The structure of g_B-perfect graphs

Abschlussarbeit
im Studiengang
Bachelor of Science in Mathematik
an der Fakultät für Mathematik und Informatik
der FernUniversität in Hagen

vorgelegt von
Edwin Lock
Matrikel-Nr. 9163921

Betreuer:
Dr. Dominique Andres

Hagen, den 16. Juni 2016

FernUniversität in Hagen
Fakultät für Mathematik und Informatik

Erklärung

Name:	Edwin Lock
Matrikel-Nr.:	9163921
Fach:	Mathematik
Modul:	Bachelorarbeit

Ich erkläre, dass ich die vorliegende Abschlussarbeit mit dem Thema

The structure of g_B-perfect graphs

selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich, inhaltlich oder sinngemäß entnommenen Stellen als solche den wissenschaftlichen Anforderungen entsprechend kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für Zeichnungen, Skizzen oder graphische Darstellungen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Arbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate überprüft und ausschließlich für Prüfungszwecke gespeichert wird.

Datum: _____ **Unterschrift:** _____

The structure of g_B-perfect graphs

Edwin Lock

Abstract

The focus of this thesis is the class of g_B-perfect graphs: graphs G such that for every induced subgraph H, a variation of Bodlaender's maker-breaker vertex colouring game played on H admits a winning strategy for the second player, the *maker*, using no more than $\omega(H)$ colours, where $\omega(\cdot)$ denotes the clique number. Our central result is to characterise g_B-perfect graphs in terms of forbidden induced subgraphs, and then to provide an explicit structural description. In doing so, we resolve an outstanding open problem posed in [S. D. Andres. On characterizing game-perfect graphs by forbidden induced subgraphs. *Contributions to Discrete Math.*, 7(1):21–34, 2012], where game-perfect graphs for six variations of Bodlaender's game are considered.

Our forbidden-graph characterisation is based on the results of an optimised computational search procedure, which, together with manual verification, yields fifteen induced subgraphs $F_1, \ldots, F_{15}$ whose exclusion is necessary for a graph to be g_B-perfect. Then, in the converse direction, we perform a structural analysis on the graphs which conform to the forbidden-subgraph criterion. In a series of case distinctions, we characterise them into thirteen possible graph classes. Finally, we prove that all instances G thereof are g_B-perfect by providing explicit winning strategies for the second player with only $\omega(G)$ colours, thereby showing that a graph is g_B-perfect if and only if it contains no induced $F_1, \ldots, F_{15}$.

The present study of g_B-perfect graphs yields as a by-product two further results, which may be of independent interest. Firstly, we establish that triangle-free graphs are g_B-perfect if and only if they are complete bipartite or *almost* complete bipartite. Secondly, through the application of our exhaustive-search algorithm, we obtain a complete list of all minimal forbidden subgraphs with up to ten vertices for all six variations of Bodlaender's game, thereby providing a valuable stepping stone for future work in the field.

Die Struktur g_B-perfekter Graphen

Edwin Lock

Zusammenfassung

Das Hauptaugenmerk dieser Arbeit liegt auf der Klasse der g_B-perfekten Graphen: ein Graph G heißt g_B-perfekt, wenn es für jeden Untergraphen H von G eine Gewinnstrategie gibt, mit der der Maker als zweiter Spieler eine abgewandelte Form von Bodlaenders Maker-Breaker-Knotenfärbungsspiel auf H mit $\omega(H)$ Farben gewinnen kann, wobei $\omega(\cdot)$ die Cliquenzahl eines Graphen darstellt. Unser Hauptresultat ist die Charakterisierung g_B-perfekter Graphen mithilfe verbotener induzierter Untergraphen und mittels expliziter struktureller Beschreibungen. Auf diese Weise beantworten wir eine offene Frage aus dem Paper [S. D. Andres. On characterizing game-perfect graphs by forbidden induced subgraphs. *Contributions to Discrete Math.*, 7(1):21–34, 2012], das spielperfekte Graphen für sechs Variationen von Bodlaenders Knotenfärbungsspiel behandelt.

Unsere Charakterisierung mithilfe verbotener induzierter Untergraphen basiert auf den von Hand verifizierten Ergebnissen einer computergestützten Suche, die 15 verbotene Graphen $F_1, \ldots, F_{15}$ ergibt, welche als induzierte Untergraphen in keinem g_B-perfekten Graphen vorhanden sein dürfen. Umgekehrt wird zunächst eine strukturelle Analyse der Graphen ohne verbotener Untergraphen durchgeführt. Eine Reihe von Fallunterscheidungen erlaubt es uns, diese Graphen in 13 verschiedene Graphenklassen zu unterteilen. Schließlich weisen wir nach, dass jeder Graph G aus einer dieser Klassen g_B-perfekt ist, indem wir für jede Klasse eine Gewinnstrategie angeben, mit der der zweite Spieler auf G mit $\omega(G)$ Farben gewinnen kann. Wir zeigen damit, dass ein Graph genau dann g_B-perfekt ist, wenn er keine induzierten $F_1, \ldots, F_{15}$ enthält.

Die Untersuchung g_B-perfekter Graphen fördert zwei weitere Ergebnisse zutage, die hier aufgezeigt werden: Erstens weisen wir nach, dass Graphen der Cliquenzahl ≤ 2 genau dann g_B-perfekt sind, wenn sie vollständig bipartit oder *fast* vollständig bipartit sind. Zweitens erhalten wir als weiteres Resultat unserer computergestützten Suchmethode eine vollständige Liste aller verbotener induzierter Graphen mit bis zu zehn Knoten für alle sechs Variationen des Spiels von Bodlaender, die für weiterführende Arbeiten in dem Gebiet von Nutzen sein kann.

Acknowledgement

I would like to thank my thesis supervisor Dr. Dominique Andres for his unwavering encouragement, wise counsel and kind support, and for challenging and motivating me with the intriguing problem that lies at the heart of this work.

Contents

List of Figures

Chapter 1

Introduction

1.1 The aim of this thesis

Suppose that Alice and Bob play a vertex colouring game g_B on a simple[1] graph G with k available colours. Bob starts by colouring a vertex, upon which Alice and Bob then take turns to colour an uncoloured vertex such that adjacent vertices do not share the same colour. The game ends as soon as no vertex is colourable any more. If at this point G is fully coloured, Alice wins, otherwise she loses. This type of game is called a maker-breaker game, as Alice tries to *make* a proper graph colouring while Bob tries to *break* it.

We call the minimum number of colours that Alice requires to beat Bob on G at the game g_B the *g_B-chromatic number* $\chi_{g_B}(G)$. A graph G is considered *g_B-perfect* if Alice wins the game on H with $\chi_{g_B}(H) = \omega(H)$ colours for every induced subgraph H of G, where $\omega(H)$ denotes the clique number of H. Section 1.2 shows, for example, that the path graph P_5 is not g_B-perfect.

The main aim of this thesis is to characterise the class of g_B-perfect graphs by means of a forbidden induced subgraph characterisation and explicit structural descriptions. In Chapter 3 we first identify a set of fifteen minimal forbidden subgraphs $F_1, \ldots, F_{15}$ for g_B-perfect graphs using an optimised computational search procedure that is able to determine the outcome of all six of Andres's vertex colouring games (see Section 1.3) given a graph and a certain number of colours. The procedure itself is presented in Appendix A. The fifteen forbidden subgraphs thus found are then manually verified and taken as a starting point for the forbidden graph characterisations in Chapters 4 and 5.

Chapter 4 shows that connected, triangle-free graphs are g_B-perfect if and only if they are complete bipartite or *almost* complete bipartite, while Chapter 5 proves our main result: a graph is g_B-perfect if and only if it contains no induced $F_1, \ldots, F_{15}$ or, equivalently, if and only if it is an instance of graph classes $E_1^{\cup}, E_2, \ldots, E_{13}$. First we show that a graph G without induced $F_1, \ldots, F_{15}$ admits a decomposition based on a dominating edge. Using this knowledge, we make a series of hierarchical case distinctions to comprehensively identify G as an instance of one of thirteen possible graph classes $E_1^{\cup}, E_2, \ldots, E_{13}$. Next we show for all instances of $E_1^{\cup}, E_2, \ldots, E_{13}$ that any subgraph is an instance of one of the same. Lastly we provide explicit strategies for Alice to win on any instance I of $E_1^{\cup}, E_2, \ldots, E_{13}$ with $\omega(I)$ colours. This concludes our characterisation of g_B-perfect graphs.

Aside from our main result concerning g_B-perfect graphs, we also present a list of all minimal forbidden graphs with up to ten vertices for all six of Andres's vertex colouring games. The script used for this is shown in Appendix B and the results themselves are presented in Appendix C.

[1] The terminology used in this thesis is defined in Chapter 2.

1.2 Playing the game g_B on a P_5 with $\omega(P_5) = 2$ colours

We play the game g_B on a path graph P_5 with $\omega(P_5) = 2$ colours, green (circle) and blue (square). Assume Bob starts with the centre vertex in green. Alice can respond by colouring either a vertex adjacent to the centre or a pendant vertex. If she colours the vertex adjacent to the centre, she must use the colour blue, as adjacent vertices cannot be coloured the same. She can colour the pendant in green or blue. This leads to the three possible moves up to isomorphism shown in Figure 1.1.

If she colours a pendant in blue, its neighbouring vertex is surrounded by two colours and cannot be coloured any more. In the other two cases, Bob makes sure that the second vertex from the right is surrounded by two colours when he colours the right-most vertex in blue. The uncolourable or surrounded vertex in each case is marked with a cross. Bob has now effectively won, as the vertex $\times$ will be left uncoloured after Alice and Bob colour the other remaining vertices, no matter how Alice proceeds.

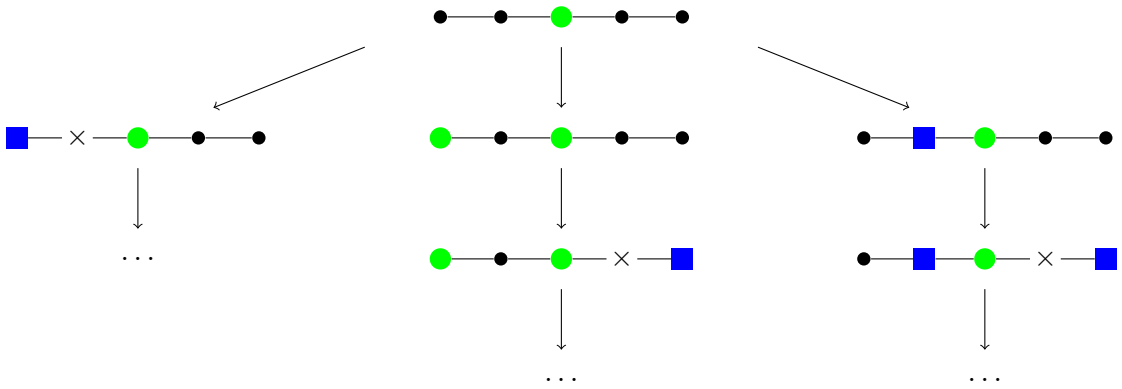


Figure 1.1: Playing the game g_B on a P_5 with $\omega(P_5) = 2$ colours.

1.3 Graph colouring games in literature

Vertex colouring games were first introduced by Brams in 1981 [27] and revived by Bodlaender [13] ten years later. One such game that goes by the name of COLOURING CONSTRUCTION GAME in [13] and which we denote by g_A requires Alice to start but is otherwise identical to the game g_B above. Four additional variants of these two games were introduced by Andres in [3] and [8], where either Alice or Bob is permitted to skip moves. We denote these games by $[A, A], [A, B], [B, A]$ and $[B, B]$: the first letter signifies which player starts and the second letter indicates the player who is allowed to skip moves. In this way one might also notate games g_A and g_B as $[A, -]$ and $[B, -]$, respectively. We refer to these six games as *Andres's vertex colouring games*.

For each game X, an interesting graph invariant is the *game chromatic number* $\chi_X(G)$, which denotes the least number of colours required for Alice to guarantee a win on a graph G.[2] Obviously, the game chromatic number of a graph G is bounded as follows:

$$\omega(G) \leq \chi(G) \leq \chi_X(G) \leq |G|, \tag{1.1}$$

where $\chi(G)$ denotes the chromatic number of G and $|G|$ is the number of vertices of G. As these strict bounds do not constitute a particularly deep result, it is more interesting to set an upper

[2]The term *game chromatic number* is frequently reserved only for the original game g_A. For variants X of the original game the term *X-chromatic number* is used.

limit k with $\omega(G) \leq k < |G|$ and study all graphs for which $\chi_X(G) \leq k$ holds. Alternatively one might take a class of graphs C and seek to determine a strict upper bound for $\{\chi_X(G) \mid G \in C\}$.

1.3.1 Characterising game-perfect graphs

The first approach is taken in [3] and [8], with an upper limit of $k = \omega(G)$, the lowest possible upper bound. As the class of graphs satisfying this upper bound does not have a nice structure in itself [8], it is additionally required that this upper bound hold for all induced subgraphs of G. In analogy to Berge's perfect graphs [11], graphs satisfying this condition are called *game-perfect* or *X-perfect*, depending on the game X.

The perfect graphs were famously characterised by Chudnovsky et al. [18] in the Strong Perfect Graph Theorem.

Theorem 1 (Chudnovsky, Robertson, Seymour, Thomas (2006))**.** *A graph is* perfect *if and only if it is* Berge*, i.e. if it contains neither odd induced cycles of size ≥ 5 nor induced complements of odd cycles of size ≥ 5.*

This motivates a characterisation of game-perfect graphs along the same lines.

Problem 2. Let X be one of Andres's six vertex colouring games. Provide a forbidden induced subgraph characterisation for X-perfect graphs.

In [8] such a forbidden induced subgraph characterisation was provided for the three games g_A, $[A, B]$ and $[B, B]$, together with an explicit structural description of the respective X-perfect graphs. This thesis presents an analogous characterisation of g_B-perfect graphs in continuation of these efforts.

While the structures of $[A, A]$ and $[B, A]$-perfect graphs have so far resisted characterisation, partial results have been achieved. In [8], a few forbidden subgraphs for both games were identified and in [3] the following two results for small classes of graphs were obtained.

Theorem 3 (Andres (2009))**.** *A triangle-free graph G is $[A, A]$-perfect if and only if every connected component of G is either a K_1, a $K_{m,n}$ or a $K_{m,n} - e$, where e is an edge.*

Theorem 4 (Andres (2009))**.** *Complements of bipartite graphs are $[A, A]$-perfect.*

1.3.2 Upper bounds for classes of graphs

A long-standing hunt initiated by Faigle et al. [25] in 1993 is to provide strict upper bounds for the g_A-chromatic number of specific classes of graphs. In [25], it was shown that $\chi_{g_A}(T) \leq 4$ for every tree T, $\chi_{g_A}(G) = O(\log |G|)$ for the class of planar graphs and $\chi_{g_A}(G) \leq 3\omega(G) - 2$ for interval graphs G. A better upper bound for the class of planar graphs is provided by Zhu in [37] and certain classes of cactus graphs are studied in [34]. These are just a few results found in the literature.

Many upper bounds for classes of graphs have been achieved with the help of another vertex colouring game, the *marking game*, which was introduced by Zhu [36]. The graph invariant of this game, the *game colouring number* $\mathrm{col}_g(G)$, is an extension of the colouring number introduced by Erdős and Hajnal in [23]. It turns out that $\chi_{g_A}(G) \leq \mathrm{col}_g(G)$, which means an upper bound for the game colouring number is also an upper bound for the game chromatic number. It is often easier to find an upper bound for $\mathrm{col}_g(G)$ for classes of graphs.

1.3.3 Related games

Apart from the marking game and Andres's five variations mentioned above, Bodlaender's vertex colouring game has also inspired other graph colouring games. One of these is the *edge colouring game*. As its name suggests, the *edge colouring game* requires Alice and Bob to colour edges instead of vertices. It was first introduced by Cai and Zhu [14] and its graph invariant is the *game chromatic index*. Various results have been achieved regarding the upper bound of the game chromatic index for special graph classes. Following initial results in [14] that certain trees of maximum vertex degree $\Delta = 3$ have a game chromatic index of at most $\Delta + 1 = 4$, Erdős et al. [24] proved that trees of maximum vertex degree $\Delta = k \geq 6$ have a game chromatic index of at most $\Delta + 1$. Andres [1] then showed that this upper bound $\Delta + 1$ holds even for all *forests* of maximum degree $\Delta \geq 5$. For wheel graphs W of order $n \geq 6$ the game chromatic index of W is n [9]. Further results for various classes of graphs can be found in [10, 15, 12].

Another type of game played on oriented graphs was introduced by Nešetřil and Sopena [29] as a game analogue to the oriented chromatic number. A few results for this game can be found in [29, 30, 31]. An entirely different extension of Bodlaender's original game g_A is played on digraphs. Introduced by Andres [2], a few additional results for this game and for various classes of graphs can be found in [5, 6, 7, 16]. Lastly, further games include *relaxed graph colouring games* [17, 22] and *incidence colouring games* [4].

Chapter 2

Terminology and graph colouring

2.1 Basic terms

A *graph* G consists of a set of *vertices* $V(G)$ and a set of *edges* $E(G)$. We call the number of vertices in G the *order* $|G|$ of G. Every edge e in $E(G)$ is associated with exactly two vertices $a, b \in V(G)$, with which it is *incident*. We say that a and b are *adjacent* or *neighbours*. If the two vertices that e is incident with are identical, we call the edge a *loop*. A *simple* graph is a graph without loops that has no more than one edge associated with any two vertices. In this thesis all graphs will be simple. This allows us to identify any edge e with its incident vertices a, b and denote it by ab.

For every vertex v in G, the *degree* of v is the number of neighbours it has. A *pendant* is a vertex of degree 1, i.e. a vertex that has only one neighbour. We also record the neighbouring vertices of v themselves in $N_G(v)$, which denotes the set of all neighbours of v in G. This definition can be extended to sets of vertices in the following way. Let S be a set of vertices in G. We define by $N_G(S)$ the set of all vertices in $V(G) \setminus S$ that are adjacent to at least one vertex in S. If it is obvious which graph we are talking about, we simply write $N(v)$ and $N(S)$.

A set of vertices $S \subseteq V(G)$ is called *independent* if none of the edges in G are incident with two vertices in S. A set of edges M in a graph is called *independent* or *a matching* if for any two edges $e, f \in M$, e and f are not incident with the same vertex. We say that M is a *matching of* $S \subseteq V(G)$ if there exists an edge in M that is incident with vertex s for every $s \in S$.

Let S be a set of vertices $S \subseteq V(G)$ of G. If we delete all vertices $V(G) \setminus S$ and incident edges from G, we call the resulting graph an *induced subgraph* H of G and say that S *induces* H. If all vertices in H are adjacent to one another, we call the vertex set S a *clique*.[1] Let $S \subseteq V(G)$ be a clique of maximum size. Then we say that $|S|$ is the *clique number* $\omega(G) = |S|$ of G.

We can also remove vertices from a graph G by specifying them directly. Let S be such a set of vertices. Then $G - S$ is the graph obtained by deleting all vertices in S and all incident edges from G. If S consists of a single vertex v, we also use the shorthand $G - v$. Lastly, if U is an induced subgraph of G and we seek to remove U from G, we write $G \setminus U$, which is equivalent to $G - V(U)$.

Let $A, B \subset V(G)$ be two disjoint sets of vertices in G. If every vertex in A is adjacent to every vertex in B we say that A and B are *completely connected*. If G has no edge incident with vertices in A and B, we say that A and B are *completely disconnected*. If A and B are neither completely

[1] Occasionally the subgraph H itself is also called a clique.

connected nor completely disconnected, we say that A and B are *partially connected*. We use the same terminology when we speak about the respective subgraphs that A and B induce.

A non-empty graph G is considered *connected* if there exists no non-trivial, proper subset A of $V(G)$ such that A and $V(G) \setminus A$ are completely disconnected. Equivalently, G is connected if for every pair of vertices a, b in G there exists a path from a to b [21]. If a graph G is not connected, it consists of two or more *components*, which are maximal connected subgraphs of G.

Let G be a connected graph. A set S of vertices in G is called a *dominating set* if every vertex not in S is adjacent to one or more vertices in S. A dominating set is called a *dominating clique* if it is a clique. A dominating clique of size two is also called a *dominating edge* and a dominating clique of size one is called a *dominating vertex*.

Finally, we define the following notation to describe non-elementary graphs. Let G, H be two graphs. Then $G \vee H$ is the graph obtained by completely connecting all vertices in G to all vertices in H. This operation $\vee$ is called a *graph join*. Another operation is the *disjoint union* $\cup$. The union of G and H is the graph $G \cup H$ that consists of two subgraphs G and H which are completely disconnected. We use the shorthand $G \vee v$ and $G \cup v$ to denote the join or union $G \vee K_1$ and $G \cup K_1$ if v is the vertex of the K_1.

2.2 Classes of graphs

The following classes of graphs make an appearance in this thesis.

A *cycle graph* C_n is a connected graph with n vertices such that every vertex has degree 2. We call C_n an *odd cycle* if n is odd.

A *path graph* P_n is a cycle graph C_n with one edge removed.

A graph is considered *complete* if every vertex is adjacent to every other vertex in the graph. A complete graph with n vertices is denoted by K_n. A K_3 is also called a *triangle*. If S is a clique of a graph G, then the subgraph H of G induced by S is a complete graph.

We say that a graph is *triangle-free* if it does not contain an induced triangle. Equivalently, a graph is triangle-free if its clique number is ≤ 2.

A graph is *bipartite* if its set of vertices can be partitioned into two disjoint sets V_1 and V_2 such that every edge in G is incident with a vertex in V_1 and a vertex in V_2. Equivalently, a graph is bipartite if and only if it does not contain any odd induced cycles [21].

Let G be a bipartite graph with disjoint vertex sets V_1 and V_2 of size m and n, respectively. If every vertex in V_1 is adjacent to every vertex in V_2, G is a *complete bipartite* graph denoted by $K_{m,n}$. The $K_{1,3}$ is also called the *claw graph*. Let $G = K_{m,n}$ be a complete bipartite graph. We call the graph obtained by deleting one edge e from G an *almost complete bipartite* graph and denote it by $K_{m,n} - e$.

Let G be a graph. The *complement* or *inverse* $\overline{G}$ of G is defined as a graph with an identical vertex set such that two vertices are adjacent if and only if they are not adjacent in G.

Lastly, the *n-spider with thin legs* is the graph obtained by taking a K_n and adding an additional pendant vertex adjacent to v for every vertex $v \in K_n$. The 3-spider with thin legs is shown in Figure 2.1.

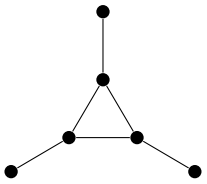


Figure 2.1: The 3-spider with thin legs.

2.3 Competitive and non-competitive graph colouring

Given a graph G, we can assign colours to vertices or edges. Vertex colourings are often required to obey certain rules. One such natural rule is the following. A *proper vertex colouring* of G is a labelling of every vertex with colours 1 to k in such a way that adjacent vertices are not labelled the same. We are often interested in the smallest number of colours required to achieve a proper colouring on G. This graph invariant is called the *chromatic number* $\chi(G)$ of G. Obviously we have $\omega(G) \leq \chi(G)$, since a clique of size n cannot properly be coloured with fewer than n colours. A graph G is called *perfect* if $\omega(H) = \chi(H)$ for every induced subgraph H of G.

As seen in Chapter 1, we can also colour the graph competitively. For each of Andres's vertex colouring games X, this naturally leads to the *game chromatic number* $\chi_X(G)$. Recall that this number is defined as the smallest number of colours Alice requires to successfully achieve a proper vertex colouring of G while her adversary Bob tries to prevent this, subject to certain rules. It follows immediately that the chromatic number is a strict lower bound for the game chromatic number and we have $\chi(G) \leq \chi_X(G)$. The game analogue to Berge's perfect graphs is defined in two steps. We say for a given vertex colouring game X that the graph G is *X-nice* if $\omega(G) = \chi_X(G)$ and that G is *X-perfect* if every induced subgraph H of G is X-nice.

Forbidden graph characterisations generally find frequent application in graph theory, most notably shining light on the nature of bipartite graphs [21], planar graphs (Kuratowski's Theorem [32]) and perfect graphs (Strong Perfect Graph Theorem [18]). We introduce the concept of a forbidden graph characterisation using *induced subgraphs* for X-perfect graphs specifically.

We call a graph H that is not X-nice a *forbidden induced subgraph* because its presence as an induced subgraph in a graph G means that G is not X-perfect. If H contains no other forbidden graph as an induced subgraph, we say that H is a *minimal forbidden induced subgraph*. Now let F be a set of minimal forbidden induced subgraphs. F is *comprehensive* if the following is true: a graph G is X-perfect if and only if it does not have any graph in F as an induced subgraph.

Lastly we present an observation that shortens the duration of Andres's vertex colouring games in certain situations. In our original definition of the game g_B we stated: "The game ends as soon as no vertex is colourable any more." We can equivalently say that the game ends as soon as all vertices *at risk* have been *neutralised* or as soon as a vertex has been *surrounded*.

Assume we are playing one of Andres's games played on a graph G with k colours. We say that a vertex v is *surrounded* if v is uncoloured and its neighbours $N(v)$ have been coloured in k different colours. Because there is no colour left with which to colour v, Bob will inevitably win and we can break off the game at this point. We have already encountered this situation in Section 1.2 for the vertices marked $\times$. Not all vertices in G necessarily have the potential of being surrounded. Pendants, for example, are never surroundable. We say that a surroundable vertex is *at risk*. Furthermore, this status can change throughout the game. A vertex v at risk can be *neutralised* either by colouring v itself or by making sure that the set of its neighbours $N(v)$ admits at most $k-1$ colours. If all vertices originally at risk have been neutralised, Bob cannot surround any vertices any more and Alice inevitably wins.

Chapter 3

Identifying forbidden subgraphs

3.1 Determining forbidden subgraphs computationally

We employ a computational search to systematically identify all minimal forbidden subgraphs of order $\leq n$ for each of Andres's six games. At the heart of our approach lies an optimised backtracking algorithm, presented in Appendix A, that is able to determine the outcome of Andres's games on a given graph and with a specified number of colours. Using this backtracking algorithm and the induced subgraph test made available by the *SageMath* software [20], a script then systematically tests all graphs up to order $\leq n$ to see whether they are minimal forbidden subgraphs. This script is shown in Appendix B. Note that there may be any number of forbidden subgraphs of order $> n$ that are not detectable in this way. The sets of forbidden graphs of order ≤ 10 for all six of Andres's vertex colouring games are shown in Appendix C.

For g_B-perfect graphs in particular, this approach detects the fifteen minimal forbidden subgraphs $F_1, \ldots, F_{15}$ of order ≤ 7 shown in Figure 3.1. We verify in Theorem 5 below that these graphs are indeed forbidden subgraphs for any g_B-perfect graph G by providing an explicit strategy for Bob to win on each of them. As we will see in Chapter 5, this set of fifteen graphs even constitutes a *comprehensive* set of minimal forbidden subgraphs for g_B-perfect graphs. Out of these fifteen graphs, the nine graphs F_1 to F_9 were already provided in [8] and the remaining graphs were newly discovered.

3.2 Proving that $F_1, ..., F_{15}$ are forbidden subgraphs

Theorem 5. *If G is a g_B-perfect graph, it contains no induced $F_1, \ldots, F_{15}$ from Figure 3.1.*

Proof. We show for each $F_i, 1 \leq i \leq 15$, that Alice cannot win on F_i with $w(F_i)$ colours. As this means that F_i is not g_B-nice, it follows immediately that any graph G with an induced F_i is not g_B-perfect.

If $\omega(F_i) < \chi(F_i)$, we are done, as we have $\omega(F_i) < \chi(F_i) \leq \chi_{g_B}(F_i)$. Otherwise we provide a concrete winning strategy for Bob with $\omega(F_i)$ colours. As $\omega(F_i) \leq 3$ for all $1 \leq i \leq 15$, we can make do with colours red, green and blue. We use the vertex labels found in Figure 3.1.

F_1: As $\omega(F_1) = 2$ and $\chi(F_1) = 2$, we provide a winning strategy for Bob with two colours, red and blue. Bob begins by colouring the isolated vertex e in red. Alice now colours any other vertex in red or blue. No matter which vertex and colour Alice chooses, Bob is able to colour a vertex of distance 2 with the other colour. The vertex in between these two differently coloured vertices cannot be coloured any more and Bob wins.

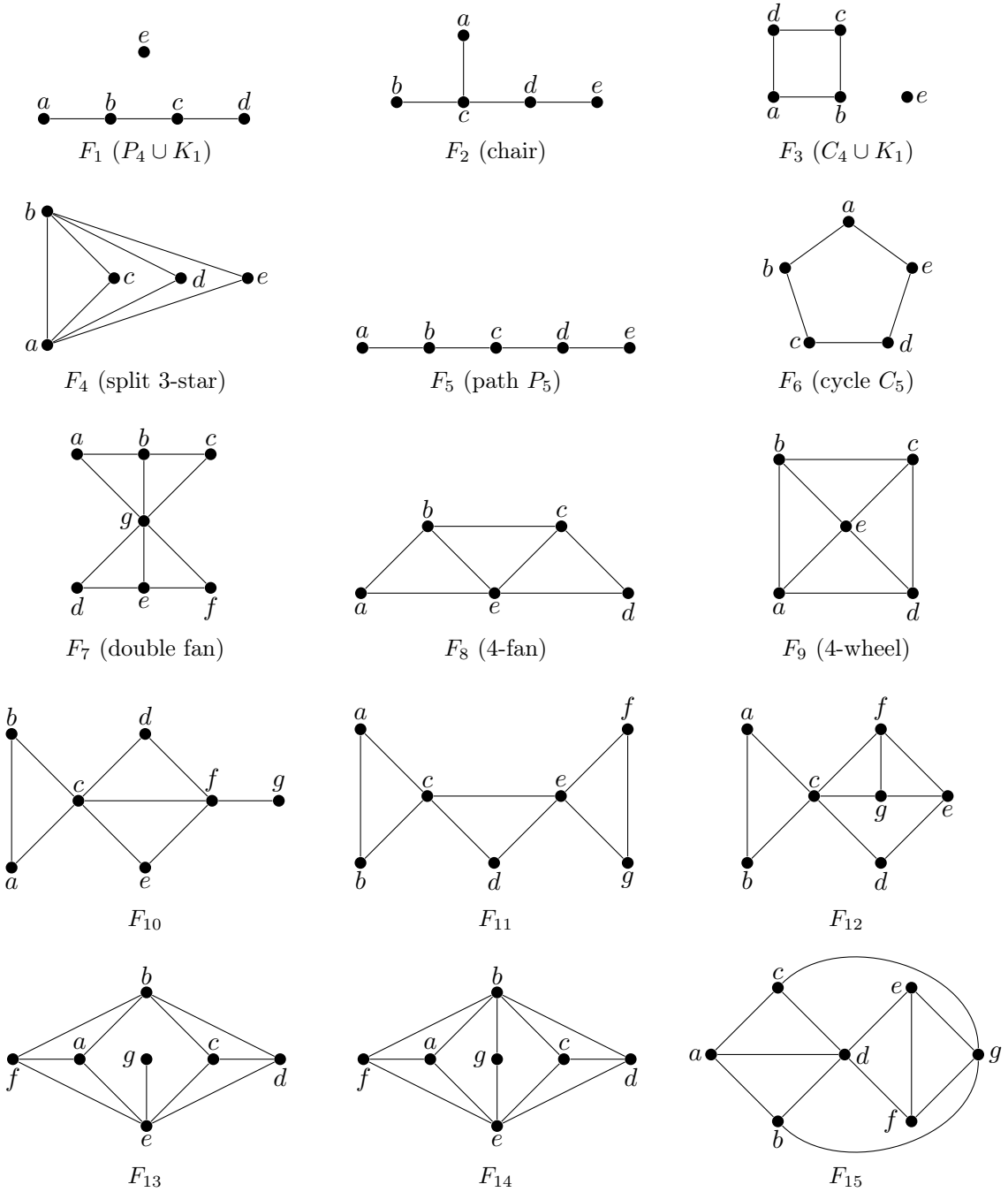


Figure 3.1: The 15 forbidden induced subgraphs for g_B-perfect graphs.

F_2: As $\omega(F_2) = 2$, we provide a winning strategy with two colours red and blue. Bob starts with a in red. If Alice colours c in blue, Bob wins with e in red because d has been surrounded. If Alice does not colour c, then Bob is free to colour either b or d in blue, surrounding c.

F_3: $\omega(F_3) = 2$. Bob starts with the isolated vertex e in red. Alice must now colour a vertex, say a, of the C_4 component of F_3. Bob's response is to colour vertex c in the other colour, thereby surrounding b and d.

F_4: $\omega(F_4) = 3$. We use colours red, green and blue. Observe that c, d and e must have the same colour in every proper vertex colouring with three colours: if w.l.o.g. c and d had different colours, then both a and b would need to be coloured in the third available colour. As a and b are adjacent, however, that leads to a contradiction. This allows Bob to win as follows: Bob starts with c in red. Regardless of Alice's next step, Bob colours either d or e with an unused colour.

F_5: $\omega(F_5) = 2$. We use colours red and blue. Bob starts by colouring c in red. If Alice colours a or b in any colour, Bob colours e in blue, surrounding d. Otherwise Bob colours a in blue, surrounding b.

F_6: As $\omega(F_6) = 2 < \chi(F_6) = 3$, no strategy is necessary.

F_7: $\omega(F_7) = 3$. Note that a and c must be coloured the same for every proper vertex colouring with three colours, as must be d and f. Bob colours a in red. Alice must colour c in red to prevent Bob from colouring c in any other colour in his next move. Bob next colours d in green. Alice must respond with f in green. Bob now wins with b in blue by surrounding g.

F_8: $\omega(F_8) = 3$. Note that in every proper vertex colouring with three colours, both pairs a, c and b, d must be uniformly coloured. Bob starts with e in red. Alice must colour a, b, c or d in green. Whichever vertex Alice has chosen, Bob colours the other vertex of the pair in blue, surrounding the uncoloured vertex in between the pair.

F_9: Bob follows the same strategy as on F_8.

F_{10}: $\omega(F_{10}) = 3$. Observe that d and e must be coloured the same. Bob starts with e in red. Alice must respond with d in red. Bob then colours b in green. If Alice now colours a in red, Bob can surround c by colouring f in blue. If Alice colours a in blue, she herself surrounds c. If she colours c in blue, Bob wins by colouring g in green. If Alice colours f or g in either possible colour, Bob can colour a in blue and surround c.

F_{11}: $\omega(F_{11}) = 3$. Bob begins with d in red. Due to symmetry, we only need to study three possibilities. If Alice colours

1. c in green, Bob responds with g in blue and surrounds e.
2. a in green, Bob responds with e in blue and surrounds c.
3. a in red, Bob colours g in green. If Alice continues with f or c in blue, she herself surrounds e, and with b in green she forces both c and e to take the same colour, a contradiction. If Alice instead colours e in blue, Bob wins with b in green by surrounding c. If Alice instead colours f in red, Bob wins with b in green because both c and e must now be coloured in blue, a contradiction. Finally, if Alice chooses c in green, Bob wins by colouring f in blue, while if Alice chooses b in blue, Bob wins with f in blue, surrounding e.

F_{12}: $\omega(F_{12}) = 3$. Note that c and e must be coloured the same. Bob starts with a in red. If Alice responds with b in green, Bob surrounds c by colouring d in blue. If Alice colours c in green, Bob wins with e in red as c and e are now differently coloured. Similarly, if Alice

colours e in any colour, Bob wins by colouring c with a different colour. Otherwise we distinguish between the following, relabelling f and g where necessary:

1. Alice colours either f or d in red. Bob responds with the other vertex in the same colour. If Alice then colours b in green, Bob wins with g in blue. If Alice chooses c in green, Bob wins with e in blue. If Alice chooses e or g in green, Bob wins with b or c in blue, respectively.

2. Alice colours either f or d in green. Bob responds with the other vertex in the same colour. If Alice then colours b in green, Bob wins with g in blue by surrounding c. If instead Alice colours b in blue, she surrounds c herself. If Alice chooses c in blue, Bob wins with e in red. If she colours e in red, Bob colours c in blue and surrounds g. If Alice colours g in blue, she surrounds c herself. Lastly, if she colours e in blue or g in red, Bob wins with b in blue by surrounding c.

F_{13}**:** $\omega(F_{12}) = 3$. Observe that b and e must be coloured the same and g must have a different colour to b and e. Bob begins with g in red. Alice cannot colour e or b, as that would allow Bob to achieve different colours for b and e on his next move. Therefore she must colour a, c, d or f. W.l.o.g. let a be the vertex she chooses. If Alice colours a in green, Bob colours b in red. As e cannot have the same colour as b any more, he wins. If Alice colours a in red, Bob applies the same colour to d. If Alice next colours b in green, Bob wins with e in blue and vice versa because c and f are surrounded. If she instead colours c in green, Bob wins by colouring f in blue and vice versa because b and e are surrounded.

F_{14}**:** Bob follows the same strategy as on F_{13}.

F_{15}**:** Observe that the pairs b, c and d, g must be uniformly coloured. Bob starts with b in red. Alice must colour c in red to stop Bob from colouring c in a different colour on his next move. Bob then colours f in red. If Alice colours a or e in green, Bob surrounds d by colouring the other vertex in blue. If Alice colours d or g in green, Bob wins by colouring the other vertex in blue.

This proves that graphs $F_1, \ldots, F_{15}$ are all forbidden induced subgraphs. □

Chapter 4

Characterising connected triangle-free g_B-perfect graphs

Before we characterise all g_B-perfect graphs in the next chapter, we investigate only a small subclass, namely the g_B-perfect graphs that are both connected and triangle-free. We prove that connected, triangle-free graphs are g_B-perfect if and only if they contain no induced $F_1, \ldots, F_{15}$ from Figure 3.1. We can restrict ourselves to the five forbidden subgraphs F_1, F_2, F_3, F_5, F_6 in Figure 4.1, as all other graphs contain triangles and cannot be induced subgraphs of any triangle-free graph. In fact, as we shall see in the proof of Theorem 6, it even suffices to restrict ourselves to F_2, F_5 and F_6 to achieve a successful characterisation. In addition, we provide a direct structural characterisation: a connected, triangle-free graph is g_B-perfect if and only if it is a complete or *almost* complete bipartite graph. Recall that an *almost* complete bipartite graph is a complete bipartite graph from which one edge has been removed (cf. page 6).

4.1 Our result

Theorem 6. *Let G be a connected, triangle-free graph. Then the following are equivalent:*

(i) G is g_B-perfect.

(ii) G contains no induced chair, P_5 or C_5 (F_2, F_5 and F_6 in Figure 4.1).

(iii) G is a $K_{m,n}$ with $m, n \geq 0$ or a $K_{m,n} - e$, where e is an edge and $m, n \geq 2$.

Proof. If G has zero or one vertices, the theorem obviously holds. We assume from now on that G has two or more vertices.

(i) ⇒ (ii): Let G be g_B-perfect. In Chapter 3, Theorem 5 we show that G cannot contain an induced $F_1, \ldots, F_{15}$, which includes the chair, the P_5 and the C_5 (i.e. F_2, F_5 and F_6).

(ii) ⇒ (iii): This follows immediately from Lemma 12 in Section 4.2.

(iii) ⇒ (i): In Section 4.3, we show that every induced subgraph H of G is g_B-nice by providing a strategy for Alice to win on H with $\omega(H)$ colours. It follows immediately that G is g_B-perfect. □

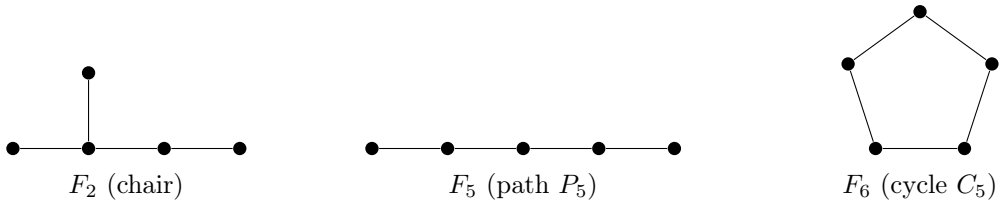


Figure 4.1: The 3 forbidden induced subgraphs for connected triangle-free g_B-perfect graphs.

4.2 The structure of triangle-free g_B-perfect graphs

Let G denote a connected triangle-free graph without an induced chair, P_5 or C_5 and with two or more vertices. This section proves Theorem 6 (ii) $\Rightarrow$ (iii).

First we note that G has a dominating edge. The following lemma by Cozzens and Kelleher [19] is crucial for this.

Lemma 7 (Cozzens and Kelleher (1990))**.** *Let G be a connected graph with two or more vertices. If G contains no induced P_5, C_5 or 3-spider with thin legs, it has a dominating edge.*

From this it follows immediately that

Corollary 8. *G has a dominating edge.*

Proof. By assumption, G has no induced P_5, C_5 or triangle. G also cannot contain an induced 3-spider with thin legs, as this graph contains a triangle. By Lemma 7, G has a dominating edge. □

From now on, let x_1 and x_2 denote the two vertices of a dominating edge x_1x_2 of G. From Corollary 8 we draw our first claim about the structure of G.

Corollary 9. *Every vertex in G is adjacent to either x_1 or x_2 but not both.*

Proof. This is obviously true for x_1 and x_2, as they are adjacent to each other and not to themselves. Let v be any other vertex in G. As x_1x_2 is a dominating edge, v must be adjacent to x_1 or x_2. If v were adjacent to both x_1 and x_2, vertices v, x_1, x_2 would induce a triangle in G, in contradiction to our assumption that G be triangle-free. □

Next we see that

Claim 10. *G is bipartite.*

Proof. By assumption, G contains no induced C_3 or C_5. G also contains no induced cycle of size ≥ 7, as these cycles contain a forbidden induced P_5 subgraph. As G is thus odd-cycle-free, we have that G is bipartite (cf. page 6). □

We propose the following proper vertex colouring of G. Colour x_1 in red and x_2 in blue. Next colour all vertices adjacent to x_2 in red and all vertices adjacent to x_1 in blue. By Corollary 9, all vertices of G are now coloured in exactly one colour. We denote the two disjoint vertex sets of colour red and blue by V_1 and V_2, respectively.

Next we note that V_1 and V_2 must be *almost* completely connected. By this we mean that no more than one edge incident with a vertex in V_1 and a vertex in V_2 may be absent.

Claim 11. *There is at most one non-adjacent pair of vertices of opposite colours in G.*

Proof. Let $(v, w) \in V_1 \times V_2$ be a non-adjacent pair of vertices with opposite colours. By Corollary 9, we have $v \neq x_1$ and $w \neq x_2$. Assume that a second non-adjacent pair of vertices (v^*, w^*) with opposite colours exists. Again we have $v^* \neq x_1$ and $w^* \neq x_2$. We distinguish between two possibilities.

First assume that $v = v^*$. As the two pairs are not identical, it follows that $w \neq w*$. By assumption, v is neither adjacent to w nor w^*. It follows that vertices v, w, w^*, x_1, x_2 induce a chair.

Now assume that $v \neq v^*$ and $w \neq w^*$. By the above, it follows that v must be adjacent to w^* and v^* must be adjacent to w. Then vertices v, v^*, w, w^*, x_1 induce a forbidden P_5. □

Summing up Claims 10 and 11, we can say that

Lemma 12. *G is a $K_{m,n}$ with $m, n \geq 1$ or G is a $K_{m,n} - e$ with $m, n \geq 2$, where e is an edge.*

Proof. By assumption, G is connected and has two or more vertices. By Claim 10, G is bipartite. If G contains no non-adjacent pair of vertices of opposite colours, we have $G = K_{m,n}$ with $m, n \geq 1$. Now assume that G contains a non-adjacent pair of vertices $(v, w) \in V_1 \times V_2$. By Claim 9, we have $v \neq x_1$ and $w \neq x_2$, so that $|V_1|, |V_2| \geq 2$. From Claim 11 we have that G cannot have any other non-adjacent pair of vertices of opposite colours. It follows that $G = K_{m,n} - e$ and $m, n \geq 2$. □

4.3 Strategies for Alice

This section proves Theorem 6 (iii) ⇒ (i).

Proof. We prove that G is g_B-perfect by showing that every induced subgraph of G is g_B-nice. Note first that every induced subgraph of a complete bipartite graph is again complete bipartite and every induced subgraph of an almost complete bipartite graph is either complete bipartite or almost complete bipartite. Let H be an induced subgraph $H = K_{m,n}$ or $H = K_{m,n} - e$ of G. If one side of the bipartition is empty, the graph consists only of independent vertices and Alice wins without making a single move. If one side of the bipartition consists of a single vertex a, Alice makes sure that a is coloured after the first round and wins. We can now assume that $m, n \geq 2$ and provide strategies for Alice to win with two colours.

Assume $H = K_{m,n}$ with $m, n \geq 2$: As a bipartite graph, H has the vertex partition sets V_1 and V_2. If Bob colours a vertex from V_1 in red, Alice colours any vertex in V_2 in blue. Bob's move neutralises all other vertices in V_1, as V_2 now only admits the colour blue. For the same reason, Alice's move neutralises all other vertices in V_2 and she wins.

Assume $H = K_{m,n} - e$ with $m, n \geq 2$: Let v, w be the pair of vertices incident with e. If Bob starts by colouring a vertex in V_1 adjacent to all vertices in V_2, Alice responds by colouring a vertex in V_2 that is adjacent to all vertices in V_1. Such a vertex exists because $m, n \geq 2$. As above, this neutralises all vertices in V_1 and V_2.

Bob might instead start by colouring the vertex v. Alice responds by colouring the other vertex w in the other colour. As v is adjacent to all vertices V_2 apart from w, and w is adjacent to all vertices V_1 apart from v, this neutralises all other vertices in G.

□

Chapter 5

Characterising g_B-perfect graphs

We now turn to the main result of this thesis, the characterisation of g_B-perfect graphs. In [8], a characterisation of non-connected g_B-perfect graphs is provided. Our result extends this to connected g_B-perfected graph also.

Theorem 13 (Andres (2012))**.** *Let G be a graph with two or more components. Then the following are equivalent.*

(i) *G is g_B-perfect.*

(ii) *G contains no induced P_4, C_4, split 3-star or double fan.*

(iii) *G consists of two or more E_1 components. The graph class E_1 is described in Section 5.3 and shown in Figure 5.1.*

5.1 The main result

Now we are ready to state the main result.

Theorem 14. *Let G be a graph. Then the following are equivalent.*

(i) *G is g_B-perfect.*

(ii) *G contains no induced $F_1, \ldots, F_{15}$ from Figure 3.1.*

(iii) *G is an instance of $E_1^{\cup}, E_2, \ldots, E_{13}$ defined in Section 5.3 and shown in Figure 5.1.*

Proof. If G has order ≤ 1, the proof is elementary. From now on assume that G has two or more vertices.

(i) $\Rightarrow$ (ii): This is Theorem 5 in Chapter 3.

(ii) $\Rightarrow$ (iii): Assume that G contains no induced $F_1, \ldots, F_{15}$. Then by Lemma 15 in Section 5.4, G is an instance of $E_1^{\cup}, E_2, \ldots, E_{13}$.

(iii) $\Rightarrow$ (i): By assumption, G is an instance of $E_1^{\cup}, E_2, \ldots, E_{13}$. Let H be an induced subgraph of G. With the help of Lemma 59 in Section 5.5 we have that H is also an instance of $E_1^{\cup}, E_2, \ldots, E_{13}$. Then by Lemma 73 in Section 5.6, H is g_B-nice. It follows immediately that G is g_B-perfect. □

5.2 A key to the graph depictions throughout this chapter

We present a concise key to the graph depictions found in Figures 5.1 to 5.11.

A large circle denotes a complete subgraph. Its order is ≥ 0 unless otherwise specified. A small dot denotes a single vertex, generally labelled with a lowercase letter. If the vertex is solid, it must be present in any graph instance of that structure. A dashed vertex together with its incident dashed edges, as in E_7 to E_9, indicates optional vertices. If a dashed vertex exists, its incident dashed edges must also exist. A dotted edge, as in E_8, may be omitted even if its incident vertices are both present.

The complete subgraphs denoted by large circles are always either completely connected to another vertex or complete subgraph or completely disconnected. If two complete subgraphs denoted by large circles are completely connected, five lines are drawn between them. If a complete subgraph denoted by a large circle is completely connected to a vertex, the circle is graphically connected to the vertex by three lines.

Take E_1, for example. The vertex x is solid and must be present in every instance of E_1. All other vertices are in one of the complete subgraphs $K_a, K_b, K_c, H_1, \ldots, H_k$. As x is completely connected to each of these subgraphs, x is adjacent to every vertex in G apart from itself and it follows that x is a dominating vertex. Lastly we see that K_b and K_c are completely disconnected from each other but both completely connected to K_a.

5.3 The graph classes E_1 to E_{13}

The class E_1 consists of all graphs of shape $K_1 \vee (H_0 \cup H_1 \cup \cdots \cup H_k)$, where $k \geq 0$ and the $H_1, \ldots H_k$ are non-null complete graphs, and where H_0 is either empty or there exist $a, b, c \geq 1$ such that $H_0 = K_a \vee (K_b \cup K_c)$. We can assume that $b \geq c$. If G is an instance of E_1, the clique number of G is $\omega(G) = \max\{a + b, |H_1|, \ldots, |H_k|\} + 1$.

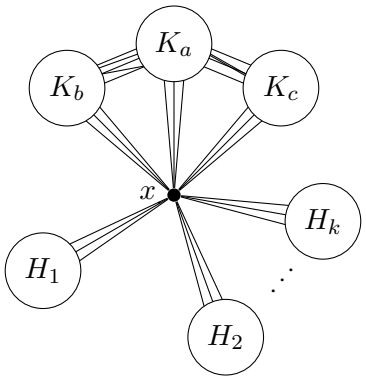
The class $E_1^{\cup}$ consists of all graphs that are instances of $E_1 \cup \ldots \cup E_1$, i.e. all graphs whose components are all instances of E_1.

The class E_2 contains all graphs obtained by taking the graph $K_r \vee (K_b \cup K_c)$, where $r \geq 2$ and $b, c \geq 1$, and adding a pendant d to a vertex $x_1 \in K_r$ and a pendant e to a different vertex $x_2 \in K_r$. $K_a := K_r - \{x_1, x_2\}$ is drawn separately from x_1 and x_2 in Figure 5.1. Due to symmetry, we can assume that $b \geq c$. For every instance G of E_2, the clique number of G is $\omega(G) = a + b + 2$.

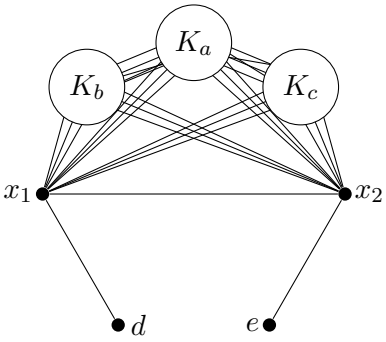
The class E_3 contains all graphs created by taking the graph $(K_m \cup b) \vee (K_r \cup a)$, where $m \geq 1$ and $r \geq 2$, and adding an additional vertex d that is adjacent to a and a single vertex c in K_r. We define $K_n := K_r - c$ and draw K_n and c separately in Figure 5.1. The clique number of an instance G of E_3 is $\omega(G) = m + n + 1$.

The class E_4 constitutes all graphs that consist of two complete subgraphs A and V partially connected in a specific way. Let $A = A_R, A_1, \ldots, A_k$ be a partition of A and $V = V_R, V_1, \ldots, V_k$ be a partition of V for some $k \geq 1$, where A_R and/or V_R may be empty. For every $1 \leq i \leq k$, A_i and V_i are completely connected, A_i is not adjacent to any other vertices in V and V_i is not adjacent any other vertices in A. Lastly, A_R is not adjacent to any vertices in V and V_R is not adjacent to any vertices in A.

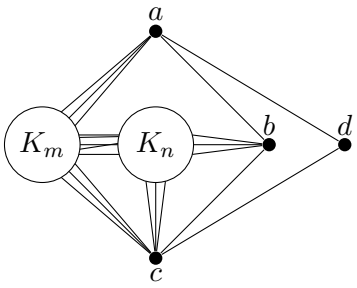
We can assume that $|A| \geq |B|$ and that $|A_1| + |V_1| \geq |A_i| + |V_i|$ for all $1 \leq i \leq k$, relabelling if necessary. The clique number of an instance G of E_4 is either $\omega(G) = |A_1| + |V_1|$ or $\omega(G) = |A|$, depending on which is larger.



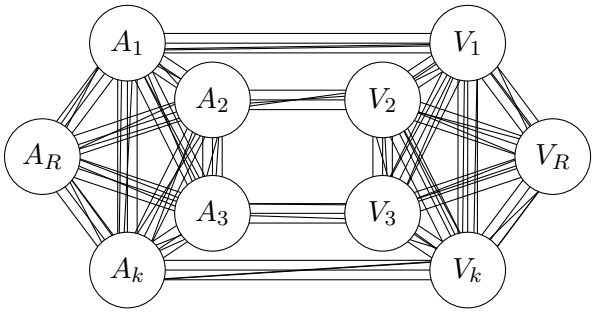
E_1 : $k \geq 0$, $a = b = c = 0$ or $a \geq 1, b \geq c \geq 1$



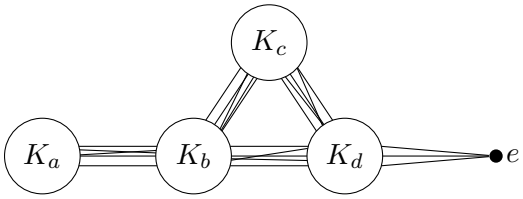
$E_2 : a \geq 0$ and $b \geq c \geq 1$



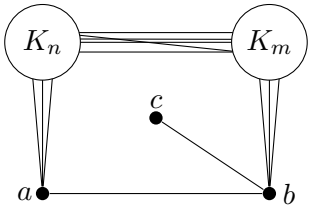
$E_3 : m, n \geq 1$



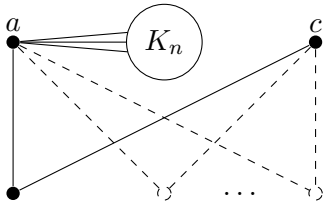
$E_4 : k \geq 1$



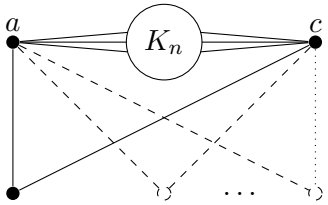
$E_5 : a, b, c, d \geq 1$



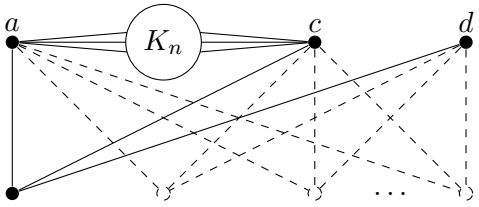
$E_6 : m, n \geq 2$



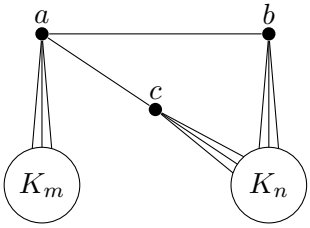
$E_7 : n \geq 2$



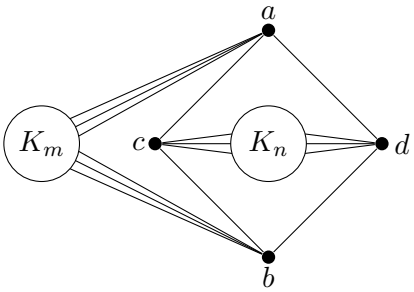
$E_8 : n \geq 2$



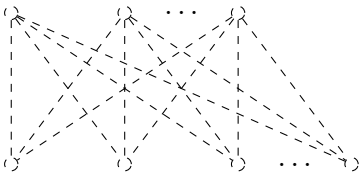
$E_9 : n \geq 2$



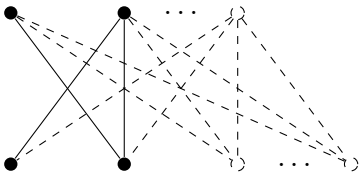
$E_{10} : m, n \geq 1$



$E_{11} : m, n \geq 1$



$E_{12} : K_{m,n}$ with $m, n \geq 0$



$E_{13} : K_{m,n} - e$ with $m, n \geq 2$

Figure 5.1: The 13 structural possibilities for connected g_B-perfect graphs.

The class E_5. We construct instances of E_5 by taking a complete subgraph $K_b \vee K_c \vee K_d$ with $b, c, d \geq 1$, completely connecting a complete subgraph K_a to the subgraph K_b and completely connecting a vertex e to the subgraph K_d. The clique number of any instance G is $\omega(G) = \max\{a, c+d\} + b$.

The class E_6 consists of the graphs constructed by joining a complete graph $K_n \vee K_m, m, n \geq 2$, with a P_3 such that one pendant a of P_3 is completely connected to K_n, the middle vertex b is completely connected to K_m and the other pendant c is only adjacent to c. The clique number of any instance G of E_6 is $\omega(G) = m + n$.

The class E_7 contains all graphs obtained by taking a complete bipartite graph $K_{2,m}, m \geq 1$ and completely connecting a complete graph $K_n, n \geq 2$ to one of two vertices that constitute one side of the bipartition. The clique number of an instance G is $\omega(G) = n + 1 \geq 3$.

The class E_8 contains all graphs obtained by taking either a complete bipartite graph $K_{2,m}$ or an almost bipartite graph $K_{2,m} - e$ with $m \geq 1$ and completely connecting a complete graph $K_n, n \geq 2$ to both of the two vertices that constitute one side of the bipartition. The edge e that might or might not be present is represented by the dotted line in Figure 5.1. For any instance G of E_8, the clique number of G is $\omega(G) = n + 1 \geq 3$.

The class E_9 contains all graphs obtained by taking a complete bipartite graph $K_{3,m}, m \geq 1$ and completely connecting a complete graph $K_n, n \geq 2$ to two of the three vertices that constitute one side of the $K_{3,m}$. Let G be an instance of E_9. Then its clique number is G is $\omega(G) = n + 1$.

The class E_{10} constitutes all graphs obtained by completely connecting the two pendants of a P_3 to one complete graph K_n and completely connecting the inner vertex of the P_3 to another complete graph K_m, where $m, n \geq 1$. The clique number of an instance G of E_{10} is $\omega(G) = \max\{m, n\} + 1$.

The class E_{11} contains all graphs obtained by completely connecting two non-adjacent vertices in a C_4 to one complete graph K_m and completely connecting the other two non-adjacent vertices to another complete graph K_n. Due to symmetry we can assume that $m \geq n$. If G is an instance of E_{11}, the clique number of G is $\omega(G) = m + 1$.

The classes E_{12} and E_{13} correspond to the complete bipartite graphs and the almost complete bipartite graphs (cf. page 6), respectively.

5.4 The 13 possible structures of g_B-perfect graphs

This section proves the following lemma.

Lemma 15. *A graph G of order ≥ 2 that contains no induced $F_1, \ldots, F_{15}$ is an instance of $E_1^{\cup}$ with one or more components or an instance of $E_2, \ldots, E_{13}$.*

Proof. Let G have two or more components. We first note that G cannot have an induced C_4 or P_4. If a component of G had an induced P_4 or C_4, G would have an $F_1 = P_4 \cup K_1$ or $F_3 = C_4 \cup K_1$, induced by the vertices of the P_4 or C_4 and any vertex in another component. By assumption, G also cannot have a split 3-star (F_4) or a double fan (F_7). By Theorem 13, G then is an instance of $E_1^{\cup}$ with two or more components.

Therefore it suffices from now on to study connected graphs. Throughout the rest of this section, let G be a connected graph of order ≥ 2 without forbidden induced subgraphs $F_1, \ldots, F_{15}$.

The proof proceeds along the following lines. First we note in 5.4.1 that G admits a decomposition that allows us to partition G into a dominating edge and three subgraphs G_1, G_2 and G_3. Then

we study the structures of G_1, G_2 and G_3 in sections 5.4.2, 5.4.3 and the relationship between them in 5.4.4 and 5.4.5. The remainder of the proof from 5.4.6 to 5.4.11 consists of a series of case distinctions that address every structural possibility for G, as outlined in Figure 5.2. Each leaf case culminates in one or more lemmas that identify the graph G as an instance of $E_1, \ldots, E_{13}$.

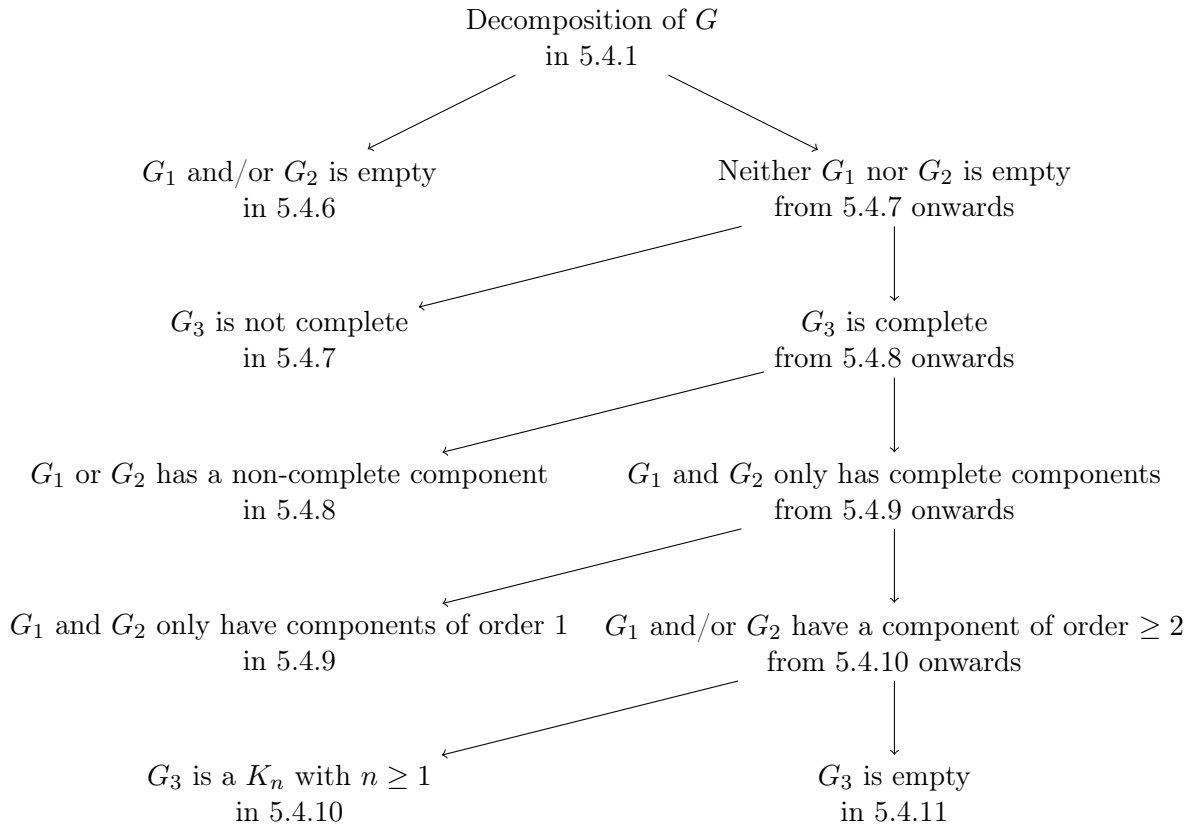


Figure 5.2: The nested case distinctions made throughout the proof.

5.4.1 The decomposition of G into a dominating edge and G_1, G_2, G_3

The following lemma from Cozzens and Kelleher [19] already used in Chapter 4 is also central to this proof, as it provides the key for our decomposition.

Lemma 16 (Cozzens and Kelleher (1990))**.** *Let G be a connected graph with two or more vertices. If G contains no induced P_5, C_5 or 3-spider with thin legs, it has a dominating edge.*

From this it follows immediately that

Corollary 17. *G has a dominating edge.*

Proof. By assumption, G has no induced F_1, P_5 or C_5. G cannot have an induced 3-spider with thin legs because if it did, it would also have an induced F_1 obtained by removing a vertex of degree 3 from the 3-spider. Hence by Lemma 16, G has a dominating edge. □

From now on, let x_1 and x_2 be the vertices of a dominating edge x_1x_2 in G. Using the existence of this dominating edge, we define subgraphs G_1, G_2 and G_3.

Definition 18. G_1 is the subgraph of G induced by all vertices in $V(G) - x_2$ adjacent to x_1 and not to x_2. G_2 is the subgraph of G induced by all vertices in $V(G) - x_1$ adjacent to x_2 and not to x_1. G_3 is the subgraph of G induced by all vertices adjacent to both x_1 and x_2.

The structures of G_1, G_2 and G_3 and the relationships between the three subgraphs obey certain rules that we investigate below. Note that by renaming x_1 to x_2, G_1 becomes G_2 and vice versa. In particular this means that all statements and proofs in 5.4.2, 5.4.3, 5.4.4 and 5.4.5 concerning G_1 and G_2 are valid also when exchanging G_1 for G_2 and vice versa.

Before we start analysing the structures of G_1, G_2 and G_3 separately, we introduce a structural lemma that is useful for understanding G_1, G_2 and G_3.

Lemma 19. *A connected, non-complete graph G that has no induced P_4, C_4, claw, split 3-star or double fan takes the shape $G = K_a \vee (K_b \cup K_c)$, where $a, b, c \geq 1$.*

Proof. Because G has no induced P_4, C_4, split 3-star or double fan, we can apply Theorem 3 from [8] which states that G has the structure $G = K_1 \vee (H_0 \cup H_1 \cup \cdots \cup H_k)$, where $k \geq 0$ and the $H_1, \ldots H_k$ are non-null complete graphs, and where H_0 is either empty or there exist $p, q, r \geq 1$ such that $H_0 = K_r \vee (K_p \cup K_q)$.

Next we make use of the fact that G cannot contain an induced S_3. If H_0 is empty, then G must have the shape $G = K_1 \vee (H_1 \cup H_2)$. It cannot have a third subgraph H_3 as the dominating K_1 together with a single vertex each from H_1, H_2 and H_3 would otherwise induce an S_3. Neither H_1 nor H_2 can be empty, as that would mean G is complete. We define $K_a := K_1, K_b := H_1$ and $K_c := H_2$ and have $G = K_a \vee (K_b \cup K_c)$.

If, on the other hand, H_0 is not empty, then G consists entirely of the H_0 and the dominating K_1, i.e. $G = K_1 \vee H_0$: assume $G = K_1 \vee (H_0 \cup H_1)$, where H_1 is not empty. Then any three vertices from K_p, K_q and H_1 together with the dominating K_1 would induce an S_3. We define $K_a := K_r \vee K_1$, $K_b := K_p$ and $K_c := K_q$ and have $G = K_a \vee (K_b \cup K_c)$. □

5.4.2 The structure of subgraphs G_1 and G_2

Recall that the subgraph G_1 is induced by all vertices in $G - x_1$ adjacent to x_1 and the subgraph G_2 is defined analogously. G_1 and G_2 need not be connected but can consist of multiple components. Of these components, we see below that no more than one can be non-complete. This non-complete component N has a specific structure.

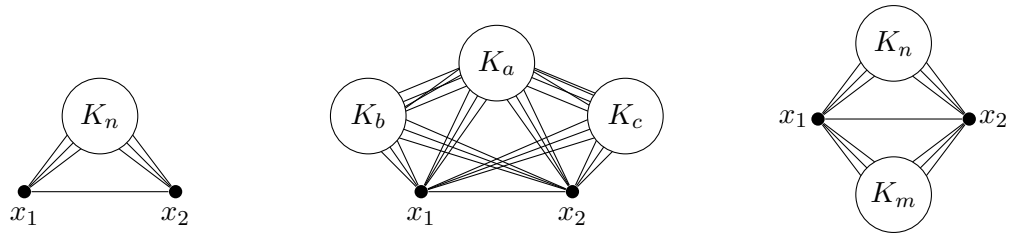
Claim 20. *G_1 and G_2 have at most one non-complete component N each. The structure of N is $N = K_a \vee (K_b \cup K_c)$ with $a, b, c \geq 1$.*

Proof. Assume that G_1 has two non-complete components. Each component contains an induced P_3. The two P_3 together with x_1 induce a double fan.

Next we examine the structure of N. First we note that N cannot contain the following induced subgraphs:

- a P_4: if N contained a P_4, it would induce an $F_1 = P_4 \vee K_1$ together with x_2.
- a C_4: if N contained a C_4, it would induce an $F_3 = C_4 \vee K_1$ together with x_2.
- an S_3: if N contained an S_3, it would induce a split 3-star together with x_1.
- any of the $F_1, \ldots, F_{15}$, by definition.

Then by Lemma 19 we have $N = K_a \vee (K_b \cup K_c), a, b, c \geq 1$. □



(a) $n \geq 1$ (b) $a, b, c \geq 1$ (c) $m, n \geq 1$

Figure 5.3: The three non-empty shapes of G_3, attached to the dominating edge x_1x_2.

5.4.3 The structure of subgraph G_3

G_3 is the subgraph induced by all vertices adjacent to both x_1 and x_2. In the following, we determine that G_3 is either empty, connected or composed of two components. In the case that G_3 is not empty, we can describe the single or the two components in greater detail.

Claim 21. *G_3 is either empty, connected or composed of two components.*

Proof. Assume that G_3 has three or more components and let a, b, c be three vertices from three different components. By assumption, a, b, c are independent and each adjacent to x_1 and x_2. It follows that a, b, c, x_1, x_2 induce a split 3-star. □

We first investigate the case where G_3 has two components and then the case where G_3 is connected.

Claim 22. *If G_3 has two components, they are both complete subgraphs.*

Proof. Let G_3 have two components. If one of these components is not complete, there exist two vertices y_1 and y_2 in this component that are not adjacent. Let y_3 be any vertex in the other component. Then x_1, x_2, y_1, y_2, y_3 induce a forbidden split 3-star. This shows that both components must be complete. □

If G_3 is connected, it can take one of two shapes.

Claim 23. *If G_3 is connected, G_3 is either complete or a graph of shape $G_3 = K_a \vee (K_b \cup K_c)$ with $a, b, c \geq 1$.*

Proof. If G_3 is complete, we are done. Thus we assume that G_3 is not complete. By assumption, G_3 cannot have an induced split 3-star or double fan. G_3 also cannot have an induced S_3, P_4 or C_4, as this induced subgraph would induce a split 3-star, 4-fan or 4-wheel together with x_1. By Lemma 19, we have $G_3 = K_a \vee (K_b \cup K_c)$ with $a, b, c \geq 1$. □

In summary, we can say that G_3 is either complete, made of two complete components or $G_3 = K_a \vee (K_b \cup K_c)$ with $a, b, c \geq 1$ as shown in Figure 5.3. Of course the graph in (b) is the same as the graph in (c) if we allow for $a = 0$.

5.4.4 The relationship between components of G_1 and G_2

We have seen that G_1 and G_2 can consist of multiple complete components and no more than one non-complete component each. We study how the components of G_1 can be connected to the components of G_2.

The first claim about the relationship between G_1 and G_2 yields that the two subgraphs must be *almost* completely connected in a very specific way. This claim occupies a central position in our approach and will be used throughout the proof.

Claim 24. *With the exception of one pair of components (X, Y), where X is a component of G_1 and Y is a component of G_2, all components of G_1 must be completely connected to all components of G_2. (X, Y) may be completely, partially or not at all connected.*

Proof. Let X and X' be two components of G_1. Assume that neither X nor X' is completely connected to G_2. Then there exists a vertex $a \in X$ not adjacent to $v \in G_2$ and a vertex $b \in X'$ not adjacent to $w \in G_2$. If we have $v = w$, then vertices a, b, v, x_1, x_2 induce a forbidden chair.

If we have $v \neq w$, we distinguish between two cases due to symmetry. If there is no edge between a and w, vertices a, b, w, x_1, x_2 induce a chair. If the edges aw and bv both exist, vertices a, b, v, w, x_1 induce a P_5 or C_5, depending on whether v and w are adjacent in G_2.

Now let Y, Y' be two components of G_2. With the same argument as above, one of them must be completely connected to G_1. This concludes the proof. □

It is interesting to see what constraints the existence of a non-complete component of G_1 places on G_2.

Claim 25. *Let G_1 have a non-complete component $N = K_r \vee (K_p \cup K_q)$ with $p, q, r \geq 1$. If both $p, q \geq 2$, then G_2 is empty. Otherwise, if $p = 1$, G_2 has at most one vertex v. This vertex v must be completely connected to K_p and K_q and completely disconnected from K_r.*

Proof. Let v be a vertex in G_2. First we prove that v must be completely connected to K_p and K_q and completely disconnected from K_r. Let a, b, c denote a P_3 in G_1 with $a \in K_p$, $b \in K_r$ and edges ab, bc. If v is adjacent to neither a, b nor c, vertices a, c, v, x_1, x_2 induce a chair. If v is adjacent only to b, vertices a, b, c, v, x_2 induce a chair. If v is only adjacent to either a or c, the same vertices induce a P_5. If v is adjacent to a and b or to b and c, vertices a, b, c, v, x_1 induce a 4-fan. If a, b, c are all adjacent to v, the same vertices induce a 4-wheel. This only leaves us with the possibility that v is adjacent to a and c and not adjacent to b. As a, b, c were chosen freely from K_p, K_r and K_q, it follows that v must be adjacent to all vertices in K_p and K_q and not adjacent to any vertices in K_r.

Next we see that G_2 can only contain one vertex. Assume G_2 has two vertices v and w. By the first part, they must both be adjacent to a and c and not adjacent to b. If v and w are not adjacent, vertices b, v, w, x_1, x_2 induce a chair. Otherwise, if v and w are adjacent, vertices a, c, v, w, x_2 induce a split 3-star.

Lastly we prove that G_2 must be empty if $p, q \geq 2$. Let a_1, a_2 be vertices in K_p and c_1, c_2 be vertices in K_q. Assume G_2 contains a vertex v. By the above, v must be adjacent to a_1, a_2, c_1 and c_2. Then vertices $a_1, a_2, c_1, c_2, v, x_1, x_2$ induce an F_{14}. □

In Claim 25 we showed that G_2 can have at most one vertex if G_1 contains a non-complete component. If we go one step further and require that G_2 contains a vertex, we see that G_1 consists entirely of the non-complete component $G_1 = K_r \vee (K_1 \cup K_p)$ with $p, r \geq 1$.

Claim 26. *Let G_1 have a non-complete component $N = K_r \vee (K_1 \cup K_p)$ with $p, r \geq 1$ and let G_2 not be empty. Then G_1 consists entirely of $G_1 = K_r \vee (K_p \cup K_1)$ and G_2 consists of a single vertex.*

Proof. Let a, b, c be a P_3 in N with edges ab, bc and let v be a vertex in G_2. By Claim 25, G_2 consists entirely of v, which must be adjacent to a, c and cannot be adjacent to b. Let d be a

vertex from another component in G_1. By Claim 24, vertices d and v must be adjacent. Then vertices a, b, d, v, x_2 induce a chair. □

The next two claims are useful in situations where we know that neither G_1 nor G_2 is empty.

Claim 27. *Let neither G_1 nor G_2 be empty. Then G_1 and G_2 can each have no more than one component of order ≥ 2.*

Proof. Assume G_1 has at least two components of order ≥ 2 and let v be a vertex of G_2. We choose two adjacent vertices from each of the two components in G_1 and denote them by a, b and c, d, respectively. By Claim 24, we know that one of the two components must be completely connected to v; we assume that a and b are both adjacent to v. Now we distinguish between three possibilities due to isomorphism. If v is adjacent to neither c nor d, vertices a, b, c, d, v, x_1, x_2 induce an F_{12}. If v is only adjacent to c and not to d, vertices a, c, d, v, x_2 induce a chair. If v is adjacent to both c and d, vertices a, b, c, d, v, x_1, x_2 induce an F_{14}. □

Claim 28. *If G_1 has a complete component A of order ≥ 2, G_2 has at most two components. Conversely, if G_2 has a complete component V of order ≥ 2, G_1 has at most two components.*

Proof. Let G_1 have a complete component A of order ≥ 2 and assume that G_2 has three vertices from three components. By Claim 24, two of the three vertices must be completely connected to A. Then these two vertices together with any two vertices from A and x_1 induce a split 3-star. □

The following important structural result holds if both G_1 and G_2 are complete subgraphs.

Claim 29. *Let G_1 and G_2 each be non-empty complete subgraphs. For a fixed $k \in \mathbb{N}$, G_1 and G_2 can be partitioned into subgraphs $G_1 = A_R, A_1, \ldots, A_k$ and $G_2 = V_R, V_1, \ldots, V_k$ such that for each $i = 1, \ldots, k$ the subgraph A_i is completely connected to V_i and completely disconnected from $V_j, j \neq i$, while A_R and V_R contain all the remaining vertices of G_1 and G_2 that are not adjacent to any vertices in G_2 and G_1, respectively. Graph E_4 in Figure 5.1 illustrates this relationship between G_1 and G_2.*

Proof. If both G_1 and G_2 consist of a single vertex, the claim is trivially true. Assume that G_1 is a complete graph of order ≥ 2 that contains vertices a and b.

First we show that if a is adjacent to a subset S_a of vertices in G_2, b is either adjacent to all of S_a or not adjacent to any vertices in S_a. Assume that a is adjacent to all vertices in S_a and b is only partially connected to S_a. Then there exists a vertex $v \in G_2$ adjacent to both a and b as well as a vertex $w \in G_2$ adjacent only to a. As G_2 is complete, vertices v and w are adjacent and it follows that vertices a, b, v, w, x_1 induce a 4-fan.

This allows us to partition G_2 into disjoint subgraphs $G_2 = V_R, V_1, \ldots, V_k$ as follows. V_R is the subgraph of G_2 induced by all vertices not adjacent to G_1 and each subgraph $V_i, 1 \leq i \leq k$, corresponds to a subgraph induced by the set of vertices S_a in G_2 adjacent to a vertex a in G_1.

Now that we have defined a partition of G_2, we can use it to partition G_1 too. First we note that every vertex a in G_1 can only be adjacent to at most one subgraph $V_i, 1 \leq i \leq k$ with the first part of this proof. We define by A_i the subgraph induced by the subset of all vertices in G_1 adjacent to V_i. Lastly, we define A_R as the subgraph of G_1 induced by all vertices not adjacent to any vertex in G_2. Note that A_R and/or V_R may be empty and $k \geq 1$. □

5.4.5 The relationship between (G_1 or G_2) and G_3

Now that we have seen how different configurations of G_1 can be connected to G_2 and vice versa regardless of the shape of G_3, we investigate how the existence and configuration of a non-empty G_3 constrains the other two subgraphs.

If G_3 is not complete, (see (b) and (c) in Figure 5.3), the case is simple.

Claim 30. *Let G_3 not be complete. Then G_3 is not connected to any vertices in G_1 and G_2. Further, neither G_1 nor G_2 can contain a non-complete component N.*

Proof. Let y_1 and y_2 be two non-adjacent vertices in G_3. These must exist as G_3 is not complete. Let a be a vertex in G_1 or G_2. If a is adjacent to y_1 alone, vertices a, x_1, x_2, y_1, y_2 induce a 4-fan. If a is adjacent to y_1 and y_2, the same vertices induce a 4-wheel. As a, y_1, y_2 were freely chosen from their respective subgraphs, this shows that a vertex in G_3 that is not adjacent to all other vertices in G_3 cannot be connected to G_1 or G_2.

Now assume that $y_3 \in G_3$ is a vertex adjacent to all other vertices in G_3. (Such a vertex need not exist.) If a is adjacent to y_3, vertices a, x_1, y_1, y_2, y_3 induce a split 3-star. This shows for all possible vertices in G_3 that they are not adjacent to any vertices in G_1 or G_2.

Lastly we note that neither G_1 nor G_2 can contain a non-complete component. Assume G_1 has a non-complete component N. We know that N must contain an induced P_3. As by the above, none of the vertices in P_3 can be adjacent to y_1 or y_2, the P_3 and vertices x_1, x_2, y_1, y_2 induce a double fan. □

If G_3 is a non-empty complete subgraph, the situation is a little more complex. We first assume that G_1 has a non-complete component.

Claim 31. *Let G_3 be a non-empty complete subgraph. If G_1 contains a non-complete component N, then G_1 and G_3 are completely disconnected from one another.*

Proof. First we show that the non-complete component $N = K_r \vee (K_p \cup K_q)$ of G_1 and G_3 are completely disconnected. We select three vertices $a \in K_p, b \in K_r$ and $c \in K_q$ with edges ab and bc. Let y be a vertex in G_3. If y is partially adjacent to a, b and c, we can assume that y is adjacent to a and not adjacent to b. Then vertices a, b, x_1, x_2, y induce a 4-fan. If y is adjacent to a, b and c, vertices a, c, x_1, x_2, y induce a split 3-star. As vertices a, b, c were chosen freely from K_p, K_r and K_q, it follows that N and G_3 must be completely disconnected.

Now let d be a vertex from another (complete) component in G_1 that is adjacent to $y \in G_3$. Then vertices a, b, c, d, x_1, x_2, y induce a double fan. This shows that G_1 and G_3 are completely disconnected. □

Claim 32. *Let G_3 be a non-empty complete subgraph. If G_1 has a non-complete component N, then G_2 must be empty.*

Proof. Let y be a vertex in G_3. We select $a \in K_p, b \in K_r$ and $c \in K_q$ from $N = K_r \vee (K_p \cup K_q)$. By Claim 31, we know that y is not adjacent to a, b or c. Assume that G_2 has a vertex v. By Claim 25, v must be adjacent to a, c and not adjacent to b. If v is not adjacent to y, vertices a, c, v, x_2, y induce a chair. If v is adjacent to y, then vertices a, b, c, v, x_1, x_2, y induce an F_{15}. □

From now on we assume that neither G_1 nor G_2 has a non-complete component. G_3 is still assumed to be complete and non-empty.

Claim 33. *Let G_3 be a non-empty complete subgraph and y be a vertex in G_3. A complete component of G_1 is either completely connected to or completely disconnected from y.*

Proof. Assume that a complete component A of G_1 is partially connected to a vertex y in G_3. Then there exist a vertex $a \in A$ adjacent to y and a vertex $b \in A$ not adjacent to y. It follows that vertices a, b, x_1, x_2, y induce a 4-fan. □

Claim 34. *Let G_3 be a non-empty complete subgraph. Only one component of G_1 can be connected to G_3. The same holds for G_2. If G_3 is connected to a component of G_1 and a component of G_2, the G_1 component must be connected to different vertices of G_3 than the G_2 component.*

Proof. First assume that a and b are vertices from two components of G_1 connected to G_3. If a and b are both adjacent to the same vertex y in G_3, vertices a, b, x_1, x_2, y induce a split 3-star. If a is adjacent to $y_1 \in G_3$ and b is adjacent to $y_2 \in G_3$, vertices a, b, x_1, y_1, y_2 induce a 4-fan.

Next assume that G_3 is connected to a component in G_1 and a component in G_2. Thus there exist two vertices $a \in G_1$ and $v \in G_2$ that are both adjacent to a vertex in G_3. If a and v are both adjacent to the same vertex y in G_3, vertices a, v, x_1, x_2, y induce a 4-fan or a 4-wheel, depending on whether the edge av is present or not. This proves the second part. □

Definition 35. Let G_3 be a non-empty complete subgraph. By Claim 34, exactly one component from each G_1 and G_2 may be connected to G_3. We denote the two components of G_1 and G_2 that are connected to G_3 by A and V, respectively. If no component in G_1 or G_2 exists that is connected to G_3, we can formally assume that A or V is empty. This allows us to partition G_3 into three complete subgraphs $G_3 = G_A \vee G_V \vee G_R$ as follows. Let G_A be the subgraph of G_3 induced by the vertices in G_3 that A is completely connected to. Next, let G_V be the subgraph of G_3 induced by the vertices in G_3 that V is completely connected to. Lastly, let G_R be the subgraph induced by all the remaining vertices in G_3 only adjacent to x_1 and x_2. If G_1 or G_2 are not connected to G_3, we formally define G_A or G_V as an empty subgraph, and if all vertices in G_3 are adjacent to either A or V, we define G_R as an empty subgraph.

If G_3 has a vertex not adjacent to any vertices in G_1 or G_2, (in other words, if G_R is not empty), we can say the following about G_1 and G_2.

Claim 36. *Assume that G_R is not empty. If G_1 and G_2 are both not empty then they are both complete.*

Proof. Let y be a vertex in G_R. By definition, y is not adjacent to any vertices in G_1 or G_2. By Claim 30 or Claim 32, we know that G_1 and G_2 only have complete components. Assume G_1 has two components and G_2 has at least one vertex v. We select two vertices a and b from the two components of G_1. If neither a nor b is adjacent to v, vertices a, b, v, x_1, x_2 induce a chair. If only one vertex is adjacent to v, vertices a, b, v, x_2, y induce an F_1. If both a and b are adjacent to v, the same vertices induce a chair. □

Claim 37. *Assume that G_R is not empty. If both G_1 and G_2 contain more than one vertex, G_1 and G_2 are completely connected to each other.*

Proof. Let y be a vertex in G_R. By Claim 36, we know that G_1 and G_2 are both complete. Let a, b be two vertices of G_1 and v, w be two vertices of G_2. We show that G_1 and G_2 must be completely connected. Due to isomorphism we distinguish between two cases only. If a, b and v, w are completely disconnected, vertices a, b, v, w, x_1, x_2, y induce an F_{11}. If a is adjacent to v and not adjacent to w, vertices a, v, w, x_1, y induce a P_5. □

If G_R is empty but G_3 is non-empty and complete, we can make the following two claims.

Claim 38. *If G_1 contains a component of order ≥ 2 and G_V is not empty, G_1 and G_2 each consist of a single component. The same holds if G_2 contains a component of order ≥ 2 and G_A is not empty.*

Proof. Let C denote the component of order ≥ 2 in G_1 with two vertices $a, b \in C$ and let y_V be a vertex in G_V. Because G_V is not empty, there must exist a vertex $v \in G_2$ adjacent to y_V. First we show that G_1 consists of a single component and is thus complete. Assume that c is a vertex from a second component in G_1. We distinguish between three cases. If C is not connected to v, c must be adjacent to v by Claim 24. Then vertices $a, b, c, v, x_1, x_2, y_V$ induce an F_{12}. Secondly, if C is partially connected to v, we can assume that a is adjacent to v and b is not. Because c and v must be adjacent, vertices a, b, c, v, y_V induce a chair. Lastly, if C and v are completely connected and c is adjacent to v, vertices $a, b, c, v, x_1, x_2, y_V$ induce an F_{14} while if c is not adjacent to v, the same vertices induce an F_{13}.

Now we show that G_2 also consists of a single component. Assume that w is a vertex from a second component in G_2. As y_V is adjacent to v, it cannot be adjacent to w by Claim 34. We show that the existence of w leads to a forbidden induced subgraph. Vertices a, b cannot both be adjacent to v and w, otherwise vertices a, b, v, w, x_1 would induce a split 3-star. We can assume that it is vertex a that is not adjacent to both v and w. If a is neither adjacent to v nor to w, vertices a, v, w, x_1, y_V induce an F_1. If a is adjacent to v only, the same vertices induce an F_3. If a is adjacent to w only, the same vertices induce a P_5. □

Claim 39. *If G_1 has a component of order ≥ 2, G_2 and G_3 are not empty but G_R is empty, then G_1 and G_2 each have at most two components.*

Proof. By Claim 28, G_2 has at most two components. We now show the same for G_1. Assume that G_1 has three components and let a, b, c be vertices from these three components. Let v be a vertex in G_2. By Claim 38, we have that G_V is empty, i.e. that $G_3 = G_A$. Let y_A be a vertex in G_A and w.l.o.g. let G_A be completely connected to a. By Claim 24, no more than one vertex of a, b, c can be non-adjacent to v. If b and c are adjacent to v, vertices b, c, v, x_2, y_A induce a chair. Otherwise we can assume that b is not adjacent to v. It follows that vertices a, b, v, x_2, y_A induce an F_3. □

5.4.6 The structure of G if G_1 or G_2 is empty

Now that we have studied the structures of G_1, G_2 and G_3 as well as the relationships between them, we start distinguishing cases that allow us to characterise G explicitly. We start with the case that either G_1 or G_2 is empty. W.l.o.g. we can assume this to be G_2. Our first structural result is to show that if G_2 is empty, G is an instance of graph E_1.

Lemma 40. *Let G_2 be empty. Then G is an instance of E_1 in Figure 5.1.*

Proof. Our aim is to show that $G = K_1 \vee (H_0 \cup H_1 \cup \cdots \cup H_k)$, where $k \geq 0$ and the $H_1, \ldots, H_k$ are non-empty complete graphs, and where H_0 is either empty or there exist $a, b, c \geq 1$ such that $H_0 = K_a \vee (K_b \cup K_c)$. We investigate the structure of G, dealing separately with the three cases that G_3 is empty, a non-empty complete subgraph and a non-complete subgraph.

If G_3 is empty, vertex x_1 is a dominating edge in G. Structurally, G is then made up of this dominating vertex x_1 completely connected to all the components of G_1 and adjacent to x_2. From Claim 20 we see that the only possible non-complete component N of G_1 matches the subgraph $H_0 = K_a \vee (K_b \cup K_c)$ of E_1 in Figure 5.1. It follows that G is an instance of E_1.

If G_3 is not complete, then by Claim 30, G_1 can only contain complete components and these components are completely disconnected from G_3. By Claims 22 and 23, we have $G_3 = K_r \vee (K_p \cup K_q)$

with $r \geq 0, p, q \geq 1$. G_3 and x_2 constitutes the subgraph $H_0 := (K_{r+1}) \vee (K_p \cup K_q)$ completely connected to x_1 and we have that G is an instance of E_1.

From now on we assume that G_3 is a non-empty complete subgraph. By Claim 20, G_1 can have at most one non-complete component N and any number of complete components. We differentiate between the two cases that G_1 has a non-complete component and that it does not.

Assume G_1 has a non-complete component. By Claim 31, G_1 and G_3 are completely disconnected. G_3 and x_2 together constitute a complete subgraph H_1. As G_1 only contains complete components apart from N, G is an instance of E_1.

Now assume that G_1 contains only complete components. By Claim 34, no more than one of these components can be connected to G_3. If none of the components of G_1 are connected to G_3, then G_3 and x_2 constitute a complete subgraph of G and G is an instance of E_1 with an empty H_0. Otherwise let A be the complete component of G_1 that is completely connected to a subset of vertices in G_3. We use the notation introduced in Definition 35 to denote by G_A the subgraph of G_3 completely connected to A and by G_R the subgraph $G_3 \setminus G_A$. Note that G_R can be empty if A is adjacent to all of G_3. We take a closer look at the subgraph induced by the x_2 and the vertices of A and G_3. Specifically, we define K_a, K_b and K_c as follows. $K_a := G_A, K_b := A$ and $K_c := G_R \vee x_2$. Then the subgraph induced by x_2 and the vertices of A and G_3 has the structure $H_0 := K_a \vee (K_b \cup K_c)$. As none of the components of G_1 apart from A are connected to G_3, we have that G is an instance of E_1. □

5.4.7 The structure of G if G_1 and G_2 are non-empty and G_3 is not complete

From now on we assume that neither G_1 nor G_2 is empty. We first deal with the straight-forward case that G_3 is not complete. We know that G_3 must contain at least two non-adjacent vertices y_1 and y_2. This allows us to describe G explicitly.

Lemma 41. *Let neither G_1 nor G_2 be empty. If G_3 is not complete, G_1 and G_2 each consist of a single vertex of degree 1. It follows that G is an instance of E_2.*

Proof. Assume, say, that G_2 contains more than one vertex. Let a be a vertex in G_1 and let v and$w$ be vertices in G_2. Further, let y_1 and y_2 be two non-adjacent vertices in G_3. By Claim 30, we have that a, v and w are not adjacent to y_1 or y_2. We first show that a cannot be adjacent to v or w. If a and v were adjacent, vertices a, v, x_1, y_1, y_2 would induce a chair. Now we need only distinguish between two cases. If v and w are not adjacent, vertices a, v, w, x_1, x_2 induce a chair. If v and w are adjacent, vertices $a, v, w, x_1, x_2, y_1, y_2$ induce an F_{10}. This shows that neither G_1 nor G_2 can have more than one vertex each and that these two vertices cannot be adjacent. As one can see by taking graphs (b) and (c) in Figure 5.3 and attaching a pendant each to x_1 and x_2, G is an instance of E_2. Note that in case (c) the subgraph K_a in E_2 is empty. □

5.4.8 The structure of G if G_1 has a non-complete component, G_2 is not empty and G_3 is complete

Having concluded the case that G_3 is not complete, we can assume from now on that G_3 is complete. We distinguish between the case that G_1 and/or G_2 have a non-complete component N and the case that neither G_1 nor G_2 has a non-complete component. We assume in this subsection that G_1 has a non-complete component. Then G admits the following structural description.

Lemma 42. *If G_1 has a non-complete component N and G_2 is not empty, G_1 takes the shape $N = K_a \vee (K_1 \cup K_b)$ with $a, b \geq 1$, G_2 consists of a single vertex and G_3 is empty. In other words, G is an instance of E_3.*

Proof. By Claim 32, G_3 is empty. The structure of G follows immediately from Claim 26. □

5.4.9 The structure of G if G_1 and G_2 are non-empty and only have K_1 components and G_3 is complete

We assume from now on that all components in G_1 and G_2 are complete and neither G_1 nor G_2 is empty. We first study the case that all components are made up of single vertices.

Our approach is to investigate in view of Definition 35 what constraints the complete subgraph $G_3 = G_A \vee G_V \vee G_R$ places on the number of K_1 components in G_1 and G_2. In particular we differentiate between the different possibilities that G_A, G_V and/or G_R are empty or not empty. This leads to a series of case distinctions. For each leaf case (1, 2.1, 2.2.1 and 2.2.2) we provide a lemma describing G structurally.

Case 1: G_3 is empty.

Lemma 43. *Let G_1 and G_2 only have K_1 components. If G_3 is empty, G is a $K_{m,n}$ with $m, n \geq 1$ or a $K_{m,n} - e$, with $m, n \geq 2$. These correspond to E_{12} and E_{13}.*

Proof. By Claim 24, no more than one pair of vertices (a, v) in $G_1 \times G_2$ is non-adjacent. If G contains no non-adjacent pair of vertices, G is a complete bipartite graph $G = K_{m,n}$ with $m, n \geq 1$. Now assume that G contains exactly one non-adjacent pair of vertices (a, v). G then is an *almost* complete bipartite graph $G = K_{m,n} - e$. Because G contains vertices x_1, x_2, a and v, we have $m, n \geq 2$. □

Case 2: G_3 is not empty.

From now on we can assume that G_3 is a complete graph with at least one vertex. By Claim 34, only one component from G_1 and G_2 each can be connected to G_3. If a vertex in G_1 exists that is connected to G_3, we denote it by a. We do the same for a vertex v in G_2. By Claim 34, a and v cannot be adjacent to the same vertex y in G_3. As per Definition 35, we partition G_3 into G_A, the subgraph of G_3 induced by the vertices of G_3 adjacent to vertex $a \in G_1$, G_V, the subgraph of G_3 induced by the vertices of G_3 adjacent to v, and lastly, G_R, the subgraph induced by all the remaining vertices in G_3 that are only adjacent to x_1 and x_2.

The following claim will prove useful in several lemmas below.

Claim 44. *If G_1 and G_2 each consist of a single vertex, then G is an instance of either E_4 or E_5.*

Proof. Let $G_3 = G_A \vee G_V \vee G_R$ be the partition of G_3 described above. The subgraphs G_A, G_V and G_R may be empty. Let a be the vertex of G_1 and v be the vertex of G_2. These two vertices may be adjacent or non-adjacent. The two resulting possibilities for G are shown in Figure 5.4 with the optional dotted edge av and $K_m := G_A \vee x_1$, $K_n := G_V \vee x_2$. Note that G_A and G_V may be empty but K_m and K_n always contain at least one vertex. If the edge av exists, G is an instance of E_4. If av does not exist and G_R is empty, G is an instance of E_4. If av does not exist and G_R is not empty, G is an instance of E_5. □

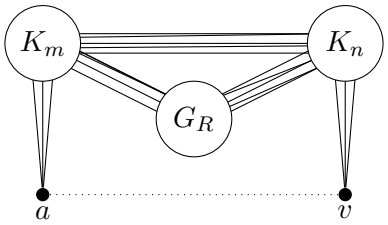


Figure 5.4: G if $G_1 = K_1$ and $G_2 = K_1$.

For our next case distinction, we distinguish between an empty and a non-empty G_R.

Case 2.1: G_R is not empty.

Assume first that G_R is not empty, which means that there exists a vertex in G_3 that is connected to neither G_1 nor G_2.

Lemma 45. *Let G_1 and G_2 have only K_1 components. If G_R is not empty, G is an instance of either E_4 or E_5.*

Proof. By Claim 36, G_1 and G_2 each have exactly one vertex and by Claim 44, G is an E_4 or E_5. □

Case 2.2: G_R is empty.

Now we assume that G_R is empty. By assumption G_3 is not empty. It follows that $G_3 = G_A \vee G_V$ and at least one of G_A and G_V is not empty. We distinguish between two cases. Either G_3 is completely connected to a single vertex in G_1 or G_2, i.e. that w.l.o.g. $G_3 = G_A$, or G_3 is connected to a and v, i.e. $G_3 = G_A \vee G_V$ and both G_A and G_V are not empty.

Case 2.2.1: $G_3 = G_A$.

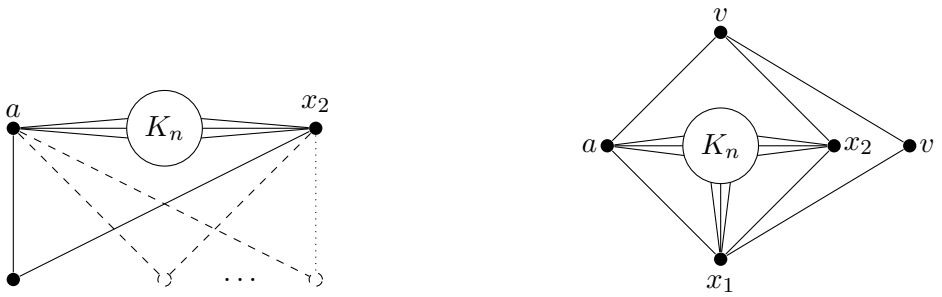
To discover what G looks like if $G_3 = G_A$, we need the following three lemmas. For these, assume that G_3 is completely connected to $a \in G_1$ and that v is a vertex in G_2. Vertices a and v can either be adjacent or non-adjacent. We investigate whether G_1 or G_2 can contain additional K_1 components.

Claim 46. *Let G_1 consist of the single vertex a and $G_3 = G_A$ be completely connected to a. Then G_2 can have any number of K_1 components but only one can be non-adjacent to a. G is then an instance of E_8.*

Proof. By Claim 24, all but one K_1 component in G_2 must be adjacent to a. By defining $K_n := G_A \vee x_1$, we see that G is an instance of E_8 as shown in Figure 5.5 (a). □

Claim 47. *Let G_A be non-empty and let v be a component in G_2. Then G_1 can have a second K_1 component b only if both a and b are adjacent to v. G_1 cannot have more than two components. If G_2 consists of a single vertex, G is an instance of E_3 as shown in Figure 5.5 (b).*

Proof. Let a and b be two K_1 components in G_1. Further let y_A be a vertex in G_A. We can assume that G_A is completely connected to a and in particular that y_A is adjacent to a. We show that a and b must both be adjacent to v. By Claim 24, at least one of them must be adjacent



(a) $G_1 = K_1$ and $G_2 = K_1^{\cup}$ (b) $G_1 = K_1 \vee K_1$ and $G_2 = K_1$

Figure 5.5: Possibilities for G if $G_3 = G_A$.

to v. If only a is adjacent to v, vertices a, b, v, x_2, y_A induce an F_3. If only b is adjacent to v, the same vertices induce a P_5. This shows that a and b must both be adjacent to v.

Next we show that G_1 cannot have more than two components. Let c denote a third component in G_1. If c is not adjacent to v, vertices a, b, c, v, y_A induce an F_1. If c is adjacent to v, the same vertices induce a chair. □

Claim 48. *Let G_A be non-empty. Then G_1 and G_2 cannot both have more than one K_1 component.*

Proof. Let y_A be a vertex in G_A. We assume that G_1 has two components a, b and G_2 has two components v, w. We can further assume that a is completely connected to G_A. By Claim 47, v and w are both adjacent to a and b. It follows that vertices b, v, w, x_1, y_A induce a chair. □

The three claims above allow us to provide an explicit description of G for the case that $G_3 = G_A$.

Lemma 49. *Let G_1 and G_2 only have K_1 components. If G_A is non-empty, then G is an instance of either E_3, E_4, E_5 or E_8.*

Proof. If G_1 and G_2 each consist of a single vertex, it follows from Claim 44 that G is an instance of E_4 or E_5. By Claim 48, G_1 and G_2 cannot both have two or more components. If G_2 has two or more K_1 components, then by Claim 46, G is an instance of E_8 as shown in Figure 5.5 (a). If on the other hand G_1 has more than one component, then by Claim 47, G_1 consists of two K_1 components and G is an instance of E_3 as shown in Figure 5.5 (b). □

Case 2.2.2: $G_3 = G_A \vee G_V$**.**

We now deal with the last case, namely that $G_3 = G_A \vee G_V$ and neither G_A nor G_V is empty.

Lemma 50. *Let G_1 and G_2 only have K_1 components. If $G_3 = G_A \vee G_V$, then G is an instance of E_3, E_4 or E_5.*

Proof. Throughout this proof, let y_A be a vertex in G_A and let y_V be a vertex in G_V. By assumption, G_1 contains a vertex a adjacent to y_A and G_2 contains a vertex v adjacent to y_V.

Assume first that a is not adjacent to v. We see that neither G_1 nor G_2 can contain any more components. Assume that b is a second component in G_1. By Claim 24, b and v must be adjacent and vertices a, b, v, y_A, y_V induce a P_5. If G_2 has two vertices, the situation is symmetrical. By Claim 44, then, G is an instance of E_4 or E_5.

Now assume that a and v are adjacent. If G_1 and G_2 each consist of a single vertex, we see with the help of Claim 44 that G is an instance of E_4 or E_5. We investigate whether G_1 and G_2 can contain additional K_1 components.

First we see that G_1 or G_2 can have no more than two components each. Let b and c be two additional components of G_1. By assumption, vertex a is still adjacent to y_A and b, c are not adjacent to y_A. By Claim 24, vertex v must be adjacent to at least two of the three vertices a, b, c. If v is adjacent to either b or c, vertices b, c, v, x_2, y_A induce an F_1. If v is adjacent to both b and c, vertices b, c, v, x_2, y_A induce a chair.

Secondly, we show that if G_1 contains a second K_1 component, it must be adjacent to v. Assume b is a second component in G_1 that is not adjacent to v. Then a, b, v, x_2, y_A induce an F_3. Next we see that if G_1 has two components, G_2 cannot also have two K_1 components. Let w be a second component of G_2. Then by the above w must be adjacent to a and vertices a, b, w, x_2, y_V induce an F_1 or chair, depending on whether w is adjacent to b or not. Conversely, if G_2 contains a second K_1 component, G_1 consists entirely of a single K_1 component due to symmetry. It follows that G is an instance of E_3 with $K_m := G_V \vee x_2$ and $K_n := G_A$. □

5.4.10 The structure of G if G_1 or G_2 has a complete component with ≥ 2 vertices, G_1 and G_2 are not empty and only contain complete components, and G_3 is complete and not empty

Now that we have dealt with the case that G_1 and G_2 contain only components of order 1, we assume that G_1 or G_2 contain at least one component of size ≥ 2. W.l.o.g. we assume that G_1 has a component of order ≥ 2 denoted by C throughout this section. By Claim 32, all components in G_1 and G_2 are complete. It is also worth repeating that G_3 can only be a complete graph at this point. We assume in this subsection that G_3 is not empty; the second case that G_3 is empty is dealt with in Subsection 5.4.11.

By Claim 27, G_1 cannot contain any further components of order ≥ 2. Just as in Subsection 5.4.9, we partition $G_3 = G_A \vee G_V \vee G_R$ using the notation introduced in Definition 18, then make a series of case distinctions that lead to structural lemmas. The first case we study is that G_R is not empty.

Case 1: G_R is not empty.

Lemma 51. *Let G_R contain one or more vertices. Then G is an instance of either E_4 or E_5.*

Proof. Let $y \in G_R$ be a vertex not adjacent to any vertices in G_1 or G_2. By Claim 36, G_1 and G_2 are both complete. We investigate the two possibilities $G_2 = K_1$ and $G_2 = K_n, n \geq 2$ separately.

If $G_2 = K_n$, then by Claim 37, G_1 and G_2 are completely connected. We define $H_2 := G_A \vee x_1$ and $H_3 := G_V \vee x_2$ and show this possibility in Figure 5.6 (a). It follows that G is an instance of E_4.

If G_2 consists of a single vertex v, then this vertex v is either completely connected or completely disconnected to G_1. Assume that G_1 and v are partially connected. Then there exist two vertices $a, b \in G_1$ such that a is adjacent to v and b is not adjacent to v and it follows that vertices a, b, v, x_2, y induce a P_5. The two resulting possibilities are shown in Figure 5.6 (b) and (c) with $H_2 := G_A \vee x_1$ and $H_3 := G_V \vee x_2$. It follows that G is an instance of E_4 or E_5. □

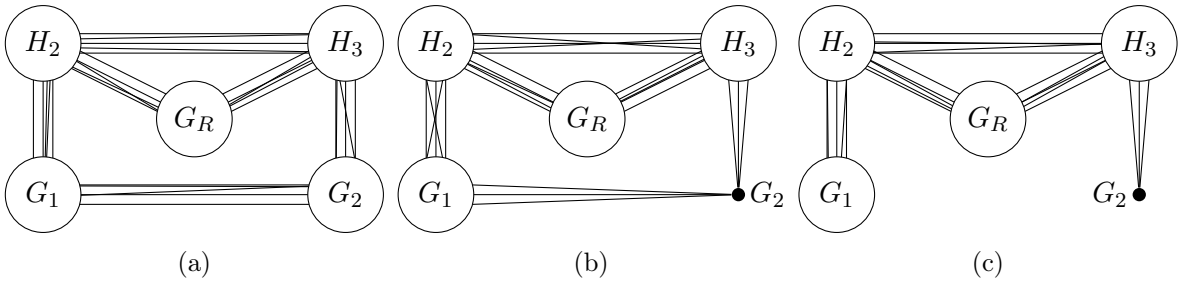


Figure 5.6: G if $G_1 = K_n$ with $n \geq 2$ and G_R is not empty.

Case 2: G_R is empty.

Now we investigate the situation that G_R is empty, i.e. that G_3 consists of $G_3 = G_A \vee G_V$ and either G_A or G_V (but not both) can be empty. By Claim 39, G_1 and G_2 contain at most two components each. By assumption, G_1 contains a complete component of order ≥ 2 and by Claim 27, the other component of G_1, if present, must be a K_1. As a result, we have either $G_1 = K_n$ or $G_1 = K_n \cup K_1$ with $n \geq 2$.

Next we determine the possibilities for G_2. Note that G_1 and G_2 cannot both contain a component of order ≥ 2 if either of them has two components. Otherwise, by Claim 38, G_A and G_V would both be empty, in contradiction to our assumption that G_3 is not empty. By Claim 27, this leads to the following possibilities for G_2. If $G_1 = K_n, n \geq 2$, we can have either $G_2 = K_m, m \geq 1$ or $G_2 = K_1 \vee K_1$. If $G_1 = K_n \vee K_1$, we have either $G_2 = K_1$ or $G_2 = K_1 \vee K_1$.

We consider all four possibilities for G_1 and G_2 to see whether they are compatible with $G_R = G_A \vee G_V$ not being empty.

Case 2.1: G_R is empty, $G_1 = K_n, n \geq 2$ and $G_2 = K_m, m \geq 1$.

Lemma 52. *If $G_1 = K_n, n \geq 2$ and $G_2 = K_m, m \geq 1$, then both G_A and G_V in $G_3 = G_A \vee G_V$ can be non-empty or either of them can be empty. G is an instance of E_4.*

Proof. Claim 29 describes the relationship between G_1 and G_2. In particular, we know that we can partition G_1 and G_2 into $G_1 = A_R, A_1, \ldots, A_k$ and $G_2 = V_R, V_1, \ldots, V_k$ such that A_R is disconnected from G_2, V_R is disconnected from G_1 and for every $i = 1, \ldots, k$ the subgraph A_i is completely connected to V_i and disconnected from $V_j, j \neq i$. We define two further subgraphs $A_{k+1} := G_A \vee x_1$ and $V_{k+1} := G_V \vee x_2$ to see that G is also an instance of E_4. □

From now on, either G_1 or G_2 has two components. This means that by Claim 38, G_V is empty and G_3 consists solely of $G_3 = G_A$ with a non-empty subgraph G_A.

Case 2.2: G_R is empty, $G_3 = G_A$, $G_1 = K_n, n \geq 2$ and $G_2 = K_1 \cup K_1$.

Lemma 53. *If $G_1 = K_n$, $G_2 = K_1 \cup K_1$ and $G_3 = G_A$ with $|G_A| \geq 1$, then G is an instance of either E_3 or E_6.*

Proof. Let v and w be the two non-adjacent vertices in G_2. By Claim 24, we can assume that v is completely connected to G_1. The second vertex w can either be completely disconnected from G_1 or adjacent to exactly one vertex a in G_1. If w were adjacent to two vertices a, b in G_1, vertices a, b, v, w, x_1 would induce a split 3-star. Both remaining possibilities are depicted in Figure 5.7. In both cases, $K_m = G_A \vee x_1$ contains G_A and x_1. In Graph (b), K_{n-1} is the subgraph $G_1 - a$. The two resulting structures correspond exactly to E_3 and E_6. □

Figure 5.7: G if $G_1 = K_n$, $G_2 = K_1 \cup K_1$ and G_3 is not empty.

Case 2.3: G_R **is empty,** $G_3 = G_A$**,** $G_1 = K_n \cup K_1, n \geq 2$ **and** $G_2 = K_1$**.**

Lemma 54. *If* $G_1 = K_n \cup K_1$*,* $G_2 = K_1$ *and* $G_3 = G_A$ *with* $|G_A| \geq 1$*, then* G *is an instance of* E_3*.*

Proof. Let a and b be two vertices of the K_n of G_1 and let c denote the K_1 in G_1. Further let v be the single vertex of G_2 and y_A be a vertex of G_A. By Definition 18, G_A must be completely connected to either the K_n or the K_1 of G_1.

We show that G_A must be completely connected to the K_n. Assume conversely that G_A is completely connected to the K_1 of G_1. By Claim 24, vertex v must be completely connected to at least one component of G_1. If v is completely connected to both the K_n and K_1, vertices $a, b, c, v, x_1, x_2, y_A$ induce an F_{15}. If v is not adjacent to a, vertices a, c, v, x_2, y_A induce an F_3. If v is not adjacent to vertex c, the same vertices induce a P_5.

As G_A is completely connected to the K_n, vertex v must also be completely connected to all of G_1, otherwise vertices a, c, y_A, v, x_2 would induce an F_1, F_3 or P_5 as above. It follows that G must be an instance of E_3. □

Case 2.4: $G_3 = G_A$**,** $G_1 = K_n \cup K_1$ **and** $G_2 = K_1 \vee K_1$**.**

Let C and b denote the K_n and the K_1 in G_1, respectively, and let v, w be the two K_1 in G_2. We can assume that w is not completely connected to C. Otherwise, if v and w were both completely connected to C, vertices v, w, x_1 and any two vertices from C would induce a split 3-star. Let c be a vertex in C that is not adjacent to w. By Claim 24, vertex b must be adjacent to v and w. Lastly, let y_A be a vertex in G_A. Then by definition, y_A must be adjacent to either b or c. If y_A is adjacent to b, vertices b, c, w, x_2, y_A induce an F_3. If y_A is adjacent to c, the same vertices induce a P_5.

This concludes our case distinctions and our structural description of G if G_3 is not empty.

5.4.11 The structure of G if G_1 or G_2 has a complete component with ≥ 2 vertices, G_1 and G_2 are not empty and only contain complete components, and G_3 is empty

Now we see what G can look like if G_3 is empty, G_1 and G_2 are both not empty and at least one of the two contains a component of order ≥ 2. By Claim 36, G_1 and G_2 can only have complete components, by Claim 28, we know that G_2 can have at most two components and by

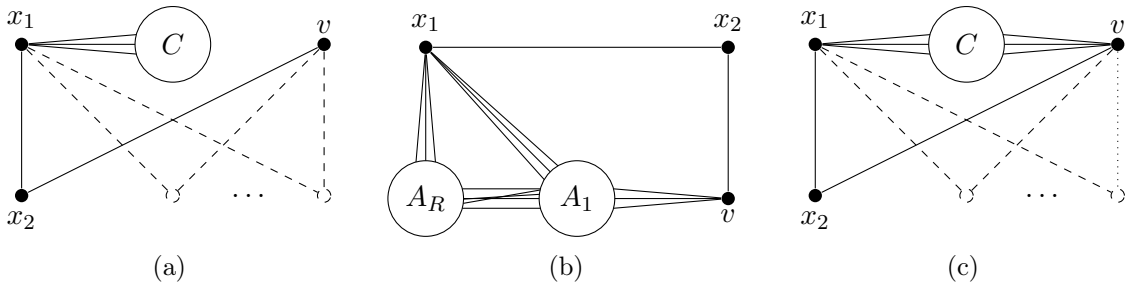


Figure 5.8: G if G_3 is empty and G_2 a single vertex.

Claim 27, G_2 cannot have two components of order ≥ 2. It follows that G_2 consist of either a single K_1, a single K_n, two K_1 components, or a K_1 and a K_n with $n \geq 2$.

We deal with each of these possibilities in turn. Because G_3 is empty, we cannot limit the number of K_1 components of G_1 a priori and must determine the structure of G_1 on a case by case basis. Note that by assumption, G_1 contains at least one component of order ≥ 2 and by Claim 27, it can only contain additional K_1 components. Our task is to determine how many K_1 components G_1 can have.

Case 1: $G_2 = K_1$**.**

Lemma 55. *If G_2 consists of a single vertex, G takes one of three shapes depicted in Figure 5.8, which correspond to E_4, E_7 and E_8.*

Proof. Assume that we have $G_2 = K_1$, denoted by the vertex v. We denote the component of G_1 of order ≥ 2 by C. Vertex v and C can be either completely disconnected, partially connected or completely connected. If v is partially connected to C, we partition C into the subgraph A_1 induced by all vertices adjacent to v and the subgraph $A_R = A \setminus A_1$. These three cases are illustrated by the solid edges and vertices of the graphs in Figure 5.8. We investigate for each of the three cases whether G_1 can contain additional K_1 components.

Case 1.1: If v and C are completely disconnected, G_1 can have any number of K_1 components that, by Claim 24, must all be adjacent to v. The result is shown in Figure 5.8 (a) and is clearly an instance of E_7.

Case 1.2: If v and C are partially connected, we show that G_1 consists entirely of C. Because v and C are partially connected, there exists a vertex $a \in A$ adjacent to v and a vertex $b \in A$ not adjacent to v. Assume d is a vertex from a second component in G_1. By Claim 24, d must be adjacent to v. It follows that vertices a, b, d, v, x_2 induce a chair. By partitioning $C = A_1 \vee A_R$, we see that the structure of G, shown in Figure 5.8 (b), is an instance of E_4.

Case 1.3: If v and C are completely connected, G_1 can have any number of K_1 components but by Claim 24, only one of them may not be adjacent to v. The resulting graph G is shown in Figure 5.8 (c) and is an instance of E_8. □

Case 2: $G_2 = K_n$ **with** $n \geq 2$**.**

Lemma 56. *If G_2 is a K_n with $n \geq 2$, G takes one of four shapes shown in Figure 5.9. G is an instance of E_3, E_4, E_6 or E_{10}.*

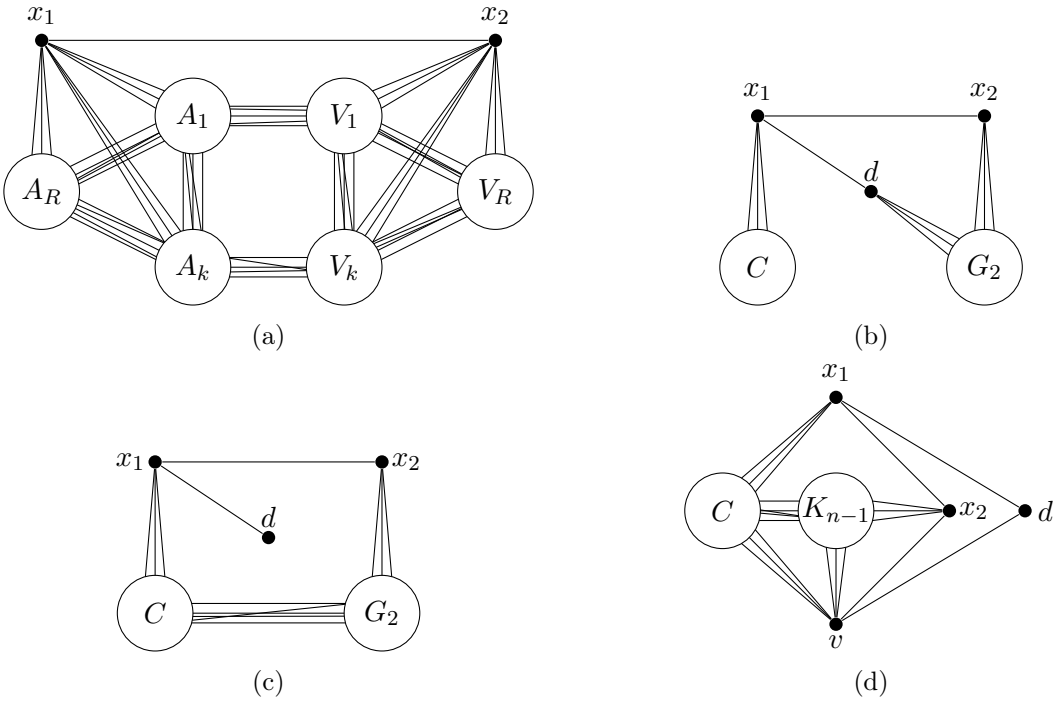


Figure 5.9: G if G_3 is empty and $G_2 = K_n$ with $n \geq 2$.

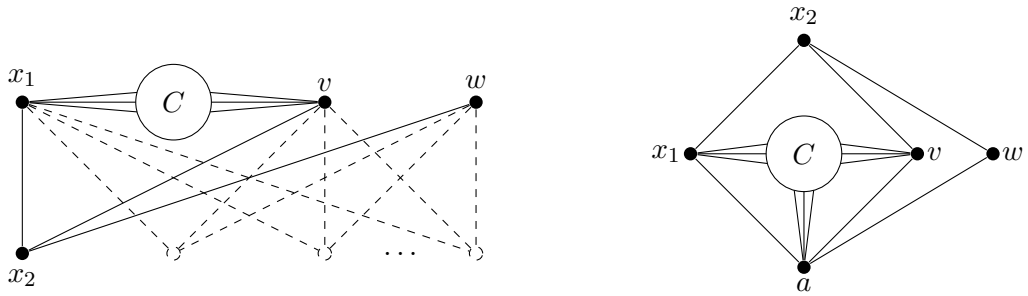
Proof. By assumption, G_1 has a complete component of order ≥ 2 and by Claim 28, G_1 can have at most one additional K_1 component. We distinguish between the case that G_1 is complete and that G_1 has an additional K_1 component.

Case 2.1: If G_1 is complete, Claim 29 provides us with the structural relationship between G_1 and G_2, as illustrated by $G_1 = A_R, A_1, \ldots, A_k$ and $G_2 = V_R, V_1, \ldots, V_k$ in Figure 5.9 (a). We define $A_{k+1} := x_1$ and $V_{k+1} := x_2$ to see that G is an instance of E_4.

Case 2.2: We now assume that G_1 consists of a component C of order ≥ 2 and a K_1 denoted by d. Then G is one of essentially two structures. First we show that C and G_2 must be either disconnected or completely connected. To do that, we choose any $a, b \in C$ and $v, w \in G_2$ and assume that they are partially connected. By Claim24, d must be adjacent to both v and w. We can dismiss all edge configurations that result in either a or b being adjacent to both v and w, as vertices d, x_1, v, w together with either a or b would otherwise induce a split 3-star. Due to symmetry, this leaves us with two possibilities. If a is adjacent to v and b is not, vertices a, b, d, v, x_2 induce a chair. If both a and b are adjacent to v (and thus non-adjacent to w), vertices a, b, d, v, w, x_1, x_2 induce an F_{15}.

Case 2.2.1: If C and G_2 are completely disconnected, it follows by Claim 24 that d must be completely connected to G_2. The graph G is shown in Figure 5.9 (b) and is an instance of E_{10}.

Case 2.2.2: If C and G_2 are completely connected, we see that d cannot be adjacent to more than one vertex v in G_2. If d were adjacent to two vertices $v, w \in G_2$, vertices d, v, w, x_2 together with any vertex $a \in C$ would induce a split 3-star. If d is not adjacent to any vertices in G_2, G is an instance of E_6 and if d is adjacent to one vertex v in G_2, G is an instance of E_3. These two cases are shown in Figure 5.9 (c) and (d). In (d), the induced complete subgraph $K_{n-1} := G_2 - v$ drawn is the subgraph G_2 without the vertex v that vertex d is attached to. □



(a) (b)

Figure 5.10: G if G_3 empty and $G_2 = K_1 \cup K_1$.

Case 3: $G_2 = K_1 \cup K_1$**.**

Lemma 57. *If G_2 consists of two K_1 components, G is an instance of E_3 or E_9, as depicted in Figure 5.10.*

Proof. We denote the two K_1 components of G_2 by v and w. By assumption, G_1 has a complete component C of order ≥ 2. By Claim 24, we can assume that v is completely connected to C. Vertex w can only be adjacent to at most one vertex in A. Assume that w is adjacent to two vertices $a, b \in C$. Then a, b, v, w, x_1 induce a split 3-star. This allows us to distinguish between two possibilities. In both cases we investigate whether G_1 can have K_1 components.

Case 3.1: If w and C are not connected, G_1 can have any number of additional K_1 components that must all be adjacent to v and w by Claim 24. These additional K_1 components are illustrated by the dashed elements in Figure 5.10 (a). G is then an instance of E_9.

Case 3.2: If w is adjacent to a single vertex a in C, then G_1 cannot have any K_1 components and must consist entirely of $G_1 = C$. Assume G_1 has a K_1 component d. By Claim 24, d must be adjacent to v and w. Let b be another vertex in C not adjacent to w. Then vertices a, b, d, w, x_2 induce a chair. The graph G with $G_1 = C$ is shown in Figure 5.10 (b) and is clearly an instance of E_3. □

Case 4: $G_2 = K_1 \cup K_n, n \geq 2$**.**

Lemma 58. *If G_2 consists of a K_1 and a K_n with $n \geq 2$, then G is an instance of E_3, E_6, E_{10} or E_{11}.*

Proof. By assumption and by Claim 28, G_1 has a component C of order ≥ 2 and no more than one additional K_1 component. We study the two possible cases $G_1 = C$ and $G_1 = C \vee K_1$.

Case 4.1 $G_1 = C$**:** If $G_1 = C$, then by relabelling G_1 to G_2 and vice versa we see that we have solved this problem in Case 2, Lemma 56. After relabelling, G becomes one of graphs (b), (c) or (d) in Figure 5.9, which correspond to instances of E_{10}, E_6 and E_3, respectively.

Case 4.2 $G_1 = C \vee K_1$**:** Now we study whether any of three possibilities (b), (c) and (d) for G in Case 4.1 allow for an additional K_1 in G_1. Assume G_1 consists of C and a K_1 component denoted by a. Let V and w denote the components of order ≥ 2 and order 1 of G_2, respectively.

If G takes shape (b), C and V are completely disconnected and w is completely connected to C. It is easily seen with the help of Claim 24 that a must be completely connected to all of G_2. The resulting graph G is shown in Figure 5.11 and is an instance of E_{11}.

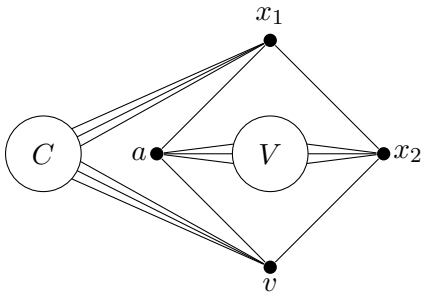


Figure 5.11: The third possibility for G if $G_2 = K_1 \cup K_n$ with $n \geq 2$.

Assume now that G is a graph of shape (c) or (d). By Case 4.1, C is completely connected to V and not completely connected to w. It follows from Claim 24 that vertex a must be adjacent to all of G_2. Then any two vertices in the K_n of G_2 together with a, x_2 and a vertex in A induce a split 3-star. □

This concludes the proof of Lemma 15: any graph G without $F_1, \ldots, F_{15}$ is an instance of $E_1^{\cup}, E_2, \ldots, E_{13}$. □

5.5 Induced subgraphs of an $E_1^{\cup}, E_2, \ldots, E_{13}$ are again instances of the same

In this section we show that graphs G that are instances of $E_1^{\cup}, E_2, \ldots, E_{13}$ only have induced subgraphs that are also instances of $E_1^{\cup}, E_2, \ldots, E_{13}$.

Lemma 59. *Let G be an instance of $E_1^{\cup}, E_2, \ldots, E_{13}$. Any induced subgraph H of G is also an instance of $E_1^{\cup}, E_2, \ldots, E_{13}$.*

Proof. Let H be a subgraph of G induced by the set of vertices $S \subseteq V(G)$. Then H is the graph obtained by deleting all vertices in $V(G) \setminus S$ and incident edges from G. In order to prove that H is an $E_1^{\cup}, E_2, \ldots, E_{13}$, we use the following inductive approach.

With the help of Claims 60 to 72 we show that removing a single vertex v and associated edges from an instance of $E_1^{\cup}, E_2, \ldots, E_{13}$ results in a graph $G-v$ that is also an $E_1^{\cup}, E_2, \ldots, E_{13}$. If we remove the vertices in $V(G) \setminus S$ from G one by one, we know after every step that the remaining graph is still an $E_1^{\cup}, E_2, \ldots, E_{13}$. This is especially true once we remove all vertices in $V(G) \setminus S$. As the remaining graph is now H, we are finished. □

Claim 60. *Let G be an $E_1^{\cup}$. Then any induced subgraph $G - v$ obtained by removing a vertex v and incident edges from G is another instance of $E_1^{\cup}$.*

Proof. Let H be the E_1 component of G that we are removing the vertex v from. If v is the only vertex in H and G consists entirely of H, $G - v$ is the empty graph and an E_{12}. Otherwise, if v is the only vertex in H and G has two or more components, $G - v$ is an $E_1^{\cup}$.

Assume from now on that H has order ≥ 2. If v is not the dominating vertex in H, then $H - v$ is an E_1. It follows that $G - v$ is an $E_1^{\cup}$. If v is the dominating vertex in H, $H - v$ is an $E_1^{\cup}$ and the same follows for $G - v$. □

Claim 61. *Let G be an E_2. Then any induced subgraph $G - v$ obtained by removing a vertex v and incident edges from G is an $E_1, E_1 \cup E_1$, E_2 or E_5.*

Proof. Let G be an E_2. $G - x_1$ consists of two E_1 components. One E_1 is made up of vertex d, the other E_1 consists of vertices e and x_2 together with K_a, K_b and K_c. Next, $G - d$ is an E_1.

Let a be a vertex in K_a. $G - a$ is an E_2. Now let v_b be a vertex in K_b. If $b = 1$, then $G - v_b$ is an E_5. If $b \geq 2$, then $G - v_b$ is an E_2. All other cases are symmetrical. $\square$

Claim 62. *Let G be an E_3. Then any induced subgraph $G - v$ obtained by removing a vertex v and incident edges from G is an E_1, E_3, E_4, E_6 or E_8.*

Proof. Let G be an E_3. $G - a$ is an E_1, $G - b$ is an E_4, $G - c$ is an E_6 and $G - d$ is an E_4. Let v_m be a vertex in K_m. If $m = 1$, $G - v_m$ is an E_4. If $m \geq 2$, $G - v_m$ is an E_3. Now let v_n be a vertex in K_n. If $n = 1$, $G - v_n$ is an E_8. If $n \geq 2$, $G - v_n$ is an E_3. □

Claim 63. *Let G be an E_4. Then any induced subgraph $G - v$ obtained by removing a vertex v and incident edges from G is an $E_1, E_1 \cup E_1$ or E_4.*

Proof. Assume first that $k = 1$. Let v be a vertex in A_1. If $|A_1| = 1$, $G - v$ has the structure $G - v = A_R \cup (V_1 \vee V_R)$. If A_R is empty, the graph is an E_1. If A_R is not empty, the graph is an instance of $E_1 \cup E_1$. If $|A_1| \geq 2$, $G - v$ is an E_4. Now let v be a vertex in A_R. Then $G - v$ is an E_4. Now assume that $k = 2$. Then for any vertex v in G the resulting graph $G - v$ is an E_4. □

Claim 64. *Let G be an E_5. Then any induced subgraph $G - v$ obtained by removing a vertex v and incident edges from G is an $E_1 \cup E_1, E_4$ or E_5.*

Proof. Let G be an E_5. $G - e$ is an E_4. Now we turn to the subgraphs K_a, K_b, K_c and K_d. If we remove a vertex v from K_a, K_b, K_c or K_d and the respective complete subgraph is not empty in $G - v$, the induced subgraph is an E_5. If K_a has only one vertex and we remove it, the result is an E_4. If K_b has only one vertex and we remove it, the result is an $E_1 \cup E_1$. If K_c has only one vertex and we remove it, the result is an E_4. If K_d has only one vertex and we remove it, the result is an instance of $E_1 \cup E_1$. □

Claim 65. *Let G be an E_6. Then any induced subgraph $G - v$ obtained by removing a vertex v and incident edges from G is an $E_1 \cup E_1, E_4, E_6, E_8$ or E_{10}.*

Proof. Let G be an E_6. $G - a$ is an E_4, $G - b$ is an $E_1 \cup E_1$ and $G - c$ is an E_4. Let v_n be a vertex in K_n. If $n = 2$, then $G - v_n$ is an E_8. If $n \geq 3$, then $G - v_n$ is an E_6. Now let v_m be a vertex in K_m. If $m = 2$, then $G - v_m$ is an E_{10}. If $m \geq 3$, then $G - v_m$ is an E_6. □

Claim 66. *Let G be an E_7. Then any induced subgraph $G - v$ obtained by removing a vertex v and incident edges from G is an $E_1 \cup E_1, E_7$ or E_{13}.*

Proof. Let G be an E_7. $G - a$ is an $E_1 \cup E_1$ and $G - c$ is an E_1. Let v_b be a vertex in the bottom row. If the bottom row has two or more vertices, $G - v_b$ is an E_7. If the bottom row only has one vertex, $G - v_b$ is an $E_1 \cup E_1$. Let v_n be a vertex in K_n. If $n = 2$, then $G - v_n$ is an E_{13}. If $n \geq 3$, then $G - v_n$ is an E_7. □

Claim 67. *Let G be an E_8. Then any induced subgraph $G - v$ obtained by removing a vertex v and incident edges from G is an $E_1, E_1 \cup E_1, E_8, E_{12}$ or E_{13}.*

Proof. Let G be an E_8. $G - a$ is an E_1 or $E_1 \cup E_1$ and $G - c$ is an E_1. Let v_n be a vertex in K_n. If $n = 2$, then $G - v_n$ is an E_{12} or E_{13}. If $n \geq 3$, then $G - v_n$ is an E_8. Now let v_b be a vertex in the bottom row. If the bottom row has only one vertex, $G - v_b$ is an E_1. If the bottom row has two or more vertices, $G - v_b$ is an E_8. □

Claim 68. *Let G be an E_9. Then any induced subgraph $G - v$ obtained by removing a vertex v and incident edges from G is an $E_1 \cup E_1, E_7, E_8, E_9$ or E_{13}.*

Proof. Let G be an E_9. $G - a$ and $G - c$ are instances of E_7 and $G - d$ is an E_8. Let v_b be a vertex in the bottom row. If the bottom row has only one vertex, $G - v_b$ is an $E_1 \cup E_1$. If the bottom row has two or more vertices, $G - v_b$ is an E_9. Now let v_n be a vertex in K_n. If $n = 2$, then $G - v_n$ is an E_{13}. If $n \geq 3$, then $G - v_n$ is an E_9. □

Claim 69. *Let G be an E_{10}. Then any induced subgraph $G-v$ obtained by removing a vertex v and incident edges from G is an $E_1, E_1 \cup E_1, E_4$ or E_{10}.*

Proof. Let G be an E_{10}. $G-a$ is an $E_1 \cup E_1$ and $G-b$ (and $G-c$) is an E_4. Let v_n be a vertex in K_n. If $n=1$, $G-v_n$ is an E_1. If $n \geq 2$, then $G-v_n$ is an E_{10}. Now let v_m be a vertex in K_m. If $m=1$, then $G-v_m$ is an E_4. If $m \geq 2$, then $G-v_m$ is an E_{10}. □

Claim 70. *Let G be an E_{11}. Then any induced subgraph $G-v$ obtained by removing a vertex v and incident edges from G is an E_8, E_{10}, E_{11} or E_{12}.*

Proof. Let G be an E_{11}. $G-a$ is an E_{10}. Let v_m be a vertex in K_m. If $m=n=1$, $G-v_m$ is an E_{12}. If $m=1$ and $n \geq 2$, $G-v_m$ is an E_8. If $m \geq 2$, $G-v_m$ is an E_{11}. All other cases are symmetrical. □

Claim 71. *Let G be an E_{12}. Then any induced subgraph $G-v$ obtained by removing a vertex v and incident edges from G is an E_{12}.*

Proof. No matter which vertex we remove, every remaining vertices in the top row is still adjacent to every remaining vertex in the bottom row. The result follows immediately with the definition of a complete bipartite graph. □

Claim 72. *Let G be an E_{13}. Then any induced subgraph $G-v$ obtained by removing a vertex v and incident edges from G is either an $E_1 \cup E_1, E_{12}$ or E_{13}.*

Proof. Let G be an E_{13} and let a, b be the two vertices that would be incident with the edge e that is missing. Then $G-a$ or $G-b$ are instances of E_{12}, as every remaining vertex in the top row is adjacent to every remaining vertex in the bottom row.

Now let v be a vertex that is not a or b. If the row we are removing v from has exactly two vertices, $G-v$ is an $E_1 \cup E_1$, where one of the E_1 is a single vertex and the other E_1 is a star. If the row we are removing v from has three or more vertices, $G-v$ is an E_{13}. □

5.6 Instances of $E_1^{\cup}, E_2, \ldots, E_{13}$ are g_B-nice

In this section we prove the following lemma.

Lemma 73. *Any instance of $E_1^{\cup}, E_2, \ldots, E_{13}$ is g_B-nice.*

Proof. Let G be an instance of $E_1^{\cup}$ with two or more E_1 components. Then by Theorem 13, G is g_B-perfect and we are done. Now let G be an instance of E_1. Then G is an induced subgraph of $G \cup G$. This graph $G \cup G$ is an instance of $E_1^{\cup}$ and as such g_B-perfect. As induced subgraphs of g_B-perfect graphs are again g_B-perfect, it follows that G is g_B-perfect.

Therefore we can assume from now on that G is an instance of $E_2, \ldots, E_{13}$. For each of these possibilities $E_i, 2 \leq i \leq 13$ we provide a strategy for Alice to win on E_i with $\omega(E_i)$ colours. This shows that E_i is g_B-nice. For the sake of completeness, we also provide a strategy for Alice to win on E_1. This strategy is easily extended to $E_1^{\cup}$: Alice simply always responds in the same component as Bob just coloured.

If Alice's approach is straightforward, we list her moves sequentially. Otherwise we define rules based on game states. Each game state, defined in bold, represents a class of partial graph colourings such that no vertex has been surrounded. For each of these game states, we provide instructions for Alice to respond to every possible move by Bob. These instructions either lead to another state or allow Alice to win immediately. It follows inductively that Alice must win after at most $|G|$ moves.

A strategy for E_1

Let G be an instance of E_1. We show that Alice can win on G with $\omega(G)$ colours. The vertices at risk are the dominating vertex x and all vertices in K_a. Recall that $b \geq$ c. Alice's main aim is to make sure that K_b and K_c contain the same colours after each round until either K_a or K_c is fully coloured. This guarantees that no vertex in K_a can be surrounded. Alice follows the following state-based rules to win.

State 1 [start]: The vertex x is not coloured.

- Assume K_a or K_c is fully coloured. If Bob colours x, all vertices have been neutralised and Alice wins. If Bob colours any other vertex, Alice colours x and wins for the same reason.
- Now assume K_a and K_c are both not fully coloured.
  - If Bob colours a vertex in K_b, Alice applies the same colour to a vertex in K_c and vice versa. Repeat state 1.
  - If Bob colours vertex x, Alice colours a vertex in K_a and vice versa. If K_a is now fully coloured, Alice wins, otherwise proceed to state 2.
  - If Bob colours a vertex in $H_1, \ldots, H_k$, Alice colours vertex x. Proceed to state 2.

State 2: The vertex x is coloured.

- Assume K_a has exactly one uncoloured vertex v_a. If Bob colours v_a, Alice wins. If Bob colours any other vertex, Alice colours v_a in K_a and wins.
- Now assume K_a has two or more uncoloured vertices.
  - If Bob colours one of these, Alice colours another. If K_a is now fully coloured, Alice wins because all vertices at risk have been coloured. Otherwise repeat state 2.
  - If Bob colours a vertex in K_b, Alice applies the same colour to a vertex in K_c and vice versa. If K_c is fully coloured, K_a has been neutralised and Alice wins. Otherwise repeat state 2.
  - If Bob colours a vertex in $H_1, \ldots, H_k$, Alice colours a vertex in K_a. Repeat state 2.

A strategy for E_2

Let G be an instance of E_2. Recall that $b \geq c$. The vertices at risk are x_1, x_2 and all vertices in K_a. Note that for any proper vertex colouring with $\omega(G)$ colours, K_c cannot have a colour not in K_b. Because of this, Alice makes sure that K_b and K_c contain the same colours after each round until K_a or K_c is full. We restrict ourselves to describing rules that apply to symmetrical situations only once.

State 1 [start if $a \geq 1$]: Vertices d, e, x_1, x_2 are uncoloured, K_a and K_c are not completely coloured. All uncoloured vertices are at risk.

- If Bob colours vertex d in a new colour, Alice applies the same colour to vertex x_2 and vice versa. If Bob instead colours vertex d in a colour already used in K_a, K_b or K_c, Alice uses a new colour on vertex x_2. In both cases proceed to state 5.

- If Bob colours a vertex in K_b, Alice applies the same colour to a vertex in K_c and vice versa. If K_c is now completely coloured, proceed to state 2. Otherwise repeat this state.
- If K_a has exactly one uncoloured vertex and Bob colours it, Alice colours vertex x_1 and wins on her next move: if Bob colours vertex x_2 she wins immediately. If Bob instead colours any other vertex, Alice colours x_2 herself.
- If K_a has two or more uncoloured vertices and Bob colours one of them, Alice colours the other. If K_a is now fully coloured, proceed to state 3. Otherwise repeat this state.

State 2: Vertices d, e, x_1, x_2 are uncoloured, K_a is not completely coloured and K_c is completely coloured. The vertices in K_a have been neutralised, only x_1 and x_2 remain at risk.

- If Bob colours vertex d in a new colour, Alice applies the same colour to vertex x_2 and vice versa. If Bob instead colours vertex d in an old colour, Alice uses a new colour on vertex x_2. In both cases Alice wins because x_1 and x_2 are no longer at risk.
- If Bob colours a vertex in K_a or K_b, Alice applies the same colour to d and wins on her next move: if Bob colours vertex x_2, Alice wins immediately. If Bob colours any other vertex, Alice colours x_2 herself.

State 3 [start if $a = 0$]: Vertices d, e, x_1, x_2 are uncoloured, K_a is completely coloured and K_c is not completely coloured. The vertices in K_a have been neutralised, only x_1 and x_2 remain at risk.

- If Bob colours vertex d in a new colour, Alice applies the same colour to vertex x_2 and vice versa. Since x_1 is now neutralised, Alice wins. If Bob colours d in a colour already used in K_a, K_b or K_c, Alice applies a new colour to x_2. Alice now wins after the next round: if Bob's move is to colour vertex x_1, Alice wins. If Bob instead colours any other vertex, Alice wins by colouring vertex x_1 herself.
- If Bob colours a vertex in K_b, Alice applies the same colour to a vertex in K_c and vice versa. If K_c is not completely coloured yet, repeat this state. Otherwise proceed to state 4.

State 4: Vertices d, e, x_1, x_2 are uncoloured, K_a and K_c are completely coloured. Only vertices x_1 and x_2 are still at risk.

- If Bob colours vertex d in a new colour, Alice applies the same colour to x_2 and vice versa. If Bob instead colours vertex d in a colour already used in K_a, K_b or K_c, Alice applies a new colour to x_2. In both cases x_1 and x_2 are now neutralised and Alice wins.
- If Bob colours a vertex in K_b, Alice colours vertex d in a colour from K_b. Alice now wins after the next round by making sure that x_2 is coloured by Bob or herself.

State 5: Vertices e, x_1 are uncoloured, d, x_2 are coloured such that d's colour is used elsewhere, K_a and K_c are not completely coloured. Only vertex x_2 has been neutralised, vertex x_1 and the uncoloured vertices in K_a are still at risk.

- If Bob colours vertex x_1, Alice colours vertex e and vice versa. Proceed to state 6.
- If Bob colours a vertex in K_b, Alice applies the same colour to a vertex in K_c and vice versa. If K_c is now fully coloured, Alice wins because vertices x_1 and K_a have been neutralised. Otherwise repeat this state.
- If K_a has one uncoloured vertex and Bob colours it, Alice colours vertex x_1 and wins.

- If K_a has two uncoloured vertices and Bob colours one, Alice colours the other and wins on her next move by making sure that x_2 is coloured by Bob or herself.
- If K_a has three or more uncoloured vertices and Bob colours one of them, Alice colours another. Repeat this state.

State 6: Vertices d, e, x_1, x_2 are coloured, K_a and K_c are not completely coloured. The uncoloured vertices in K_a are at risk.

- If Bob colours a vertex in K_a and it is fully coloured, Alice wins. Otherwise Alice colours another vertex in K_a. If K_a is now fully coloured, she wins. Otherwise repeat this state.
- If Bob colours a vertex in K_b, Alice applies the same colour to K_c and vice versa. If K_c is now completely coloured, Alice wins. Otherwise repeat this state.

A strategy for E_3

Let G be an instance of E_3. The only vertex not at risk is d. For a colouring with $\omega(G) = m+n+1$ colours, vertex b must be coloured the same as a vertex in K_m, vertex a must be coloured the same as c or a vertex in K_n, and vertex d must be coloured the same as a vertex in K_m or K_n. This leads to the following strategy for Alice.

State 1 [start]: None of the vertices in G are coloured.

- If Bob colours vertex a, Alice applies the same colour to vertex c and vice versa. Proceed to state 5.
- If Bob colours vertex b, Alice applies the same colour to a vertex in K_m and vice versa. If $m \geq 5$, proceed to state 3. If $m = 1$, K_m is now fully coloured, a and K_n are neutralised and Alice wins by making sure that c is coloured by Bob or herself in the next round.
- If Bob colours vertex d, Alice applies the same colour to a vertex in K_n and vice versa. If K_n is now fully coloured, proceed to state 4. Otherwise proceed to state 2.

State 2: Vertices a, b, c and K_m are uncoloured, d is coloured in same colour as a vertex in K_n, which is not completely coloured. All uncoloured vertices are still at risk.

- If Bob colours a, Alice applies the same colour to c and vice versa. Proceed to state 5.
- If Bob colours b, Alice applies the same colour to K_m and vice versa. If K_m is now fully coloured, Alice wins. Otherwise proceed to state 6.
- If Bob colours the last uncoloured vertex in K_n, Alice colours a in the same colour and then wins by making sure that vertex c is coloured by Bob or herself in the next round.
- If Bob colours the penultimate uncoloured vertex in K_n, Alice colours the last uncoloured vertex. Proceed to state 4.
- If K_n contains more than two uncoloured vertices and Bob colours one, Alice colours another. Repeat this state.

State 3: Vertices a, c, d and K_n are uncoloured, b is coloured in the same colour as a vertex in K_m and K_m is not completely coloured. Vertices a, b and K_n are neutralised, vertex c and the uncoloured vertices in K_m are at risk.

- If Bob colours vertex c, Alice wins by applying the same colour to a and vice versa.
- If Bob colours vertex d in a colour previously used, Alice colours a vertex in K_m in a new colour. If Bob instead colours vertex d in a new colour, Alice applies the same colour to K_m. If K_m is now completely coloured, Alice wins. Otherwise proceed to state 6.
- If Bob colours a vertex in K_m or K_n, Alice applies the same colour to d. If K_m is now completely coloured, Alice wins. Otherwise proceed to state 6.

State 4: Vertices a, b, c and K_m are uncoloured, d is coloured in same colour as a vertex in K_n, which is fully coloured. Only K_n is neutralised.

- If Bob colours a, Alice wins by applying the same colour to c and vice versa.
- If Bob colours b, Alice applies the same colour to K_m and vice versa. If K_m is now fully coloured, Alice wins. Otherwise proceed to state 6.

State 5: Vertices a, c are coloured the same, b and K_m are uncoloured and K_n is not completely coloured. Vertices a, b, c and K_m are neutralised, only the uncoloured vertices in K_n are at risk.

- If Bob colours vertex b, Alice applies the same colour to K_m and vice versa. As K_n is now neutralised, she wins.
- If Bob colours a vertex in d or K_n, Alice colours a vertex in K_n. If K_n was already fully coloured or is fully coloured now, she wins. Otherwise repeat this state.

State 6: Vertices a, c are uncoloured, b is coloured in the same colour as a vertex in K_m, K_m is not completely coloured, d is coloured in the same colour as a vertex in K_n or K_m. Only the uncoloured vertices in K_m are at risk.

- If Bob colours vertex a in a colour present in K_n, Alice wins. If Bob instead colours vertex a in a new colour, Alice wins by applying the same colour to c and vice versa.
- If Bob colours a vertex in K_m or K_n, Alice colours a vertex in K_m. If K_n was already fully coloured or if K_m is fully coloured, Alice wins. Otherwise repeat this state.

A strategy for E_4

Let G be an instance of E_4. We saw in Section 5.3 that the clique number of G is either $\omega(G) = |A_1| + |V_1|$ or $|A|$, depending on which is larger. We will treat these two cases separately.

The case $|A_1| + |V_1| > |A|$

Assume that $|A_1| + |V_1| > |A|$. As by assumption we have $|A| \geq |V|$, it follows immediately that $|A_1| > |V| - |V_1|$ and $|V_1| > |A| - |A_1|$. We first show constructively that there exists a vertex colouring for G with $\omega(G) = |A_1| + |V_1|$ colours: colour the vertices of $A_1 \vee V_1$ in consecutive colours $1, \ldots, \omega(G)$. Next colour every vertex in $V \setminus V_1$ in consecutive colours from A_1. Lastly, colour every vertex in $A \setminus A_1$ in consecutive colours from V_1.

Note that in this colouring every vertex in $A \setminus A_1$ and $V \setminus V_1$ has exactly one identically-coloured vertex in V_1 or A_1, respectively. Alice recreates this colouring with her strategy. We describe Alice's responses if Bob colours a vertex in V. The case for A is symmetrical.

- If Bob colours a vertex in $V \setminus V_1$, Alice applies the same colour to a vertex in A_1.

- Assume that $A \setminus A_1$ is not fully coloured. If Bob colours a vertex in V_1, Alice applies the same colour to a vertex in $A \setminus A_1$. If $V \setminus V_1$ is now fully coloured, $A \setminus A_1$ and all of V have been neutralised.
- Assume that $A \setminus A_1$ is fully coloured. Only vertices V_1 are at risk. If Bob colours a vertex in V_1, Alice colours another vertex in V_1. If V_1 is fully coloured, Alice wins.

The case $|A_1| + |V_1| \leq |A|$

We assume that $\omega(G) = |A| \geq |A_1| + |V_1|$, which means we have exactly $|A|$ colours available in the game. As above, we first show that there exists a proper vertex colouring of G with $\omega(G)$ colours, before providing a strategy for Alice to achieve this colouring competitively. We employ Hall's Marriage Theorem [28] for this.

Theorem 74 (Hall (1935))**.** *Let G be a bipartite graph with disjoint vertex sets V_1 and V_2 and $|V_1| \leq |V_2|$. G has a matching of V_1 if and only if for every subset S of V_1:*

$$|S| \leq |N(S)|.$$

We are now ready to show that there exists a proper vertex colouring with $\omega(G)$ colours.

Claim 75. *Let $|A_1| + |V_1| \leq |A|$. G admits a proper vertex colouring with $|A|$ colours.*

Proof. Let $\overline{G}$ be the complement of G. Note that $\overline{G}$ is bipartite, allowing us to apply Hall's Marriage Theorem.

Let S be a subset of V_i, $1 \leq i \leq k$. Then its neighbouring vertices in $\overline{G}$ are $N_{\overline{G}}(S) = V(A \setminus \overline{A_i})$. By assumption we have $|A_1| + |V_1| \leq |A|$, which implies $|V_1| \leq |A \setminus A_i|$. It follows that $|S| \leq |N_{\overline{G}}(S)|$. Now let S be a set of V that is not a subset of any $V_i, 1 \leq i \leq k$. Then we have $N_{\overline{G}}(S) = V(A)$. As by assumption $|V| \leq |A|$, it follows that $|S| \leq |N_{\overline{G}}(S)|$. Applying Hall's Marriage Theorem, we know that there exists a matching that entirely covers V.

The matching pairs up every vertex v in V with a vertex a_v in A. As the two vertices a_v and v are adjacent in $\overline{G}$, they are not adjacent in G. This allows us to achieve a proper vertex colouring of G. First we colour all vertices in A in consecutive colours. Then for every vertex in v in V, we retrieve the colour of its paired vertex a_v in A and apply it to v. □

Now that we have a vertex colouring of G that fulfils the above requirements, we note the following. Every vertex in V is assigned a colour also used in A but not every colour in A is also present in V. If a vertex a in A has a vertex with the same colour in V, we say that a and v are *paired*. Otherwise we call the vertex *unpaired*. The following modification of the vertex colouring results in another valid vertex colouring: Let a be an unpaired vertex in A_i and let v be a vertex in $V \setminus V_i$ that is paired with $b \in A$. We can pair a and v by assigning v the colour of a. This means v and b are no longer coloured the same, and b is now an unpaired vertex.

Once we know that a non-competitive vertex colouring of G exists as described above, Alice can use the following strategy to win.

If Bob colours a vertex v in V, Alice applies the same colour to the vertex $a \in A$ that v is paired with. If Bob colours any vertex a in A that is paired with a vertex v in V, Alice applies the same colour to v. If V is now fully coloured, Alice wins as Bob cannot surround any vertices any more.

If Bob colours any vertex a in $A_i, 1 \leq i \leq k$ that is not paired with a vertex in $V \setminus V_i$, Alice applies the same colour to any other vertex v in $V \setminus V_i$. This pairs up vertices a and v and unpairs

the vertex that v was originally paired with. If V is full, Alice wins as Bob cannot surround any vertices any more.

If Alice cannot respond in this way because $V \setminus V_i$ is already fully coloured, we note the following. Firstly, all vertices in V and the vertices in $A \setminus A_i$ are neutralised and only the vertices in A_i are still at risk. Secondly, if a vertex in A_i is uncoloured, it must be an unpaired vertex. For this reason, Alice responds to Bob's move by colouring another vertex in A_i. If all vertices in A_i are fully coloured, Alice wins.

A strategy for E_5

Let G be an instance of E_5. First we assume that $a > c + d$. Then the clique number of G is $\omega(G) = a + b \geq b + c + d + 1$. The only vertices at risk are the vertices of K_b. Alice adheres to the following strategy to win. If Bob colours the vertex e or a vertex in K_b, Alice colours another vertex in K_b. If Bob instead colours a vertex in K_a, Alice applies the same colour to $K_c \vee K_d$ and vice versa. Alice wins as soon as K_b or $K_c \vee K_d$ is fully coloured.

If $a \leq c + d$, Alice follows a slightly more complicated strategy. The clique number of G in that case is $\omega(G) = b + c + d$. The vertices at risk are in K_b and K_d. In general, Alice makes sure that K_a always contains the same colours as $K_c \vee K_d$ after every round. Similarly, she ensures that vertex e is coloured the same as a vertex in K_b or K_c. If both of these objectives have been achieved, she wins as K_b and K_d are now neutralised.

State 1 [start]: K_a, K_b and K_d are not completely coloured and vertex e is uncoloured. K_a and $K_c \vee K_d$ contain the same colours. The uncoloured vertices in K_b and K_d are at risk.

- If Bob colours e in a new colour, Alice applies the same colour to a vertex in K_b and vice versa. If Bob colours e in a colour already used in K_b or K_c, Alice colours a vertex in K_b in a new colour. In both cases this neutralises K_d. If K_b is now fully coloured, Alice wins. Otherwise proceed to state 2.
- If Bob colours a vertex in K_a, Alice applies the same colour to a vertex in K_d and vice versa. If K_a and K_d are fully coloured, Alice wins. If only K_a is fully coloured, proceed to state 3. If only K_d is fully coloured, proceed to state 4. If neither K_a nor K_d are fully coloured, repeat this state.
- If Bob colours a vertex in K_c, Alice applies the same colour to a vertex in K_a. If K_a is now fully coloured, proceed to state 3, otherwise repeat this state.

State 2: K_a, K_b and $K_c \vee K_d$ are not completely coloured and vertex e is coloured in a colour from K_b. K_a and $K_c \vee K_d$ contain the same colours. Only the uncoloured vertices in K_b are at risk.

- If Bob colours a vertex in K_c or K_d, Alice applies the same colour to K_a and vice versa. If K_a is fully coloured, K_b has been neutralised and Alice wins. Otherwise repeat this state.
- If Bob colours a vertex in K_b, Alice also colours a vertex in K_b. If K_b was already fully coloured or if K_b is now fully coloured, Alice wins, otherwise repeat this state.

State 3: K_a is completely coloured with colours from $K_c \vee K_d$, K_b and K_d are not completely coloured and vertex e is uncoloured. Only the uncoloured vertices in K_d are at risk.

- If Bob colours a vertex in K_b or K_c, Alice wins by applying the same colour to e.

- If Bob colours vertex e in a colour used in K_b or K_c, Alice wins. If he uses a new colour, Alice applies that colour to K_b and wins.
- If Bob colours a vertex in K_d, Alice colours another vertex in K_d. If K_d was already fully coloured or if K_d is now fully coloured, Alice wins. Otherwise repeat this state.

State 4: K_d is completely coloured, K_a and K_b are not completely coloured and vertex e is uncoloured. K_a and $K_c \vee K_d$ contain the same colours. Only the uncoloured vertices in K_b are at risk.

- If Bob colours vertex e or a vertex in K_b, Alice colours a vertex in K_b. If K_b is fully coloured, Alice wins. Otherwise repeat this state.
- If Bob colours a vertex in K_c, Alice applies the same colour to K_a and vice versa. If K_a is fully coloured, Alice wins. Otherwise repeat this state.

A strategy for E_6

Let G be an instance of E_6. Because $m, n \geq 2$, the clique number of G is $\omega(G) = m + n \geq \max\{m, n\} + 2$. The vertices at risk are possibly b and all vertices in K_m and K_n. Every vertex at risk can be neutralised by colouring two neighbours the same.

State 1 [start]: G is completely uncoloured. Vertices b, K_m and K_n are at risk.

- If Bob colours vertex a, Alice applies the same colour to a vertex in K_m and vice versa. Proceed to state 4.
- If Bob colours vertex b, Alice applies the same colour to a vertex in K_n and vice versa. Proceed to state 3.
- If Bob colours vertex c, Alice applies the same colour to vertex a. Proceed to state 2.

State 2: Vertices a and c are coloured the same, all other vertices are uncoloured. The vertices in K_m and K_n are at risk.

- If Bob colours vertex b, Alice applies the same colour to a vertex in K_n and vice versa. Alice wins on her next move by making sure that a vertex in K_m has the same colour as vertex a.
- If Bob colours a vertex in K_m in the colour of vertex a, Alice colours another vertex in K_m in a new colour. If Bob colours a vertex in K_m in a new colour, Alice colours another vertex in K_m in the colour of vertex a. In both cases Alice wins if K_m is completely coloured. Otherwise proceed to state 4.

State 3: Vertex b is coloured in a colour from K_n, K_n is partially coloured, a and K_m are uncoloured. Only the uncoloured vertices in K_n are still at risk.

- If Bob colours vertex a, Alice wins by applying the same colour to a vertex in K_m and vice versa.
- If Bob colours vertex c or a vertex in K_n, Alice colours another vertex in K_n. If K_n was already fully coloured or if K_n is now fully coloured, Alice wins. Otherwise repeat this state.

State 4: Vertices b and K_n are uncoloured, K_m is partially coloured, a is coloured in a colour from K_m. Only the uncoloured vertices in K_m are still at risk.

- If Bob colours vertex b, Alice wins by applying the same colour to a vertex in K_n and vice versa.

- If Bob colours vertex c or a vertex in K_m, Alice colours another vertex in K_m. If K_m was already fully coloured or if K_m is now fully coloured, Alice wins. Otherwise repeat this state.

A strategy for E_7

Let G be an instance of E_7. The only vertices of G that are at risk are a and c. Alice's strategy is to ensure that both of them are coloured after two rounds. If Bob starts with vertex a, Alice wins by applying the same colour to vertex c and vice versa. If Bob starts with another vertex, Alice responds with vertex a in a different colour. If Bob's second move is vertex c, Alice wins immediately. If Bob's second move is another vertex, Alice can colour vertex c herself as at most two neighbouring vertices have been coloured and $\omega(G) \geq 3$.

A strategy for E_8

Let G be an instance of E_8. The vertices at risk are a, c and every vertex in K_n. Let BR denote the bottom row of vertices and let p denote the right-most vertex in BR that may or may not be adjacent to c. We play the game with $n+1$ colours.

Alice adheres to the following principle: she makes sure that K_n and BR always contain the same colours until K_n contains $n-1$ colours or BR is fully coloured, whichever case arises first. Then she proceeds to state 2 or 3 to play her winning end game.

State 1 [start]: Vertices a, c are uncoloured, K_n has two or more uncoloured vertices, BR is not fully coloured. BR only contains colours from K_n. Vertices a, c and the uncoloured vertices in K_n are at risk.

- If Bob colours vertex a, Alice applies the same colour to vertex c and vice versa. As this neutralises K_n, Alice wins.
- If Bob colours a vertex in K_n, Alice applies the same colour to a vertex in BR. If BR is now fully coloured, proceed to state 3. If BR is not fully coloured and K_n now contains exactly one uncoloured vertex, proceed to state 2. Otherwise repeat this state.
- If BR contains exactly one uncoloured vertex and Bob colours it in a colour already found in K_n, Alice colours a vertex in K_n in a new colour. If Bob instead uses a new colour, Alice applies the same colour to a vertex in K_n. In both cases proceed to state 3.
- BR contains two or more uncoloured vertices and Bob colours one in a colour already found in K_n, Alice colours another vertex in BR with the same colour. If BR is now fully coloured, proceed to state 3. Otherwise repeat this state. If instead Bob uses a new colour, Alice applies the same colour to K_n. If K_n now contains exactly one uncoloured vertex, proceed to state 2. Otherwise repeat this state.

State 2: Vertices a, c are uncoloured, K_n has exactly one uncoloured vertex, BR is not fully coloured. BR only contains colours from K_n. Vertices a, c and the uncoloured vertex in K_n are at risk.

- If Bob colours vertex a, Alice wins by applying the same colour to vertex c and vice versa.

- If Bob colours the last vertex in K_n, Alice colours vertex a in its last legal colour and wins as vertex c can now only take the colour of a and the bottom row cannot surround c any more.
- If Bob colours vertex p in a new colour, Alice colours vertex c in its last legal colour. Vertex a cannot be surrounded any more as none of the remaining vertices in BR nor the vertex in K_n can be coloured in the colour of vertex c. K_n cannot be surrounded any more as vertex a cannot be coloured in any colour of BR and must take the colour of vertex c. It follows that Alice wins.
- If Bob colours a vertex in BR other than p in a new colour, Alice colours vertex a in its last legal colour. Alice wins for the same reason as above.
- If BR contains exactly one uncoloured vertex and Bob colours it in a colour already used in BR, Alice wins by colouring the last uncoloured vertex in K_n.
- If BR contains two or more uncoloured vertices and Bob colours one of them in a colour already used in BR, Alice applies the same colour to another vertex in BR. If BR is now fully coloured, proceed to state 3. Otherwise repeat this state.

State 3: Vertices a, c are uncoloured, K_n is not fully coloured, BR is fully coloured with colours from K_n. Only the uncoloured vertices in K_n are at risk.

- If Bob colours vertex a, Alice wins by applying the same colour to vertex c and vice versa.
- If Bob colours a vertex in K_n, Alice colours another vertex in K_n. If K_n was already fully coloured or if K_n is now fully coloured, Alice wins. Otherwise repeat this state.

A strategy for E_9

Let G be an instance of E_9. We play the vertex colouring game on G with $n+1$ colours. If Bob starts with vertex d, Alice applies the same colour to a vertex in K_n. As no other vertex in G can be coloured in that colour any more, this reduces the situation to a game with n colours on an instance of E_{12} (if $n = 2$) or E_8 (if $n \geq 3$).

We can now assume that Bob starts with a vertex that is not d. The following additions to Alice's strategy for E_8 allow her to win on G.

States 1 and 2: If Bob colours d, Alice colours a vertex in BR in a colour already found in BR. If BR is now fully coloured, proceed to state 3. Otherwise repeat the respective state.

State 3: If Bob colours d, Alice colours a vertex in K_n. If K_n is full, Alice wins. Otherwise repeat state 3.

A strategy for E_{10}

Let G be an instance of E_{10}. We first deal with the case $m = n = 1$. Then G is a C_4 induced by vertices a, b, c and K_n with a pendant K_m attached to vertex a and the clique number of G is $\omega(G) = 2$. Alice's strategy is straightforward: If Bob starts with vertex a in colour 1, Alice colours vertex b in colour 2 and vice versa. If Bob starts with K_m in colour 1, Alice colours K_n in colour 2 and vice versa.

If $m \geq 2$ and $m > n \geq 1$, Alice's strategy is even simpler. Only vertex a is at risk, so Alice wins by making sure that a is coloured after the first round, either by Bob or by herself.

We can now assume that $n \geq m \geq 1, n \geq 2$. The vertices at risk are a, b and c as well as all of K_n. Alice's strategy consists of three sets of instructions. States 1 and 2 concern the opening game and allow Alice to win or ensure that a and a vertex in K_n are coloured the same after at most two rounds. State 3 describes how she should proceed in subsequent rounds of the game.

State 1 [start]: G is completely uncoloured.

- If Bob colours b, Alice applies the same colour to c and vice versa. She wins on her next move by making sure that a is coloured either by Bob or Alice herself.
- If Bob colours a vertex in K_m, Alice colours vertex a in a different colour. Proceed to state 2.
- If Bob colours a, Alice applies the same colour to K_n and vice versa. Proceed to state 3.

State 2: Vertex a is coloured, one vertex in K_m is coloured, all other vertices are uncoloured. Vertices b, c and K_n are still at risk.

- If Bob colours b, Alice wins by applying the same colour to c and vice versa.
- If Bob colours another vertex in K_m, Alice applies the colour of a to K_n. Proceed to state 3.
- If Bob colours a vertex in K_n in the colour of a, Alice colours a second vertex in K_n in any colour. If Bob instead colours a vertex in K_n in a different colour, Alice colours a second vertex in K_n in the colour of a. Such a move is possible as K_n contains at least two uncoloured vertices. If K_n is now fully coloured, Alice wins. Otherwise proceed to state 3.

State 3: Vertices b, c are uncoloured, vertex a and a vertex in K_n are coloured the same, K_n is not completely coloured. Only the uncoloured vertices in K_n are still at risk.

- If Bob colours b, Alice wins by applying the same colour to c and vice versa.
- If Bob colours a vertex in K_m or K_n, Alice colours a vertex in K_n. If K_n was already fully coloured or if K_n is now fully coloured, Alice wins. Otherwise repeat this state.

A strategy for E_{11}

Let G be an instance of E_{11}. Due to symmetry it is sufficient to distinguish between the two cases $m = n$ and $m > n$.

If $m = n$, Alice adheres to the following strategy below. In general, Alice makes sure that K_m and K_n contain the same colours throughout the game. Similarly, if Bob colours a, Alice applies the same colour to b and vice versa. The same applies to c and d. The two exceptions to these rules are in states 2 and 4, which allow Alice to win after at most two moves. Note that Alice might start her strategy with state 1 or 2, depending on whether $m = 1$ or $m \geq 2$.

State 1 [start if $m \geq 2$]: Vertices a, b, c, d are uncoloured and K_n, K_m each have two or more uncoloured vertices. All uncoloured vertices are at risk.

- If Bob colours a vertex in K_m, Alice applies the same colour to K_n and vice versa. If K_n and K_m now have exactly one uncoloured vertex each, proceed to state 2. Otherwise repeat this state.

- If Bob colours the vertex a, Alice applies the same colour to b. If K_n and K_m now have exactly two uncoloured vertex each, proceed to state 3. Otherwise proceed to state 4.

State 2 [start if $m = 1$]: Vertices a, b, c, d are uncoloured and K_n, K_m each have exactly one uncoloured vertex. All uncoloured vertices are at risk.

- If Bob colours the last vertex of K_m, Alice wins by colouring the last vertex of K_n in a different colour and vice versa.
- If Bob colours vertex a, Alice colours vertex c in the only possible colour. Vertex d now only admits the colour of c and vertex b only admits the colour of a. It follows that Alice wins.

State 3: Vertices a, b are coloured, c, d are uncoloured and K_m, K_n each have exactly two uncoloured vertices. Vertices c, d and the uncoloured vertices in K_n are at risk.

- If Bob colours c, Alice wins by applying the same colour to d and vice versa.
- If Bob colours a vertex in K_m, Alice completes the colouring of K_m. She is now one round away from winning: if Bob next colours vertex c, she wins by applying the same colour to vertex d and vice versa. If Bob instead colours one of the two uncoloured vertices in K_n, Alice colours the other uncoloured vertex, making sure that the colour of a is present in K_n.
- If Bob colours a vertex in K_n, Alice completes the colouring of K_n, making sure that the colour of a is present in K_n. Alice wins because c, d and K_n have been neutralised.

State 4: Vertices a, b are coloured, c, d are not and K_m, K_n each have three or more uncoloured vertices. Vertices c, d and the uncoloured vertices in K_n are at risk.

- If Bob colours c, Alice wins by applying the same colour to d and vice versa.
- If Bob colours a vertex in K_m, Alice applies the same colour to a vertex in K_n. If K_m and K_n still have three or more uncoloured vertices each, repeat this state. Otherwise proceed to state 3.

If $m > n$, Alice's strategy is a little more complex but works by the same principle. The five possible states and accompanying rules for Alice are listed below. The game starts in state 1.

In every state the following rule applies: If Bob colours vertex a, Alice wins by applying the same colour to vertex b and vice versa.

State 1 [start]: Vertices a, b, c, d are uncoloured, K_m and K_n contain identical colours, K_m has two or more uncoloured vertices and K_n has fewer uncoloured vertices than K_m but at least one. Vertices a, b, c and the uncoloured vertices in K_m are at risk.

- If Bob colours c, Alice applies the same colour to d. If that colour was previously used in K_m, proceed to state 5. Otherwise proceed to state 3.
- If Bob colours a vertex in K_m, Alice responds with a vertex in the same colour in K_n and vice versa. If K_n is not fully coloured yet, repeat this state. If K_n is fully coloured, proceed to state 3.

State 2: Vertices a, b, c, d are uncoloured, K_m has at least one uncoloured vertex and K_n is fully coloured. Vertices a, b, c, d and the uncoloured vertices in K_m are at risk.

- If Bob colours vertex c with a colour from K_m, Alice applies the same colour to vertex d and vice versa. Proceed to state 5.

- Assume that K_m has exactly one uncoloured vertex:
  - If Bob colours vertex c in a colour not found in K_m, Alice colours vertex a in its only remaining legal colour. Vertex b now has $m-1$ neighbouring colours and cannot be surrounded any more, as neither vertex c nor d nor any vertices in K_m can take the colour of a. Similarly, d and the last vertex in K_m cannot be surrounded any more either. Alice wins.
  - If Bob colours the uncoloured vertex in K_m, Alice wins by colouring vertex a in the only colour possible.
- Assume K_m has exactly two uncoloured vertices:
  - If Bob colours one of the two uncoloured vertices in K_m with a new colour, Alice colours vertex c in the same colour and vice versa. She then wins on her next move: if Bob colours vertex a, Alice applies the same colour to vertex b and vice versa. If Bob instead colours the last vertex of K_m in any colour, Alice applies the same colour to d and vice versa.
- Should K_m have three or more uncoloured vertices:
  - If Bob colours vertex c in a colour not found in K_m, Alice applies the same colour to vertex d and vice versa. Proceed to state 4.
  - If Bob colours one of the vertices in K_m, Alice colours another. Repeat this state.

State 3: Vertices a, b are uncoloured, c, d are coloured in a colour not found in K_m, K_m has two or more uncoloured vertices and K_n has fewer uncoloured vertices than K_m but at least one. K_m and K_n contain identical colours. Vertices a, b and the uncoloured vertices in K_m are at risk.

- Assume K_m has three or more vertices:
  - If Bob colours a vertex in K_m, Alice applies the same colour to a vertex in K_n and vice versa. If K_n is now fully coloured, proceed to state 4. Otherwise repeat this state.
- Assume K_m has exactly two vertices:
  - If Bob colours one of the two uncoloured vertices in K_m, Alice makes sure that K_m contains the colour of vertices c and d when she colours another vertex in K_m. Alice wins because K_m is now fully coloured and vertices a, b are neutralised.
  - If Bob colours a vertex in K_n, Alice colours a vertex in K_m in the colour of vertex c. Proceed to state 5.

State 4: Vertices a, b are uncoloured, c, d are coloured in a colour not found in K_m, K_m has two or more uncoloured vertices and K_n is fully coloured. Vertices a, b and the uncoloured vertices in K_m are at risk.

- If Bob colours one of the two uncoloured vertices in K_m, Alice makes sure that K_m contains the colour of vertices c and d when she colours another vertex in K_m. Vertices a and b have now been neutralised. If K_m is fully coloured, Alice wins. Otherwise proceed to state 5.

State 5: Vertices a, b are uncoloured, c, d are coloured in a colour also found in K_m and K_m contains at least one uncoloured vertex. Only the uncoloured vertices in K_m are at risk.

- If K_m has only one uncoloured vertex and Bob colours it, Alice wins.

- If K_m has two uncoloured vertices and Bob colours one of them, Alice colours the other one to win.
- If K_m has more than two uncoloured vertices, Bob and Alice each colour a vertex in K_m and we return to this state again.

A strategy for E_{12} and E_{13}

See Section 4.3 in Chapter 4.

This concludes the proof of Lemma 73. □

Chapter 6

Outlook

6.1 Open problems

For four of Andres's six vertex colouring games, their respective class of game-perfect graphs has been successfully characterised. The characterisation of g_A, $[A, B]$ and $[B, B]$-perfect graphs can be found in [8], while the g_B-perfect graphs has been characterised in this thesis.

Two classes of graphs have resisted characterisation so far: the $[A, A]$ and the $[B, A]$-perfect graphs. Hence the following open problems are apparent.

Problem 76. Characterise $[A, A]$-perfect graphs explicitly or by means of forbidden (induced) subgraphs.

Problem 77. Characterise $[B, A]$-perfect graphs explicitly or by means of forbidden (induced) subgraphs.

Problem 76 seems particularly interesting, as the number of minimal forbidden induced subgraphs for the game $[A, A]$ is not finite. In particular, all odd anti-holes $\overline{C}_{2k+7}, k \geq 0$ are minimal forbidden induced subgraphs for $[A, A]$-perfect graphs [8]. It is not known whether the number of minimal forbidden subgraphs for $[A, A]$-perfect graphs is finite apart from the odd anti-holes or even whether F consists of a finite number of forbidden graphs and (infinite) forbidden graph classes such as the class of odd anti-holes.

Problem 78. Is the number of minimal forbidden subgraphs for $[A, A]$-perfect graphs finite apart from the odd anti-holes $\overline{C}_{2k+7}, k \geq 0$?

Problem 79. Does there exist another interesting infinite class of graphs that constitute a set of minimal forbidden subgraphs such as the class of odd anti-holes mentioned above?

Looking at the computational results obtained in Appendix C, we see that $[A, A]$ has an intriguing wealth of forbidden graphs ≤ 10 compared to the other five games. These results may aid the hunt for further forbidden graphs, especially if further (infinite) classes of graphs can be identified.

Using our computational results for the game $[B, A]$, it can be shown that the odd anti-holes $\overline{C}_7, \overline{C}_9$ and $\overline{C}_{11}$ are minimal forbidden subgraphs. This raises the following question.

Problem 80. Are all odd anti-holes $\overline{C}_{2k+7}, k \geq 0$ minimal forbidden subgraphs for $[B, A]$-perfect graphs?

We note that $[A, A]$ and $[B, A]$-perfect graphs do not admit the same structural decomposition used in our characterisation of g_B-perfect graphs. The 3-spider with thin legs, for example, has no dominating edge and is $[A, A]$-perfect. It remains open whether all $[A, A]$-perfect graphs

contain any dominating clique of size k for some $k \in \mathbb{N}$, perhaps allowing for an analogous but more complex decomposition.

Problem 81. Does there exist a constant $k \in \mathbb{N}$ such that all $[A, A]$ or $[B, A]$-perfect graphs have a dominating clique of size k?

6.2 Algorithmic challenges

If we know that a given graph G is an instance of $E_1^{\cup}$ or E_i, where $2 \leq i \leq 13$, we can easily turn Alice's strategy from Section 5.6 into an algorithm for Alice to win on G with $\omega(G)$ colours. This algorithm then runs in linear time $O(n)$, where $n = |G|$ is the number of vertices of G.

This motivates the following challenge.

Problem 82. Provide an efficient algorithm that identifies a given graph as an instance of $E_1^{\cup}$ or E_i, where $2 \leq i \leq 13$.

Of course, this problem is a strong version of the problem of determining whether a given graph is g_B-perfect. More generally we can pose the problem of determining whether a given graph is X-perfect for any of Andres's colouring games.

Problem 83. Provide an efficient algorithm that determines whether a given graph is game-perfect for any of Andres's six vertex colouring games.

A promising angle on Problem 82 is the observation that every class of graphs $E_i, 1 \leq i \leq 13$, has a relatively simple structure. For every E_i, it might be possible to employ a modular decomposition technique first presented by Gallai [26]. This technique recursively decomposes a graph into components that correspond to maximal modules, determining a unique encoding of G into a hierarchical tree. Given a graph class $E_i, 1 \leq i \leq 13$, the unique tree encoding for every G in E_i should be essentially the same. This modular decomposition is possible in linear time [35].

Appendix A

The vertex colouring game implemented as a backtracking algorithm

The following implementation of a simple backtracking algorithm determines the outcome of all six of Andres's vertex colouring games under the assumption that both Alice and Bob play optimally. Written in Python, it exhaustively plays all possible moves allowed in the game X with k colours. This allows us to test whether a given graph G is X-nice by simulating the game X on G with $\omega(G)$ colours.

The implementation works as a standalone program or as a function that can be called in a different script. This second usage is demonstrated in Appendix B. The code is included as **game.py** on the CD accompanying this thesis. The focus of the code lies on readability, not speed, although the canonical pruning technique used significantly improves computation time compared to an entirely brute-force approach.

A.1 Basic principles

We first describe the algorithm without the pruning technique presented in A.2. We will refer to **game.py**'s **_move()** function below to highlight the relevant code during the explanation.

In principle, the algorithm works by playing through every single possible colouring sequence, working through what is essentially a search tree. The leaves of the search tree consist of all possible full graph colourings and all possible partial graph colourings with one surrounded vertex. We work backwards from these leaves in order to determine which branch will allow the player to win. At any stage in the tree, if Alice finds a subtree that will allow her to complete the graph colouring, she will choose it. Similarly, if Bob finds a subtree that will allow him to win, he will choose that one.

We start the game by calling **_move(player, game_state)**, passing the first player and an uncoloured graph as parameters. The function **_move()** simulates the decision process of Alice or Bob when considering which branch to follow. At any point in the game, a player will generally be able to colour multiple vertices in multiple legal colours. For each possible vertex and colour, he or she colours it and calls **_move()** to pass on the game to the other player [lines 101-108]. If the current player is allowed to skip, he or she also calls **_move()** and passes on the game to the next player without colouring a vertex [lines 89-91].

Every call of **_move()** continues this recursive process until the graph colouring hits one of two possible base cases: If the graph is fully coloured, the function returns *True* [lines 74-75]. If a player detects that the graph contain a vertex that is *surrounded*, i.e. a vertex that cannot be coloured any more, the functions returns *False* [lines 78-80].

Assume the player is Alice. If one of her **_move()** calls return *True*, she also returns *True* [lines 96-98, 109-111], otherwise she returns *False* [line 116]. Bob does the exact opposite: if any of his child calls return *False*, he also returns *False*, otherwise he returns *True*.

If the root **_move()** call returns *True*, Alice wins, otherwise Bob does.

A.2 Pruning

The number of calls grows extremely quickly even for relatively small graphs. Because of this, we prune the search tree to stop Alice and Bob from following branches where the outcome has essentially been determined.

A first observation is that we can limit the number of colours available to colour a vertex if fewer than $k-1$ colours have been used, allowing us to curtail the number of child calls that a player makes. For any partial colouring of G we can assume that it contains consecutive colours from 1 to $i, i < k$; if necessary, by relabelling colours. Throughout the game, we keep track of the largest colour l used. If a player uses a colour $> l$ on a vertex, we can relabel the resulting colouring to one with consecutive colours up to $l+1$. For that reason we restrict each player to colours up to $l+1$ in the first place.

Secondly, we employ a canonization technique to detect branches of the search tree we can prune. A partial vertex colouring with i colours divides the set of vertices into $i+1$ partitions: one partition for each colour and one partition for the uncoloured vertices. We say that two partial colourings are *isomorphic* or *essentially the same* if the partitions they define are identical. We define a canonization function such that two partial colourings are mapped to the same canonical colouring if and only if they are essentially the same. When given a partial colouring as a Python list, we can achieve such a function by relabelling the colours in such a way that they appear consecutively in the list. Example: [0,2,1,0,2] becomes [0,1,2,0,1]. This is implemented as **_canonize()** in **game.py** and shown in Appendix A. Whenever a function call returns *True* or *False*, it stores this result in a hash table, using the current player and the canonised colouring as the key. As soon as the same player follows another branch and is faced with essentially the same graph colouring, they can retrieve the result from the hash table and save the program from traversing this branch.

A.3 The file *game.py*

```
1  #!/usr/bin/env python
2  #file: game.py
3  #author: Edwin Lock
4  #date: 22 May, 2016
5
6  """
7  An implementation of a simple backtracking algorithm to play all six variations
8  [A,-], [B,-], [A,A], [A,B], [B,A],[B,B] of Bodlaender's vertex colouring game
9  presented by Stephan Dominique Andres in [0]. It uses a pruning technique to cut
10 down the number of branches to follow in the recursive search tree.
```

```
[0]: Stephan Dominique Andres. On characterizing game-perfect graphs by
forbidden induced subgraphs. Contributions to Discrete Mathematics, 7(1), 2012.

How to use:
A) To use standalone, run "python game.py FirstPlayer SkippingPlayer graph k",
where graph should be given as an adjacency list and k is the number of
colours to play the game with. FirstPlayer can be "Alice" or "Bob,
SkippingPlayer can be "Alice", "Bob" or "-".
Example: "python game.py Bob - [[3,4],[4],[],[0],[0,1]] 2".

B) To use in another program, import this file and run play(g,k,player1,
skipping_player). player1 should be True or False, for Alice and Bob,
respectively. skipping_player should be True, False or None.
Example: import game; game.play([[3,4],[4],[],[0],[0,1]],2, False, None).
"""

def play(g, k, player1 = True, skipping_player = None):
    """
    Provides the starting point for the colouring game and calls _move() to
    start off player one. If player1 and skipping_player are not given, it
    defaults to Bodlaender's original game where Alice starts and neither Alice
    nor Bob may skip moves.
    Args:
        adjacency list of graph g: [[neighbours of vertex 0],
                                    [neighbours of vertex 1], ... ]
        integer k: the number of colours to play with
        (optional) boolean player1: Alice = True, Bob = False. Defaults to True
        (optional) skipping_player: Alice = True, Bob = False, Defaults to None.
    Returns:
        True if Alice wins, False if Bob wins.
    """
    #hashtable for pruning
    hashtable = {}

    #prepare initial game state
    game_state = _create_game_state(g, k, skipping_player)

    #start game with first move
    result = _move(player1, game_state, hashtable)
    return result

def _move(player, game_state, hashtable):
    """
    Plays a single move of a player in the graph colouring game. The player
    colours every possible vertex colouring in turn and hands over the result to
    the other player.
    If all vertices are coloured, Alice wins down this path. If a vertex has
    been surrounded, Bob wins on this path.
    Args:
        boolean player: True (Alice) or False (Bob),
        tuple game_state: (adjacency_list, colouring, uncoloured_vertices,
```

```
                        legal_colours, largest_colour, skipping_player)
        dict hashtable: previous (player,game_states) tuples for pruning
    Returns:
        boolean: True if Alice wins down this path, otherwise False.
    """
    #unpack values from tuple
    uncoloured_vertices = game_state[2]
    legal_colours = game_state[3]
    largest_colour = game_state[4]
    skipping_player = game_state[5]

    #If all vertices are coloured, Alice wins down this path
    if not uncoloured_vertices:
        return True

    #If a vertex is surrounded, return False and update hashtable
    if _has_surrounded_vertices(game_state):
        _update_hashtable(player, game_state, hashtable, False)
        return False

    #Canonical pruning: If (player, game_state) is recorded in hash table,
    #return previous result, otherwise just continue
    try: return _check_hashtable(player, game_state, hashtable)
    except KeyError: pass

    #if player is allowed to skip, hand over game without doing anything,
    #then proceed as normal
    if player == skipping_player:
        new_game_state = _duplicate_game_state(game_state)
        result = _move(not player, new_game_state, hashtable)
        #If Alice (player = True) finds a child that returns True, she
        #also returns True. If Bob (player = False) finds a child that
        #return False, he also returns False.
        #In both cases: if result == player, return player.
        if result == player:
            _update_hashtable(player, game_state, hashtable, result)
            return result

    #colour all possible vertices with all (sensible) possible colours
    for vertex in uncoloured_vertices:
        for colour in legal_colours[vertex]:
            #due to symmetry any larger colour is equivalent.
            if colour <= largest_colour+1:
                #colour vertex and update game state
                new_game_state = _update_game_state(game_state, vertex, colour)
                #hand over game to other player with new game state
                result = _move(not player, new_game_state, hashtable)
                if result == player:
                    _update_hashtable(player, game_state, hashtable, result)
                    return result
    #If neither Alice nor Bob find a child branch that allows them to win,
```

```
113         #they return True for Bob or False for Alice.
114         result = not player
115         _update_hashtable(player, game_state,hashtable, result)
116         return result
117 
118     def _canonize(colouring, largest_colour):
119         """
120         Takes a partially coloured graph and relabels its colour sets so that they
121         are sorted in ascending order according to the graph's vertex ordering.
122 
123         This canonizes the colouring of a graph, i.e. if two graphs have isomorphic
124         colour sets, the result after canonising is identical.
125 
126         Why is this useful? We can prune large parts of the tree that develop when
127         we run my game algorithm as what is essentially the same branch does not
128         need to be traversed again.
129         """
130         copy = list(colouring)
131         n = len(copy)
132         highest = 0
133         i = 0
134         while i < n:
135             v = copy[i]
136             if v > highest:
137                 highest += 1
138                 if v > highest:
139                     for j in range(i, n):
140                         if copy[j] == v:
141                             copy[j] = highest
142                         elif copy[j] == highest:
143                             copy[j] = v
144             if highest == largest_colour-1:
145                 break
146             i += 1
147         return tuple(copy)
148 
149     def _update_hashtable(player, game_state, hashtable, result):
150         """
151         Updates hashtable with an entry (player, game_state) and its branch result.
152         """
153         colouring = game_state[1]
154         largest_colour = game_state[4]
155         canonical = _canonize(colouring, largest_colour)
156         hashtable[(player, canonical)] = result
157 
158     def _check_hashtable(player, game_state, hashtable):
159         """ checks whether (player, game_state) has already occurred previously """
160         colouring = game_state[1]
161         largest_colour = game_state[4]
162         canonical = _canonize(colouring, largest_colour)
163         try:
```

```
164             return hashtable[(player, canonical)]
165         except:
166             raise KeyError("This combination of player and game state not found.")
167 
168 def _has_surrounded_vertices(game_state):
169     """ checks to see whether any vertices have been surrounded """
170     uncoloured_vertices = game_state[2]
171     legals = game_state[3]
172     for vertex in uncoloured_vertices:
173         if not legals[vertex]:
174             return True
175     return False
176 
177 def _create_game_state(g, k, skipping_player):
178     adjacency_list = g
179     n = len(g)
180     colouring = [0]*n
181     uncoloured_vertices = range(n)
182     legal_colours = [range(1,k+1) for i in range(n)]
183     largest_colour = 0
184     return (adjacency_list, colouring, uncoloured_vertices,
185             legal_colours, largest_colour, skipping_player)
186 
187 def _update_game_state(game_state, vertex, colour):
188     """
189     colours in a vertex with given colour and updates game state accordingly
190     """
191     adjacency_list = game_state[0]
192     colouring = game_state[1]
193     uncoloured_vertices = game_state[2]
194     legal_colours = game_state[3]
195     largest_colour = game_state[4]
196     skipping_player = game_state[5]
197 
198     if colour == largest_colour + 1:
199         new_largest_colour = largest_colour + 1
200     else:
201         new_largest_colour = largest_colour
202 
203     #make copies to hand over
204     new_colouring = list(colouring)
205     new_legal_colours = [l[:] for l in legal_colours] #deep copy
206     new_uncoloured_vertices = list(uncoloured_vertices)
207 
208     #colour in vertex with colour
209     new_colouring[vertex] = colour
210     new_uncoloured_vertices.remove(vertex)
211 
212     #update legal colours of neighbours
213     for i in adjacency_list[vertex]:
214         try:
```

```
215                 new_legal_colours[i].remove(colour)
216             except ValueError: #colour already removed previously
217                 pass
218         return (adjacency_list, new_colouring, new_uncoloured_vertices,
219                         new_legal_colours, new_largest_colour, skipping_player)
220 
221     def _duplicate_game_state(game_state):
222         """ makes an exact copy of game state to be handed on to the next player """
223         adjacency_list = game_state[0]
224         colouring = game_state[1]
225         uncoloured_vertices = game_state[2]
226         legal_colours = game_state[3]
227         largest_colour = game_state[4]
228         skipping_player = game_state[5]
229 
230         #make copies to hand over
231         new_colouring = list(colouring)
232         new_uncoloured_vertices = list(uncoloured_vertices)
233         new_legal_colours = [l[:] for l in legal_colours] #deep copy
234         new_largest_colour = largest_colour
235 
236         return (adjacency_list, new_colouring, new_uncoloured_vertices,
237                         new_legal_colours, new_largest_colour, skipping_player)
238 
239     def _plot_game_state(game_state):
240         """
241         plots a graph reflecting the current game state using SageMath's plot()
242         function
243         """
244         adjacency_list = game_state[0]
245         colouring = game_state[1]
246         largest_colour = game_state[4]
247 
248         vertex_colours = {i:[] for i in range(largest_colour+1)}
249         for index, value in enumerate(colouring):
250             vertex_colours[value].append(index)
251 
252         g = sage.all.Graph({i: adjacency_list[i] for i in range(len(adjacency_list))})
253 
254         g.show(vertex_colors = vertex_colours)
255 
256 
257     #if executed standalone:
258     import sys, ast
259     if __name__ == "__main__":
260         try:
261             player1 = str(sys.argv[1])
262             skipping_player = str(sys.argv[2])
263             g = ast.literal_eval(sys.argv[3])
264             k = int(sys.argv[4])
265         except:
```

```
266         sys.exit('''Please enter the first player [Alice or Bob], the skipping
267         player [Alice, Bob, or -], a graph as an adjacency list and the number
268         of colours to play the game with.
269 
270         Example: "python game.py Alice - [[1,2],[0,2],[0,1]] 3".
271         This plays the original game on the complete graph K3 using 3 colours.''')
272 
273     if player1 == "Alice":
274         player1 = True
275     elif player1 == "Bob":
276         player1 = False
277     else:
278         raise ValueError("I don't recognise the name '{0}'.".format(player1))
279 
280     if skipping_player == "Alice":
281         skipping_player = True
282     elif skipping_player == "Bob":
283         skipping_player = False
284     elif skipping_player == "-":
285         skipping_player = None
286     else:
287         raise ValueError("I don't recognise the name '{0}'.".format(skipping_player))
288 
289     result = play(g, k, player1, skipping_player)
290     if result:
291         print "Alice wins."
292     else:
293         print "Bob wins."
```

Appendix B

Systematically finding forbidden induced subgraphs

In order to generate forbidden induced subgraphs for each of Andres's vertex colouring games, we make use of the mathematics software *SageMath* [20], which provides built-in functions for testing induced subgraph isomorphism and calculating the clique number of a graph. Another useful features is the inclusion of *nauty*[33], a program that allows very quickly generation of all non-isomorphic graphs of order n for a reasonable value of n.[1]

The script used to compute all minimal forbidden induced subgraphs can be found in **forbidden.py** on the CD and consists of a single main function **generate()** and two helper functions. It makes use of **game.py** (see Appendix A) to test for X-niceness. We use this script to determine a minimal comprehensive set of forbidden induced subgraphs with up to 10 vertices for each of Andres's games in Appendix C.

B.1 The file *forbidden.py*

```
#!/usr/bin/env python
#file: forbidden.py
#author: Edwin Lock
#date: 22 May, 2016

import sage.all
import game

"""
This script utilises the SageMath mathematical software which provides
extensive support for graphs. In particular we use the integration of nauty,
a tool that allows us to (fairly) rapidly generate all non-isomorphic graphs
with k vertices.

SageMath is available at http://www.sagemath.org. The nauty package can be
added with 'sage -p nauty'.
```

[1] Very quickly as opposed to equivalent graph generators such as the native generator implemented in SageMath. Even with nauty it is still practically impossible to generate and test all graphs with more than 10 vertices comprehensively on a personal computer. Instead, one might customise the generator to only grow those graph structures one is interested in.

```
17
18  How to use:
19  This script must be run from within SageMath. The easiest way is to run
20  'sage -python forbidden.py Player1 SkippingPlayer n' from the command
21  line, where n is the largest order of graphs to be tested.
22
23  Alternatively, we can import the script from within Sage. Launch Sage from the
24  command line using the 'sage' command. Then load this script with
25  'load("path/to/forbidden.py"). The function generate() is now
26  available to use.
27  """
28
29  def generate(max_order, player1=True, skipping_player=None):
30      """
31      Generates and returns a set of minimal forbidden subgraphs (as Sage graph
32      objects) with up to n vertices. Defaults to Bodlaender's original game.
33      """
34      forbidden_graphs=[]
35      for n in range(max_order+1):
36          param = str(n)
37          #param = str(n)+" -c" if only connected graphs
38          #generate all graphs of order n
39          gen = sage.all.graphs.nauty_geng(param)
40          for g in gen:
41              #first make sure g doesn't contain a forbidden subgraph.
42              if _doesnt_contain_forbidden(g,forbidden_graphs):
43                  #test whether g is g-nice:
44                  clique_no = g.clique_number()
45                  g_adj = _to_adjacency_list(g)
46                  is_gnice = game.play(g_adj, clique_no, player1, skipping_player)
47                  if not is_gnice:
48                      forbidden_graphs.append(g)
49      return forbidden_graphs
50
51  def _to_adjacency_list(g):
52      """
53      Turns a SageMath graph object into an adjacency list that our implementation
54      of the game playing algorithm understands.
55      """
56      copy = g.copy()
57      copy.relabel()
58      d = copy.to_dictionary()
59      array = [None]*len(d)
60      for key in d:
61          array[key]=d[key]
62      return array
63
64  def _doesnt_contain_forbidden(g,f):
65      """
66      Takes a SageMath graph object g and a set of graph objects f.
67      Tests whether the graph g contains any induced subgraph from f.
```

```
68      """
69      for s in f:
70          subg = g.subgraph_search(s, induced=True)
71          if subg:
72              return False
73      return True
74
75  # if the script is called stand-alone
76  if __name__ == "__main__":
77      import sys
78
79      def _output_forbidden(forbidden_graphs):
80          """ Outputs all forbidden graphs as adjacency lists to stdout. """
81          for f in forbidden_graphs:
82              print _to_adjacency_list(f)
83
84      try:
85          player1 = str(sys.argv[1])
86          skipping_player = str(sys.argv[2])
87          n = int(sys.argv[3])
88      except:
89          sys.exit("\nPlease enter parameters as follows: first_player"
90          " [Alice, Bob], skipping_player [Alice, Bob, or -] and the maximum size"
91          " of graphs to be tested."
92
93          "\nExample: >> sage -python forbidden.py Alice - 7."
94          "This finds all forbidden induced subgraphs for the original game"
95          "with up to 7 vertices.")
96
97      #convert player1 and skipping_player for use with generate()
98      if player1 == "Alice":
99          player1 = True
100     elif player1 == "Bob":
101         player1 = False
102     else:
103         raise ValueError("I don't recognise the name '{0}'.".format(player1))
104
105     if skipping_player == "Alice":
106         skipping_player = True
107     elif skipping_player == "Bob":
108         skipping_player = False
109     elif skipping_player == "-":
110         skipping_player = None
111     else:
112         raise ValueError("I don't recognise the name '{0}'.".format(skipping_player))
113
114     forbidden_graphs = generate(n, player1, skipping_player)
115
116     print "{0} forbidden graphs were found:".format(len(forbidden_graphs))
117     _output_forbidden(forbidden_graphs)
```

Appendix C

Computational forbidden subgraph results for Andres's six vertex colouring games

We run our script for each of Andres's six vertex colouring games to find minimal forbidden induced graphs with up to 10 vertices. The unix tool *time* shows how long each computation took on a slightly below average laptop computer.[1]

C.1 A brief note on the results

The forbidden graphs identified for the games g_A, $[A, B]$ and $[B, B]$ correspond with those presented in [8]. The results for the game g_B are shown in Figure 3.1 and were used throughout this thesis.

Of more interest to future characterisations are the results for the games $[A, A]$ and $[B, A]$, they include forbidden subgraphs not previously found in literature. The 75 forbidden subgraphs yielded for the game $[A, A]$ in particular hint at a rich structure of $[A, A]$-perfect graphs.

The game $g_A = [A, -]$

```
$ time sage -python forbidden.py Alice - 10

5 forbidden graphs were found:
[[2, 3], [3], [0], [0, 1]]

[[2, 3], [2, 3], [0, 1], [0, 1]]

[[3, 4, 5], [3, 4, 5], [3, 4, 5], [0, 1, 2, 4, 5], [0, 1, 2, 3, 5], [0, 1, 2, 3,
↪  4]]

[[4, 6, 7], [4, 6, 7], [5, 6, 7], [5, 6, 7], [0, 1, 6, 7], [2, 3, 6, 7], [0, 1,
↪  2, 3, 4, 5, 7], [0, 1, 2, 3, 4, 5, 6]]
```

[1] Processor: 2.4 GHz Intel Core 2 Duo, Memory: 8 GB 1067 MHz DDR3, Disk: Samsung SSD 840 EVO.

```
[[6, 8], [6, 8], [6, 8], [7, 9], [7, 9], [7, 9], [0, 1, 2, 8], [3, 4, 5, 9], [0,
↪  1, 2, 6], [3, 4, 5, 7]]

real        347m54.052s
user        329m4.419s
sys        1m23.774s
```

The game $[A, A]$

```
$ time sage -python forbidden.py Alice Alice 10

75 forbidden graphs were found:
[[3, 4], [4], [4], [0], [0, 1, 2]]

[[2, 4], [3, 4], [0], [1], [0, 1]]

[[2, 3], [3, 4], [0, 4], [0, 1], [1, 2]]

[[3, 4, 5], [3, 4, 5], [3, 4, 5], [0, 1, 2, 4, 5], [0, 1, 2, 3, 5], [0, 1, 2, 3,
↪  4]]

[[3, 4, 5, 6], [4, 5, 6], [6], [0, 5, 6], [0, 1, 5, 6], [0, 1, 3, 4], [0, 1, 2,
↪  3, 4]]

[[3, 4, 5, 6], [4, 5, 6], [5, 6], [0, 5, 6], [0, 1, 5, 6], [0, 1, 2, 3, 4], [0,
↪  1, 2, 3, 4]]

[[3, 6], [4, 5, 6], [4, 5, 6], [0, 6], [1, 2, 5, 6], [1, 2, 4], [0, 1, 2, 3, 4]]

[[3, 4, 5, 6], [3, 4, 5, 6], [5, 6], [0, 1, 5, 6], [0, 1, 6], [0, 1, 2, 3], [0,
↪  1, 2, 3, 4]]

[[3, 4, 5, 6], [3, 4, 5, 6], [5, 6], [0, 1, 5], [0, 1, 6], [0, 1, 2, 3, 6], [0,
↪  1, 2, 4, 5]]

[[3, 4, 5, 6], [3, 4, 5, 6], [6], [0, 1, 5, 6], [0, 1, 5, 6], [0, 1, 3, 4], [0,
↪  1, 2, 3, 4]]

[[3, 4, 5, 6], [3, 4, 5, 6], [5, 6], [0, 1, 5, 6], [0, 1, 5, 6], [0, 1, 2, 3,
↪  4], [0, 1, 2, 3, 4]]

[[3, 4, 5, 6], [3, 4, 5, 6], [4, 6], [0, 1, 5], [0, 1, 2, 6], [0, 1, 3], [0, 1,
↪  2, 4]]

[[3, 4, 5, 6], [3, 4, 5, 6], [4, 6], [0, 1, 5, 6], [0, 1, 2, 6], [0, 1, 3], [0,
↪  1, 2, 3, 4]]

[[3, 4, 5, 6], [3, 4, 5, 6], [4, 5, 6], [0, 1, 5, 6], [0, 1, 2, 6], [0, 1, 2,
↪  3], [0, 1, 2, 3, 4]]
```

```
[[3, 4, 5, 6], [3, 4, 5, 6], [4, 5, 6], [0, 1, 6], [0, 1, 2, 5, 6], [0, 1, 2,
↪  4], [0, 1, 2, 3, 4]]

[[3, 4, 5, 6], [3, 4, 5, 6], [5, 6], [0, 1, 4, 5], [0, 1, 3, 6], [0, 1, 2, 3,
↪  6], [0, 1, 2, 4, 5]]

[[3, 4, 5, 6], [3, 5, 6], [4, 5, 6], [0, 1, 4, 5], [0, 2, 3, 6], [0, 1, 2, 3,
↪  6], [0, 1, 2, 4, 5]]

[[3, 4, 5, 6], [3, 4, 5, 6], [4, 5, 6], [0, 1, 4, 5], [0, 1, 2, 3, 6], [0, 1, 2,
↪  3, 6], [0, 1, 2, 4, 5]]

[[3, 4, 5, 6], [3, 4, 5, 6], [3, 4, 5, 6], [0, 1, 2, 5, 6], [0, 1, 2, 6], [0, 1,
↪  2, 3], [0, 1, 2, 3, 4]]

[[3, 4, 5, 6], [3, 4, 5, 6], [3, 4, 5, 6], [0, 1, 2, 5, 6], [0, 1, 2, 5, 6], [0,
↪  1, 2, 3, 4], [0, 1, 2, 3, 4]]

[[2, 3, 4, 6], [3, 4, 5, 6], [0, 4, 5, 6], [0, 1, 5, 6], [0, 1, 2], [1, 2, 3],
↪  [0, 1, 2, 3]]

[[2, 3, 4, 5], [3, 4, 5, 6], [0, 4, 5, 6], [0, 1, 5, 6], [0, 1, 2, 6], [0, 1, 2,
↪  3], [1, 2, 3, 4]]

[[4, 6, 7], [5, 6, 7], [5, 6, 7], [5, 6, 7], [0, 6, 7], [1, 2, 3, 7], [0, 1, 2,
↪  3, 4], [0, 1, 2, 3, 4, 5]]

[[4, 6, 7], [5, 6, 7], [5, 6, 7], [5], [0, 6, 7], [1, 2, 3, 6, 7], [0, 1, 2, 4,
↪  5, 7], [0, 1, 2, 4, 5, 6]]

[[4, 6, 7], [5, 6, 7], [5, 6, 7], [5, 7], [0, 6, 7], [1, 2, 3, 6, 7], [0, 1, 2,
↪  4, 5, 7], [0, 1, 2, 3, 4, 5, 6]]

[[4, 6, 7], [4, 6, 7], [5, 6, 7], [5, 6, 7], [0, 1, 6, 7], [2, 3, 6, 7], [0, 1,
↪  2, 3, 4, 5], [0, 1, 2, 3, 4, 5]]

[[4, 6, 7], [4, 6, 7], [5, 6, 7], [5, 6, 7], [0, 1, 6, 7], [2, 3, 6, 7], [0, 1,
↪  2, 3, 4, 5, 7], [0, 1, 2, 3, 4, 5, 6]]

[[3, 6], [4, 6, 7], [5, 7], [0, 6], [1], [2, 7], [0, 1, 3, 7], [1, 2, 5, 6]]

[[3, 5, 7], [4, 6, 7], [5, 6, 7], [0, 5, 7], [1, 6, 7], [0, 2, 3, 6, 7], [1, 2,
↪  4, 5, 7], [0, 1, 2, 3, 4, 5, 6]]

[[3, 5, 6, 7], [4, 5, 6, 7], [4, 5, 6, 7], [0, 5, 6, 7], [1, 2, 6, 7], [0, 1, 2,
↪  3], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

[[5, 6, 7, 8], [5, 6, 7, 8], [5, 6, 7, 8], [7], [8], [0, 1, 2, 6], [0, 1, 2, 5],
↪  [0, 1, 2, 3, 8], [0, 1, 2, 4, 7]]
```

```
[[4, 6, 7], [5, 6, 7, 8], [5, 6, 7, 8], [8], [0, 6, 7], [1, 2], [0, 1, 2, 4, 8],
↪  [0, 1, 2, 4, 8], [1, 2, 3, 6, 7]]

[[4, 5, 8], [6, 7, 8], [6, 7, 8], [6, 7], [0, 5, 8], [0, 4, 8], [1, 2, 3, 7, 8],
↪  [1, 2, 3, 6, 8], [0, 1, 2, 4, 5, 6, 7]]

[[4, 7, 8], [5, 6, 7, 8], [5, 6, 7, 8], [5, 6, 7, 8], [0, 7, 8], [1, 2, 3, 7,
↪  8], [1, 2, 3], [0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]]

[[4, 7, 8], [5, 6, 7, 8], [5, 6, 7, 8], [5, 6, 7, 8], [0, 7, 8], [1, 2, 3, 6],
↪  [1, 2, 3, 5], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

[[4, 7, 8], [5, 6, 7, 8], [5, 6, 7, 8], [5, 6, 7], [0, 7, 8], [1, 2, 3, 6, 8],
↪  [1, 2, 3, 5, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 4, 5, 6, 7]]

[[4, 5, 6, 7, 8], [5, 6, 7, 8], [5, 6, 7, 8], [8], [0, 5, 6, 7, 8], [0, 1, 2, 4,
↪  8], [0, 1, 2, 4, 8], [0, 1, 2, 4], [0, 1, 2, 3, 4, 5, 6]]

[[4, 5, 6, 7, 8], [5, 6, 7, 8], [5, 6, 7, 8], [6, 7, 8], [0, 5, 6, 7, 8], [0, 1,
↪  2, 4, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4, 5, 6]]

[[4, 5, 6, 7, 8], [5, 6, 7, 8], [5, 6, 7, 8], [6, 7, 8], [0, 5, 6, 7, 8], [0, 1,
↪  2, 4], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4, 6, 7]]

[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [5, 6, 7, 8], [8], [0, 1, 5, 6, 7, 8], [0, 1,
↪  2, 4, 7], [0, 1, 2, 4, 8], [0, 1, 2, 4, 5], [0, 1, 2, 3, 4, 6]]

[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [5, 6, 7, 8], [6, 7, 8], [0, 1, 5, 6, 7, 8],
↪  [0, 1, 2, 4], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4, 6]]

[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [5, 6, 7, 8], [6, 7], [0, 1, 5, 6, 7, 8], [0,
↪  1, 2, 4], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 4, 6, 7]]

[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [5, 6, 7, 8], [6, 7, 8], [0, 1, 5, 6, 7, 8],
↪  [0, 1, 2, 4, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4, 5, 6]]

[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [5, 6, 7, 8], [6, 7, 8], [0, 1, 5, 6, 7, 8],
↪  [0, 1, 2, 4], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4, 6, 7]]

[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [5, 6, 7, 8], [5, 6, 7], [0, 1, 5, 6, 7, 8],
↪  [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4], [0, 1, 2, 4, 5, 6]]

[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [5, 6, 7, 8], [5, 6, 7, 8], [0, 1, 5, 6, 7,
↪  8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4,
↪  5, 6]]

[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 5, 6, 7], [8], [0, 1, 2, 7, 8], [0, 1, 2,
↪  7, 8], [0, 1, 2, 8], [0, 1, 2, 4, 5, 8], [0, 1, 3, 4, 5, 6, 7]]

[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 5, 6, 7], [7, 8], [0, 1, 2, 7, 8], [0, 1,
↪  2, 7, 8], [0, 1, 2, 8], [0, 1, 2, 3, 4, 5, 8], [0, 1, 3, 4, 5, 6, 7]]
```

```
[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 5, 6, 7], [8], [0, 1, 2, 6, 8], [0, 1, 2,
↪  7, 8], [0, 1, 2, 4, 8], [0, 1, 2, 5, 8], [0, 1, 3, 4, 5, 6, 7]]

[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 5, 6, 7], [7, 8], [0, 1, 2, 6, 8], [0, 1,
↪  2, 7, 8], [0, 1, 2, 4, 8], [0, 1, 2, 3, 5, 8], [0, 1, 3, 4, 5, 6, 7]]

[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 5, 6, 7], [5, 6, 7], [0, 1, 2, 7, 8], [0,
↪  1, 2, 3, 6, 8], [0, 1, 2, 3, 5, 8], [0, 1, 2, 3, 4, 8], [0, 1, 4, 5, 6, 7]]

[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 5, 6, 7], [4, 5, 6, 7], [0, 1, 2, 3, 7,
↪  8], [0, 1, 2, 3, 8], [0, 1, 2, 3, 8], [0, 1, 2, 3, 4, 8], [0, 1, 4, 5, 6,
↪  7]]

[[4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 5, 6, 7], [4, 5, 6, 7], [0, 1, 2, 3, 6,
↪  8], [0, 1, 2, 3, 7, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 5, 8], [0, 1, 4, 5,
↪  6, 7]]

[[3, 6, 7], [4, 6, 8], [5, 7, 8], [0, 6, 7], [1, 6, 8], [2, 7, 8], [0, 1, 3, 4,
↪  7, 8], [0, 2, 3, 5, 6, 8], [1, 2, 4, 5, 6, 7]]

[[3, 5, 8], [4, 6, 7], [6, 7, 8], [0, 5, 8], [1, 6, 7], [0, 3, 8], [1, 2, 4, 7,
↪  8], [1, 2, 4, 6, 8], [0, 2, 3, 5, 6, 7]]

[[3, 5, 6], [4, 7, 8], [5, 6, 7, 8], [0, 5, 6], [1, 7, 8], [0, 2, 3, 7, 8], [0,
↪  2, 3, 7, 8], [1, 2, 4, 5, 6], [1, 2, 4, 5, 6]]

[[3, 5, 7, 8], [4, 6, 8], [5, 6, 7, 8], [0, 5, 7, 8], [1, 6, 8], [0, 2, 3, 6,
↪  7], [1, 2, 4, 5, 7, 8], [0, 2, 3, 5, 6], [0, 1, 2, 3, 4, 6]]

[[3, 5, 6, 7, 8], [4, 6, 7, 8], [5, 6, 7, 8], [0, 5, 6, 7, 8], [1, 6, 7, 8], [0,
↪  2, 3, 7], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 6]]

[[3, 5, 6, 7, 8], [4, 5, 6, 7, 8], [6, 7, 8], [0, 5, 6, 7, 8], [1, 5, 6, 7, 8],
↪  [0, 1, 3, 4, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4, 5, 6]]

[[3, 5, 6, 7, 8], [4, 5, 6, 7, 8], [6, 7, 8], [0, 5, 6, 7, 8], [1, 5, 6, 7, 8],
↪  [0, 1, 3, 4], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4, 6, 7]]

[[3, 5, 6, 7], [4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [0, 5, 6, 7], [1, 2], [0, 1, 2,
↪  3, 8], [0, 1, 2, 3, 8], [0, 1, 2, 3, 8], [1, 2, 5, 6, 7]]

[[3, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [0, 5, 6, 7, 8], [1, 2, 8],
↪  [0, 1, 2, 3, 8], [0, 1, 2, 3, 8], [0, 1, 2, 3], [0, 1, 2, 3, 4, 5, 6]]

[[3, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [0, 5, 6, 7, 8], [1, 2, 7,
↪  8], [0, 1, 2, 3, 6], [0, 1, 2, 3, 5], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4,
↪  7]]
```

```
[[3, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 6, 7, 8], [0, 5, 6, 7, 8], [1, 2, 5, 6,
↪  7, 8], [0, 1, 3, 4], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4,
↪  6]]

[[3, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 6, 7, 8], [0, 5, 6, 7, 8], [1, 2, 5, 6,
↪  7, 8], [0, 1, 3, 4, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4,
↪  5, 6]]

[[3, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 6, 7, 8], [0, 5, 6, 7, 8], [1, 2, 5, 6,
↪  7, 8], [0, 1, 3, 4], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4,
↪  6, 7]]

[[3, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [0, 5, 6, 7, 8], [1, 2, 5,
↪  6, 7, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3,
↪  4, 5]]

[[3, 5, 6, 7, 8], [4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [0, 5, 6, 7, 8], [1, 2, 5,
↪  6, 7, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4], [0, 1, 2,
↪  3, 4, 5, 6]]

[[3, 4, 5, 6, 7], [4, 5, 6, 7, 8], [5, 6, 7, 8], [0, 4, 5, 6, 7], [0, 1, 3, 7,
↪  8], [0, 1, 2, 3, 6, 8], [0, 1, 2, 3, 5, 8], [0, 1, 2, 3, 4, 8], [1, 2, 4, 5,
↪  6, 7]]

[[3, 4, 5, 6, 7], [4, 5, 6, 7, 8], [4, 5, 6, 7, 8], [0, 4, 5, 6, 7], [0, 1, 2,
↪  3, 6, 8], [0, 1, 2, 3, 7, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 5, 8], [1, 2,
↪  4, 5, 6, 7]]

[[2, 3, 4, 5, 6, 7], [3, 4, 5, 6, 7, 8], [0, 4, 5, 6, 7, 8], [0, 1, 5, 6, 7, 8],
↪  [0, 1, 2, 6, 7, 8], [0, 1, 2, 3, 7, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4,
↪  5], [1, 2,
3, 4, 5, 6]]

[[6, 8], [6, 8], [6, 8], [7, 9], [7, 9], [7, 9], [0, 1, 2, 8], [3, 4, 5, 9], [0,
↪  1, 2, 6], [3, 4, 5, 7]]

[[4, 6, 7, 8, 9], [5, 9], [6, 7, 8, 9], [7, 8], [0, 6, 7, 8, 9], [1, 9], [0, 2,
↪  4], [0, 2, 3, 4, 9], [0, 2, 3, 4, 9], [0, 1, 2, 4, 5, 7, 8]]

[[4, 6, 7, 8, 9], [5, 9], [6, 7, 8, 9], [6, 7, 8, 9], [0, 6, 7, 8, 9], [1, 9],
↪  [0, 2, 3, 4, 8], [0, 2, 3, 4, 9], [0, 2, 3, 4, 6], [0, 1, 2, 3, 4, 5, 7]]

[[3, 6, 7, 8, 9], [4, 6, 7, 8, 9], [5, 9], [0, 6, 7, 8, 9], [1, 6, 7, 8, 9], [2,
↪  9], [0, 1, 3, 4, 8], [0, 1, 3, 4, 9], [0, 1, 3, 4, 6], [0, 1, 2, 3, 4, 5,
↪  7]]

real        1135m42.488s
user        1059m52.558s
sys         3m57.298s
```

The game $[A, B]$

```
$ time sage -python forbidden.py Alice Bob 10

5 forbidden graphs were found:
[[2, 3], [3], [0], [0, 1]]

[[2, 3], [2, 3], [0, 1], [0, 1]]

[[3, 4, 5], [3, 4, 5], [3, 4, 5], [0, 1, 2, 4, 5], [0, 1, 2, 3, 5], [0, 1, 2, 3,
↪  4]]

[[4, 6, 7], [4, 6, 7], [5, 6, 7], [5, 6, 7], [0, 1, 6, 7], [2, 3, 6, 7], [0, 1,
↪  2, 3, 4, 5, 7], [0, 1, 2, 3, 4, 5, 6]]

[[6, 8], [6, 8], [6, 8], [7, 9], [7, 9], [7, 9], [0, 1, 2, 8], [3, 4, 5, 9], [0,
↪  1, 2, 6], [3, 4, 5, 7]]

real        406m34.237s
user        356m6.363s
sys        1m22.497s
```

The game $g_B = [B, -]$

```
$ time sage -python forbidden.py Bob - 10

15 forbidden graphs were found:
[[3, 4], [4], [], [0], [0, 1]]

[[3, 4], [4], [4], [0], [0, 1, 2]]

[[3, 4], [3, 4], [], [0, 1], [0, 1]]

[[3, 4], [3, 4], [3, 4], [0, 1, 2, 4], [0, 1, 2, 3]]

[[2, 4], [3, 4], [0], [1], [0, 1]]

[[2, 3], [3, 4], [0, 4], [0, 1], [1, 2]]

[[2, 3, 4], [3, 4], [0, 4], [0, 1, 4], [0, 1, 2, 3]]

[[2, 3, 4], [2, 3, 4], [0, 1, 4], [0, 1, 4], [0, 1, 2, 3]]

[[4, 6], [5, 6], [5, 6], [5], [0, 6], [1, 2, 3, 6], [0, 1, 2, 4, 5]]

[[4, 6], [4, 6], [5, 6], [5, 6], [0, 1, 6], [2, 3, 6], [0, 1, 2, 3, 4, 5]]

[[3, 5], [4, 6], [5, 6], [0, 5], [1, 6], [0, 2, 3, 6], [1, 2, 4, 5]]

[[3, 5, 6], [4, 6], [5, 6], [0, 5, 6], [1, 6], [0, 2, 3], [0, 1, 2, 3, 4]]
```

```
[[3, 5, 6], [4, 5, 6], [6], [0, 5, 6], [1, 5, 6], [0, 1, 3, 4], [0, 1, 2, 3, 4]]

[[3, 5, 6], [4, 5, 6], [5, 6], [0, 5, 6], [1, 5, 6], [0, 1, 2, 3, 4], [0, 1, 2,
↪  3, 4]]

[[3, 5, 6], [4, 5, 6], [4, 5, 6], [0, 5, 6], [1, 2, 6], [0, 1, 2, 3], [0, 1, 2,
↪  3, 4]]

real        398m44.609s
user        370m36.084s
sys         1m40.220s
```

The game $[B, A]$

```
$ time sage -python forbidden.py Bob Alice 10

16 forbidden graphs were found:
[[3, 4], [4], [4], [0], [0, 1, 2]]

[[3, 4], [3, 4], [3, 4], [0, 1, 2, 4], [0, 1, 2, 3]]

[[2, 4], [3, 4], [0], [1], [0, 1]]

[[2, 3], [3, 4], [0, 4], [0, 1], [1, 2]]

[[4, 6], [5, 6], [5, 6], [5], [0, 6], [1, 2, 3, 6], [0, 1, 2, 4, 5]]

[[4, 6], [4, 6], [5, 6], [5, 6], [0, 1, 6], [2, 3, 6], [0, 1, 2, 3, 4, 5]]

[[3, 5], [4, 6], [5, 6], [0, 5], [1, 6], [0, 2, 3, 6], [1, 2, 4, 5]]

[[3, 4, 5, 6], [4, 5, 6], [6], [0, 5, 6], [0, 1, 5, 6], [0, 1, 3, 4], [0, 1, 2,
↪  3, 4]]

[[3, 4, 5, 6], [4, 5, 6], [5, 6], [0, 5, 6], [0, 1, 5, 6], [0, 1, 2, 3, 4], [0,
↪  1, 2, 3, 4]]

[[3, 6], [4, 5, 6], [4, 5, 6], [0, 6], [1, 2, 5, 6], [1, 2, 4], [0, 1, 2, 3, 4]]

[[3, 4, 5, 6], [3, 4, 5, 6], [5, 6], [0, 1, 5, 6], [0, 1, 6], [0, 1, 2, 3], [0,
↪  1, 2, 3, 4]]

[[3, 4, 5, 6], [3, 4, 5, 6], [6], [0, 1, 5, 6], [0, 1, 5, 6], [0, 1, 3, 4], [0,
↪  1, 2, 3, 4]]

[[3, 4, 5, 6], [3, 4, 5, 6], [5, 6], [0, 1, 5, 6], [0, 1, 5, 6], [0, 1, 2, 3,
↪  4], [0, 1, 2, 3, 4]]

[[2, 3, 4, 6], [3, 4, 5, 6], [0, 4, 5, 6], [0, 1, 5, 6], [0, 1, 2], [1, 2, 3],
↪  [0, 1, 2, 3]]
```

```
[[2, 3, 4, 5], [3, 4, 5, 6], [0, 4, 5, 6], [0, 1, 5, 6], [0, 1, 2, 6], [0, 1, 2,
↪  3], [1, 2, 3, 4]]

[[2, 3, 4, 5, 6, 7], [3, 4, 5, 6, 7, 8], [0, 4, 5, 6, 7, 8], [0, 1, 5, 6, 7, 8],
↪  [0, 1, 2, 6, 7, 8], [0, 1, 2, 3, 7, 8], [0, 1, 2, 3, 4, 8], [0, 1, 2, 3, 4,
↪  5], [1, 2, 3, 4, 5, 6]]

real        898m52.379s
user        848m49.826s
sys        2m38.561s
```

The game $[B, B]$

```
$ time sage -python forbidden.py Bob Bob 10

4 forbidden graphs were found:
[[2, 3], [3], [0], [0, 1]]

[[2, 3], [2, 3], [0, 1], [0, 1]]

[[3, 4], [3, 4], [3, 4], [0, 1, 2, 4], [0, 1, 2, 3]]

[[4, 6], [4, 6], [5, 6], [5, 6], [0, 1, 6], [2, 3, 6], [0, 1, 2, 3, 4, 5]]

real        477m48.122s
user        376m1.383s
sys        2m54.458s
```

Bibliography

[1] S. D. Andres. The game chromatic index of forests of maximum degree $\Delta \geq 5$. *Discrete Applied Mathematics*, 154(9):1317–1323, 2006.

[2] S. D. Andres. *Digraph Coloring Games and Game-Perfectness*. PhD thesis, Universität zu Köln, Verlag Dr. Hut, München, 11 2007.

[3] S. D. Andres. Game-perfect graphs. *Math. Methods Oper. Res.*, 69:235–250, 2009.

[4] S. D. Andres. The incidence game chromatic number. *Discrete Applied Math.*, 157(9):1980–1987, 2009.

[5] S. D. Andres. The positive lightness of digraphs, embeddable in a surface, without 4-cycles. *Discrete Math.*, 309:3594–3579, 2009.

[6] S. D. Andres. Directed defective asymmetric graph coloring games. *Discrete Applied Math.*, 158:251–260, 2010.

[7] S. D. Andres. Game-perfect digraphs. *Math. Meth. Oper. Res.*, 76:321–341, 2012.

[8] S. D. Andres. On characterizing game-perfect graphs by forbidden induced subgraphs. *Contributions to Discrete Math.*, 7(1):21–34, 2012.

[9] S. D. Andres, W. Hochstättler, and C. Schallück. The game chromatic index of wheels. *Discrete Applied Math.*, 159(16):1660–1665, 2011. 8th Cologne/Twente Workshop on Graphs and Combinatorial Optimization (CTW 2009).

[10] T. Bartnicki and J. Grytczuk. A note on the game chromatic index of graphs. *Graphs and Combinatorics*, 24(2):67–70, 2008.

[11] C. Berge. Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind. *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe*, 10:114, 1961.

[12] A. Beveridge, T. Bohman, A. Frieze, and O. Pikhurko. Game chromatic index of graphs with given restrictions on degrees. *Theoretical Computer Science*, 407(1):242–249, 2008.

[13] H. L. Bodlaender. On the complexity of some coloring games. In R. H. Möhring, editor, *Graph-Theoretic Concepts in Computer Science: 16th International Workshop WG '90 Berlin, Germany, June 20–22, 1990 Proceedings*, pages 30–40, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[14] L. Cai and X. Zhu. Game chromatic index of k-degenerate graphs. *Journal of Graph Theory*, 36(3):144–155, 2001.

[15] W. H. Chan and G. Nong. The game chromatic index of some trees of maximum degree 4. *Discrete Applied Mathematics*, 170:1–6, 2014.

[16] W. Chan, W. Shiu, and X. Zhu. The strong game colouring number of directed graphs. *Discrete Math.*, 313:1070–1077, 2013.

[17] C.-Y. Chou, W. Wang, and X. Zhu. Relaxed game chromatic number of graphs. *Discrete Math.*, 262:89–98, 2003.

[18] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006.

[19] M. B. Cozzens and L. L. Kelleher. Dominating cliques in graphs. *Discrete Mathematics*, 86(1):101–116, 1990.

[20] T. S. Developers. *SageMath, the Sage Mathematics Software System (Version 6.10)*, 2015. `http://www.sagemath.org`.

[21] R. Diestel. *Graph Theory*. Electronic library of mathematics. Springer, 2006.

[22] C. Dunn. The relaxed game chromatic index of k-degenerate graphs. *Discrete Math.*, 307(14):1767–1775, 2007.

[23] P. Erdős and A. Hajnal. On chromatic number of graphs and set-systems. *Acta Math. Acad. Sci. Hungar.*, 17:61–99, 1966.

[24] P. L. Erdős, U. Faigle, W. Hochstättler, and W. Kern. Note on the game chromatic index of trees. *Theoretical Computer Science*, 313(3):371–376, 2004. Algorithmic Combinatorial Game Theory.

[25] U. Faigle, U. Kern, H. Kierstead, and W. Trotter. On the game chromatic number of some classes of graphs. *Ars Combin.*, 35:143–150, 1993.

[26] T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungaricae*, 18:25–66, 1967.

[27] M. Gardner. Mathematical games. *Scientific American*, 23, 1981.

[28] P. Hall. On representatives of subsets. *J. London Math. Soc.*, 10:26–30, 1935.

[29] J. Nešetřil and E. Sopena. On the oriented game chromatic number. *Electronic J. Comb.*, 8(2):R14, 2001.

[30] H. Kierstead and W. Trotter. Competitive colorings of oriented graphs. *Electronic J. Comb.*, 8(2), 2001.

[31] H. Kierstead and Z. Tuza. Marking games and the oriented game chromatic number of partial k-trees. *Graphs Comb.*, 19:121–129, 2003.

[32] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fund. Math.*, 15:271–283, 1930.

[33] B. D. McKay and A. Piperno. Practical graph isomorphism, ii. *J. Symbolic Computation*, 60:94–112, 2013.

[34] E. Sidorowicz. The game chromatic number and the game colouring number of cactuses. *Information Processing Letters*, 102(4):147–151, 2007.

[35] M. Tedder, D. Corneil, M. Habib, and C. Paul. *Simpler Linear-Time Modular Decomposition Via Recursive Factorizing Permutations*, pages 634–645. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[36] X. Zhu. The game coloring number of planar graphs. *Journal of Combinatorial Theory, Series B*, 75(2):245–258, 1999.

[37] X. Zhu. Refined activation strategy for the marking game. *Journal of Combinatorial Theory, Series B*, 98(1):1–18, 2008.