\mathcal{L}$-tree problem are known. Thus, the question arises whether the $\mathcal{L}$-tree recognition problem is easy in general for every graph search.

If we compare the complexity of both search tree problems with the end-vertex problem (results here can be found in [9]), we see that in some cases the end-vertex problem is harder than recognising graph search trees; for example the end-vertex problem is hard for DFS on split graphs. However, there is no known combination of graph class and search for which the end-vertex problem is easy, but the tree-recognition problem is hard.

Moreover, we have also considered the rooted search tree problem. As we have already seen in Sections 5.0.1 and 5.1, if we can solve the rooted problem in polynomial time, we can also solve the unrooted problem efficiently by solving it for every vertex as the starting point of the search. Nevertheless, it could be possible that the unrooted problem is easier than the rooted problem, i.e., maybe it is easy to find a search order with arbitrary root that generates the tree, but it is $\mathcal{NP}$-hard to find one that uses the given root. For all cases discussed in this chapter, the results imply that both problems have the same complexity status. However, there might be a combination of graph class and search algorithm for which this is not the case.

As a second variant, we have considered the unlabelled problem, i.e., no labelled spanning tree is given, but an unlabelled tree with a matching number of vertices. Thus, we are looking for a search tree which is isomorphic to the given tree. Obviously, this problem is $\mathcal{NP}$-hard for $\mathcal{L}$-trees of DFS, since it includes the hamiltonian path problem. We have shown that this is not the case for graph classes closed under the insertion of leaves. However, many important classes such as interval graphs or comparability graphs do not fulfil this property, and it remains open whether there is a non-trivial combination of a search and a graph class where the unlabelled case is easy or even easier than the labelled one.

Finally, the search tree recognition problem can be easily translated to the setting of directed graphs and it is an interesting question whether this would lead to results that differ significantly from the undirected case.

In the literature, spanning trees with special properties and corresponding optimisation problems are well studied. Examples are the maximum leaf spanning tree problem [68] and distance approximating spanning trees [120]. Are there graph classes where search trees of the investigated graph searches solve or at least lead to an approximate solution of such problems?

Bibliography

[1] Pierre Aboulker, Pierre Charbit, Nicolas Trotignon, and Kristina Vušković. Vertex elimination orderings for hereditary graph classes. *Discrete Math.*, 338(5):825–834, 2015. ISSN 0012-365X.

[2] Louigi Addario-Berry, Maria Chudnovsky, Frédéric Havet, Bruce Reed, and Paul Seymour. Bisimplicial vertices in even-hole-free graphs. *J. Comb. Theory, Ser. B*, 98(6):1119–1164, 2008. ISSN 0095-8956.

[3] Liliana Alcón, Boštjan Brešar, Tanja Gologranc, Marisa Gutierrez, Tadeja Kraner Šumenjak, Iztok Peterin, and Aleksandra Tepeh. Toll convexity. *Eur. J. Comb.*, 46:161–175, 2015. ISSN 0195-6698.

[4] Jørgen Bang-Jensen, Jing Huang, and Erich Prisner. In-tournament digraphs. *J. Comb. Theory, Ser. B*, 59(2):267–287, 1993. ISSN 0095-8956.

[5] Michael Barot, Christof Geiss, and Andrei Zelevinsky. Cluster algebras of finite type and positive symmetrizable matrices. *J. Lond. Math. Soc., II. Ser.*, 73(3): 545–564, 2006. ISSN 0024-6107; 1469-7750/e.

[6] Jesse Beisegel. Characterising AT-free graphs with BFS. In *Graph-theoretic concepts in computer science. 44th international workshop, WG 2018, Cottbus, Germany, June 27–29, 2018. Proceedings*, pages 15–26. Cham: Springer, 2018. ISBN 978-3-030-00255-8/pbk; 978-3-030-00256-5/ebook.

[7] Jesse Beisegel, Maria Chudnovsky, Vladimir Gurvich, Martin Milanič, and Mary Servatius. Avoidable vertices and edges in graphs. In *Proceedings of the Algorithms and Data Structures Symposium (WADS 2019)*, 2019. To appear.

[8] Jesse Beisegel, Carolin Denkert, Ekkehard Köhler, Matjaž Krnc, Nevena Pivač, Robert Scheffler, and Martin Strehler. Recognizing graph search trees. In *Proceedings of the X Latin and American Algorithms, Graphs and Optimization Symposium (LAGOS 2019)*, 2019. To appear. Preprint on arXiv: `https://arxiv.org/abs/1811.09249`.

[9] Jesse Beisegel, Carolin Denkert, Ekkehard Köhler, Matjaž Krnc, Nevena Pivač, Robert Scheffler, and Martin Strehler. On the End-Vertex Problem of Graph Searches. *Discrete Mathematics & Theoretical Computer Science*, vol. 21 no. 1, ICGT 2018, June 2019. URL `https://dmtcs.episciences.org/5572`.

[10] Seymour Benzer. On the topology of the genetic fine structure. *Proceedings of the National Academy of Sciences of the United States of America*, 45(11):1607, 1959.

[11] Claude Berge. Sur une conjecture relative au problème des codes optimaux. *Comm. 13ieme Assemblee Gen. URSI*, pages 51–229, 1962.

[12] A. Berry, R. Krueger, and G. Simonet. Maximal label search algorithms to compute perfect and minimal elimination orderings. *SIAM J. Discrete Math.*, 23(1): 428–446, 2009. ISSN 0895-4801; 1095-7146/e.

[13] Anne Berry and Jean-Paul Bordat. Separability generalizes Dirac's theorem. *Discrete Appl. Math.*, 84(1-3):43–53, 1998. ISSN 0166-218X.

[14] Anne Berry, Jean R. S. Blair, Pinar Heggernes, and Barry W. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39(4):287–298, 2004. ISSN 0178-4617; 1432-0541/e.

[15] Anne Berry, Jean R. S. Blair, Jean-Paul Bordat, and Geneviève Simonet. Graph extremities defined by search algorithms. *Algorithms (Basel)*, 3(2):100–124, 2010. ISSN 1999-4893/e.

[16] Binay K. Bhattacharya and Damon Kaller. An $O(m + n \log n)$ algorithm for the maximum-clique problem in circular-arc graphs. *J. Algorithms*, 25(2):336–358, 1997. ISSN 0196-6774.

[17] Jean R. S. Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation. Proceedings of a workshop that was an integral part of the 1991-92 IMA program on "Applied linear algebra", Minneapolis, MN (USA)*, pages 1–29. New York: Springer-Verlag, 1993. ISBN 0-387-94131-2/hbk.

[18] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13:335–379, 1976. ISSN 0022-0000.

[19] Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31(1):212–232, 2001. ISSN 0097-5397; 1095-7111/e.

[20] Vincent Bouchitté and Ioan Todinca. Listing all potential maximal cliques of a graph. *Theor. Comput. Sci.*, 276(1-2):17–32, 2002. ISSN 0304-3975.

[21] Andreas Brandstädt, Feodor F. Dragan, and Falk Nicolai. LexBFS-orderings and powers of chordal graphs. *Discrete Math.*, 171(1-3):27–42, 1997. ISSN 0012-365X.

[22] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey.* Philadelphia, PA: SIAM, 1999. ISBN 0-89871-432-X/pbk; 978-0-89871-979-6/ebook.

[23] Andreas Brandstädt, Feodor F. Dragan, and Ekkehard Köhler. Linear time algorithms for Hamiltonian problems on (claw, net)-free graphs. *SIAM J. Comput.*, 30(5):1662–1677, 2000. ISSN 0097-5397; 1095-7111/e.

[24] Boštjan Brešar, Tatiana Romina Hartinger, Tim Kos, and Martin Milanič. 1-perfectly orientable K_4-minor-free and outerplanar graphs. *Discrete Appl. Math.*, 248:33–45, 2018. ISSN 0166-218X.

[25] Gunnar Brinkmann, Kris Coolsaet, Jan Goedgebeur, and Hadrien Mélot. House of Graphs: a database of interesting graphs. *Discrete Appl. Math.*, 161(1-2):311–314, 2013. ISSN 0166-218X.

[26] Hajo Broersma, Ton Kloks, Dieter Kratsch, and Haiko Müller. Independent sets in asteroidal triple-free graphs. *SIAM J. Discrete Math.*, 12(2):276–287, 1999. ISSN 0895-4801; 1095-7146/e.

[27] C. Carathéodory. Über den Variabilitätsbereich der Koeffizienten von Potenzreihen, die gegebene Werte nicht annehmen. *Math. Ann.*, 64:95–115, 1907. ISSN 0025-5831; 1432-1807/e.

[28] Jou-Ming Chang, Ton Kloks, and Hung-Lung Wang. Gray codes for AT-free orders via antimatroids. In *Combinatorial algorithms. 26th international workshop, IWOCA 2015, Verona, Italy, October 5–7, 2015. Revised selected papers*, pages 77–87. Cham: Springer, 2016. ISBN 978-3-319-29515-2/pbk; 978-3-319-29516-9/ebook.

[29] Maw-Shang Chang. Efficient algorithms for the domination problems on interval and circular-arc graphs. *SIAM J. Comput.*, 27(6):1671–1694, 1998. ISSN 0097-5397; 1095-7111/e.

[30] Pierre Charbit, Michel Habib, and Antoine Mamcarz. Influence of the tie-break rule on the end-vertex problem. *Discrete Math. Theor. Comput. Sci.*, 16(2):57–72, 2014. ISSN 1365-8050/e.

[31] Victor Chepoi, Feodor F. Dragan, Bertrand Estellon, Michel Habib, and Yann Vaxès. Notes on diameters, centers, and approximating trees of δ-hyperbolic geodesic spaces and graphs. In *The international conference on topological and geometric graph theory. Papers from the conference (TGGT 2008) held at the École Normale Supérieure, Paris, France, May 19–23, 2008*, pages 231–234. Amsterdam: Elsevier, 2008.

[32] Maria Chudnovsky, Gérard Cornuéjols, Xinming Lu, Paul Seymour, and Kristina Vušković. Recognizing Berge graphs. *Combinatorica*, 25(2):143–186, 2005. ISSN 0209-9683; 1439-6912/e.

[33] Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Ann. Math. (2)*, 164(1):51–229, 2006. ISSN 0003-486X; 1939-8980/e.

[34] Vašek Chvátal. Antimatroids, betweenness, convexity. In *Research trends in combinatorial optimization. Papers based on the presentations at the workshop Bonn,*

Germany, 2008. Dedicated to Bernard Korte on the occasion of the 70th birthday., pages 57–64. Berlin: Springer, 2009. ISBN 978-3-540-76795-4/hbk.

[35] Vašek Chvátal, Irena Rusu, and R. Sritharan. Dirac-type characterizations of graphs without long chordless cycles. *Discrete Math.*, 256(1-2):445–448, 2002. ISSN 0012-365X.

[36] Derek G. Corneil. A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. *Discrete Appl. Math.*, 138(3):371–379, 2004. ISSN 0166-218X.

[37] Derek G. Corneil and Richard M. Krueger. A unified view of graph searching. *SIAM J. Discrete Math.*, 22(4):1259–1276, 2008. ISSN 0895-4801; 1095-7146/e.

[38] Derek G. Corneil and Juraj Stacho. Vertex ordering characterizations of graphs of bounded asteroidal number. *J. Graph Theory*, 78(1):61–79, 2015. ISSN 0364-9024; 1097-0118/e.

[39] Derek G. Corneil, Stephan Olariu, and Lorna Stewart. Asteroidal triple-free graphs. *SIAM J. Discrete Math.*, 10(3):399–430, 1997. ISSN 0895-4801; 1095-7146/e.

[40] Derek G. Corneil, Stephan Olariu, and Lorna Stewart. Linear time algorithms for dominating pairs in asteroidal triple-free graphs. *SIAM J. Comput.*, 28(4): 1284–1297, 1999. ISSN 0097-5397; 1095-7111/e.

[41] Derek G. Corneil, Feodor F. Dragan, and Ekkehard Köhler. On the power of BFS to determine a graph's diameter. *Networks*, 42(4):209–222, 2003. ISSN 0028-3045; 1097-0037/e.

[42] Derek G. Corneil, Ekkehard Köhler, Stephan Olariu, and Lorna Stewart. Linear orderings of subfamilies of AT-free graphs. *SIAM J. Discrete Math.*, 20(1):105–118, 2006. ISSN 0895-4801; 1095-7146/e.

[43] Derek G. Corneil, Stephan Olariu, and Lorna Stewart. The LBFS structure and recognition of interval graphs. *SIAM J. Discrete Math.*, 23(4):1905–1953, 2009. ISSN 0895-4801; 1095-7146/e.

[44] Derek G. Corneil, Ekkehard Köhler, and Jean-Marc Lanlignel. On end-vertices of lexicographic breadth first searches. *Discrete Appl. Math.*, 158(5):434–443, 2010. ISSN 0166-218X.

[45] Derek G. Corneil, Barnaby Dalton, and Michel Habib. LDFS-based certifying algorithm for the minimum path cover problem on cocomparability graphs. *SIAM J. Comput.*, 42(3):792–807, 2013. ISSN 0097-5397; 1095-7111/e.

[46] Pilu Crescenzi, Roberto Grossi, Michel Habib, Leonardo Lanzi, and Andrea Marino. On computing the diameter of real-world undirected graphs. *Theor. Comput. Sci.*, 514:84–95, 2013. ISSN 0304-3975.

[47] George B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. Activity Analysis of Production and Allocation, Chap. XXI, 339-347 (1951)., 1951.

[48] Reinhard Diestel. *Graph theory*, volume 173. Berlin: Springer, 5th edition, 2017. ISBN 978-3-662-53621-6/hbk; 978-3-662-53622-3/ebook.

[49] G. A. Dirac. On rigid circuit graphs. *Abh. Math. Semin. Univ. Hamb.*, 25:71–76, 1961. ISSN 0025-5858; 1865-8784/e.

[50] Rodney G. Downey and Michael R. Fellows. *Fundamentals of parameterized complexity.* London: Springer, 2013. ISBN 978-1-4471-5558-4/hbk; 978-1-4471-5559-1/ebook.

[51] Feodor F. Dragan. Almost diameter of a house-hole-free graph in linear time via LexBFS. *Discrete Appl. Math.*, 95(1-3):223–239, 1999. ISSN 0166-218X.

[52] Feodor F. Dragan, Falk Nicolai, and Andreas Brandstädt. Lexbfs-orderings and powers of graphs. In Fabrizio d'Amore, Paolo Giulio Franciosa, and Alberto Marchetti-Spaccamela, editors, *Graph-Theoretic Concepts in Computer Science*, pages 166–180, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[53] Feodor F. Dragan, Falk Nicolai, and Andreas Brandstädt. Convexity and HHD-free graphs. *SIAM J. Discrete Math.*, 12(1):119–135, 1999. ISSN 0895-4801; 1095-7146/e.

[54] Pierre Duchet. Convex sets in graphs. II: Minimal path convexity. *J. Comb. Theory, Ser. B*, 44(3):307–316, 1988. ISSN 0095-8956.

[55] Jérémie Dusart and Michel Habib. A new LBFS-based algorithm for cocomparability graph recognition. *Discrete Appl. Math.*, 216:149–161, 2017. ISSN 0166-218X.

[56] Paul H. Edelman and Robert E. Jamison. The theory of convex geometries. *Geom. Dedicata*, 19:247–270, 1985. ISSN 0046-5755; 1572-9168/e.

[57] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. Assoc. Comput. Mach.*, 19:248–264, 1972. ISSN 0004-5411.

[58] Shimon Even. *Graph algorithms. Edited by Guy Even. With a foreword by Richard M. Karp. 2nd ed.* Cambridge: Cambridge University Press, 2nd ed. edition, 2012. ISBN 978-0-521-73653-4/pbk; 978-0-521-51718-8/hbk; 978-1-139-21056-0/ebook.

[59] Martin Farber and Robert E. Jamison. Convexity in graphs and hypergraphs. *SIAM J. Algebraic Discrete Methods*, 7:433–444, 1986. ISSN 0196-5212.

[60] Fedor V. Fomin and Dieter Kratsch. *Exact exponential algorithms.* Berlin: Springer, 2010. ISBN 978-3-642-16532-0/hbk; 978-3-642-16533-7/ebook.

[61] Fedor V. Fomin and Yngve Villanger. Finding induced subgraphs via minimal triangulations. In *STACS 2010. 27th international symposium on theoretical aspects of computer science, Nancy, France, March 4–6, 2010.*, pages 383–394. Wadern: Schloss Dagstuhl – Leibniz Zentrum für Informatik, 2010. ISBN 978-3-939897-16-3.

[62] Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM J. Comput.*, 44(1):54–87, 2015. ISSN 0097-5397; 1095-7111/e.

[63] Andras Frank. Some polynomial algorithms for certain graphs and hypergraphs. Proc. 5th Br. comb. Conf., Aberdeen 1975, 211-226 (1976)., 1976.

[64] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pac. J. Math.*, 15:835–855, 1965. ISSN 0030-8730.

[65] Harold N Gabow. Searching. In *Handbook of graph theory*, pages 953–984. Springer, 2013.

[66] T. Gallai. Maximum-Minimum Sätze über Graphen. *Acta Math. Acad. Sci. Hung.*, 9:395–434, 1958. ISSN 0001-5954.

[67] T. Gallai. Transitiv orientierbare Graphen. *Acta Math. Acad. Sci. Hung.*, 18: 25–66, 1967. ISSN 0001-5954.

[68] Michael R. Garey and David S. Johnson. *Computers and Intractability*. W. H. Freeman, 29th edition, 2002.

[69] Fanica Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Comb. Theory, Ser. B*, 16:47–56, 1974. ISSN 0095-8956.

[70] Fănică Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.*, 1:180–187, 1972. ISSN 0097-5397; 1095-7111/e.

[71] Alain Ghouila-Houri. Caractérisation des graphes non orientes dont on peut orienter les aretes de maniere à obtenir le graphe d'une rélation d'ordre. *C. R. Acad. Sci., Paris*, 254:1370–1371, 1962. ISSN 0001-4036.

[72] P. C. Gilmore and A. J. Hoffman. A characterization of comparability graphs and of interval graphs. *Can. J. Math.*, 16:539–548, 1964. ISSN 0008-414X; 1496-4279/e.

[73] John Gimbel. End vertices in interval graphs. *Discrete Appl. Math.*, 21(3):257–259, 1988. ISSN 0166-218X.

[74] M. C. Golumbic. The complexity of comparability graph recognition and coloring. *Computing*, 18:199–208, 1977. ISSN 0010-485X; 1436-5057/e.

[75] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs. 2nd ed.* Amsterdam: Elsevier, 2nd ed. edition, 2004. ISBN 0-444-51530-5/hbk.

[76] Martin Charles Golumbic and Clinton F. Goss. Perfect elimination and chordal bipartite graphs. *J. Graph Theory*, 2:155–193, 1978. ISSN 0364-9024; 1097-0118/e.

[77] Jan Gorzny and Jing Huang. End-vertices of LBFS of (AT-free) bigraphs. *Discrete Appl. Math.*, 225:87–94, 2017. ISSN 0166-218X.

[78] Martin Grötschel, László Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. IIASA Collab. Proc. Ser. CP-81-S1, 511-546 (1981)., 1981.

[79] M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theor. Comput. Sci.*, 234(1-2):59–84, 2000. ISSN 0304-3975.

[80] Hugo Hadwiger, Hans Debrunner, and Victor Klee. *Combinatorial geometry in the plane*. Courier Corporation, 2015.

[81] Torben Hagerup and Manfred Nowak. Recognition of spanning trees defined by graph searches. Technical Report A 85/08, Universität des Saarlandes, 1985.

[82] András Hajnal and János Surányi. Über die Auflösung von Graphen in vollständige Teilgraphen. *Ann. Univ. Sci. Budap. Rolando Eötvös, Sect. Math.*, 1:113–121, 1958. ISSN 0524-9007.

[83] György Hajós. Über eine Art von Graphen. *Intern. Math. Nachr.*, 11(65), 1957.

[84] Tatiana Romina Hartinger and Martin Milanič. 1-perfectly orientable graphs and graph products. *Discrete Math.*, 340(7):1727–1737, 2017. ISSN 0012-365X.

[85] Tatiana Romina Hartinger and Martin Milanič. Partial characterizations of 1-perfectly orientable graphs. *J. Graph Theory*, 85(2):378–394, 2017. ISSN 0364-9024; 1097-0118/e.

[86] Ryan Hayward. Generating weakly triangulated graphs. *J. Graph Theory*, 21(1): 67–69, 1996. ISSN 0364-9024; 1097-0118/e.

[87] Pinar Heggernes. Minimal triangulations of graphs: a survey. *Discrete Math.*, 306 (3):297–317, 2006. ISSN 0012-365X.

[88] Harald Hempel and Dieter Kratsch. On claw-free asteroidal triple-free graphs. *Discrete Appl. Math.*, 121(1-3):155–180, 2002. ISSN 0166-218X.

[89] Dorit S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Appl. Math.*, 6:243–254, 1983. ISSN 0166-218X.

[90] John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[91] John Hopcroft and Robert Tarjan. Efficient planarity testing. *J. Assoc. Comput. Mach.*, 21:549–568, 1974. ISSN 0004-5411.

[92] Wen-Lian Hsu and Jeremy P. Spinrad. Independent sets in circular-arc graphs. *J. Algorithms*, 19(2):145–160, 1995. ISSN 0196-6774.

[93] B. Jamison and S. Olariu. On the semi-perfect elimination. *Adv. Appl. Math.*, 9 (3):364–376, 1988. ISSN 0196-8858.

[94] Robert E. Jamison-Waldner. A perspective on abstract convexity: classifying alignments by varieties. Convexity and related combinatorial geometry, Proc. 2nd Conf., Oklahoma 1980, Lect. Notes Pure Appl. Math. 76, 113-150 (1982)., 1982.

[95] Frank Kammer and Torsten Tholey. Approximation algorithms for intersection graphs. *Algorithmica*, 68(2):312–336, 2014. ISSN 0178-4617; 1432-0541/e.

[96] Victor Klee. What are the intersection graphs of arcs in a circle? *The American Mathematical Monthly*, 76(7):810–813, 1969.

[97] Ekkehard Köhler. Recognizing graphs without asteroidal triples. *J. Discrete Algorithms*, 2(4):439–452, 2004. ISSN 1570-8667.

[98] Ekkehard Köhler. Linear structure of graphs and the knotting graph. In *Gems of Combinatorial Optimization and Graph Algorithms*, pages 13–27. Springer, 2015.

[99] Ekkehard G Köhler. *Graphs without asteroidal triples.* Cuvillier, 1999.

[100] Ephraim Korach and Zvi Ostfeld. DFS tree construction: Algorithms and characterizations. In Jan van Leeuwen, editor, *Graph-Theoretic Concepts in Computer Science*, pages 87–106, Berlin, Heidelberg, 1989.

[101] Bernhard Korte, László Lovász, and Rainer Schrader. *Greedoids.*, volume 4. Springer, Berlin, 1991. ISBN 3-540-18190-3.

[102] Norbert Korte and Rolf H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.*, 18(1):68–81, 1989. ISSN 0097-5397; 1095-7111/e.

[103] Dieter Kratsch and Jeremy Spinrad. Between $O(nm)$ and $O(n^{\alpha})$. *SIAM J. Comput.*, 36(2):310–325, 2006. ISSN 0097-5397; 1095-7111/e.

[104] Dieter Kratsch and Lorna Stewart. Domination on cocomparability graphs. *SIAM J. Discrete Math.*, 6(3):400–417, 1993. ISSN 0895-4801; 1095-7146/e.

[105] Dieter Kratsch, Mathieu Liedloff, and Daniel Meister. End-vertices of graph search algorithms. In *Algorithms and complexity. 9th international conference, CIAC 2015, Paris, France, May 20–22, 2015. Proceedings*, pages 300–312. Cham: Springer, 2015. ISBN 978-3-319-18172-1/pbk; 978-3-319-18173-8/ebook.

[106] R. Krithika, Rogers Mathew, N. S. Narayanaswamy, and N. Sadagopan. A Dirac-type characterization of k-chordal graphs. *Discrete Math.*, 313(24):2865–2867, 2013. ISSN 0012-365X.

[107] P. Sreenivasa Kumar and C. E. Veni Madhavan. Minimal vertex separators of chordal graphs. *Discrete Appl. Math.*, 89(1-3):155–168, 1998. ISSN 0166-218X.

[108] C. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundam. Math.*, 15:271–283, 1930. ISSN 0016-2736; 1730-6329/e.

[109] C. G. Lekkerkerker and J. Ch. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundam. Math.*, 51:45–64, 1962. ISSN 0016-2736; 1730-6329/e.

[110] Peter J. Looges and Stephan Olariu. Optimal greedy algorithms for indifference graphs. *Comput. Math. Appl.*, 25(7):15–25, 1993. ISSN 0898-1221.

[111] T. Ma. unpublished manuscript.

[112] Udi Manber. Recognizing breadth-first search trees in linear time. *Inf. Process. Lett.*, 34(4):167–171, 1990. ISSN 0020-0190.

[113] Ross M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003. ISSN 0178-4617; 1432-0541/e.

[114] Brendam D. McKay and Gordon F. Royle. The transitive graphs with at most 26 vertices. *Ars Comb.*, 30:161–176, 1990. ISSN 0381-7032.

[115] Daniel Meister. Recognition and computation of minimal triangulations for AT-free claw-free and co-comparability graphs. *Discrete Appl. Math.*, 146(3):193–218, 2005. ISSN 0166-218X.

[116] Lalla Mouatadid. Private Communication, 2015.

[117] Tatsuo Ohtsuki, Lap Kit Cheung, and Toshio Fujisawa. Minimal triangulation of a graph and optimal pivoting order in a sparse matrix. *J. Math. Anal. Appl.*, 54: 622–633, 1976. ISSN 0022-247X.

[118] Stephan Olariu. An optimal greedy heuristic to color interval graphs. *Inf. Process. Lett.*, 37(1):21–25, 1991. ISSN 0020-0190.

[119] James B. Orlin. Max flows in $O(nm)$ time, or better. In *Proceedings of the 45th annual ACM symposium on theory of computing, STOC '13. Palo Alto, CA, USA, June 1–4, 2013*, pages 765–774. New York, NY: Association for Computing Machinery (ACM), 2013. ISBN 978-1-4503-2029-0.

[120] Erich Prisner. Distance approximating spanning trees. In Rüdiger Reischuk and Michel Morvan, editors, *STACS 97*, pages 499–510, Berlin, Heidelberg, 1997. Springer. ISBN 978-3-540-68342-1.

[121] Srinivasa Rao Arikati and C. Pandu Rangan. Linear algorithm for optimal path cover problem on interval graphs. *Inf. Process. Lett.*, 35(3):149–153, 1990. ISSN 0020-0190.

[122] F. S. Roberts. Indifference graphs. Proof Tech. Graph Theory, Proc. 2nd Ann Arbor Graph Theory Conf. 1968, 139-146 (1969)., 1969.

[123] D. J. Rose. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.*, 32:597–609, 1970. ISSN 0022-247X.

[124] Donald J. Rose. *Symmetric elimination on sparse positive definite systems and the potential flow network problem.* ProQuest LLC, Ann Arbor, MI, 1970. Thesis (Ph.D.)–Harvard University.

[125] Donald J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. Graph Theory Comput., 183-217 (1972)., 1972.

[126] Donald J. Rose, R. Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:266–283, 1976. ISSN 0097-5397; 1095-7111/e.

[127] Inc. SageMath. *CoCalc Collaborative Computation Online*, 2018. `https://cocalc.com/`.

[128] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th annual ACM symposium on theory of computing, STOC'78, Dan Diego, CA, USA, May 1–3, 1978*, pages 216–226. New York, NY: Association for Computing Machinery (ACM), 1978.

[129] D. R. Shier. Some aspects of perfect elimination orderings in chordal graphs. *Discrete Appl. Math.*, 7:325–331, 1984. ISSN 0166-218X.

[130] Wei-Kuan Shih and Wen-Lian Hsu. An 0(n log n+m log log n) maximum weight clique algorithm for circular-arc graphs. *Inf. Process. Lett.*, 31(3):129–134, 1989. ISSN 0020-0190.

[131] Dale J. Skrien. A relationship between triangulated graphs, comparability graphs, proper interval graphs, proper circular-arc graphs, and nested interval graphs. *J. Graph Theory*, 6:309–316, 1982. ISSN 0364-9024; 1097-0118/e.

[132] Jeremy Spinrad and R. Sritharan. Algorithms for weakly triangulated graphs. *Discrete Appl. Math.*, 59(2):181–191, 1995. ISSN 0166-218X.

[133] Jeremy P. Spinrad. *Efficient graph representations.*, volume 19. Providence, RI: American Mathematical Society (AMS), 2003. ISBN 0-8218-2815-0/hbk.

[134] E. Steinitz. Bedingt konvergente Reihen und konvexe Systeme. (Teil I.). *J. Reine Angew. Math.*, 143:128–175, 1913. ISSN 0075-4102; 1435-5345/e.

[135] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972. ISSN 0097-5397; 1095-7111/e.

[136] Robert E. Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13:566–579, 1984. ISSN 0097-5397; 1095-7111/e.

[137] Robert Endre Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Inf.*, 6:171–185, 1976. ISSN 0001-5903; 1432-0525/e.

[138] Alan Tucker. Characterizing circular-arc graphs. *Bull. Am. Math. Soc.*, 76:1257–1260, 1970. ISSN 0002-9904; 1936-881X/e.

[139] Alan Tucker. Matrix characterizations of circular-arc graphs. *Pac. J. Math.*, 39: 535–545, 1971. ISSN 0030-8730.

[140] Alan Tucker. Structure theorems for some circular-arc graphs. *Discrete Math.*, 7: 167–195, 1974. ISSN 0012-365X.

[141] Alan Tucker. Circular arc graphs: New uses and a new algorithm. Theor. Appl. Graphs, Proc. Kalamazoo 1976, Lect. Notes Math. 642, 580-589 (1978)., 1978.

[142] Alan Tucker. An efficient test for circular-arc graphs. *SIAM J. Comput.*, 9:1–24, 1980. ISSN 0097-5397; 1095-7111/e.

[143] J. Urrutia and F. Gavril. An algorithm for fraternal orientation of graphs. *Inf. Process. Lett.*, 41(5):271–274, 1992. ISSN 0020-0190.

[144] M. L. J. van de Vel. *Theory of convex structures.* Amsterdam: North-Holland, 1993. ISBN 0-444-81505-8/hbk.

[145] Vijay V. Vazirani. *Approximation algorithms.* Berlin: Springer, 1999. ISBN 3-540-65367-8/hbk.

[146] V. I. Voloshin. Properties of triangulated graphs. Operations research and programming, Interuniv. Collect., Kishinev 1982, 24-32 (1982)., 1982.

[147] Douglas B. West. *Introduction to graph theory. 2nd ed.* New Delhi: Prentice-Hall of India, 2nd ed. edition, 2005. ISBN 81-203-2142-1.

[148] Yuli Ye and Allan Borodin. Elimination graphs. *ACM Trans. Algorithms*, 8(2):23, 2012. ISSN 1549-6325; 1549-6333/e.

Index