School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCE

A Study on Cops and Robbers: Tree Width and Algorithms

Kavan Price

Submitted in accordance with the requirements for the degree of MSci, BSc Computer Science with Mathematics

2019/20

40 credits

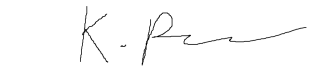
The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Deliverable 1	Report	Minerva (06/05/20)
Deliverable 2	Report	Minerva (06/05/20)
Deliverable 3	Report	Minerva (06/05/20)
Deliverable 4	Report	Minerva (06/05/20)

Type of project: Theoretical Study

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) 

© 2019/20 The University of Leeds and Kavan Price

Summary

Cops and Robbers is a long-studied, discrete pursuit-evasion game played on a graph, with applications in networking and routing, search-and-rescue, and graph searching (e.g. artificial intelligence and robotics). It is of theoretical interest as many variants are closely related to the computationally hard problems of tree- and path-decompositions. This project will explore the theoretical literature and results surrounding an "infinite velocity" variant of Cops and Robbers and the effects of the properties of the graph on which it is played.

Acknowledgements

With thanks to Dr. Isolde Adler for her guidance and supervision during this project and Professor Kristina Vušković for her feedback on the intermediate report. A special thank you to my parents for their continued support throughout my university studies.

Contents

Chapter 1

Introduction

1.1 Planning and Goals

Before we begin with the main content of the report, we outline and plan what we wish to achieve in this document.

1.1.1 Objectives

The goal of this project is to present an understanding and formulation of the common "cops and robbers" combinatorial game played on a graph, including the theoretical structures and definitions involved, the proceedings of the game itself as played by 2 opponents, and the implications that various structural qualities of the graph can have on the game. This report will detail the findings of the author's research into graph structures and algorithms concerning the game. The project is fully theoretical, in that it involves no implementation, and will hence replace such a deliverable with detailed theoretical results herein. The core outcomes for this project are as follows:-

1. To present a thorough mathematical understanding of fundamental concepts in graph theory relating to the problem;
2. To construct an expository report recounting research into the game of "cops and robbers;"
3. To evaluate the final project based on expected outcomes.

These objectives will be addressed and fulfilled by the following deliverables of this project:

1. Explanations and examples of the studied game;
2. An exposition of basic theory and advanced research materials related to "cops and robbers;"
3. Presentation and explanation of the author's own proofs of theoretical results;
4. A conclusion of the findings of this report and evaluative summary of the goals met.

The deliverables 1-4 are contained wholly within this report, and hence this document should be considered to constitute the project itself.

1.1.2 Schedule

Fortnightly supervision meetings are scheduled to aid progress and guidance with this project. Shown in figure 1.1 is a Gantt chart to outline the task scheduling and time allocation. The final week will be used for evaluating the project. The allocation of time to each area is approximate and may deviate from the planned schedule due to various circumstances that arise during the

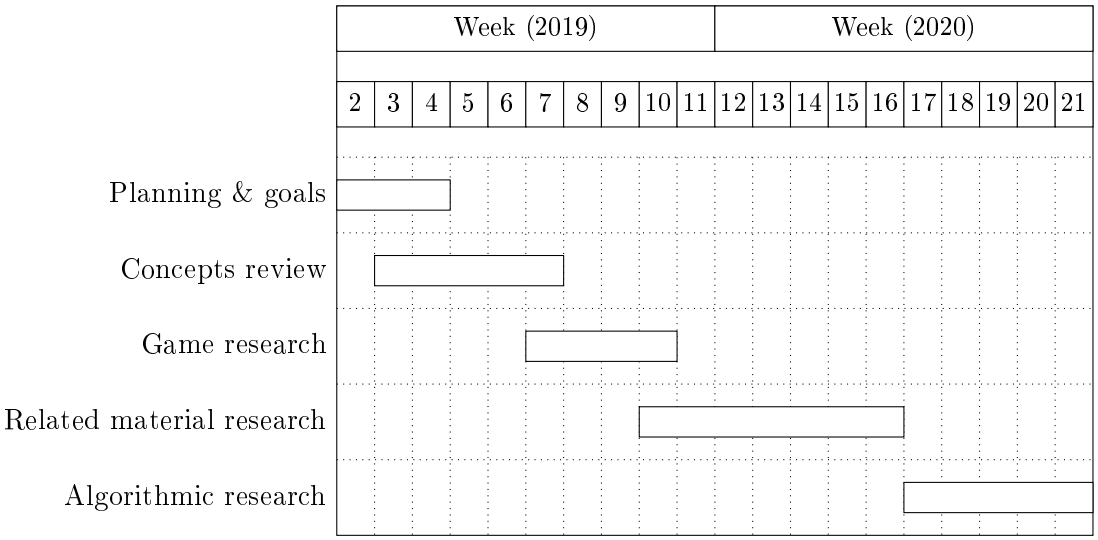


Figure 1.1: The planned schedule for this project.

course of the project, for example difficulty of literature or deviations in research topics. In this instance, "research" entails both the reading and exposition of mathematical literature, as well as the construction of theorems and original proofs.

1.1.3 Ethical Considerations

Due to the theoretical nature of this project, no consent for user testing, data collection, or photography and imaging is required. No personal data or transcripts - other than the names and institutions of cited academics - are obtained for this project and hence we need not consider the privacy and security of sensitive information. Where individuals involved with this project, for example supervisor and assessor, wish to remain anonymous or have their input unspecified, these arrangements are made. Clearly, since this document involves the citation of multiple academic articles, plagiarism of academic work is a major ethical issue for this project. In order to ensure that this document nor the author are guilty of academic misconduct through plagiarism, all material - for example theorems, concepts, proofs - not original to the author or considered "common knowledge" is properly and appropriately referenced, so as to credit the original source.

This consideration also constitutes the legal and professional issues concerning this project, since, though academic plagiarism is not understood to be illegal in itself, it could be considered fraudulent and result in a legal case if pursued. Academic plagiarism is not considered professional behavior and likely results in the removal of the author from their academic institution should it be committed. These points serve to exacerbate the need for proper referencing within this project.

To address another ethical and professional issue - where work not attributable to the author has been performed, for example giving advice and checking proofs, it is properly acknowledged and credit given. This is to ensure that the author is not given undue credit for the efforts of others.

1.2 Fundamental Definitions

We begin the main content of this report with some basic definitions on which the entirety of this project is based.

Definition 1.2.1. A (*undirected*) *graph* G is an ordered triple $(V(G), E(G), \psi_G)$, with $V(G) = \{v_1, v_2, ..., v_n\}$ being the set of *vertices* of G, $E(G) = \{e_1, e_2, ..., e_n\}$ the set of *edges* of G between any two vertices, and ψ_G the "incidence function" of G. When it is clear we are considering the graph G, we often use V and E in place of $V(G)$ and $E(G)$ respectively.

Definition 1.2.2. Given a graph $G = (V, E, \psi)$, ψ is the *incidence function* that associates an edge in E with some unordered pair of (not necessarily distinct) vertices in V.

Definition 1.2.3. Let $G = (V, E, \psi)$ be a graph. Then if, for some $e \in E$ and $u, v \in V$, we have $\psi(e) = \{u, v\}$: e is said to *join* u and v; e is *incident* to both u and v; u and v are *endpoints* of e; u and v are *adjacent* vertices.

Note. We often omit ψ_G from the definition of G by simply having $E(G) \subseteq \binom{V(G)}{2}$ where, for $u, v \in V(G)$, $\{u, v\} \in E(G)$ if and only if u and v are adjacent in G. Further, we denote by uv the unordered pair $\{u, v\}$.

Definition 1.2.4. Let e be an edge in a graph G with endpoints u and v. Then e is a *loop* if and only if $u = v$.

Definition 1.2.5. Let $G = (V, E)$ be a graph and $v \in V$. Then $d : V \to \mathbb{N}$ is the *degree function* defined by $d(v) = n \in \mathbb{N}$ if v is incident to n edges in G, where loops contribute 2. We also have $N_G : V \to \mathcal{P}(V)$ being the *neighbourhood function*, where $N_G(v) = \{u \in V : \ uv \in E\}$.

Definition 1.2.6. For a graph G, we define the *order* and *size* of G to be $v(G) = |V(G)|$ and $e(G) = |E(G)|$ respectively.

Definition 1.2.7. A *simple graph* is a graph in which there are no loops and no "parallel edges" (two edges are *parallel* if they share both endpoints).

1.3 Simple Results

We are now ready to prove some results that follow immediately from our current set of definitions.

Theorem 1.3.1 (Degree-Sum Formula)**.** *Let $G = (V, E)$ be a graph with $V = \{v_1, v_2, ..., v_n\}$. Then $\sum_{i=1}^{n} d(v_i) = 2e(G)$.*

Proof. Begin by noting that each edge in G has 2 endpoints. Then it follows that summing over the degree of each vertex in G counts each edge twice. Hence, $\sum_{i=1}^{n} d(v_i) = 2e(G)$. □

Corollary 1.3.2. *The sum of the degrees of all vertices in a graph is even.*

Proof. Let G be a graph. Then, by the Degree-Sum formula,

$$\sum_{v \in V(G)} d(v) = 2e(G).$$

Now $e(G) \in \mathbb{N}$ since $E(G)$ is countable, so $\sum_{v \in V(G)} d(v)$ is even by the definition of an even number. □

Theorem 1.3.3. *Let G be a graph. Then there is an even number of vertices of odd degree in G.*

Proof. We have already shown that the sum of the degrees of all vertices in G must be even. Let D_{odd} be the set of vertices in G with odd degree and D_{even} be the set of vertices in G with even degree. Then define $S_{odd} := \sum_{v \in D_{odd}} d(v)$ and $S_{even} := \sum_{v \in D_{even}} d(v)$. Then, since (D_{odd}, D_{even}) is a partition of $V(G)$, i.e. $D_{odd} \cup D_{even} = V(G)$ and $D_{odd} \cap D_{even} = \varnothing$,

$$\sum_{v \in V(G)} d(v) = 2e(G) = S_{odd} + S_{even}.$$

From this we see that either both S_{odd} and S_{even} are odd or they are both even. Since S_{even} is a summation of even numbers, i.e. for all $v \in D_{even}$, $d(v)$ is even, S_{even} must be even, from which it follows that S_{odd} is necessarily also even. Since S_{odd} is a summation of odd numbers in D_{odd}, $|D_{odd}|$ must be even, otherwise such a sum would be odd. By our definition of D_{odd}, we conclude that the number of vertices in G with odd degree is even. □

Theorem 1.3.4. *Let G be a simple graph. Then $e(G) \leq \binom{v(G)}{2}$.*

Proof. Since G is simple, each edge is a unique unordered pair. Hence, the number of edges in G is bound by the number of unordered pairs of vertices in G (given by $\binom{v(G)}{2}$). That is, $e(G) \leq \binom{v(G)}{2}$. □

1.4 Special Classes of Graph

We are ready to define some simple classes of graph. These classes are fundamental to understanding certain problems and larger graphs can be decomposed into such classes.

Definition 1.4.1 (Directed Graph)**.** A *directed graph* $G = (V, E)$ is a graph with vertex set $V = \{v_1, ..., v_n\}$ and edge set $E \subseteq V \times V$, where the ordered pair $(u, v) \in E$ denotes a *directed edge* from u to v.

Remark. We can encode the meaning of an undirected graph using a directed graph by using two directed edges in place of one undirected edge. e.g. For a directed graph $G_D = (V(G_D), E(G_D))$ and undirected graph $G_U = (V(G_U), E(G_U))$, where $V(G_D) = V(G_U)$, we make G_D "equivalent" to G_U by the following rule - whenever we have $\{u, v\} \in E(G_U)$, we put (u, v) and (v, u) into $E(G_D)$. This allows us to travel in both directions between u and v in G_D just as we can in G_U, encoding the undirected edge $\{u, v\}$ in G_U by the directed edges (u, v) and (v, u) in G_D.

Definition 1.4.2 (Bipartite Graph)**.** A *bipartite graph* $G[X, Y] = (V, E)$ is a simple graph with a bipartition (X, Y) of V, i.e. $X \cup Y = V$ and $X \cap Y = \varnothing$, such that all edges have one endpoint in X and one endpoint in Y. That is, no edge has both endpoints in X or both endpoints in Y.

Definition 1.4.3 (Complete Graph)**.** A *complete graph* $G = (V, E)$ is a simple graph in which, given any vertex $u \in V$, u is adjacent to all $v \in V$. That is, $E = \binom{V}{2}$ in a simple graph. We

denote by K_n, the complete graph of order n, and by $K_{n,m}$, the bipartite graph $G[X,Y]$ with $|X| = n$ and $|Y| = m$, such that all vertices in X are adjacent to all vertices in Y.

Definition 1.4.4 (Regular Graph)**.** A graph G is *regular* if for all $v_1, v_2, ..., v_n \in V(G)$, $d(v_1) = d(v_2) = ... = d(v_n)$. If $d(v_1) = ... = d(v_n) = k$ for some $k \in \mathbb{N}$, then G is said to be *k-regular*.

Definition 1.4.5. For a graph G, we have the following definitions:

- A *walk* (in G) is a sequence $w = (v_0, e_1, v_1, e_2, ..., e_n, v_n)$ of alternating vertices and edges, where $v_i \in V(G)$ and $e_i \in E(G)$ for all $i \leq n$, and e_j joins v_{j-1} and v_j for all $1 \leq j \leq n$. The length of w is the number of edges in w.
- A *path* (in G) $p = (v_0, e_1, ..., e_n, v_n)$ is a walk (in G) where all vertices in p are distinct, except possibly v_0 and v_n.
- A *closed walk* (in G) $w = (v_0, ..., v_n)$ is a walk (in G) such that $v_0 = v_n$.
- A *cycle* (in G) $c = (v_0, ..., v_n)$ is a closed path (in G) with length at least 1.

Note. If G is a simple graph, then we may denote a walk w, of length n, informally by $w = v_0, v_1, ..., v_n$.

Definition 1.4.6 (Cycle graph)**.** A *cycle graph* G is a graph consisting of a single cycle. We denote by C_n, the cycle graph of order n.

Going forward in this report, for simplicity, we will use the term "graph" to mean a "simple, undirected graph," unless stated otherwise. We now prove, as before, some useful results following from these definitions.

Lemma 1.4.7. *Let $G[X,Y]$ be a bipartite graph. Then*

$$\sum_{x \in X} d(x) = \sum_{y \in Y} d(y).$$

Proof. Since G is bipartite, each edge in $E(G)$ has one endpoint in X and one in Y. Then summing over the the degrees of the vertices in X counts all edges in $E(G)$, and similarly for summing over the vertices in Y. That is, $\sum_{x \in X} d(x) = e(G) = \sum_{y \in Y} d(y)$. □

Corollary 1.4.8. *Let $G[X,Y]$ be a k-regular bipartite graph with $k > 0$. Then $|X| = |Y|$.*

Proof. By the previous lemma, $\sum_{x \in X} d(x) = \sum_{y \in Y} d(y)$, hence $k \cdot |X| = k \cdot |Y|$, implying $|X| = |Y|$ after division by $k > 0$. □

Lemma 1.4.9. *Let G be a graph consisting of only a path. Then G is bipartite.*

Proof. Since G is a path, we can list the vertices in $V(G)$ in a sequence $(v_1, ..., v_n)$, such that v_i is adjacent to only v_{i-1} and v_{i+1} for $1 < i < n$. So, we have that for all $v_i, v_j \in V(G)$, v_i is adjacent to v_j if and only if i is odd and j is even, or i is even and j is odd. From this, it follows that we have the bipartition (X,Y) of $V(G)$, where $X \subseteq V(G)$ is the set of vertices of odd enumeration and $Y \subset V(G)$ is the set of vertices of even numeration. □

Theorem 1.4.10. *Let K_n be the complete graph of order n. Then $e(K_n) = \frac{n(n-1)}{2}$.*

Proof. In K_n, we must place an edge between every pair of vertices. Since the pairs are not ordered, we have that $E(K_n) = \binom{V(G)}{2}$, so $e(K_n) = \binom{v(G)}{2} = \binom{n}{2} = \frac{n(n-1)}{2}$. □

Remark. Theorem 1.4.10 is a special case of Theorem 1.3.4, namely the case in which equality holds.

We have now proven and reinforced a good basis of knowledge on which we can build intuition about combinatorial games on graphs and the algorithms and theoretical results surrounding them. In the following chapter, we will describe the rules of the game Cops and Robbers in full and mathematically formulate said rules using our definitions given in this chapter. We will relate the graph classes we have studied to parameters within the game which we will describe later.

Chapter 2

Cops and Robbers

2.1 An Informal Description

Cops and Robbers is a combinatorial game played by two adversaries on a graph (simple and undirected). One player controls a set of cops and the other controls a single robber. The aim of the player controlling the cops is to "catch" the robber and the robber's aim is to flee capture for as long as possible (possibly indefinitely). At the beginning of the game, the robber may place himself anywhere in the graph, and then players must alternate turns moving between vertices of the graph. Following the robber's turn, the cops may drop, via helicopter, onto any set of vertices in the graph but must first announce such vertices and also allow the robber to move along edges of the graph to a new location before they land. The robber may not travel to a vertex with a cop on it and may only travel along edges of the graph, i.e. the robber cannot travel between different "components" of a graph, but does so instantaneously. Only one cop may occupy a single vertex. Multiple cops can be moved in a single turn and must be moved via helicopter. The game ends when a cop lands on the same vertex as the robber, in which case the cops win, or when the cops resign upon realising it is not possible to catch the robber, in which case the robber wins. Both the cops and the robber remain visible to all parties throughout the game.

2.2 Preliminary Definitions

We give some definitions necessary to give our described game a precise formulation.

Definition 2.2.1. For a (simple, undirected) graph $G = (V, E)$, we have the following:

- H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, under the constraint that for all $uv \in E(G)$, if $uv \in E(H)$ then $u, v \in V(H)$.
- For $u, v \in V$, u and v are *connected* if there exists a walk from u to v in G. u and v are *disconnected* if there exists no such walk.
- G is a *connected graph* if for all $u, v \in V$, u and v are connected.
- G is a *disconnected graph* if there exist $u, v \in V$ such that u and v are disconnected.

Definition 2.2.2 (Bondy, Murty)**.** Let G be a disconnected graph. Then the *connected components*, or simply *components*, of G are the vertex-disjoint subgraphs of G such that their union is G [5].

Remark. This definition has multiple equivalent forms. For example, given a graph G, a component of G is a maximally connected subgraph of G, i.e. a subgraph in which all vertices are connected and adding a further vertex would disconnect the subgraph. We can define connected components of G with an equivalence relation $\sim$ on $V(G)$, where for $u, v \in V(G)$,

$u \sim v$ if and only if u and v are connected. Then the equivalence class, notated $[u]_\sim$, of a vertex $u \in V(G)$ is the vertex set of the connected component that u belongs to, since $[u]_\sim = \{v : u \sim v\} = \{v : u \text{ and } v \text{ are connected}\}$. To form the edges of the component created by $[u]_\sim$, we simply place an edge between $v, w \in [u]_\sim$ if $vw \in E(G)$. So we get all connected components of G by iterating this process on all vertices of G.

Definition 2.2.3 (Seymour, Thomas [13])**.** Let G be a simple, undirected graph. Then we have the following definitions:

- $G \setminus X$ is the graph obtained from G by deleting X, where X may be a vertex, an edge, or a set of vertices or edges.
- The vertex set of a component of $G \setminus X$ is called an *X-flap*.
- For a set A and $k \in \mathbb{N}$, we denote by $[A]^{<k}$ the set of all subsets of A of cardinality less than k. That is, $[A]^{<k} = \{S \in \mathcal{P}(A) : |S| < k\} \subseteq \mathcal{P}(A)$.

2.3 Formal Rules of the Game

We are now ready to give a mathematical formulation to the rules of the game (attributed to Seymour and Thomas [13]). Let $G = (V, E)$ be a simple, undirected graph. Given $k - 1 \in \mathbb{N}$ cops, the game is denoted $\mathcal{CR}(G, k-1)$, and a position in $\mathcal{CR}(G, k-1)$ is an ordered pair (X, R) where $X \in [V(G)]^{<k}$ and R is an X-flap. X is the set of vertices currently occupied by cops and R is the vertex set of the component of $G \setminus X$ occupied by the robber. Since, as described before, the robber may run arbitrarily fast, it suffices to identify the component they occupy, not a specific vertex of that component, since they may travel instantaneously to any vertex in that component. The game consists of a sequence of positions $(X_0, R_0), (X_1, R_1), ..., (X_n, R_n)$, where on the i^{th} step of the game we begin with (X_{i-1}, R_{i-1}) and progress to position (X_i, R_i). (X_0, R_0) is the initial position of the game with $X_0 = \varnothing$ and R_0 chosen by the robber player. In each subsequent step, the cop player chooses a new set $X_i \in [V(G)]^{<k}$ such that $X_{i-1} \subseteq X_i$ or $X_i \subseteq X_{i-1}$, after which the robber player chooses an X_i-flap R_i such that $R_i \subseteq R_{i-1}$ or $R_{i-1} \subseteq R_i$. The cop player wins if $V(R_{i-1}) \subseteq X_i$, that is, the cops cover all vertices of the robber's component. The robber wins if they can manage to avoid capture indefinitely.

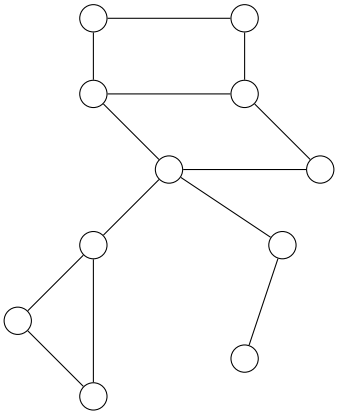
Definition 2.3.1. Let G be a graph. The *cop number* of G, $c(G)$, is the minimum number of cops such that there is a winning strategy for the cop player.

In order to clarify the procedure we have just formulated, we will step through a visual example in the next section. This will give intuition to the game we have created and put us in a good position to begin discussing cop strategies for specific graphs.

2.4 An Example Game

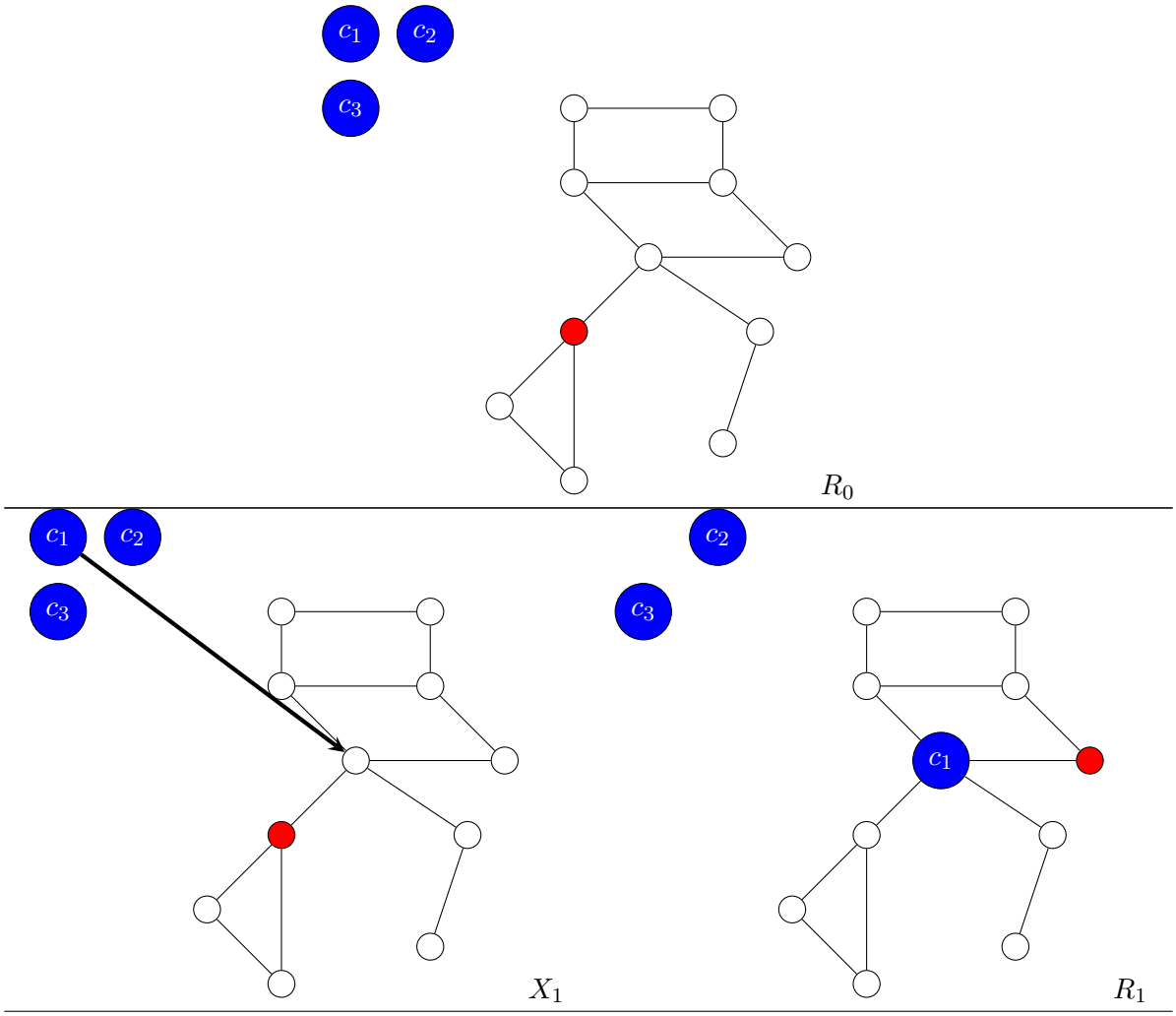
In this section we demonstrate an example pertaining to the Cops and Robbers game we have described. We will first show the graph and then each position on the graph as the game progresses.

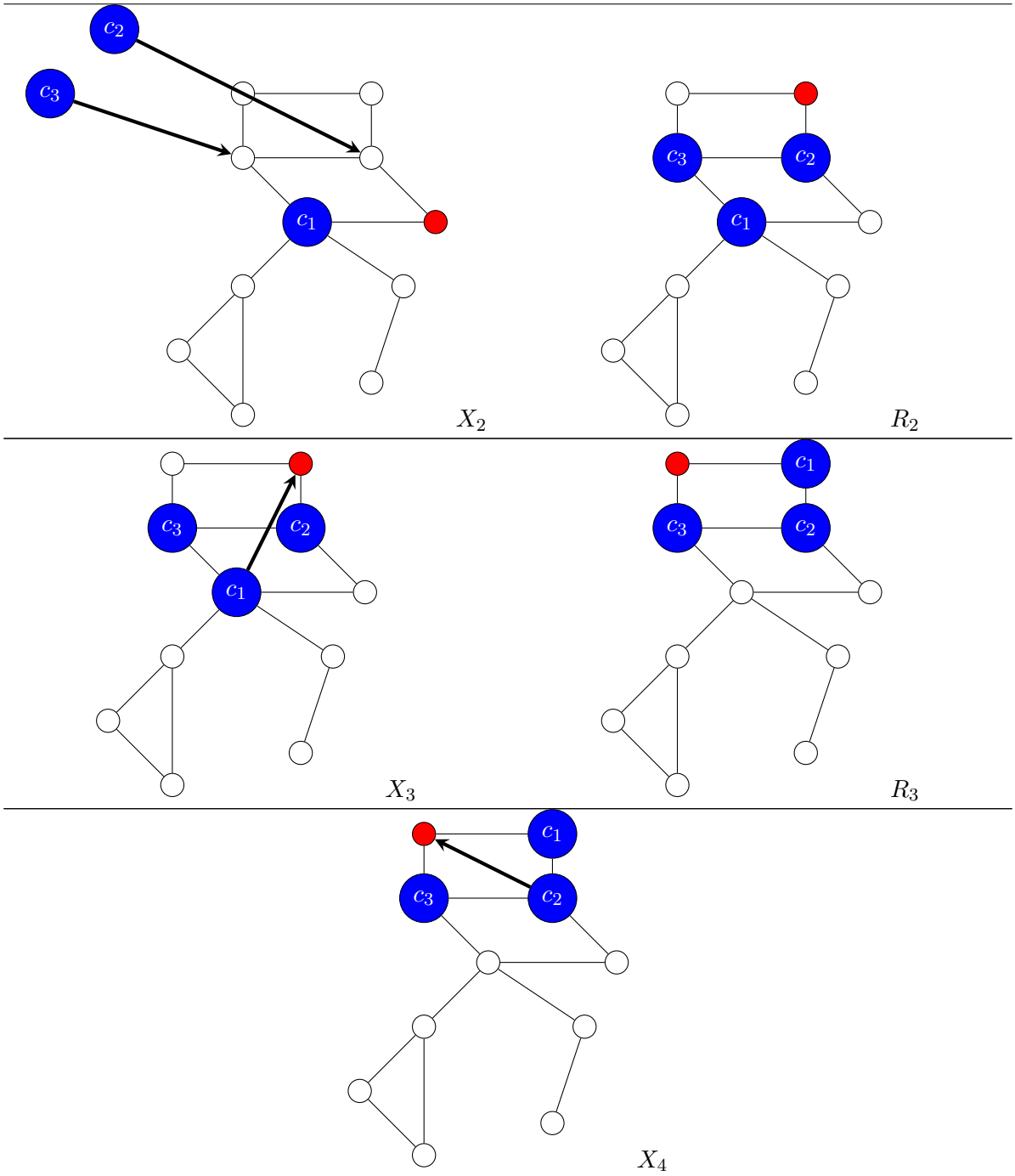
Example 2.4.1. We define the graph G below and consider the game $\mathcal{CR}(G, 3)$.



G

We proceed with the game below with blue cops $\{c_1, c_2, c_3\}$. For simplicity, we denote the robber's position with only a single red vertex, though we consider the whole X_i-flap containing that red vertex to be his true position for some cop move X_i.





Hence we see that the cops win since $R_3 \subseteq X_4$. We clearly see the importance of controlling the central vertex (the vertex first occupied by c_1) to cut the graph into components and force the robber into a smaller area. We see the same method applied in move X_2 with c_2 and c_3.

For readability, we place two further examples in Appendix A. We see a similar method in Example A.0.1 whereby the central vertex is controlled to restrict the movement of the robber. In all of these examples we see a strategy employed where cops adjacent to the robber's current position are fixed in place to keep the robber cornered and the behind cops are moved to unoccupied spots to displace the robber and further restrict his movement.

2.5 Implications of Graph Classes

In this section we present the cop number for some aforementioned classes of graphs through the use of cop strategies.

Definition 2.5.1. Let G be a graph. G is a *forest* if it has no cycles. G is a *tree* if it is a forest and connected. A *leaf* is a vertex of degree 1 in a forest.

Definition 2.5.2. Given $n > 0 \in \mathbb{N}$, the *path graph* P_n is the tree of order n such that, for all $v \in V(P_n)$, $d(v) < 3$ [6].

Lemma 2.5.3. *For $n > 0 \in \mathbb{N}$, $c(P_n) \leq 2$.*

Proof. We will prove this by giving a winning strategy in $\mathcal{CR}(P_n, 2)$ for the cop player with the set of cops $\{c_1, c_2\}$. We enumerate $V(P_n) = \{v_1, ...v_n\}$ such that, $\{v_\ell, v_m\} \in E(P_n)$ if and only if $m = \ell + 1$, for $1 \leq \ell \leq m \leq n$. If $n = 1$, then the cop player can immediately win by declaring $X_1 = V(P_n)$, otherwise the cop player declares $X_1 = \{v_1, v_2\}$. If $n = 2$ then the cop player has won, otherwise the cop player may execute the following strategy. Beginning with c_1, and alternating between moving c_1 and c_2, place the cop on the next available vertex with lowest enumeration. With this strategy, it is clear that the cops can win by reducing the order of P_n by one vertex in each move until a cop has been placed on the single vertex the robber will eventually occupy. Hence, for $n > 0$, $c(P_n) \leq 2$. In particular, $c(P_1) = 1$ and $c(P_2) = 2$, special cases of a theorem for complete graphs that we will see soon. □

Theorem 2.5.4. *For $n > 1$, $c(P_n) = 2$.*

Proof. We will show that for $n > 2$, a single cop is insufficient to catch the robber. Take $\mathcal{CR}(P_n, 1)$. As an initial position, we have $(\varnothing, P_n)$. Now the cop player selects a vertex of P_n that partitions P_n into two paths. The strategy for the robber, on each selection of X_i by the cop, is to select R_i to be the path with maximum length once P_n is split. Since, for any P an X_i-flap of P_n, $|V(P)| \geq 1$ and the cop player has no spare cop, there is no strategy for the cop to win, since the robber can continuously choose the larger X_i-flap as P_n is connected. Hence $c(P_n) \neq 1$ for $n > 1$, so $c(P_n) = 2$ for $n > 1$ by Lemma 2.5.3. □

Corollary 2.5.5. *$c(C_n) = 3$ for $n > 2$.*

Proof. First consider $\mathcal{CR}(C_n, 2)$. We begin with the initial position $(\varnothing, C_n)$. Any X_i chosen by the cop player splits C_n into two paths. As before, since C_n is connected, the robber player may continuously choose the path with longer length and so will never be caught. So $c(C_n) > 2$. Now consider $\mathcal{CR}(C_n, 3)$. We begin again with the position $(\varnothing, C_n)$. The cop player places a cop on an arbitrary vertex $v \in V(C_n)$, so $(X_1, R_1) = (v, C_n \setminus v)$, hence we obtain the graph P_{n-1}. By Theorem 2.5.4, $c(P_{n-1}) = 2$, thus, fixing the cop on v, there is a winning strategy for the two remaining cops. Hence we have $c(C_n) = 3$. □

Theorem 2.5.6. *For $n > 0$, $c(K_n) = n$.*

Proof. We will consider a game containing $n - 1$ cops and show that the cop player cannot win. In this game we take the initial position $(\varnothing, K_n)$. The cop player takes their turn and places $n-1$

cops on an arbitrary set of vertices $X_1 \subset V(K_n)$. Now, since $|X_1| = n - 1 \neq v(K_n)$, the robber player may choose $R_1 = K_n \setminus X_1$. Indeed, since all vertices of K_n are adjacent, the robber may freely choose any vertex not chosen by the cop player, that is $R_i = K_n \setminus X_i$. Hence the robber may avoid capture indefinitely, so $c(K_n) \neq n - 1$. Clearly, since $v(K_n) = n$, n cops may win on the first turn, hence $c(K_n) = n$.

□

We have proven the cop number for specific graph classes, but we can in fact work in the opposite direction. That is, we may describe the properties of graphs that have a given cop number. In subsequent chapters we will introduce stronger "if and only if" statements to do precisely this. A natural comparison of the cop number of a graph is to that of its subgraphs. More precisely, we seek to answer the following question.

Question:
Let G and H be graphs with G a subgraph of H.
Is $c(G) \leq c(H)$?

It would appear that the answer to this question is obvious, since G is in some sense "simpler" than H (e.g. has less vertices or edges) and should therefore require no more cops to catch a robber in such a graph. However, our only knowledge of determining the cop number of given graphs so far is through constructing winning strategies for the players. While these are valid proofs, in order to more easily construct a formal proof of the answer to our question, we require an additional property of graphs, described in the following chapter.

Chapter 3

Tree Width

In this chapter we will present the results of Seymour and Thomas concerning the "tree width" property of a graph and its relation to the cop number of said graph [13].

3.1 Introduction

We first describe the concept of monotonicity and monotonely searching a graph.

Definition 3.1.1. Given a graph G, $k \in \mathbb{N}$ cops can *monotonely search* G if and only if the cop player can win in $\mathcal{CR}(G,k)$ with a sequence of moves $X_0, X_1, ..., X_n$ such that $X_i \cap X_{i''} \subseteq X_{i'}$ for $0 \leq i \leq i' \leq i'' \leq n$. Denote by $c_M(G)$ the minimum number of cops such that they have a monotone winning strategy in G.

Remark. More intuitively, this is equivalent to the statement "k cops can search G monotonely if and only if they can search G without returning to any vertices." It appears that this property of searching is in some sense "stricter" than searching non-monotonely, but they are in fact equivalent. We will later see a theorem that shows this.

Now we give definitions to describe the tree width of a graph.

Definition 3.1.2. Given a graph G, a *tree-decomposition* of G is a pair (T, W) where T is a tree and $W = \{W_t : t \in V(T)\}$ is a family of subsets of $V(G)$ such that:

- $\bigcup_{t \in V(T)} W_t = V(G)$;
- for all $\{u, v\} \in E(G)$, $u, v \in W_t$ for some W_t;
- if $t, t', t'' \in V(T)$ and t' lies on the path from t to t'', then $W_t \cap W_{t''} \subseteq W_{t'}$. $(*)$

The third condition can be restated as:

- for all $v \in V(G)$, the set of vertices $\{t \in V(T) : v \in W_t\}$ induces a connected subgraph of T. $(**)$

We prove the equivalence of $(*)$ and $(**)$ below.

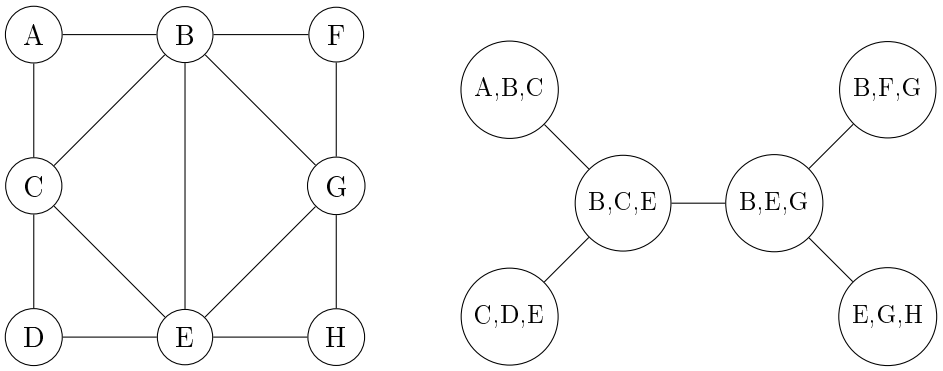
Proof. Let G be a graph and (T, W) a pair where T is a tree and $W = \{W_t : t \in V(T)\}$ a family of subsets of $V(G)$. Assume $(*)$ holds for G and (T, W). Take a vertex $x \in V(G)$ and consider the set $A = \{t \in V(T) : x \in W_t\}$. We must show that there is a path between any two vertices in A. If $|A| = 1$, A induces a connected subgraph of T. Otherwise, take arbitrary and distinct $u, v \in A$. A path from u to v exists in T, since T is connected, so for any $t \in V(T)$, $W_u \cap W_v \subseteq W_t$. Since $x \in W_u$ and $x \in W_v$, $x \in W_t$ for all $t \in V(T)$ in the path from u to v. Hence each vertex on that path is a member of A, so any $u, v \in A$ are connected by vertices in A. Thus, $(**)$ is true. Now assume $(**)$ holds for G and (T, W). Choose arbitrary $t, t', t'' \in V(T)$

where t' lies on the path between t and t''. For all vertices $v \in V(G)$ such that $v \in W_t \cap W_{t''}$ we have that $t, t'' \in \{x \in V(T) : v \in W_x\}$, which induces a connected subgraph of T by $(**)$. Hence, $v \in W_u$ for all $u \in V(T)$ on the path between t and t'', specifically $v \in W_{t'}$. So we have, for all $v \in W_t \cap W_{t''}$, $v \in W_{t'}$, so $W_t \cap W_{t''} \subseteq W_{t'}$. Thus we have $(*)$ if and only if $(**)$. □

Definition 3.1.3. Let G be a graph and (T, W) a tree-decomposition of G. Then the *width* of (T, W), denoted $w(T, W)$, is given by $w(T, W) = \max\{|W_t| - 1 : t \in V(T)\}$.

Definition 3.1.4. The *tree-width* of a graph G, $\mathcal{W}(G)$, is the minimum width of a tree-decomposition of G. That is, $\mathcal{W}(G) = \min\{w(T, W) : (T, W) \text{ is a tree-decomposition of } G\}$.

Example 3.1.5. Shown below is an example tree-decomposition with the original graph on the left and the decomposition on the right. This tree-decomposition has width 2 [7].



3.2 Effects on Cops and Robbers

Now that we have fully described the tree-width property, we can discuss the implications of this on the game we are studying. The following is an important result by Seymour and Thomas.

Theorem 3.2.1. *Let G be a graph and $k \in \mathbb{N}$. Then the following are equivalent:*

(i) $c(G) = k$;

(ii) $c_M(G) = k$;

(iii) $\mathcal{W}(G) = k - 1$.

Proof. See [13]. □

LaPaugh showed that *(i)* if and only if *(ii)* in a game where the robber is invisible [12]. Thus, this is true for our game in which the robber is visible, since the cops may employ the same search strategy as if the robber were invisible. We will see why *(ii)* implies *(iii)* in the next section when we introduce cliques. The proof of *(iii)* implies *(i)* is much longer, involving "screens," "havens," and "jump-searching." The idea is to show that, for any "screen" S of a graph G and $k \in \mathbb{N} \setminus \{0\}$, there is no $S' \supseteq S$, where for all $X \in [V(G)]^{<k}$ and $H \in S'$, $X \cap H \neq \varnothing$, if and only if G has a tree-decomposition (T, W) where every $t \in V(T)$ with $|W_t| \geq k$ both has "valency" 1 and $W_t \cap H = \varnothing$ for any $H \in S$. It is then possible to set $S = \varnothing$ in this result to obtain

the proof of (*iii*) implies (*i*). This is the main focus of Seymour and Thomas [13] and involves multiple intermediate steps to attain this result. We refer to their paper for details of the proof and the terms we neglected to define, though we will later meet some of them when discussing strategies for the robber player.

From this theorem, we have established the link between the cop number and tree-width of a graph. We can use this link to answer our earlier question concerning subgraphs:

Question:
Let G and H be graphs with G a subgraph of H.
Is $c(G) \leq c(H)$?

We confirm that the answer to this question is positive in the following theorem.

Theorem 3.2.2. *Let H be a graph and G a subgraph of H. Then $c(G) \leq c(H)$.*

Proof. If $V(H) = \varnothing$, then $V(G) = \varnothing$, so $c(H) = c(G) = 0$. If $V(G) = \varnothing$, $c(G) = 0 \leq c(H)$. Otherwise, let (T, W) be a tree-decomposition of H such that $w(T, W) = \mathcal{W}(H)$. Then we may construct the set $W' = \{W'_t \subseteq V(G) : t \in V(T)\}$ where, for each $t \in V(T)$, $W'_t = W_t \cap V(G)$. We have $|W'_t| = |W_t \cap V(G)| \leq |W_t|$ for all $t \in V(T)$, so $w(T, W') \leq w(T, W)$. It remains to show that (T, W') is in fact a tree-decomposition of G.

- We have $\bigcup_{t \in V(T)} W_t = V(H) \supseteq V(G)$. Hence, for all $v \in V(G)$, $v \in W_t$ for some $W_t \in W$, so $v \in (W_t \cap V(G)) = W'_t \in W'$. So each $v \in V(G)$ is a member of some $W'_t \in W'$, i.e. $\bigcup_{t \in V(T)} W'_t = V(G)$.
- Let $e = \{u, v\} \in E(G)$ where $u, v \in V(G)$. Then $e \in E(H)$ since $E(G) \subseteq E(H)$, and $u, v \in V(H)$. So $u, v \in W_t$ for some $W_t \in W$ where $t \in V(T)$. Then $u, v \in (W_t \cap V(G))$, where we have $(W_t \cap V(G)) = W'_t$. Hence, for any edge $\{u, v\} \in E(G)$, $u, v \in W'_t$ for some $W'_t \in W'$.
- We already have that for all $v \in V(H)$, $\{t \in V(T) : v \in W_t\}$ induces a connected subgraph of T. Hence this property holds for all $v \in V(G)$ since $V(G) \subseteq V(H)$, so $\{t \in V(T) : v \in W_t\}$ induces a subgraph of T. Since, for any $v \in V(G)$, $v \in W_t$ for some $t \in V(T)$ only if $v \in W'_t$, it follows that $\{t \in V(T) : v \in W'_t\}$ induces a subgraph of T.

Hence, comparing with Definition 3.1.2, (T, W') is a tree-decomposition of G and we have $w(T, W') \leq w(T, W)$. So

$$c(H) = \mathcal{W}(H) + 1 = w(T, W) + 1 \geq w(T, W') + 1 \geq \mathcal{W}(G) + 1 = c(G).$$

□

An intuitive outline of the previous proof is that we take the minimum width tree-decomposition of the supergraph and remove the vertices not in the subgraph from the sets associated with the vertices of the tree in the decomposition. From this we obtain a new tree-decomposition of the subgraph with a smaller width.

Corollary 3.2.3. *Let H be a graph and G a subgraph of H. Then $\mathcal{W}(G) \leq \mathcal{W}(H)$.*

Proof. The result follows immediately from Theorem 3.2.1 and Theorem 3.2.2. □

Using Theorem 3.2.1 and Theorem 3.2.2, we can characterise the properties of graphs with a given cop number.

Corollary 3.2.4. *Let G be a graph. Then $c(G) = 1$ if and only if $e(G) = 0$.*

Proof. We will show that $\mathcal{W}(G) = 0$ if and only if the size of G is 0. Suppose $\mathcal{W}(G) = 0 = w(T, W)$ for a tree decomposition (T, W) of G with minimum width. Then for any $W_t \in W$, $|W_t| = 1$ and hence no W_t contains the endpoints of an edge. Since both endpoints of each edge of G are in some $W_t \in W$, G has no edges, so $e(G) = 0$. Now suppose $e(G) = 0$, so G has no edges. Hence, taking an arbitrary tree T such that $v(T) = v(G)$, we may construct a tree-decomposition (T, W) of minimum width by assigning each vertex in $V(G)$ to a different set $W_t \in W$. Thus, we have $w(T, W) = \max\{|W_t| - 1 : t \in V(T)\} = 0$. Hence, $\mathcal{W}(G) = 0$, so $c(G) = 1$. It now follows that $c(G) = 1$ if and only if $\mathcal{W}(G) = 0$ if and only if $e(G) = 0$. □

Corollary 3.2.5. *Let G be a graph. Then $c(G) = 2$ if and only if G is a forest with $e(G) > 0$.*

Proof. We prove the "if" direction first. A graph T is a tree on two or more vertices (i.e. $e(T) > 0$) only if $\mathcal{W}(T) = 1$ [6], so $c(T) = 2$. Hence if G is a forest with $e(G) > 0$, then $c(G) \geq 2$ by Theorem 3.2.2. Indeed $c(G) \leq 2$ since the robber must select a tree of G and 2 cops can catch the robber within this tree. Thus we have $c(G) = 2$. Conversely, assume $c(G) = 2$ and that G is not a forest with $e(G) > 0$. If $e(G) = 0$ then $c(G) = 1$ by Corollary 3.2.4. Otherwise, G contains a cycle since it is not a forest, so C_n is a subgraph of G for some natural number $n \leq v(G)$. $c(C_n) = 3$ by Theorem 2.5.5, so $c(G) \geq 3$ by Theorem 3.2.2. In either case we have a contradiction, so G is a forest with $e(G) > 0$. □

These two corollaries, following from the powerful Theorems 3.2.1 and 3.2.2, have allowed us to completely characterise the graphs with cop number two or less. In general, determining the tree-width - and therefore the cop number - of an arbitrary graph is an $\mathbb{NP}$-complete problem. The next chapter will discuss the meaning of this classification of the problem and algorithms designed to solve it.

3.3 An Equivalent Definition

In this section, we will state a more intuitive description of the tree-width property that will make Theorem 3.2.1 clearer.

Definition 3.3.1. Given a graph G, a *clique* $C \subseteq V(G)$ of G is a set of vertices such that all vertices in C are adjacent in G. Equivalently, the subgraph of G induced by C is complete. The *clique number* of G, $\omega(G)$, is the size of the largest clique in G.

Definition 3.3.2. Given a cycle c, a *chord* of c is an edge that joins two vertices of c, but is not in c itself.

Definition 3.3.3. Let G be a graph. Then G is *chordal* if every cycle in G, of length greater than 3, has at least one chord.

From Definition 3.1.2, Example 3.1.5, and Definition 3.3.1, it should be clear that a tree-decomposition of a graph in some sense separates that graph into cliques, represented by the tree

structure of the decomposition. In fact, the tree-width of a graph is definable in this way, as we shall see.

Definition 3.3.4. Let S and I be sets and $\{S_i \subseteq S : i \in I\}$ a family of subsets of S. Then S satisfies the *Helly property* if, for all subsets $J \subseteq I$ such that for all $j, k \in J$, $S_j \cap S_k \neq \varnothing$, we have that $\bigcap_{j \in J} S_j \neq \varnothing$ holds.

Theorem 3.3.5. *Let T be a tree and I a set with $J = \{T_i \subseteq V(T) : i \in I\}$ a family of subsets of vertices of T, inducing subtrees. Then J has the Helly property.*

Proof. See [10]. □

Lemma 3.3.6 (Clique containment lemma [4])**.** *Let G be a graph, (T, W) a tree-decomposition of G, and $C \subseteq V(G)$ a clique in G. Then for some $W_t \in W$, $C \subseteq W_t$.*

Proof. Let $T_v = \{t \in T : v \in W_t\}$, so $\{T_v : v \in C\}$ is a family of induced subtrees of T. By Theorem 3.3.5, $\{T_v : v \in C\}$ has the Helly property. Hence, we have for all $u, v \in C$, $u, v \in W_t$ for some $t \in V(T)$ since $uv \in E(G)$, so $t \in T_u \cap T_v$, implying $\bigcap_{v \in C} T_v \neq \varnothing$. That is, there exists some $t \in V(T)$ such that for all $v \in C$, $v \in W_t$. □

Lemma 3.3.7. *Given a graph G, $\mathcal{W}(G) \geq \omega(G) - 1$.*

Proof. Let (T, W) be a tree-decomposition of G of minimum width. By Lemma 3.3.6, there is some $t \in V(T)$ with $|W_t| = \omega(G)$ since each clique of G is assigned to a vertex of T. That is $w(T, W) \geq |W_t| - 1 = \omega(G) - 1$. Hence we have $\mathcal{W}(G) = w(T, W) \geq \omega(G) - 1$. □

We now show that, for chordal graphs, equality holds in the previous lemma. We may then use this result to complete this section by relating tree-width, clique number, subgraphs, and chordal graphs in our equivalent formulation of tree-width. To do this we need the following lemma.

Lemma 3.3.8. *A graph G is chordal if and only if G has a tree-decomposition (T, W) such that, for all $W_t \in W$, W_t is a clique in G.*

Proof. See [6]. □

Theorem 3.3.9. *Let G be a chordal graph. Then $\mathcal{W}(G) = \omega(G) - 1$.*

Proof. By Lemma 3.3.7 we have $\mathcal{W}(G) \geq \omega(G) - 1$. By Lemma 3.3.8, G has a tree-decomposition (T, W) such that every $W_t \in W$ is a clique of G. Hence, for all $W_t \in W$, $|W_t| \leq \omega(G)$. Thus, $\mathcal{W}(G) \leq w(T, W) \leq \omega(G) - 1$, so we have $\mathcal{W}(G) = \omega(G) - 1$. □

Theorem 3.3.10. *Let G be a graph and define a chordal graph H, with the smallest $\omega(H)$, such that G is a subgraph of H. Then $\mathcal{W}(G) = \omega(H) - 1$.*

Proof. Let A be the set of chordal supergraphs of G. Then we seek to prove that $\mathcal{W}(G) = \min\{\omega(H) - 1 : H \in A\}$. Each graph $H \in A$ has a tree-decomposition $(T, W)_H$ such that $w(T, W)_H = \omega(H) - 1$ since H is chordal. Hence, since G is a subgraph of each $H \in A$, we have $\mathcal{W}(G) \leq \mathcal{W}(H) \leq w(T, W)_H = \omega(H) - 1$. Now we construct a chordal graph H such that $\mathcal{W}(G) \geq \omega(H) - 1$. For a tree-decomposition (T, W) of G such that $w(T, W) = \mathcal{W}(G)$,

we let $V(H) = V(G)$ and $E(H) = \bigcup_{t \in V(T)} \binom{W_t}{2}$. (T, W) is a tree-decomposition of H since: $\bigcup_{t \in V(T)} W_t = V(H)$; for all $uv \in E(H)$, $u, v \in W_t$ for some $t \in V(T)$; for all $v \in V(H)$, $\{t \in V(T) : v \in W_t\}$ is a connected subgraph of T, following from (T, W) being a tree-decomposition of G. Hence, H is chordal (by Lemma 3.3.8) since (T, W) is a tree-decomposition of cliques of H, so $\mathcal{W}(G) = w(T, W) \geq \omega(H) - 1$. Thus $\mathcal{W}(G) = \omega(H) - 1$.

□

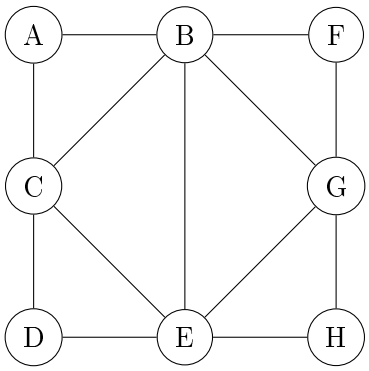
The intuition for this formulation of tree-width is that the width of any tree-decomposition of a graph must be at least the size of the largest clique in that graph. Conveniently, we may bound this above by the clique number of chordal supergraphs to attain our resulting equality. For additional clarity and completeness, we present the following proposition.

Proposition 3.3.11. *Let G and H be graphs with G a subgraph of H. Then $\omega(G) \leq \omega(H)$.*

Proof. Let $C \subseteq V(G)$ be a clique in G such that $\omega(G) = |C|$. Since G is a subgraph of H, for any edge $e \in E(G)$, $e \in E(H)$. Similarly, for any vertex $v \in V(G)$, $v \in V(H)$. Thus, it follows that C is also a clique of H. Hence we have $\omega(G) = |C| \leq \omega(H)$. □

Establishing that the clique number doesn't increase when taking subgraphs should now clarify the fact that cop number also does not increase, since if the robber player wishes to play optimally, he will exploit the cliques of a graph to allow the most "freedom of movement" in some sense. We finalise this section by returning to the graph shown in Example 3.1.5 and demonstrating the use of the results we have presented in this chapter.

Example 3.3.12. We saw previously that the below graph has a tree-decomposition of width 2. Thus, by Definition 3.1.4, the tree-width of this graph is ≤ 2. Hence, by Theorem 3.2.1, its cop number is ≤ 3.



We observe that this graph contains multiple instances of K_3 as subgraphs. We saw from Theorem 2.5.6 that $c(K_3) = 3$, and so, by Theorem 3.2.2, the cop number of this graph is ≥ 3. So we deduce that the minimum number of cops that may win on this graph is 3.

From this example, we have demonstrated the usefulness of tree-decompositions and subgraphs to bound the cop number of a given graph. While subgraphs can be readily observed from diagrams, it is not so obvious how we may obtain tree-decompositions with the required width in order to sufficiently bound the cop number of a graph. This will be the subject of the following chapter.

Chapter 4

Algorithms for Tree Width

In this chapter we will introduce the foundations for studying algorithms related to Cops and Robbers and tree-width.

4.1 Formalising Problems

Loosely, a problem is a question we seek to answer, described by its parameters and the properties that a solution must satisfy. An instance of a problem is a specific question obtained through the specification of values for the problem parameters. A decision problem is a question that must be answered with "yes" or "no."

Definition 4.1.1. A *decision problem* Π is a set of instances D_Π, the domain of Π, consisting of a set of "yes-instances" $Y_\Pi \subseteq D_\Pi$ and a set of "no-instances" $N_\Pi = D_\Pi \setminus Y_\Pi$.

Definition 4.1.2. An *algorithm* is a specified sequence of steps taken to solve a problem given an initial input. An algorithm solves a decision problem Π if, given an instance $I \in D_\Pi$, the algorithm determines if $I \in Y_\Pi$ or $I \in N_\Pi$.

We want to describe the efficiency of an algorithm, that is, the number of steps that it must take before it terminates. We can do this by measuring the growth of the number of basic operations as a function of the size of the input instance. This function is a measure of the worst-case instances for each input size, i.e. the maximum time the algorithm would take to terminate for an instance of that size.

Definition 4.1.3. Given two functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$, f is $\mathcal{O}(g)$ if there exist $c \in \mathbb{Z}^+$ and $n_0 \in \mathbb{N}$ such that, for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$.

Definition 4.1.4. An algorithm A (or the time complexity of an algorithm A) is $\mathcal{O}(g)$ for some function g if A performs $f(n)$ operations, in the worst case, on an input of size $n \in \mathbb{N}$ and f is $\mathcal{O}(g)$.

Definition 4.1.5. An algorithm A is a *polynomial time algorithm* if A is $\mathcal{O}(p(n))$ for some polynomial function $p : \mathbb{N} \to \mathbb{N}$.

We wish to speak about deterministic and non-deterministic algorithms, but the specific details are outside the scope of this report. We shall simply say that deterministic algorithms are algorithms that can be simulated by a “basic-while-program” on a Turing machine with a non-ambiguous grammar, and contrarily, a non-deterministic algorithm can be simulated by a "basic-while-program" on a Turing machine with an ambiguous grammar. Here, deterministic is taken, roughly, to mean that after completion of a step in the algorithm, there is only a single possible next step, i.e. there is no decision by the machine as to which step should be taken or parallelisation of steps, and vice versa for non-determinism. For greater depth, we refer to the works of Goldreich [9].

4.2 Complexity Classes

Here we describe the complexity classes $\mathbb{P}$ and $\mathbb{NP}$.

Definition 4.2.1. A *certificate* of a problem Π is an instance that is guessed to be in Y_Π.

Definition 4.2.2. The *complexity class* $\mathbb{P}$ is the class of problems solvable by a deterministic algorithm in polynomial time.

Definition 4.2.3. The *complexity class* $\mathbb{NP}$ is the class of problems Π such that, given a certificate $c \in D_\Pi$, there is a non-deterministic algorithm that can verify that the certificate is in Y_Π in polynomial time. Equivalently, $\mathbb{NP}$ is the class of problems solvable by a non-deterministic algorithm in polynomial time.

Clearly, by these definitions, $\mathbb{P} \subseteq \mathbb{NP}$, since any problem solvable, in polynomial time, by a deterministic algorithm can be solved similarly by a non-deterministic algorithm.

Definition 4.2.4. A *polynomial transformation* from a decision problem Π to a decision problem Ψ is a function $f : D_\Pi \to D_\Psi$ such that f is computable in polynomial time by a deterministic algorithm and for an instance $I \in D_\Pi$, $I \in Y_\Pi$ if and only if $f(I) \in Y_\Psi$. We write $\Pi \leq_m^p \Psi$ and say that Π reduces to Ψ.

Definition 4.2.5. The *complexity class* of $\mathbb{NP}$-hard problems is the class of problems Ψ such that, for any problem Π in $\mathbb{NP}$, $\Pi \leq_m^p \Psi$. The complexity class of $\mathbb{NP}$-complete problems is the class of $\mathbb{NP}$-hard problems also in $\mathbb{NP}$.

The following decision problem is an $\mathbb{NP}$-complete problem[2]:

TW Tree-width
Instance: A graph G and $k \in \mathbb{N}$.
Question: Does G have tree-width at most k?

That is, every decision problem in $\mathbb{NP}$ is reducible to TW, i.e. TW is at least as hard as every other problem in $\mathbb{NP}$. This is a significant barrier for us if we wish to study the tree-wdith of graphs in general. However, algorithms have been designed to exploit the structural properties of given graphs or to bound tree-width below a parameter of a certain form, and can even produce tree-decompositions with width of such a parameter. In particular Bodlaender produced a linear time algorithm that, for a fixed constant $k \in \mathbb{N}$, given a graph G, determines whether the tree-width of G is at most k, and if so, finds a tree-decomposition of G with tree-width at most k [3]. It does this by partitioning the vertex set of a graph into vertices of "high" and "low" degree, then using the theory of matchings and I-simplicial vertices to prove that a related graph has bounded tree-width from which the final result may be derived. This is a complex result and certainly outside the scope of this report.

$\mathbb{NP}$-complete problems such as TW are of particular algorithmic interest, since if it is possible to create a polynomial time algorithm to solve an $\mathbb{NP}$-complete problem Π, that is we may add Π to the more specific complexity class $\mathbb{P}$, then it is in fact the case that $\mathbb{P} = \mathbb{NP}$. This is because, for any decision problem $\Psi \in \mathbb{NP}$, we have $\Psi \leq_m^p \Pi$ since Π is $\mathbb{NP}$-complete. We confirm that this is true with the following results.

Lemma 4.2.6. *Let f and g be two functions computable in polynomial time. Then $g \circ f$ is computable in polynomial time.*

Proof. Suppose f is computable in time $\mathcal{O}(n^p)$ and g is computable in time $\mathcal{O}(n^q)$, with n the size of the input of f and g, and $p, q \in \mathbb{N}$. Then there exist $c_f \in \mathbb{Z}^+$ and $n_f \in \mathbb{N}$ such that f is computable in time $c_f \cdot n^p$ for inputs of size $n \geq n_f$. Similarly, there exist $c_g \in \mathbb{Z}^+$ and $n_g \in \mathbb{N}$ such that g is computable in time $c_g \cdot n^q$ for inputs of size $n \geq n_g$. Now choose $n_0 = \max\{n_f, n_g\}$, so the previous two statements still hold for inputs of size $n \geq n_0$. Then for an input $x \in \mathrm{dom}(f)$ of size $n \geq n_0$, since we must compute $f(x)$ first and then $g(f(x))$, we find that $(g \circ f)(x) = g(f(x))$ is computable in time $c_f \cdot n^p + c_g(c_f \cdot n^p)^q \leq c_f \cdot n^{pq} + c_f^q c_g \cdot n^{pq} = (c_f + c_f^q c_g) n^{pq} = \mathcal{O}(n^{pq})$ for $n \geq n_0$. Hence $g \circ f$ is computable in polynomial time. □

Theorem 4.2.7. *Given decision problems Π and Ψ, if $\Psi \in \mathbb{P}$ and $\Pi \leq_m^p \Psi$, then $\Pi \in \mathbb{P}$.*

Proof. Let A be an algorithm that solves Ψ in polynomial time and let $f : D_\Pi \to D_\Psi$ be the polynomial time transformation from Π to Ψ. Given $I \in D_\Pi$, we first compute $f(I)$ and then compute A on $f(I)$, solving Π. This is done in polynomial time by Lemma 4.2.6, hence $\Pi \in \mathbb{P}$. □

So we see that if we can prove, for some $\mathbb{NP}$-complete decision problem Π, that $\Pi \in \mathbb{P}$, then for any decision problem $\Psi \in \mathbb{NP}$, $\Psi \in \mathbb{P}$. Thus we now have $\mathbb{NP} \subseteq \mathbb{P}$ as well as $\mathbb{P} \subseteq \mathbb{NP}$ as we established before. Hence $\mathbb{P} = \mathbb{NP}$. This is, of course, yet to be proven or disproven, but such a proof would have serious implications for many graph theoretical problems, a large percentage of which are $\mathbb{NP}$-hard in the general case.

4.3 Separators

Here we will describe the concept of separators of graphs and vertex sets and their relation to tree-width.

Definition 4.3.1. Given a graph $G = (V, E)$, a set $S \subseteq V$ is a *separator* of G if $G \setminus S$ has more components than G. If S is a separator of G then S is said to separate G.

Definition 4.3.2. Let $G = (V, E)$ be a graph, with a separator $S \subseteq V$ of G, and two sets $A \subseteq V$ and $B \subseteq V$. Then S *separates* A from B if, for any vertices $a \in A \setminus S$ and $b \in B \setminus S$, there is no walk from a to b (or b to a) in $G \setminus S$.

Definition 4.3.3. Let $G = (V, E)$ be a graph and $W \subseteq V$. A *W-separator (balanced)* in G is a set $S \subseteq V$ such that, for every S-flap, C, of G, $|C \cap W| \leq \frac{|W|}{2}$ holds. The *separator width* of G, $\mathcal{SW}(G)$, is given by

$$\mathcal{SW}(G) = \min\{n \in \mathbb{N} : \text{for each } W \subseteq V \text{ there is a } W\text{-separator } S \text{ with } |S| \leq n\}.$$

Definition 4.3.4. Let $G = (V, E)$ be a graph and $W \subseteq V$. A *nearly balanced separation* of W in G is a triple (X, S, Y) where $X \subseteq W$, $Y \subseteq W$, and $S \subseteq V$ such that: S separates X and Y; $W = X \cup (S \cap W) \cup Y$; $0 < |X| < \frac{2}{3}|W|$ and $0 < |Y| < \frac{2}{3}|W|$.

This will allow us to describe an algorithm for creating tree-decompositions. The following two lemmas concerning tree-width are presented without proof (for proof see [1]) and will allow us to bound the tree-width of a graph in the subsequent theorem.

Lemma 4.3.5. *Let $G = (V, E)$ be a graph with $\mathcal{W}(G) = k$ and $W \subseteq V$. Then there exists a W-separator $S \subseteq V$ such that $|S| \leq k + 1$.*

Lemma 4.3.6. *Let $k \in \mathbb{N}$ and $G = (V, E)$ be a graph where, for all $W \subseteq V$ such that $|W| = 2k + 1$, there is a W-separator S in G with $|S| \leq k$. Then $\mathcal{W}(G) \leq 3k$.*

We can now bound the tree-width using separator width in the following theorem.

Theorem 4.3.7. *Let G be a graph. Then $\mathcal{SW}(G) - 1 \leq \mathcal{W}(G) \leq 3\mathcal{SW}(G)$.*

Proof. Assume $\mathcal{W}(G) = n \in \mathbb{N}$. Then by Lemma 4.3.5, given $W \subseteq V(G)$, there is a W-separator, S, in G with $|S| \leq n + 1$. $\mathcal{SW}(G) \leq |S|$ by Definition 4.3.3, so $\mathcal{SW}(G) \leq |S| \leq n + 1 = \mathcal{W}(G) + 1$. Hence, $\mathcal{SW}(G) - 1 \leq \mathcal{W}(G)$. We take $k = \mathcal{SW}(G)$ in Lemma 4.3.6 to attain the upper bound. Thus, we have $\mathcal{SW}(G) - 1 \leq \mathcal{W}(G) \leq 3\mathcal{SW}(G)$. □

4.4 The Separator Algorithm

In this section we present an algorithm, based on separators, to construct tree-decompositions of a specific width. This algorithm executes the steps of the proof of Lemma 4.3.6 presented in [1], in which it is shown that, for a graph G, if every $A \subseteq V(G)$ with $|A| = 2k + 1$ has a balanced separator S with $|S| \leq k$, then there is a tree-decomposition (T, W) of G such that $w(T, W) \leq 3k$ and $A = W_t \in W$ for some $t \in V(T)$. The algorithm in question, DECOMP, is shown in the figure below.

input : (G, A, k), where G is a graph, $k \in \mathbb{N}$, $A \subseteq V(G)$ with $|A| \leq 3k + 1$.
output: A tree-decomposition of width less than $4k + 1$ or determines that $\mathcal{W}(G) > k$.

```
1  begin
2      if e(G) > k · v(G) then
3          Return W(G) > k
4      else if v(G) ≤ 4k + 2 then
5          Return trivial tree-decomposition of G
6      else
7          Choose A' ⊇ A with |A'| = 3k + 1
8          if there exists a nearly balanced A'-separation (X, S, Y) with |S| ≤ k + 1 then
9              Define C_1, ..., C_m as S-flaps of G
10             for i ← 1 to m do
11                 Define G_i subgraph of G induced by (C_i ∪ S)
12                 A_i ← (C_i ∩ A') ∪ S
13                 (T_i, W_i) ← DECOMP(G_i, A_i, k)
14             end for
15             T ← (⋃_{i=1}^m V(T_i) ∪ {s, s'}, ⋃_{i=1}^m E(T_i) ∪ {ss'} ∪ {st_i : t_i ∈ V(T_i) for 1 ≤ i ≤ m})
16             W ← (⋃_{i=1}^m W_i ∪ {W_s = (A ∪ S), W_s' = A})
17             Return tree-decomposition (T, W)
18         else
19             Return W(G) > k
20         end if
21     end if
22 end
```

Algorithm 1: DECOMP

To obtain a tree-decomposition of G of width at most $4k+1$, we call DECOMP$(G, \varnothing, k)$. If such a tree-decomposition does not exist, then the algorithm determines that $\mathcal{W}(G) > k$. We now prove a bound on the running time of this algorithm using a recurrence relation. First note that lines 2-5 run in time $\mathcal{O}(k)$. Next we assume a worst case of $|S| = k+1$ within the algorithm and thus determine the S-flaps $C_1, ..., C_m$ of G as on line 11. We wish to choose the worst case for determining the vertices of S and applying DECOMP to each subgraph G_i on line 13. We do this by choosing vertices of S such that the operations performed by each call DECOMP(G_i, A_i, k) is maximised. In order to determine the running time of line 15, we need an additional theorem shown in [1] and extending the work of Ford and Fulkerson in [8], stated without proof:

Theorem 4.4.1. *Let G be a graph, $k \in \mathbb{N}$, and $W \subseteq V(G)$ such that $|W| = 3k+1$. If there exists a W-separator $S \subseteq V(G)$ with $|S| \leq k+1$, then S can be calculated in time $\mathcal{O}(3^{3k} \cdot k \cdot e(G))$.*

Thus, we immediately conclude that line 15 runs in time $\mathcal{O}(3^{3k} \cdot k \cdot e(G))$. Hence, we are able to formulate the recurrence relation R as

$$R(v(G)) = \begin{cases} \mathcal{O}(k) & \text{if } v(G) \leq 4k+2 \\ \max\{\sum_{i=1}^{m} R(v(G_i)) : m \geq 2\} + \mathcal{O}(3^{3k} \cdot k \cdot e(G)) & \text{otherwise} \end{cases}$$

We have $\sum_{i=1}^{m}(v(G_i) - (k+1)) = v(G) - (k+1)$ since each subgraph G_i is induced by $C_i \cup S$ with $C_i \cap S = \varnothing$. We now define a new relation R' such that $R'(n) = R(n+k+1)$. So we have

$$\begin{aligned} R'(v(G)) =& R(v(G)+k+1) \\ =& \begin{cases} \mathcal{O}(k) & \text{if } (v(G)+k+1) \leq 4k+2 \\ \max\{\sum_{i=1}^{m} R(v(G_i)+k+1) : m \geq 2\} + \mathcal{O}(3^{3k} \cdot k \cdot (e(G)+k+1)) & \text{otherwise} \end{cases} \\ =& \begin{cases} \mathcal{O}(k) & \text{if } v(G) \leq 3k+1 \\ \max\{\sum_{i=1}^{m} R(v(G_i)+k+1) : m \geq 2\} + \mathcal{O}(3^{3k} \cdot k \cdot (e(G)+k+1)) & \text{otherwise} \end{cases} \\ =& \mathcal{O}(3^{3k} \cdot k \cdot v(G)^2) \end{aligned}$$

Then we have $R(v(G)) = R'(v(G)-k-1) = \mathcal{O}(3^{3k} \cdot k \cdot v(G)^2)$ by the definition of R'. Hence, we conclude that DECOMP runs in time $\mathcal{O}(3^{3k} \cdot k \cdot v(G)^2)$. Since this bound is exponential in k, this should only be implemented when k is very small. For example, taking $k = 4$, we perform $\mathcal{O}(3^{12} \cdot 4 \cdot v(G)^2) = \mathcal{O}(2125764 \cdot v(G)^2)$ operations, even without yet considering the number of vertices in the input graph. In the next section we will give an example of a faster algorithm, but one that is not guaranteed to result in a tree-decomposition of the input graph.

4.5 k-good Tree-decompositions

In this section we will define some properties of graphs that will be used to produce a further algorithm.

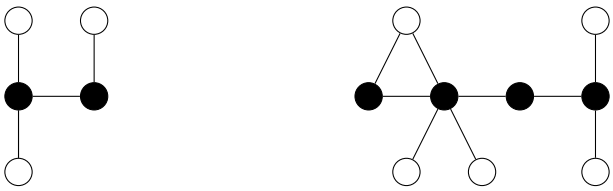
Definition 4.5.1. Let G be a graph. The *chordality* of G is the length of the longest chordless cycle (a cycle with no chord) in G. G is said to be *k-chordal* if it has chordality k for some $k \in \mathbb{N}$.

Note. A graph being "chordal" by Definition 3.3.3 does not imply that it is 3-chordal, since a chordal graph may have no cycles. The converse is true and follows directly from the definitions.

Definition 4.5.2. For integer $k > 1$, a *k-caterpillar* is a graph G such that there exists a set $P \subseteq V(G)$ inducing a path in G of length $< k$ where, for every vertex $v \in V(G)$, either $v \in P$ or there exists $v' \in P$ such that $v \in N_G(v')$.

Caterpillars are an important idea in the functionality of the algorithm we wish to create, so for clarity we illustrate the concept of caterpillars in the following example. They will be used to formulate a possible tree-decomposition to bound the tree-width of a graph, as will see in a later theorem.

Example 4.5.3. 3- and 5-caterpillars, the former of which also happens to be a tree.



Definition 4.5.4. A tree-decomposition (T, W) of a graph G is *k-good* if for all $W_t \in W$, the vertices in W_t induce a k-caterpillar in G.

As a matter of fact, if we can confirm that there exists a k-good tree-decomposition of a graph, then we can bound its tree-width from above. We confirm this with the subsequent proposition.

Proposition 4.5.5. *Let G be a graph with a k-good tree-decomposition and Δ the maximum degree of any vertex in G. Then $\mathcal{W}(G) \leq (k-1)(\Delta-1)+2$.*

Proof. Let (T, W) be a k-good tree-decomposition of G for some $k > 1$. We will find an upper bound for the size of a bag in (T, W) and, therefore, an upper bound on $w(T, W)$. For any set $W_t \in W$, W_t induces a k-caterpillar. That is, W_t contains an induced path P of order $\leq k-1$. Suppose, towards an upper bound, there exists W_u, for some $u \in V(T)$, containing a path of order $k-1$. We must have, for any $v \in P$, $d_G(v) \leq \Delta$, so assume a worst case of $d_G(v) = \Delta$ for all $v \in P$. Then, the two endpoints of P are adjacent to at most $\Delta - 1$ vertices not in P. All other vertices of P are adjacent to at most $\Delta - 2$ vertices not in P. Thus it follows

$$\begin{aligned}
|W_u| &\leq 2(\Delta-1) + (k-3)(\Delta-2) + (k-1) \\
&= 2\Delta - 2 + k\Delta - 2k - 3\Delta + 6 + k - 1 \\
&= k\Delta - k - \Delta + 3 \\
&= (k\Delta - k - \Delta + 1) + 2 \\
&= (k-1)(\Delta-1) + 2
\end{aligned}$$

Hence we have

$$\mathcal{W}(G) \leq w(T, W) \leq |W_u| \leq (k-1)(\Delta-1)+2$$

□

While this may seem a useful result upon first sight, it is in fact simply confirming an obvious fact - since we have an upper bound on the vertex degree and an upper bound on a specific set

of vertices to which others must be adjacent, we can easily attain an upper bound on the size of a bag by multiplying these parameters in some sense. If we have already deduced that a graph admits a k-good tree-decomposition by finding an example, then it is likely that our previous result will not help us at all, since it only states the maximum width. Nevertheless, it is included to demonstrate rigor in our concepts.

We present the following theorem without proof. The extensive proof, attributed to Kosowski, Li, Nisse, and Suchan, can be seen in [11].

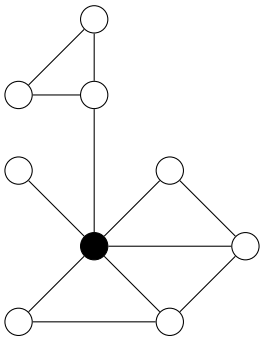
Theorem 4.5.6. *Let G be a graph and $k > 2$ an integer. Then there exists an algorithm that, given inputs G and k, returns either a chordless cycle of length at least $k + 1$ or a k-good tree-decomposition of G.*

Using this, we may now make a simple deduction about the tree-width of k-chordal graphs:

Corollary 4.5.7. *Let G be a k-chordal graph for integer $k > 1$ with maximum degree Δ. Then $\mathcal{W}(G) \leq (k-1)(\Delta-1)+2$.*

Proof. Running the algorithm from Theorem 4.5.6 returns either a chordless cycle of length at least $k+1$ in G or a k-good tree-decomposition of G. Since G is k-chordal, it contains no chordless cycle of length greater than k, so must have a k-good tree-decomposition. Thus, we get $\mathcal{W}(G) \leq (k-1)(\Delta-1)+2$ by Proposition 4.5.5. □

Example 4.5.8. We show the use of our corollary on the 3-chordal graph below. The marked vertex has the highest degree of 6.



We determine from Corollary 4.5.7 that this graph has tree-width at most $2 \cdot 5 + 2 = 12$. From inspection we see that this graph is also chordal, and so we obtain a tree-width of 2 by Theorem 3.3.9.

Thus we see the value in different approaches to determining tree-width - it is unnecessary to apply lengthy algorithms to small graphs where one can easily derive the answer through inspection, and it is also imperative to choose the correct result to apply when deriving tree-width manually as one can obtain vastly different estimations depending on the structural qualities of the graph in question.

4.6 k-good Algorithm

We now formulate the algorithm DECOMP2 described in [11] and proven to exist in Theorem 4.5.6 based upon k-good tree-decompositions. It is shown in the following figure. Note that on line 14 we assume the existence of a leaf bag $W_{t_i} \supset S_i$, which we may do by the proof of correctness by Kosowski, Li, Nisse, and Suchan. We see that line 2 and lines 5-8 run in time $\mathcal{O}(e(G))$ since we determine both the components of $G \setminus N_G(v)$ and bags W_t by iterating edges. For the remainder of the algorithm, each pass of the loop on line 10 takes $\mathcal{O}(e(G)^2)$ time and we perform $v(G) - 1$ loops. So, for this formulation of the algorithm we obtain a time-complexity of $\mathcal{O}(v(G)e(G)^2)$, but it is suggested in [11] that a faster implementation can be achieved.

input : (G, k), where G is a graph and $k > 2$ an integer.
output: A chordless cycle in G of length $> k$ or a k-good tree-decomposition of G.

```
1  begin
2      Choose arbitrary v ∈ V(G) and compute components C_1, ...C_j of G \ N_G(v)
3      T ← (V(T), E(T)) = ({t}, ∅)
4      W ← {W_t} where W_t = N_G(v)
5      for i ← 1 to j do
6          Add t_i to V(T) adjacent to t
7          Add W_{t_i} = {v} ∪ {w ∈ N_G(v) : N_G(w) ∩ C_i ≠ ∅} to W
8      end for
9      G_0 ← {v}
10     for v' ∈ V(G) \ {v} do
11         Compute components C_1, ..., C_ℓ of G \ G_0
12         for i ← 1 to ℓ do
13             S_i ← {u ∈ V(G_0) : u adjacent to a vertex of C_i}
14             P_i ← W_{t_i} \ S_i
15             Choose arbitrary w ∈ S_i such that Q = P_i ∪ {w} is a chordless path
16             Add a vertex t'_i adjacent to t_i in T
17             Add W_{t'_i} = (Q ∪ W_{t_i} ∪ (N(w) ∩ C_i)) to W
18             if |Q| > k then
19                 Return induced cycle W_{t'_i}
20             else
21                 Compute components C'_1, ..., C'_r of (C_i ∪ W_{t_i}) \ W_{t'_i}
22                 for h ← 1 to r do
23                     S'_h ← {u ∈ S_i : u is adjacent to a vertex of C'_h}
24                     Q_h ←smallest subgraph of Q such that every vertex of S'_h is in Q_h or is
                        adjacent to a vertex in Q_h
25                     Add vertex t''_h adjacent to t'_i in T
26                     Add W_{t''_h} = Q_h ∪ S'_h to W
27                 end for
28             end if
29         end for
30         Add v' to G_0
31     end for
32     Return k-good tree-decomposition (T, W)
33 end
```

Algorithm 2: DECOMP2

To find a k-good tree-decomposition of a graph G using this algorithm, it is required to run DECOMP2(G, k) with increasing values of k until a success is found. However, unlike with the previous algorithm DECOMP, DECOMP2 does not determine if the tree-width of a graph is greater

than our chosen parameter, and so it may be required to first find the upper limit at which to halt. Since it may be the case that, for any given k, our input graph G does indeed satisfy $\mathcal{W}(G) \leq k$ but does not have a k-good tree-decomposition, using this algorithm to estimate tree-width may lead to an overestimation.

We now briefly discuss how these algorithms compare. The running time of DECOMP is exponential in k, so, as discussed before, it is inefficient to run this algorithm for large values of k. Contrarily, the running time of DECOMP2 does not depend on k, and so higher values of k do not increase the running time of this algorithm. Thus it is perhaps more efficient to run DECOMP2 instead of DECOMP for high values of k. However, as previously stated, DECOMP2 may not even produce a tree-decomposition of the input graph - so it is recommended, to save running DECOMP2 an excessive number of times, that DECOMP2 is run only on graphs that have already been determined to have a k-good tree-decomposition or DECOMP2 is only run up to a pre-determined bound on k derived from inspection of the graph. Of course, if a k-good tree-decomposition is required, then DECOMP2 is the preferable option, since DECOMP is not guaranteed to produce such a tree-decomposition. In contrast to high values of k, we may also consider high values of $v(G)$ and $e(G)$ for an input graph G. Since for a simple graph G we have $e(G) \leq v(G)^2$ by Theorem 1.3.4, we obtain a running time of $\mathcal{O}(v(G)^5)$ for DECOMP2. So, for low values of k and high values of $v(G)$ or $e(G)$, we see that it is better to choose DECOMP. Specifically we require $3^{3k} \cdot k \leq v(G)^3$ for DECOMP to be the better choice.

Chapter 5

Strategies for Robbers

We have previously discussed strategies for cops and their relation to tree-decompositions. We now wish to outline a strategy for the robber player to evade capture. We said briefly that a robber player should utilise the cliques of a graph to effectively elude cops, and in this chapter we shall clarify this statement.

5.1 Havens and Screens

To describe evasion strategies for robbers we must define a new type of function. The robber can use this function to tell him where to move to avoid capture.

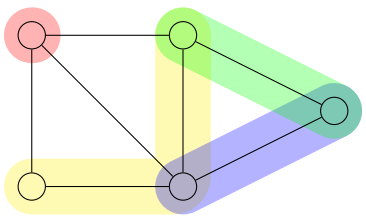
Definition 5.1.1. Given an graph G, two subsets $X, Y \subseteq V(G)$ *touch* if either $X \cap Y \neq \varnothing$ or there exist vertices $u \in X$ and $v \in Y$ such that $uv \in E(G)$.

Definition 5.1.2. Let G be a graph. A *haven* of order $k \in \mathbb{N}$ in G is a function β that associates each subset $X \in [V(G)]^{<k}$ with an X-flap $\beta(X)$, such that $\beta(X)$ touches $\beta(Y)$ for all $X, Y \in [V(G)]^{<k}$

We may also describe a robber's strategy in terms of touching sets of vertices.

Definition 5.1.3. Given a graph G, a *screen* $S \subseteq \mathcal{P}(V(G))$ in G is a set of subsets of connected vertices in G such that, for any two sets $A, B \in S$, A and B touch. We say S has *thickness* at least $k \in \mathbb{N}$ ($\tau(S) \geq k$) if, for all $X \in [V(G)]^{<k}$ there is some $H \in S$, $X \cap H = \varnothing$.

Example 5.1.4. Here is an example of a screen in a graph. Each differently coloured set of vertices is a different set in the screen.



From inspection we see that this screen has thickness ≥ 3, since we cannot choose a set of 1 or 2 vertices that intersects all sets in the screen. We may, however, choose a set of 3 vertices that intersects all sets in the screen, so this screen must indeed have thickness 3. Note that this is not necessarily the only screen of thickness 3 in this graph.

The two previous definitions are in fact equivalent, as we prove with the following lemma. Seymour and Thomas used screens in their paper, but many authors prefer to use "brambles," which are also equivalent to screens and havens. They are sets of connected, touching subsets of vertices similar to screens, but their order is defined to be the size of the smallest set of vertices intersecting all sets in the bramble.

Lemma 5.1.5. *Let G be a graph. Then G has a screen of thickness at least k if and only if G has a haven of order at least k.*

Proof. First assume that G has a screen S such that $\tau(S) \geq k \in \mathbb{N}$. Then, by definition, for all $X \in [V(G)]^{<k}$, there exists $H \in S$ with $X \cap H = \varnothing$. Now let β be a function mapping $X \in [V(G)]^{<k}$ to the X-flap containing such an $H \in S$. We claim that β is a haven of order k and now prove that claim. Take $X, Y \in [V(G)]^{<k}$ with $H_X, H_Y \in S$ such that $X \cap H_X = \varnothing$ and $Y \cap H_Y = \varnothing$, so we have X-,Y-flaps $\beta(X) \supseteq H_X$ and $\beta(Y) \supseteq H_Y$. Now, since H_X and H_Y touch, if $H_X \cap H_Y \neq \varnothing$, we have $\beta(X) \cap \beta(Y) \neq \varnothing$ and we are done. Otherwise there exist vertices $u \in H_X$ and $v \in H_Y$ with $uv \in E(G)$. Hence, we also have $u \in \beta(X)$ and $v \in \beta(Y)$ with $uv \in E(G)$. So in either case $\beta(X)$ and $\beta(Y)$ touch. Thus β is indeed a haven of order k. Now assume that β is a haven of order at least k in G. We claim that the set $S = \{\beta(X) : X \in [V(G)]^{<k}\}$ is a screen with $\tau(S) \geq k$ and prove this. Since β is a haven of order at least k, we deduce that each $\beta(X) \in S$ is an X-flap (and is therefore connected by Definition 2.2.3) and all sets in S are touching. Now for any subset $X \in [V(G)]^{<k}$ we have that$X \cap \beta(X) = \varnothing$, and hence S is a screen in G with $\tau(S) \geq k$. □

Thus far, the concepts we have established may not seem obviously connected to any form of strategy in Cops and Robbers, but we may extend our earlier theorem to include our new developments.

Theorem 5.1.6 (3.2.1 extended)**.** *Let G be a graph and $k \in \mathbb{N}$. Then the following are equivalent:*

(*i*) $c(G) = k$;

(*ii*) $c_M(G) = k$;

(*iii*) $\mathcal{W}(G) = k - 1$;

(*iv*) *G has a screen of thickness k;*

(*v*) *G has a haven of order k.*

We have already proven (*iv*) if and only if (*v*) in Lemma 5.1.5 and (*iii*) implies (*iv*) is the main focus of [13]. So we are left to prove (*v*) implies (*i*). From this theorem we conclude that a graph with a haven of order $> k$ or screen of thickness $> k$ does not permit a k-cop winning strategy, and hence there may be some special property of these concepts that a robber may utilise for success. This statement will become clear when we prove (*v*) implies (*i*) in the next section.

5.2 Robber Strategies from Havens

In this section we prove the link between havens and cop number seen in Theorem 5.1.6 and discuss how this gives the robber an evasion strategy. This, however, in a game with infinite moves and a perfect cop player, is the only way for the robber player to not lose. If such a haven or screen does not exist then the robber player is certain to lose, and so it is imperative that he takes advantage of the strategy that we shall demonstrate.

Lemma 5.2.1. *Let G be a graph and $k \in \mathbb{N}$. If G has a haven of order k, then $c(G) = k$.*

Proof. Assume G has a haven β of order k. We will first show that $< k$ cops cannot win and then show that k cops can win. Consider the game $\mathcal{CR}(G, k-1)$ with a sequence of moves $(\varnothing, R_0), (X_1, R_1), ...$ possibly infinite. For any cop move X_i, we have that $X_i \in [V(G)]^{<k}$ by definition, and hence we may apply β to such a move to obtain an X_i-flap. So the robber may choose $R_i = \beta(X_i)$ at any step in the game. He can, of course, do this because the component of the previous move $R_{i-1} = \beta(X_{i-1})$ touches R_i since β is a haven. Thus, since each robber move R_i is an X_i-flap, the cop player can never achieve the condition $R_{i-1} \subseteq X_i$ and so the cop player can never win. Now consider $\mathcal{CR}(G, k)$. Since β is not a haven of order $k+1$ in G, there exists some cop move X_i for which no X_i-flap touches another X_j-flap for some cop move X_j. Executing the move X_i allows the cop to corner the robber, since they now cannot move from the X_i-flap they have chosen. Hence, the cop player has created a new induced graph G' with vertices equal to the X_i-flap in which they may now win. □

With this lemma we complete the proof of Theorem 5.1.6. The proof of this lemma has also given us the strategy for the robber player: for a graph G with haven β, choose the move $R_i = \beta(X_i)$ for $i \geq 1$. This will ensure the survival of the robber should the cop player have fewer cops than the order of the haven. We also see that the robber can survive if and only if such a haven exists, so capture is inevitable in $\mathcal{CR}(G, k)$ if no haven of order $> k$ exists. It is easy to see the comparative use of screens in a robber strategy. That is, for any graph G with screen S such that $\tau(S) > k$, the robber can survive in $\mathcal{CR}(G, k)$ by selecting their move R_i to be $R_i = H \in S$ such that $X_i \cap H = \varnothing$ which is guaranteed to exist by the definition of S. Hence, the cop may never achieve $X_i \supseteq R_{i-1}$ to win.

To conclude this chapter, we will, as discussed earlier, comment on how cliques may be used by the robber player to evade capture. The following propositions will allow us to do this.

Proposition 5.2.2. *Let G be a graph with a clique C and G_C the subgraph of G induced by C. Then there exists a haven β_C of order $|C|$ in G_C.*

Proof. We claim that $\beta_C(X) = V(G_C) \setminus X$ for $X \in [V(G_C)]^{<|C|}$ is a haven of order $|C|$. Take arbitrary $X, Y \in [V(G_C)]^{<|C|}$ and consider the X-,Y-flaps $\beta_C(X) = V(G_C) \setminus X$ and $\beta_C(Y) = V(G_C) \setminus Y$ in G_C. If we find that $\beta_C(X) \cap \beta_C(Y) \neq \varnothing$ then we are done. Otherwise, if we find that $\beta_C(X) \cap \beta_C(Y) = \varnothing$, then, since for all distinct $u, v \in V(G_C)$, $uv \in E(G_C)$, we have $uv \in E(G_C)$ for all $u \in \beta_C(X)$ and $v \in \beta_C(Y)$. Hence $\beta_C(X)$ and $\beta_C(Y)$ touch for any $X, Y \in [V(G_C)]^{<|C|}$, so β_C is a haven of order $|C|$ in G_C. □

Proposition 5.2.3. *Let G be a graph with a clique C and G_C the subgraph of G induced by C. Then there exists a screen S_C of thickness $|C|$ in G_C.*

Proof. We claim that $S = \{\{v\} : v \in V(G_C)\}$ is a screen of thickness $|C|$ in G_C. Each singleton in S is a connected set of vertices and since we have $uv \in E(G_C)$ for all distinct $u, v \in V(G_C)$, all sets in S touch. Now take arbitrary $X \in [V(G_C)]^{<|C|}$. Since $|X| < |C| = |V(G_C)|$, there exists some vertex $v \in V(G_C)$ not in X. We have $\{v\} \in S$ by definition, so we have $X \cap \{v\} = \varnothing$. Hence S is a screen in G_C with $\tau(S) = |C|$. □

From these propositions, we determine that, for a graph G with a clique of size $k+1$, the robber can evade capture in $\mathcal{CR}(G,k)$ by using the haven or screen for the clique that we demonstrated in the propositions above. Thus, in a graph with a clique of size larger than the number of available cops, the cop player will never win if playing against a robber who uses such a strategy. We see that this is consistent with Theorem 3.2.2 since cliques induce complete graphs, which then force the cop number of the supergraph to be at least the order of the complete subgraph.

Chapter 6

Conclusion

To conclude this report, we reflect on the results we have observed herein. We follow this with a personal reflection on the achievements of this project.

6.1 Main results

In Chapter 1 we simply presented the foundations for the theoretical knowledge required to understand the material in the following chapters. Chapter 2 saw the formulation of the game that forms the focus of this report and original proofs for the cop number of some graphs discussed in Chapter 1. We also posed a crucial question that we wished to answer. In Chapter 3 we described some key concepts that allowed us to gain a deeper understanding of Cops and Robbers, for example tree-decompositions and tree-width, and also discussed a very important result in Theorem 3.2.1. These concepts allowed us to answer our question from Chapter 2 and to precisely determine the properties of graphs with a given cop number. We then explored how our new results can be expressed in terms of cliques and chordal graphs, culminating in the equally important Theorem 3.3.10. Chapter 4 introduced algorithmic concepts to study the $\mathbb{NP}$-complete problem of determining the tree-width of a graph, demonstrating two contrasting algorithms for tree-decompositions that help solve this problem. We also saw the useful bound in Theorem 4.3.7 and discussed the underlying theoretical results that these algorithms utilise to produce tree-decompositions. Finally, Chapter 5 saw the discussion of strategies for the robber player, previously left without much mention. Here we presented the extended Theorem 5.1.6 and proved how the robber can use two equivalent concepts to continually evade the cops.

6.2 Evaluation and Self-assessment

Firstly, addressing the deliverables stated at the beginning of this report, I believe all promised deliverables have been met and that this has indeed resulted in the completion of the stated outcomes. I formulated these outcomes and deliverables before prior research into the subject of this project (although having met the cops and robbers game and graph theory before) and so in hindsight they are very vague, though they are perhaps the best I could create given very little knowledge of the subject. Since I decided that I wanted this project to be a fully theoretical one - purely out of interest in theoretical computer science - it was difficult to accurately pin down precise goals for myself since there would be no practical outcomes such as "create a piece of software to demonstrate the concepts involved in cops and robbers" and practical deliverables such as "User testing consent forms and feedback." This is also reflected in the planning section which, since I needed to perform extensive research into the game and related concepts, is also kept vague. I found the schedule to be of little use, since the outlining of goals took very little time - as did the review of basic concepts in graph theory since I had already encountered the fundamentals throughout previous years of my education - leaving the remaining time to "research" unknown material. The meetings with my supervisor were very helpful in this regard,

as my supervisor has extensive knowledge in this field and could give me guidance as to what I should look into when lacking direction.

It could be argued that much of the first chapter of this document could be removed without the loss of any depth of knowledge, however I believe that ensuring that I demonstrate my knowledge of the fundamentals of this topic is key to showing the reader that I am able to tackle the harder material that follows. I also think that including basic concepts at the outset aids to keep this document more self-contained and less reliant on work done by others. I think I have demonstrated a high level of understanding through the proofs I have written throughout this report, and the explanations and examples show that I can also explain the material at a more basic level that is easier to follow.

I have learned much about the theory presented in this project, as well as the key papers and authors in the field, and would enjoy continuing to study this area of computer science. This material was challenging at first, but quickly becomes intuitive once you begin to grasp the concepts. I quickly discovered that this topic becomes very circular, in that many earlier results can be easily proved by later ones, and seemingly unrelated properties can be connected to the far-reaching topic of tree-decompositions - e.g. cop number, chordal graphs, cliques, separators, havens. I believe I have discussed all of the topics that are needed, though given more time I would have liked to explore havens and screens a little further, and I have included both a good depth and breadth of knowledge in this report. There is perhaps also the potential to study the link between cops and robbers and other graph theoretic problems, such as graph colouring and graph connectivity, through the lens of tree-decompositions and tree-width.

In terms of ethical considerations, I believe I have adhered to the issues raised at the beginning of this document. One social/ethical issue I did not anticipate at the beginning of this project was the accessibility issue surrounding the use of colour in diagrammatic examples. This may be a concern for individuals with certain conditions that wish to read this report, e.g. colour blindness or partial-sightedness. To address this, I have ensured that wherever colour is used, the colours are highly contrasting, the objects are also labeled with text, and that the actual colours themselves are not the main focus of the example (i.e. we don't refer to a specific object by its colour, merely that there are different contrasting colours present, e.g. Example 5.1.4). Where we refer to a specific, single, unlabeled object in a diagram, we simply use black and white, e.g. Example 4.5.8.

The language used throughout and the style of this report are appropriate in my opinion. I have attempted to use formal and precise mathematical language which is suited to the type of project and aids with clarity. The "theorem-proof" style of formatting and writing is obviously appropriate for mathematics and is kept consistent throughout. This is also useful for cross-referencing, as specific theorems and results can be referenced easily for immediate access to the specified information. Both writing this document and reading the relevant literature have been a helpful insight into how academic articles should be written, and it can become quite an easy style to replicate.

I believe this document, if edited slightly to be made more suitable, could serve as some form of coagulation of literature on the topic of cops and robbers and tree-width. This could be useful as an entry point for studying the topic, since this report reads similarly to lecture notes anyway. Though this project does not contain any new breakthrough results, it is a useful guide

through the already existing material that, in my opinion, is also easy to read and comprehend. Were I to continue further work on this document, I would like to emphasise more the circular nature of the main results discussed in this project. There are many deep equivalences that occur throughout this material that take a great deal of work to prove, e.g. the equivalence of tree-width and havens from [13], which I would like to either discuss or prove. There are also much more complex algorithms than the ones demonstrated in this report, e.g. those of Bodlaender, to which the content of this document may be extended, though they are significantly harder to grasp and explain.

In all, despite the vague objectives of this project, I believe it has been a successful attempt at an exposition of the literature surrounding cops and robbers. It has been a challenging but enjoyable experience to write a mathematical report at such a high level and an interesting insight into academic reading and writing.

References

[1] I. Adler. Lecture notes on tree decompositions, algorithms and logic, 2014. Goethe University, Frankfurt.

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, 1987.

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

[4] H. L. Bodlaender and R. H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. on Discrete Mathematics*, 6(2):301–309, 1990.

[5] J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer, 2008.

[6] R. Diestel. *Graph Theory*. Springer, 2005.

[7] D. Eppstein. File:tree decomposition.svg — wikimedia commons, the free media repository, 2015. `https://commons.wikimedia.org/wiki/File:Tree_decomposition.svg` [Online; accessed 29-November-2019].

[8] L. R. J. Ford and D. R. Fulkerson. A simple algorithm for finding maximal network flows and an application to the hitchcock problem. *Canadian Journal of Mathematics*, 9:210–218, 1957.

[9] O. Goldreich. *P, NP, and NP-Completeness*. Cambridge University Press, 2010.

[10] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Elsevier, 2004.

[11] A. Kosowski, B. Li, N. Nisse, and K. Suchan. k-chordal graphs: From cops and robber to compact routing via treewidth. *Algorithmica*, 72:610–622, 2012.

[12] A. S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40(2):224–245, 1993.

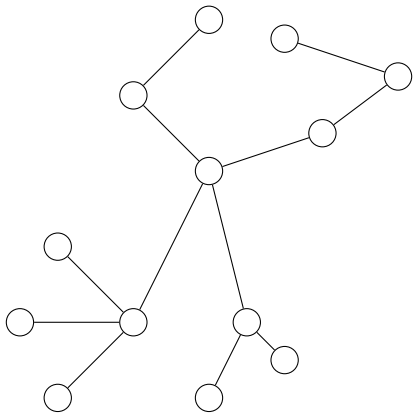
[13] P. D. Seymour and R. Thomas. Graph searching, and a min-max theorem for tree-width. *J. Combin. Theory Ser. B*, 58(1):22–23, 1993.

Appendices

Appendix A

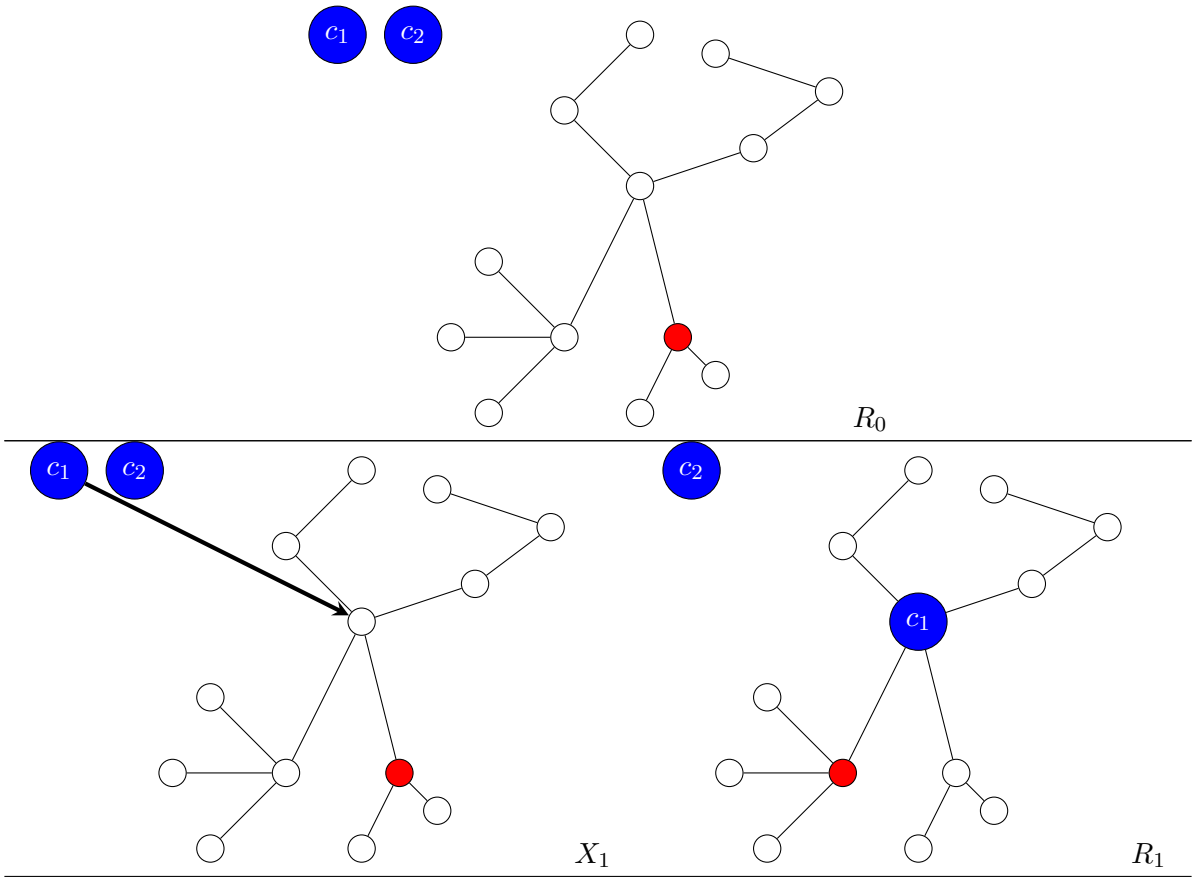
Further Examples

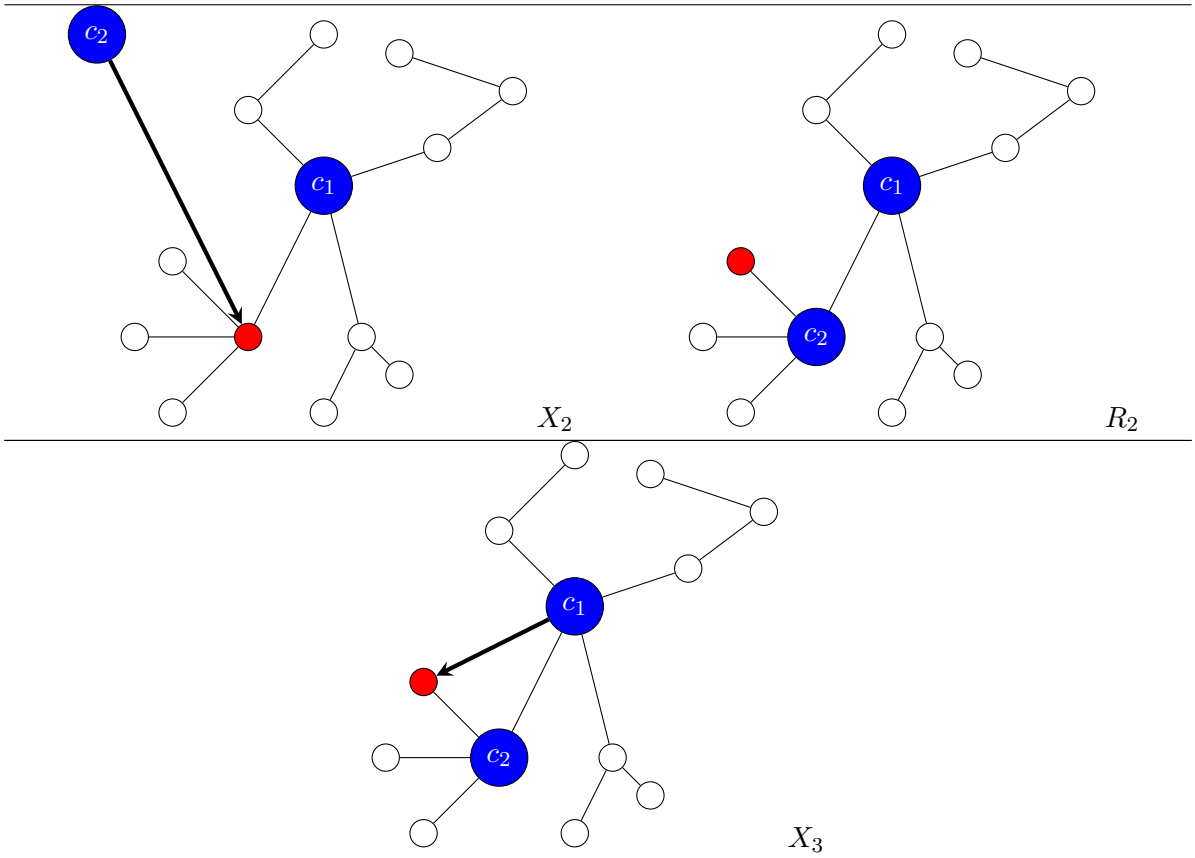
Example A.0.1. We define the tree T below and consider $\mathcal{CR}(T, 2)$.



T

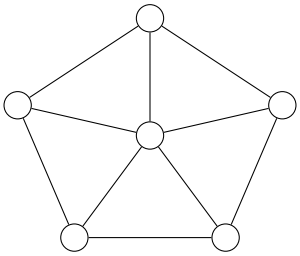
We now proceed with cops $\{c_1, c_2\}$.





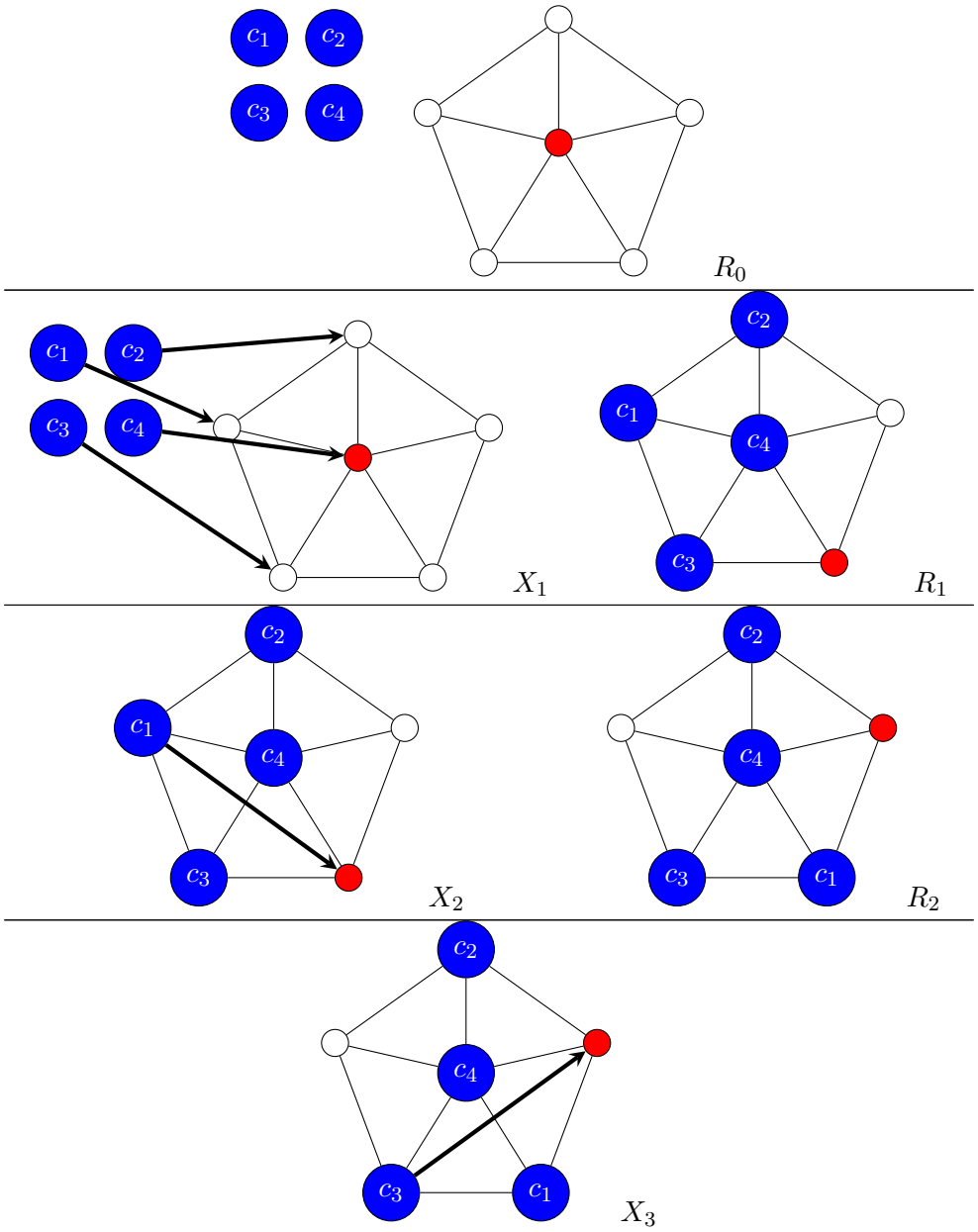
We have $R_2 \subseteq X_3$ and hence the cop player wins.

Example A.0.2. We define the graph G below and consider $\mathcal{CR}(G, 4)$.



G

We now proceed with cops $\{c_1, c_2, c_2, c_4\}$.



We have $R_2 \subseteq X_3$ and hence the cop player wins.