Contents

Contributors

1

Graph Searching Games

Stephan Kreutzer

Abstract

This chapter provides an introduction to graph searching games, a form of one- or two-player games on graphs that have been studied intensively in algorithmic graph theory. The unifying idea of graph searching games is that a number of searchers wants to find a fugitive on an arena defined by a graph or hypergraph. Depending on the precise definition of moves allowed for the searchers and the fugitive and on the type of graph the game is played on, this yields a huge variety of graph searching games.

The objective of this chapter is to introduce and motivate the main concepts studied in graph searching and to demonstrate some of the central ideas developed in this area.

1.1 Introduction

Graph searching games are a form of two player games where one player, the *Searcher* or *Cop*, tries to catch a *Fugitive* or *Robber*. The study of graph searching games dates back to the dawn of mankind: running after one another or after an animal has been one of the earliest activities of mankind and surely our hunter-gatherer ancestors thought about ways of optimising their search strategies to maximise their success.

Depending on the type of games under consideration, more recent studies of graph searching games can be traced back to the work of Pierre Bouger, who studied the problem of a pirate ship pursuing a merchant vessel, or more recently to a paper by Parsons [1978] which, according to Fomin and Thilikos [2008], was inspired by a paper by Breisch in the Southwestern Cavers Journal where a problem similar to the following problem was considered:

suppose after an accident in a mine some workers are lost in the system of tunnels constituting the mine and a search party is sent into the mine to find them. The problem is to devise a strategy for the searchers which guarantees that the lost workers are found but tries to minimise the number of searchers that need to be sent into the mine. Graph-theoretically, this leads to the following formulation in terms of a game played on a graph which is due to Golovach [1989]. The game is played on a graph which models the system of tunnels, where an edge corresponds to a tunnel and a vertex corresponds to a crossing between two tunnels. The two players are the Fugitive and the Searcher. The Fugitive, modelling the lost worker, hides in an edge of the graph. The Searcher controls a number of searchers which occupy vertices of the graph. The Searcher knows the graph, i.e. the layout of the tunnels, but the current position of the fugitive is unknown to the Searcher. In the course of the game the searchers search edges by moving along an edge from one endpoint to another trying to find the fugitive.

This formulation of the game is known as *edge searching*. More popular in current research on graph searching is a variant of the game called *node searching* which we will describe now in more detail.

Node Searching

In node searching, both the fugitive and the searchers occupy vertices of the graph. Initially, the fugitive can reside on any vertex and there are no searchers on the graph. In each round of the play, the Searcher can lift some searchers up or place new searchers on vertices of the graph. This can happen within one move, so in one step the searcher can lift some searchers up and place them somewhere else. However, after the searchers are lifted from the graph but before they are placed again the fugitive can move. He can move to any vertex in the graph reachable from his current position by a path of arbitrary length without going through a vertex occupied by a searcher remaining on the board. In choosing his new position, the fugitive knows where the searchers want to move. This is necessary to prevent "lucky" moves by the searchers where they accidentally land on a fugitive. The fugitive's goal is to avoid capture by the searchers. In our example above, the fugitive or lost miner would normally not try to avoid capture. But recall that we want the search strategy to succeed independent of how the lost miner moves, and this is modelled by the fugitive trying to escape. If at some point of the game the searchers occupy the same vertex as the fugitive then they have won. Otherwise, i.e. if the fugitive can escape forever, then he wins. The fact that the fugitive tries to avoid capture by a number of searchers has led to these kind of games being known as *Cops and Robber* games in the

literature and we will at various places below resort to this terminology. In particular, we will refer to the game described above as the *invisible Cops and Robber Game*. The name "Cops and Robber game", however, has also been used for a very different type of games. We will describe the differences in Section 1.3.1.

Optimal Strategies

Obviously, by using as many searchers as there are vertices we can always guarantee to catch the fugitive. The main challenge with any graph searching game therefore is to devise an optimal search strategy. There are various possible optimisation goals. One is to minimise the number of searchers used in the strategy. Using as few searchers as possible is clearly desirable in many scenarios, as deploying searchers may be risky for them, or it may simply be costly to hire the searchers. Closely related to this is the question whether with a given bound on the number of searches the graph can be searched at all.

Another very common goal is to minimise the time it takes to search the graph or the number of steps taken in the search. In particular, often one would want to avoid searching parts of the graph multiple times. Think for instance of the application where the task is to clean a system of tunnels of some pollution which is spreading through the tunnels. Hence, every tunnel, once cleaned, must be protected from recontamination which can only be done by sealing off any exit of the tunnel facing a contaminated tunnel. As cleaning is likely to be expensive, we would usually want to avoid having to clean a tunnel twice. Search strategies which avoid having to clean any edge or vertex twice are called *monotone*.

On the other hand, sealing off a tunnel might be problematic or costly and we would therefore aim at minimising the number of tunnels that have to be sealed off simultaneously. In the edge searching game described above, sealing off a tunnel corresponds to putting a searcher on a vertex incident to the edge modelling the tunnel. Hence, minimising this means using as few searchers as possible. Ideally, therefore, we aim at a search strategy that is monotone and at the same time minimises the number of searchers used. This leads to one of the most studied problems with graph searching games, the *monotonicity problem*, the question whether for a particular type of game the minimal number of searchers needed to catch the fugitive is the same as the minimal number of searchers needed for a monotone winning strategy. Monotonicity of a type of games also has close connections to the complexity of deciding whether k searchers can catch a fugitive on a given graph – monotone strategies are usually of length linear in the size of the

graph – and also to decompositions of graphs. As we will see below, for the node searching game considered above this is indeed the case. The first monotonicity proof, for the edge searching variant, was given by LaPaugh [1993] and since then monotonicity has been established for a wide range of graph searching games.

Monotonicity of graph searching games will play an important part of this chapter and we will explore this in detail in Section 1.4.

Applications

The goal of graph searching games is to devise a winning strategy for the searchers that uses as few searchers as possible. The minimal number of searchers needed to guarantee capture of the fugitive on a particular graph thereby yields a complexity measure for the graph, which we call the *search width*. This measure, obviously, depends on the type of game being considered. The search width of a graph measures the connectivity of a graph in some way and it is therefore not surprising that there is a close relationship between width measures defined by graph searching games and other complexity or width measures for graphs studied in the literature, such as the *tree-width* or the *path-width* of a graph. This connection is one of the driving forces behind graph searching games and we will explore it in Section 1.6 below.

Graph searching games have found numerous applications in computer science. One obvious application of graph searching games is to all kinds of search problems and the design of optimal search strategies. In games with an invisible fugitive, searching can also be seen as conquering and an optimal search strategy in this context is a strategy to conquer a country so that at each point of time the number of troops needed to hold the conquered area is minimised.

Furthermore, graph searching games have applications in Robotics and the planning of robot movements, as it is explored, for instance, by Guibas et al. [1996]. Another example of this type is the use of graph searching games to network safety as explored by Franklin et al. [2000] where the fugitive models some information and the searchers model intruders, or infected computers, trying to learn this information. The goal here is not to design an optimal search strategy but to improve the network to increase the search number. Graph searching games have also found applications in the study of sequential computation through a translation from pebbling games. We will give more details in Section 1.3.2.

Other forms of graph searching games are closely related to questions in logic. For instance the *entanglement* of a graph is closely related to ques-

tions about the variable hierarchy in the modal μ-calculus, as explored by Berwanger and Grädel [2004].

See the annotated bibliography of graph searching by Fomin and Thilikos [2008] for further applications and references.

As different applications require different types of games, it is not surprising that graph searching games come in many different forms. We will give an overview of some of the more commonly used variants of games in the Section 1.3.

Organisation. This chapter is organised as follows. In Section 1.2 we first define graph searching games in an abstract setting and we introduce formally the concept of monotonicity. We also explore the connection between graph searching and reachability games and derive a range of general complexity results about graph searching games. In Section 1.3 we present some of the more commonly used variants of graph searching games. The monotonicity problem and some important tools to show monotonicity are discussed in Section 1.4. Formalisations of winning strategies for the fugitive in terms of *obstructions* are discussed in Section 1.5. We will explore the connections between graph searching and graph decompositions in Section 1.6. Finally, in Section 1.7 we study the complexity of deciding the minimal number of searchers required to search a graph in a given game and we close this chapter by stating open problems in Section 1.8. Throughout the chapter we will use some concepts and notation from graph theory which we recall in the appendix.

Acknowledgement. I would like to thank Isolde Adler for carefully proof reading the manuscript.

1.2 Classifying Graph Searching Games

In the previous section we have described one particular version of graph searching, also known as the *Invisible Cops and Robber* games. Possible variants of this game arise from whether or not the fugitive is invisible, from the type of graph the game is played on, i.e. undirected or directed, a graph or a hypergraph, whether the searchers can move freely to any position or whether they can only move along one edge at a time, whether searchers only dominate the vertex they occupy or whether they dominate other vertices as well, whether the fugitive or the searchers can move in every round or only once in a while, and many other differences. The great variations in graph

searching games has made the field somewhat confusing. The fact that the same names are often used for very different games does not help either. In this section we will introduce some of the main variants of the game and attempt a classification of graph searching games.

Most variations do not fundamentally change the nature of the game. The notable exception is between games with a visible fugitive and those where the fugitive is invisible. Essentially, the game with a visible fugitive is a two-player game of perfect information whereas games with an invisible fugitive are more accurately described as one-player games on an (exponentially) enlarged game graph or as two-player games of imperfect information. This difference fundamentally changes the notion of strategies and we therefore introduce the two types of games separately.

1.2.1 Abstract Graph Searching Games

We find it useful to present graph searching games in their most abstract form and then explain how some of the variants studied in the literature can be derived from these abstract games. This will allow us to introduce abstract notions of strategies which then apply to all graph searching games. We will also derive general complexity results for variants of graph searching games. Similar abstract definitions of graph searching games have very recently be given by Amini et al. [2009], Adler [2009] and Lyaudet et al. [2009] for proving very general monotonicity results. Our presentation here only serves the purpose to present the games considered in this paper concisely and in a uniform way and we therefore choose a presentation of abstract graph searching games which is the most convenient for our purpose.

Definition 1.1 An abstract graph searching game is a tuple $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ where

- V is a set
- $\mathcal{S} \subseteq \mathrm{Pow}(V) \times \mathrm{Pow}(V)$ is the *Searcher admissibility relation* and
- $\mathcal{F} : \mathrm{Pow}(V)^3 \to \mathrm{Pow}(V)$ is the *Fugitive admissibility function* and
- $c : \mathrm{Pow}(V) \to \mathbb{N}$ is the *complexity function*.

In the following we will always assume that for every $X \in \mathrm{Pow}(V)$ there is an $X' \in \mathrm{Pow}(V)$ such that $(X, X') \in \mathcal{S}$. This is not essential but will avoid certain notational complications in the definition of strategies below as they otherwise would have to be defined as partial functions.

To give a first example, the invisible Cops and Robber game on a graph

G introduced in the introduction can be rephrased as as abstract graph searching game $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ as follows.

The set V contains the positions the searchers and the fugitive can occupy. In our example, this is the set $V(G)$ of vertices of G.

The Searcher admissibility relation defines the possible moves the searchers can take. As in our example the searchers are free to move from any position to any other position, the Searcher admissibility relation is just $\mathcal{S} := \mathrm{Pow}(V) \times \mathrm{Pow}(V)$.

The fugitive admissibility function models the possible moves of the fugitive: if the searchers currently reside on $X \subseteq V(G)$, the fugitive currently resides somewhere in $R \subseteq V$ and the searchers announce to move to $X' \subseteq V$, then $\mathcal{F}(X, R, X')$ is the set of positions available to the fugitive during the move of the searchers. In the case of the invisible Cops and Robber game described above $\mathcal{F}(X, R, X')$ is defined as

$$\{v \in V : \text{there is } u \in R \text{ and a path in } G \setminus (X \cap X') \text{ from } v \text{ to } u \}$$

the set of positions reachable from a vertex in R by a path that does not run through a searcher remaining on the board, i.e. a searcher in $X \cap X'$.

Finally, the complexity function c is defined as $c(X) := |X|$ – the number of vertices in X. The complexity function tells us how many searchers are needed to occupy a position X of the Searcher. On graph searching games played on graphs this is usually the number of vertices in X. However, on games played on hypergraphs searchers sometimes occupy hyper-edges and then the complexity would be the number of edges needed to cover the set X of vertices.

Based on the definition of abstract graph searching games we can now present the rules for invisible and visible games.

1.2.2 Invisible Abstract Graph Searching Games

Let $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ be an abstract graph searching game. In the variant of graph searching with an invisible fugitive, the searchers occupy vertices in V. The Fugitive, in principle, also occupies a vertex in V but the searchers do not know which one. It is therefore much easier to represent the position of the Fugitive not by the actual position $v \in V$ currently occupied by the fugitive but by the set R of all positions where the fugitive could currently be. This is known as the **fugitive space**, or **robber space**. The goal of the searchers in such a game therefore is to systematically search the set V so that at some point the robber space will be empty.

The rules of the **invisible abstract graph searching game on $\mathcal{G}$** are

defined as follows. The initial position of the play is $(X_0 := \varnothing, R_0 := V)$, i.e. initially there are no searchers on the board and the Fugitive can reside on any position in V.

Let $X_i \subseteq V$ be the current position of the searchers and $R_i \subseteq V$ be the current fugitive space. If $R_i = \varnothing$ then the Searcher has won and the game is over. Otherwise, the Searcher chooses $X_{i+1} \subseteq V$ such that $(X_i, X_{i+1}) \in \mathcal{S}$. Afterwards, $R_{i+1} := \mathcal{F}(X_i, R_i, X_{i+1})$ and the play continues at (X_{i+1}, R_{i+1}). If the fugitive can escape forever, then he wins.

Formally, a play in $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ is a finite or infinite sequence $\mathcal{P} := (X_0, R_0), \ldots$ such that, for all i, $(X_i, X_{i+1}) \in \mathcal{S}$ and $R_{i+1} := \mathcal{F}(X_i, R_i, X_{i+1})$. Furthermore, if $\mathcal{P}$ is infinite then $R_i \neq \varnothing$, for all $i \geq 0$, and if $\mathcal{P} := (X_0, R_0), \ldots, (X_k, R_k)$ is finite then $R_k = \varnothing$ and $R_i \neq \varnothing$ for all $i < k$. Hence, the Searcher wins all finite plays and the Fugitive wins the infinite plays.

Note that as $R_0 := V$ and $R_{i+1} := \mathcal{F}(X_i, R_i, X_{i+1})$, the entire play is determined by the actions of the Searcher and we can therefore represent any play $\mathcal{P} := (X_0, R_0), \ldots$ by the sequence $X_0, X_1, ...$ of searcher positions. Hence, invisible graph searching games are essentially one-player games of perfect information. Alternatively, we could have defined invisible graph searching games as a game between two players where the fugitive also chooses a particular vertex $v_i \in R_i$ at each round but this information is not revealed to the searchers. This would yield a two-player game of partial information. For most applications, however, it is easier to think of these games as one-player games.

We now formally define the concept of strategies and winning strategies. As we are dealing with a one-player game, we will only define strategies for the Searcher.

Definition 1.2 A strategy for the Searcher in an invisible abstract graph searching game $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ is a function $f : \mathrm{Pow}(V) \times \mathrm{Pow}(V) \to \mathrm{Pow}(V)$ such that $(X, f(X, R)) \in \mathcal{S}$ for all $X, R \subseteq V$.

A finite or infinite play $\mathcal{P} := (X_0, R_0), ...$ is consistent with f if $X_{i+1} := f(X_i, R_i)$, for all i.

The function f is a winning strategy if every play $\mathcal{P}$ which is consistent with f is winning for the Searcher.

If in a play the current position is (X, R), i.e. the searchers are on the vertices in X and the Fugitive space is R, then a strategy for the Searcher tells the Searcher what to do next, i.e. to move the searchers to the new position X'.

Note that implicitly we have defined our strategies to be *positional strate-*

gies in the sense that the action taken by a player does only depend on the current position in the play but not on the history. We will see in Section 1.2.6 that this is without loss of generality as graph searching games are special cases of reachability games for which such positional strategies suffice.

Example: The Invisible Cops and Robber Game

We have already seen how the invisible Cops and Robber game on a graph G described in the introduction can be formulated as an abstract invisible Cops and Robber game $(V, \mathcal{S}, \mathcal{F}, c)$ where $V := V(G)$ is the set of positions, $\mathcal{S} := \mathrm{Pow}(V) \times \mathrm{Pow}(V)$ says that the cops can move freely from one position to another and $\mathcal{F}(X, R, X') := \{v \in V :$ there is a path in $G \setminus (X \cap X')$ from some $u \in R$ to $v\ \}$. This game was first described as node searching by Kirousis and Papadimitriou [1986]. Here the searchers try to systematically search the vertices of the graph in a way that the space available to the fugitive shrinks until it becomes empty.

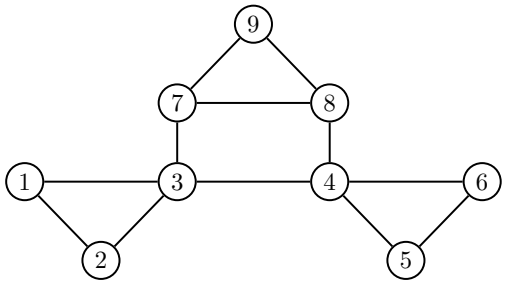


Figure 1.1 Example for an invisible Cops and Robber game

To give an example, consider the graph depicted in Figure 1.1. We will describe a winning strategy for 4 cops in the invisible cops and robber game. The first row contains the cop positions and the second row the corresponding robber space.

X_i :	$\{1,2,3\}$	$\{3,4\}$	$\{3,4,5,6\}$	$\{3,4,7\}$	$\{4,7,8\}$	$\{7,8,9\}$
R_i :	$\{4,5,6,7,8,9\}$	$\{5,6,7,8,9\}$	$\{7,8,9\}$	$\{8,9\}$	$\{9\}$	$\varnothing$

Note that we only have used all four cops once, at position $\{3,4,5,6\}$. It is not too difficult to see that we cannot win with 3 cops. For, consider the edge $3,4$ and assume that cops are placed on it. The graph $G \setminus \{3,4\}$ contains three components, $\{1,2\}$, $\{5,6\}$ and $\{7,8,9\}$. Each of these requires at least 3 cops for clearing but as soon as one of them is cleared the vertices of the

edge $\{3,4\}$ adjacent to a vertex in the component must be guarded until at least a second component of $G \setminus \{3,4\}$ is cleared. For instance, if we first clear the triangle $\{1,2,3\}$ then the vertex 3 needs to be guarded until 4 and 7 are clear but then there are not enough cops left to clear the rest of the graph.

To formally prove that we cannot search the graph with only three cops we will exhibit structural properties of graphs, called *blockages*, which guarantee the existence of a winning strategy for the robber. This leads to the concept of *obstructions* and corresponding duality theorems which we will study in more detail in Section 1.5.

1.2.3 Visible Abstract Graph Searching Games

In this section we describe a variant of graph searching games where the fugitive is visible to the searchers. This fundamentally changes the nature of the game as now the searchers can adapt their strategy to the move of the fugitive. Such graph searching games are now truly two-player games which necessitates some changes to the concepts of strategies.

In particular, it no longer makes sense to represent the position of the fugitive as a fugitive space. Instead we will have to consider individual positions of the fugitive.

Given an abstract game $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$, the rules of the **visible abstract graph searching game on** $\mathcal{G}$ are defined as follows. Initially, the board is empty[1]. In the first round the Searcher first chooses a set $X_0 \subseteq V$ and then the Fugitive chooses a vertex $v_0 \in V$.

Let $X_i \subseteq V$ and $v_i \in V$ be the current positions of the searchers and the fugitive respectively. If $v_i \in X_i$ then the Searcher has won and the game is over. Otherwise, the Searcher chooses $X_{i+1} \subseteq V$ such that $(X_i, X_{i+1}) \in \mathcal{S}$. Afterwards, the fugitive can choose any vertex $v_{i+1} \in \mathcal{F}(X_i, \{v_i\}, X_{i+1})$. If there is none or if $\mathcal{F}(X_i, \{v_i\}, X_{i+1}) \subseteq X_{i+1}$, then again the Searcher wins. Otherwise, the play continues at (X_{i+1}, v_{i+1}). If the fugitive can escape forever, then he wins.

Formally, a *play* in $\mathcal{G}$ is a finite or infinite sequence $\mathcal{P} := (X_0, v_0), \ldots$ such that $(X_i, X_{i+1}) \in \mathcal{S}$ and $v_{i+1} \in \mathcal{F}(X_i, \{v_i\}, X_{i+1})$, for all $i \geq 0$. Furthermore, if $\mathcal{P}$ is infinite then $v_i \notin X_i$, for all $i \geq 0$, and if $\mathcal{P} := (X_0, v_0), \ldots, (X_k, v_k)$ is finite then $v_k \in X_k$ and $v_i \notin X_i$ for all $i < k$. Hence, the Searcher wins all finite plays and the Fugitive wins the infinite plays.

We now define strategies and winning strategies for the Searcher. In con-

[1] There are some variants of games where the robber chooses his position first, but this is not relevant for our presentation.

trast to the invisible case, there is now also a meaningful concept of strategies for the fugitive. However, here we are primarily interested in searcher strategies but we will come back to formalisations of fugitive strategies later in Section 1.5.

Definition 1.3 A **strategy** for the Searcher in a visible abstract graph searching game $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ is a function $f : \mathrm{Pow}(V) \times V \to \mathrm{Pow}(V)$ such that for all $X \subseteq V$ and $v \in V$, $(X, f(X, v)) \in \mathcal{S}$.

A finite or infinite play $\mathcal{P} := (X_0, v_0), ...$ is **consistent with** f if $X_{i+1} := f(X_i, v_i)$, for all i.

f is a **winning strategy** if every play $\mathcal{P}$ which is consistent with f is winning for the Searcher.

If in a play the current position is (X, v), i.e. the searchers are on the vertices in X and the Fugitive is on v, then a strategy for the Searcher tells the Searcher what to do next, i.e. to move the searchers to the new position X'.

Note that implicitly we have defined our strategies to be *positional strategies* in the sense that the action taken by a player does only depend on the current position in the play but not on the history. We will see below that this is without loss of generality as graph searching games are special cases of reachability games for which such positional strategies suffice. Furthermore, the determinacy of reachability games implies that in any visibly graph searching game exactly one of the two players has a winning strategy (see Corollary 1.11).

It is worth pointing out the fundamental difference between strategies for the visible and invisible case. In the invisible case, a strategy for the Searcher uniquely defines a play. Therefore, as we have done above, we can represent a strategy for the Searcher in an invisible graph searching game as a sequence $X_0, X_1, \ldots$ or Searcher positions.

In the visible case, however, the next searcher position may depend on the choice of the fugitive. Therefore, a Searcher strategy f in the visible case can be described by a rooted directed tree T as follows. The nodes $t \in V(T)$ are labelled by $\mathit{cops}(t) \subseteq V$ and correspond to Searcher positions. The individual edges correspond to the possible robber moves. More formally, the root $r \in V(T)$ of T is labelled by $\mathit{cops}(r) := X_0$ the initial cop position. For every $v \in V \setminus X_0$ there is a successor t_v such that $\mathit{cops}(t_v) := f(\mathit{cops}(t), v)$. The edge (t, t_v) is labelled by v. Now, for every $u \in \mathcal{F}(X_0, v, \mathit{cops}(t_v))$ there is a successor t_u of t_v labelled by $\mathit{cops}(t_u) := f(\mathit{cops}(t_v), u)$. Continuing in this way we can build up a strategy tree which is finite if, and only if, f is a winning strategy. More formally, we define a strategy tree as follows.

Definition 1.4 Let $(V, \mathcal{S}, \mathcal{F}, c)$ be an abstract visible graph searching game. An **abstract strategy tree** is a rooted directed tree T whose nodes t are labelled by $\mathit{cops}(t) \subseteq V$ and whose edges e are labelled by $\mathit{search}(e) \in V$ as follows.

1 $\mathit{search}(e) \notin \mathit{cops}(s)$ for all edges $e := (s,t) \in E(T)$.
2 If r is the root of T then for all $v \in V \setminus \mathit{cops}(r)$ there is a successor t_v of r in T and $\mathit{search}(r, t_v) := v$.
3 If t is a node with predecessor s and $v := \mathit{search}((s,t))$ then for each $u \in \mathcal{F}(\mathit{cops}(s), v, \mathit{cops}(t))$ there is a successor t_u of t in T so that $\mathit{search}(t, v_u) := u$.

Often this tree can be further simplified. Suppose for instance that there is an edge $(s,t) \in E(T)$ and that there are vertices $u_1, u_2 \in \mathcal{F}(\mathit{cops}(s), v, \mathit{cops}(t)) \setminus \mathit{cops}(t)$ such that $\mathcal{F}(\mathit{cops}(t), u_1, X) = \mathcal{F}(\mathit{cops}(t), u_2, X)$, for all $X \subseteq V(G)$. In this case the two vertices u_1 and u_2 are equivalent in the sense that it makes no sense for the Searcher to play differently depending on whether the fugitive moves to u_1 or u_2 and likewise for the robber. We therefore do not need to have separate sub-trees corresponding to the two different moves.

Example: The Visible Cops and Robber Game

Let us illustrate the definition of abstract graph searching games. In Seymour and Thomas [1993], a graph searching game called **Cops and Robber Game** is considered, where searchers and the fugitive reside on vertices of a graph $G = (V, E)$. From a position (X, v), where $X \subseteq V$ are the positions of the searchers and $v \in V$ is the current fugitive position, the game proceeds as follows. The searchers can move freely from position $X \subseteq V$ to any other position $X' \subseteq V$. But they have to announce this move publicly and while the searchers move from X to X' the fugitive can choose his new position from all vertices v' such that there is a path in G from v to v' not containing a vertex from $X \cap X'$.

Formulated as an abstract graph searching game, $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ we let $V := V(G)$ and $\mathcal{S} := \mathrm{Pow}(V) \times \mathrm{Pow}(V)$, indicating that there is no restriction on the moves of the searchers. The function $\mathcal{F}$ is then defined as

$$\mathcal{F}(X, \{v\}, X') := \{u \in V : \text{there is a path in } G \setminus (X \cap X') \text{ from } v \text{ to } u\ \}.$$

The complexity function c is defined as $c(X) := |X|$.

To illustrate the game we will show a winning strategy for 3 cops in the visible Cops and Robber game played on the graph G depicted in Figure 1.1. Initially the cops go on the vertices $\{3, 4\}$. Now the robber has a choice to go in one of the three components of $G \setminus \{3, 4\}$. If he chooses a vertex in $\{1, 2\}$

then the next cop move is to play $\{3, 1, 2\}$. As the cop 3 remains on the board the robber cannot escape and is trapped. Analogously, if the robber chooses a vertex in $\{5, 6\}$ then the cops go to $\{4, 5, 6\}$. Finally, suppose the robber chooses a vertex in $\{7, 8, 9\}$. Now the cops have to be a little more careful. If they would lift up a cop on the board, say the cop on vertex 3 to place it on 7, then the robber could escape through a path from his current vertex over the vertex 3 to the component $\{1, 2, 3\}$. So the cop on 3 has to remain on the board and the same for 4. To continue with the winning strategy for the cops we place the third cop on the vertex 7. Now the robber can only move to one of $8, 9$. We can now lift the cop from 3 and place it on 8, as the cops remaining on 7 and 4 block all exists from the component containing the robber. Putting the cop on 7 leaves only vertex 9 for the robber and in the next move he will be caught by moving the cop from 4 to 9.

Recall that in the invisible graph searching game we needed 4 cops to catch the invisible robber whereas here, knowing the robber position, allows us to save one cop. This example also shows that strategies for the cops are trees rather than just simple sequences of cop positions.

1.2.4 Complexity of Strategies

We now define the complexity of a strategy for the Searcher.

Definition 1.5 Let $\mathcal{P} := (X_0, R_0), \ldots,$ where $R_i := \{v_i\}$ in case of visible games, be a finite or infinite play in a graph searching game $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$. The complexity of $\mathcal{P}$ is defined as

$$comp(\mathcal{P}) := \max\{c(X_i) : (X_i, R_i) \in \mathcal{P}\}.$$

The *complexity of a winning strategy* f for the Searcher is

$$comp(f) := \max\{comp(\mathcal{P}) : \mathcal{P} \text{ is an } f\text{-consistent play }\}.$$

As outlined in the introduction, the computational problem associated with a graph searching game is to determine a winning strategy for the Searcher that uses as few searchers as possible, i.e. is of lowest complexity.

Definition 1.6 Let $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ be an abstract graph searching game. The search-width of $\mathcal{G}$ is the minimal complexity of all winning strategies for the Searcher, or ∞ if the Searcher does not have any winning strategies.

A natural computational problem, therefore, is to compute the search-width of a graph searching game. More often we are interested in the corresponding decision problem to decide, given an abstract graph searching

game $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ and $k \in \mathbb{N}$, if there is a winning strategy in $\mathcal{G}$ for the Searcher of complexity at most k. We will usually restrict this problem to certain classes of graph searching games, such as visible Cops and Robber games. In these cases we will simply say "the visible Cops and Robber game has complexity $\mathcal{C}$". Furthermore, often this problem is further restricted to games with a fixed number of Searchers.

Definition 1.7 Let $k \in \mathbb{N}$. The *k-searcher game* on $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ is defined as the graph searching game $\mathcal{G}' := (V, \mathcal{S}', \mathcal{F}, c)$ on the restriction of $\mathcal{G}$ to $\mathcal{S}' := \{(X, X') : (X, X') \in \mathcal{S} \text{ and } c(X), c(X') \leq k\}$.

1.2.5 Monotonicity

In this section we formally define the concept of monotone strategies. Let $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ be an abstract graph searching game.

Definition 1.8 A play $\mathcal{P} := (X_0, R_0), \ldots,$ where $R_i := \{v_i\}$ in case of visible graph searching games, is cop-monotone if for all $v \in V$ and $i \leq l \leq j$: if $v \in X_i$ and $v \in X_j$ then $v \in X_l$.

$\mathcal{P}$ is robber-monotone if $\mathcal{F}(X_i, R_i, X_{i+1}) \supseteq \mathcal{F}(X_{i+1}, R_{i+1}, X_{i+2})$, for all $i \leq 0$.

A strategy is cop- or robber-monotone if any play consistent with the strategy is cop- or robber-monotone.

As outlined above, monotone winning strategies have the advantage of being efficient in the sense that no part of the graph is searched more than once. In most games, this also means that the strategies are short, in the sense that they take at most a linear number of steps.

Lemma 1.9 *Let $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ be an abstract graph searching game with the property that the robber space does not decrease if the cops do not move. Then every play consistent with a cop-monotone winning strategy f will end after at most $|V|$ steps.*

Proof Note that by definition of Searcher strategies the move of the searchers only depends on the current searcher position and the fugitive space or position. Hence, from the assumption that no part of the graph is cleared if the searchers do not move, we can conclude that if at some point the searchers do not move and the fugitive stands still, the play would be infinite and hence losing for the searchers.

Therefore, the cops have to move at every step of the game and as they can never move back to a position they left previously, they can only take a linear number of steps. □

Almost all games considered in this chapter have the property that no player is forced to move and therefore, if the searchers do not move, the fugitive space does not decrease. An exception is the game of entanglement studied by Berwanger and Grädel [2004] where the fugitive has to move at every step and therefore it can be beneficial for the searchers not to move.

A similar lemma as before can often be shown for robber monotone strategies as the robber space is non-increasing. However, this would require the game to be such that there is a bound on the number of steps the cops have to make to ensure that the robber space actually becomes smaller. In almost all games such a bound can easily be found, but formalising this in an abstract setting does not lead to any new insights.

1.2.6 Connection to Reachability Games

In this section we rephrase graph searching games as reachability games and derive some consequences of this. A *reachability game* is a game played on an arena $\mathfrak{G} := (A, V_0, E, v_0)$ where (A, E) is a directed graph, $V_0 \subseteq A$ and $v_0 \in A$. We define $V_1 := A \setminus V_0$. The game is played by two players, Player 0 and Player 1, who push a token along edges of the digraph. Initially the token is on the vertex v_0. In each round of the game, if the token is on a vertex $v_i \in V_0$ then Player 0 can choose a successor v_{i+1} of v_i, i.e. $(v_i, v_{i+1}) \in E$, and push the token along the edge to v_{i+1} where the play continues. If the token is on a vertex in V_1 then Player 1 can choose the successor. The winning condition is given by a set $X \subseteq A$. Player 0 wins if at some point the token is on a vertex in X or if the token is on a vertex in V_1 which has no successors. If the token never reaches a vertex in X or if at some point Player 0 cannot move anymore, then Player 1 wins. See [Grädel et al., 2002, Chapter 2] for details of reachability games.

A positional strategy for Player i in a reachability game can be described as a function $f_i : V_i \to A$ assigning to each vertex v where the player moves a successor $f(v)$ such that $(v, f(v)) \in E$. f_i is a winning strategy if the player wins every play consistent with this strategy. For our purposes we need two results on reachability games, *positional determinacy* and the fact that the winning region for a player in a reachability game can be computed in linear time.

Lemma 1.10 *1 Reachability games are positionally determined, i.e. in every reachability game exactly one of the players has a winning strategy and this can be chosen to be positional.*

2 There is a linear time algorithm which, given a reachability game (A, V_0, E, v_0)

and a winning condition $X \subseteq A$, decides whether Player 0 has a winning strategy in the game.

Let $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ be a visible graph searching game. We associate with $\mathcal{G}$ a game arena $\mathfrak{G} := (A, V_0, E, v_0)$ where

$$\begin{aligned} A := \quad & \mathrm{Pow}(V) \times V \cup \\ & \{(X, v, X') \in \mathrm{Pow}(V) \times \mathrm{Pow}(V) \times V : (X, X') \in \mathcal{S}\}. \end{aligned}$$

Nodes $(X, v) \in \mathrm{Pow}(V) \times V$ correspond to positions in the graph searching games. A node $(X, v, X') \in \mathrm{Pow}(V) \times V \times \mathrm{Pow}(V)$ will correspond to the intermediate position where the searchers have announced that they move from X to X' and the fugitive can choose his new position $v' \in \mathcal{F}(X, \{v\}, X')$. There is an edge from (X, v) to (X, v, X') for all X' such that $(X, X') \in \mathcal{S}$. Furthermore, there is an edge from (X, v, X') to (X', v') for all $v' \in \mathcal{F}(X, \{v\}, X')$.

All nodes of the form (X, v) belong to Player 0 and nodes (X, v, X') belong to Player 1. Finally, the winning condition contains all nodes (X, v) which $v \in X$.

Now, it is easily seen that from any position (X, v) in the graph searching game, the Searcher has a winning strategy if, and only if, Player 0 has a winning strategy in $\mathfrak{G}$ from the node (X, v). Lemma 1.10 implies the following corollary.

Corollary 1.11 *For every fixed k, in every visible graph searching game exactly one of the two players has a winning strategy in the k-searcher game.*

Similarly, for the invisible graph searching game, we define a game arena $\mathfrak{G}$ as follows. The vertices are pairs (X, R) where $X, R \subseteq V$ and there is an edge between (X, R) and (X', R') if $(X, X') \in \mathcal{S}$ and $R' := \mathcal{F}(X, R, X')$. All nodes belong to Player 0. Again it is easily seen that Player 0 has a winning strategy from node (X, R) in $\mathfrak{G}$ if, and only if, the Searcher has a winning strategy in the invisible graph searching game starting from (X, R).

1.3 Variants of Graph Searching Games

In this section we present some of the main variants of games studied in the literature.

1.3.1 A Different Cops and Robber Game

Nowakowski and Winkler [1983] and study a graph searching game, also called *Cops and Robber game*, where the two players take turns and both players are restricted to move along an edge. More formally, starting from a position (X, r), first the Searcher moves and can choose a new position X' obtained from X by moving some searchers to neighbours of their current position. Once the searchers have moved the fugitive can then choose a neighbour of his current position, provided he has not already been caught. See Alspach [2006] for a survey of this type of games.

In our framework of graph searching games, this game, played on a graph G, can be formalised as $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ where

- $V := V(G)$
- A pair (X, X') is in $\mathcal{S}$ if there is a subset $Y \subseteq X$ (these are the searchers that move) and a set Y' which contains for each $v \in Y$ a successor v' of v, i.e. a vertex with $(v, v') \in E(G)$, and $X' \subseteq Y' \cup X \setminus Y$.
- For a triple (X, v, X') we define $\mathcal{F}(X, v, X')$ to be empty if $v \in X'$ and otherwise the set of vertices u s.t. $u \notin X'$ and $(v, u) \in E(G)$.
- Finally, $c(X) := |X|$ for all $X \subseteq V$.

We will refer to this type of games as *turn-based*. Goldstein and Reingold [1995] study turn-based Cops and Robber games on directed graphs and establish a range of complexity results for variations of this game ranging from LOGSPACE-completeness to EXPTIME-completeness. Among other results they show the following theorem.

Theorem 1.12 (Goldstein and Reingold [1995]) *The turn-based Cops and Robber game on a strongly connected digraph is* EXPTIME*-complete.*

The study of this type of games forms a rich and somewhat independent branch of graph searching games. To keep the presentation concise, we will mostly be focusing on games where the two players (essentially) move simultaneously and are not restricted to moves of distance one. See Alspach [2006] and Fomin and Thilikos [2008] and references therein for a guide to the rich literature on turn-based games.

1.3.2 Node and Edge Searching with an Invisible Fugitive

We have already formally described the rules of the (non turn-based) invisible Cops and Robber game in Section 1.2.2. This game has been introduced as *node-searching* by Kirousis and Papadimitriou [1986] who showed that it

is essentially equivalent to pebbling games used to analyse the complexity of sequential computation.

Pebble games are played on an acyclic directed graph. In each step of a play we can remove a pebble from a vertex or place a new pebble on a vertex provided that all its predecessors carry pebbles. The motivation for pebble games comes from the analysis of register allocation for sequential computation, for instance for computing arithmetical expressions. The vertices of a directed acyclic graph corresponds to sub-terms that have to be computed. Hence, to compute the value of a term represented by a node t we first need to compute the value of its immediate sub-terms represented by the predecessors of t. A pebble on a node means that the value of this node is currently contained in a register of the processor. To compute a value of a term in a register the values of its sub-terms must also be contained in registers and this motivates the rule that a pebble can only be placed if its predecessors have been pebbled.

Initially the graph is pebble free and the play stops once all vertices have been pebbled at least once. The minimal number of pebbles needed for a directed graph representing an expression t is the minimal number of registers that have to be used for computing t. Kirousis and Papadimitriou [1986] show that pebble games can be reformulated as graph searching games with an invisible fugitive and therefore register analysis as described above can be done within the framework of graph searching games.

In the same paper they show that *edge searching* and *node searching* are closely related. Recall from the introduction that the edge searching game is a game where the robber resides on edges of the graph. The searchers occupy vertices. In each move, the searchers can clear an edge by sliding along it, i.e. if a searcher occupies an endpoint of an edge then he can move to the other endpoint and thereby clears the edge. As shown by Kirousis and Papadimitriou [1986], if G is a graph and G' is the graph obtained from G by sub-dividing each edge twice, then the minimal number of cops required to catch the fugitive in the node searching game on G, called the **node search number** of G, is one more than the minimal number of searchers required in the edge searching game on G', called the **edge search number** of G'. Conversely, if G is a graph and G' is obtained from G by replacing each edge by three parallel edges, then the edge search number of G' is one more than the node search number of G.

LaPaugh [1993] proved that the edge searching game is monotone thereby giving the first monotonicity proof for a graph searching game. Using the correspondence between edge searching and node searching, Kirousis and Papadimitriou [1986] establish monotonicity for the node searching game.

Bienstock and Seymour [1991] consider a version of invisible graph searching, called *mixed searching*, where the searcher can both slide along an edge or move to other nodes clearing an edge as soon as both endpoints are occupied. They give a simpler monotonicity proof for this type of games which implies the previous two results.

A very general monotonicity proof for games with an invisible robber based on the concept of sub-modularity was given by Fomin and Thilikos [2003]. We will present an even more general result using sub-modularity in Section 1.4 below.

Using a reduction from the Min-Cut Into Equal-Sized Subsets problem, Megiddo et al. [1988] showed that edge searching is NP-complete. Using the correspondence between edge and node searching outlined above, this translates into NP-completeness of the node searching variant, i.e. the invisible Cops and Robber game defined above.

1.3.3 Visible Robber Games

We have already introduced the visible cops and robber game above. This game was studied by Seymour and Thomas [1993] in relation to tree-width, a connection which we will present in more depth in Section 1.6. In this paper they introduce a formalisation of the robber strategies in terms of *screens*, nowadays more commonly referred to as *brambles*, and use this to prove monotonicity of the visible cops and robber game. A monotonicity proof unifying this result and the results obtained for invisible robber games has been given by Mazoit and Nisse [2008]. We will review this proof method in Section 1.4 below.

Arnborg et al. [1987] proved by a reduction from the Minimum Cut Linear Arrangement problem that determining for a given graph the minimal k such that G can be represented as a *partial k-tree* is NP-complete. This number is equal to the tree-width of G and therefore deciding the tree-width of a graph is NP-complete. We will see in Section 1.6 that the minimal number of searchers, called the visible search width of G, required to catch a visible fugitive in the visible Cops and Robber game on a graph G is equal to the tree-width of G plus one. Hence, deciding the visible search width of a graph is NP-complete.

1.3.4 Lazy or Inert Fugitives

In the games studied so far the fugitive was allowed to move at every step of the game. The inert variant of visible and invisible graph searching is

obtained by restricting the fugitive so that he can only move if a searcher is about to land on his position. More formally, the inert graph searching game $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ is defined as an abstract graph searching game where for all $X, X' \subseteq V$ and $v \in V$, $\mathcal{F}(X, v, X') = v$ if $v \notin X'$.

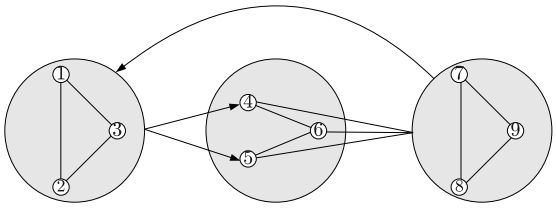


Figure 1.2 A visible directed reachability game

Consider the directed graph depicted in Figure 1.3.5. An undirected edge indicates a directed edge in both directions. The graph consists of two cliques of 3 vertices each which we will call $C_L := \{1, 2, 3\}$ and $C_R := \{7, 8, 9\}$. An edge connecting a clique to a specific vertex means that every vertex of the clique is connected to this vertex. That is, every vertex of C_l has a directed edge to 4 and 5 and every vertex of C_R has a directed edge to every vertex in C_L and also an undirected edge (two directed edges in either direction) to the vertices $4, 5, 6$ in the middle.

On this graph, 5 cops have a winning strategies against the robber as follows. As every vertex in C_R has an edge to every other vertex, the cops must first occupy all vertices in C_R, which takes 3 cops. In addition they put two cops on 4 and 5. Now the robber has a choice to either move to 6 or to a vertex in the clique C_L. If he goes to C_L we lift all cops from C_R and place them on C_L capturing the robber as the only escape route from C_L is through the vertices 4 and 5 which are both blocked.

If on the other hand the robber decides to move to 6 then we lift the two cops from 4 and 5 and place one of them on 6. Now the robber can move to one of 4 or 5 but whatever he does we can then place the space cop on the chosen vertex capturing the robber.

Note that this strategy is non-monotone as the robber can reach the vertices 4 and 5 after they have already been occupied by a cop. Kreutzer and Ordyniak [2008] show that there is no monotone strategy with 5 cops on this graph showing that the directed reachability game is non-monotone.

This example also demonstrates the crucial difference between games played on undirected and directed graphs. For, let G be an undirected graph with some cops being on position X and let R be the robber space, i.e. the component of $G \setminus X$ containing the robber. Now, for every $Y \subseteq V(G)$, if the Cop player places cops on $X \cup Y$ and then removes them from Y again, i.e. moves back to position X, then the robber space is exactly the same space R as before. Intuitively, this is the reason why non-monotone moves

are never necessary in the undirected cops and robber game. For a game played on directed graphs, this is not the case as the example above shows. If $X := \{6, 7\}$ and $Y := C_R$ then once the cops go on $X \cup Y$ and the robber has moved to C_L, the cops can be lifted from C_R without the robber being able to regain control of the lost vertices.

To see that the two variants of directed graph searching games presented above are very different consider the class of *trees with backedges* as studied, e.g., by Berwanger et al. [2006]. The idea is to take a tree and add an edge from every node to any of its (direct or indirect) predecessors up to the root. Then it is easily seen that two searchers suffice to catch a visible fugitive in the SCC game on these trees but to catch the fugitive in the reachability game we need at least as many cops as the height of tree. (It might be a good exercise to prove both statements.) Hence, the difference between the two game variants can be arbitrarily large. On the other hand, we never need more searchers to catch the fugitive in the SCC game than in the reachability game as every move allowed to the fugitive in the latter is also a valid move in the former.

The visible SCC game has been introduced in connection to **directed tree-width** by Johnson et al. [2001]. Barát [2006] studies the invisible reachability game and established its connection to *directed path-width*. Finally, the visible reachability game was explored by Berwanger et al. [2006] and its inert variant by Hunter and Kreutzer [2008]. See also Hunter [2007].

As we have seen above, the visible, invisible and inert invisible graph searching games as well as their edge and mixed search variants are all monotone on undirected graphs. For directed graphs the situation is rather different. Whereas Barát [2006] proved that the invisible reachability game is monotone, all other game variants for directed graphs mentioned here have been shown to be non-monotone. For the SCC game this was shown by Johnson et al. [2001] for the case of searcher monotonicity and by Adler [2007] for fugitive monotonicity. However, Johnson et al. [2001] proved that the visible SCC game is at least *approximately monotone*. We will review the proof of this result in Section 1.4 below.

The visible reachability game as well as the inert reachability game were shown to be non-monotone by Kreutzer and Ordyniak [2008].

1.3.6 Games Played on Hypergraphs

Graph searching games have also found applications on hypergraphs. Gottlob et al. [2003] study a game called the **Robber and Marshal game** on hypergraphs. In the game, the fugitive, here called the robber, occupies

vertices of the hypergraph whereas the searchers, here called marshals, occupy hyper-edges. The game is somewhat different from the games discussed above as a marshal moving from a hyper-edge e to a hyper-edge h still blocks the vertices in $e \cap h$. In particular, one marshal is enough to search an acyclic graph, viewed as hypergraph in the obvious way, whereas we always need at least two cops for any graph containing at least one edge in the visible cops and robber game.

Formally, give a hypergraph $H := (V(H), E(H))$ the Robber and Marshal game on H is defined as $\mathcal{G}_H := (V, \mathcal{S}, \mathcal{F}, c)$ where

- $V := V(H) \dot\cup E(H)$
- $(X, X') \in \mathcal{S}$ if $X, X' \subseteq E(H)$
- $\mathcal{F}(X, R, X') := \varnothing$ if $R \not\subseteq V(H)$ or $R \subseteq \{v \in V(H) : \exists e \in X, v \in e\}$ and otherwise $\mathcal{F}(X, R, X') := \{v \in V(H) :$ there is a path in H from a vertex $u \in R$ to v not going through any vertex in $\bigcup X \cap \bigcup X'$
- $c(X) := |X|$.

Robber and Marshal games have been studied in particular in connection to hypergraph decompositions such as hypertree-width and generalised hypertree-width and approximate duality theorems similar to the one we will establish in Section 1.4.2 and 1.6 have been proved by Adler et al. [2005].

1.3.7 Further Variants

Finally, we briefly comment on further variants of graph searching. Here we concentrate on games played on undirected graphs, but some of the variants translate easily to other types of graphs such as digraphs or hypergraphs.

An additional requirement sometimes imposed on the searchers is that at every step in a play the set of vertices occupied by searchers needs to be **connected**. This is, for instance, desirable if the searchers need to stay within communication range. See e.g. Fomin and Thilikos [2008] for references on connected search.

Another variation is obtained by giving the searchers a greater radius of visibility. For instance, we can consider the case where a searcher not only dominates his own vertex but also all vertices adjacent to it. That is, to catch the robber it is only necessary to trap the robber in the neighbourhood of a searcher. In particular in the invisible fugitive case, such games model the fact that often searchers can see further than just their current position, for instance using torch lights, but they still cannot see the whole system of tunnels they are asked to search. Such games, called **domination games** were introduced by Fomin et al. [2003]. Kreutzer and Ordyniak [2009] study

complexity and monotonicity of these games and show domination games are not only algorithmically much harder compared to classical cops and robber games, they are also highly non-monotone (see Section 1.4.3 below).

Besides graph searching games inspired by applications related to graph searching, there are also games which fall under the category of graph searching games but were inspired by applications in logic. In particular, Berwanger and Grädel [2004] introduce the game of entanglement and its relation to the variable hierarchy of the modal μ-calculus.

1.4 Monotonicity of Graph Searching

As mentioned before, monotonicity features highly in research on graph searching games for a variety of reasons. In this section we present some of the most important techniques that have been employed for proving monotonicity results in the literature.

In Section 1.4.1, we first introduce the concept of *sub-modularity*, which has been used (at least implicitly) in numerous monotonicity results, and demonstrate this technique by establishing monotonicity of the visible cops and robber game discussed above.

Many graph searching games on undirected graphs have been shown to be monotone. Other games, for instance many games on directed graphs, are not monotone and examples showing that searchers can be saved by playing non-monotonic have been given. In some cases, however, at least *approximate monotonicity* can be retained in the sense that there is a function $f : \mathbb{N} \to \mathbb{N}$ such that if k searchers can win by a non-monotone strategy then no more then $f(k)$ searchers are needed to win by a monotone strategy. Often f is just a small constant. Many proofs of approximate monotonicity use the concept of *obstructions*. We will demonstrate this technique in Section 1.4.2 for the case of directed graph searching.

1.4.1 Monotonicity by Sub-Modularity

The aim of this section is to show that the visible cops and robber game on undirected graphs is monotone. The proof presented here essentially follows Mazoit and Nisse [2008]. We will demonstrate the various constructions in this part by the following example.

Recall the representation of winning strategies for the Cop player in terms of strategy trees in Definition 1.4. In this tree, a node t corresponds to a cop position $\mathit{cops}(t)$ and an out-going edge $e := (t, s)$ corresponds to a

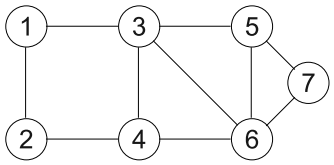


Figure 1.3 Example graph G for monotonicity proofs.

robber move to a vertex $\mathit{search}(e) := v$. Clearly, if the cops are on vertices $X := \mathit{cops}(t)$ and u, v are in the same component of $G \setminus X$, then it makes no difference for the game whether the robber moves to u or v, because whatever the cops do next, the robber can move to exactly the same positions. We therefore do not need to have two separate sub-trees for u and v and can combine vertices in the same component. Thus, we can rephrase strategy trees for the visible cops and robber game as follows. To distinguish from search trees defined below we deviate from the notation of Definition 1.4 and use $\mathit{robber}(e)$ instead of $\mathit{search}(e)$.

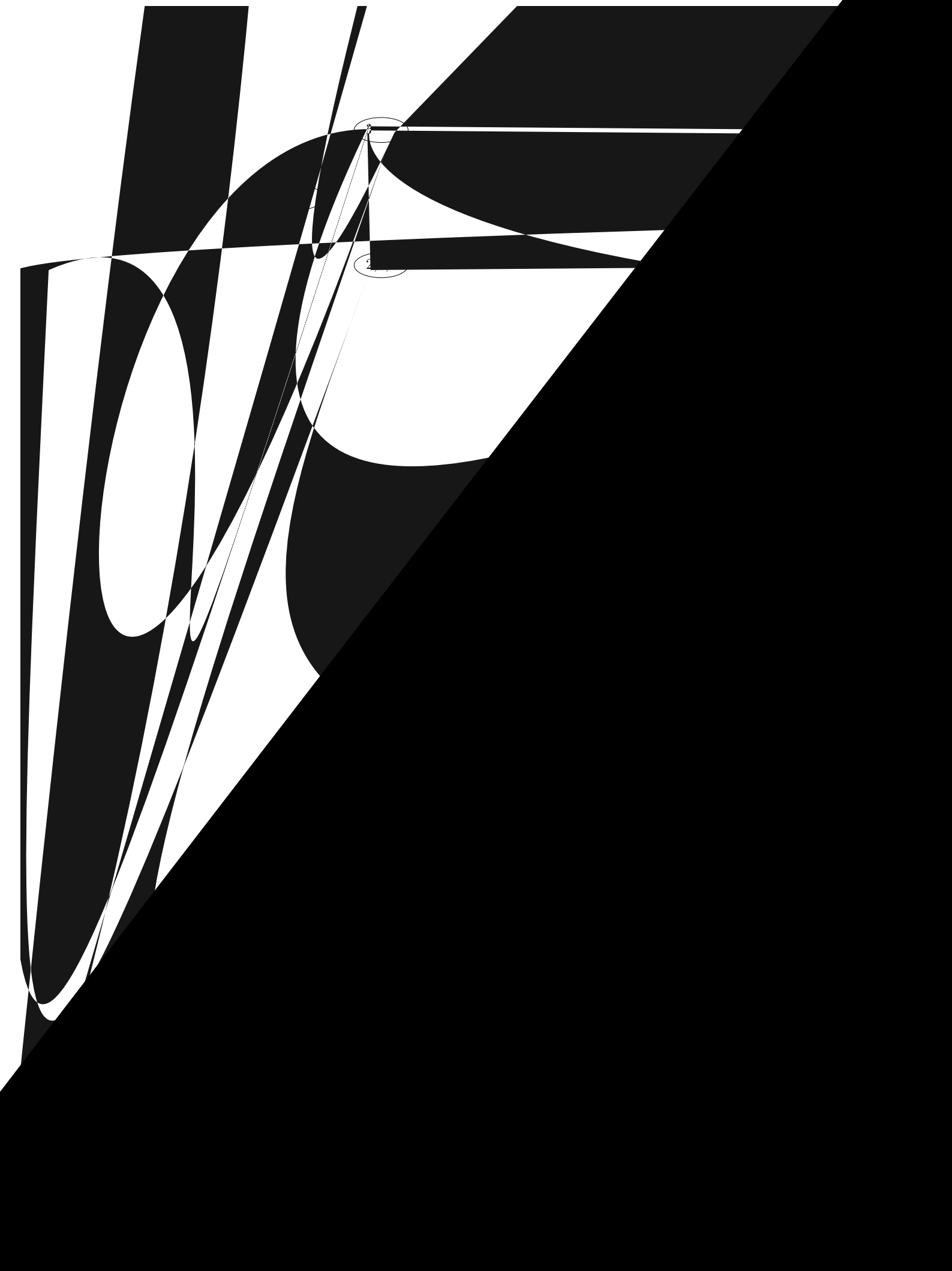
Definition 1.14 Let G be an undirected graph. A strategy tree is a rooted directed tree T whose nodes t are labelled by $\mathit{cops}(t) \subseteq V(G)$ and whose edges $e \in E(T)$ are labelled by $\mathit{robber}(e) \subseteq V(G)$ as follows.

1 If r is the root of T then for all components C of $G \setminus \mathit{cops}(r)$ there is a successor t_C of r in T and $\mathit{robber}(r, t_C) := V(C)$.
2 If t is a node with predecessor s and $C' := \mathit{robber}((s,t))$ then for each component C of $G \setminus \mathit{cops}(t)$ contained in the same component of $G \setminus (\mathit{cops}(s) \cap \mathit{cops}(t))$ as C' there is an edge $e_C := (t, t_C) \in E(T)$ that $\mathit{robber}(e_C) := V(C)$.

A strategy tree is monotone if for all $(s,t), (t,t') \in E(T)$ $\mathit{robber}(s,t) \supseteq \mathit{robber}(t,t')$.

Towards proving monotonicity of the game it turns out to be simpler to think of the cops and the robber as controlling edges of the graph rather than vertices. We therefore further reformulate strategy trees into what we will call *search trees*. Here, a component housing a robber, or a robber space in general, will be represented by the set of edges contained in the component plus the edges joining this component to the cop positions guarding the robber space. We will next define the notion of a border for a set of edges.

Definition 1.15 Let E be a set. We denote the set of partitions $\mathcal{P} := (X_1, \ldots, X_k)$ of E by $\mathcal{P}(E)$, where we do allow degenerated partitions, i.e. $X_i = \varnothing$ for some i.



2 $\mathit{search}(e) \cap \mathit{clear}(t) = \varnothing$ for every edge $e := (s,t) \in E(T)$.

Let $t \in V(T)$ be a node with out-going edges $e_1, \ldots, e_r$. We define

$$\mathit{guard}(t) := V[\mathit{new}(t)] \cup \delta\big(\mathit{search}(e_1), \ldots, \mathit{search}(e_r), \mathit{clear}(t)\big)$$

and the width $w(t)$ of a node t as $w(t) := |\mathit{guard}(t)|$. The width of a search tree is $\max\{w(t) : t \in V(T)\}$.

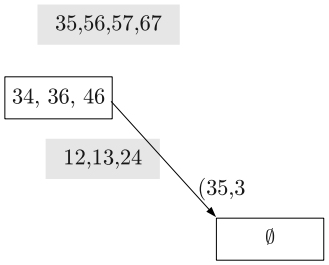
An edge $e := (s,t) \in V(T)$ is called *monotone* if $\mathit{search}(e) \cup \mathit{clear}(t) = E(G)$. Otherwise it is called *non-monotone*. We call $\mathcal{T}$ monotone if all edges are monotone.

It is not too difficult to see that any strategy tree $(T, \mathit{cops}, \mathit{robber})$ corresponds to a search tree $(T, \mathit{new}, \mathit{search}, \mathit{clear})$ over the same underlying directed tree T, where

$$\begin{aligned}
\mathit{new}(t) &:= \{e = \{u,v\} \in E(G) : u, v \in \mathit{cops}(t)\} \\
\mathit{search}(s,t) &:= \{e = \{u,v\} \in E(G) : u \in \mathit{robber}(e) \text{ or } v \in \mathit{robber}(e)\} \\
\mathit{clear}(t) &:= E(G) \setminus \big(\mathit{new}(t) \cup \bigcup_{(t,t') \in E(T)} \mathit{search}(t,t')\big).
\end{aligned}$$

Figure 1.5 shows the search tree corresponding to the strategy tree in Figure 1.4. Here, the node labels correspond to $\mathit{new}(t)$, e.g. the label "34,36,46" of the root corresponds to the edges $(3,4), (3,6)$ and $(4,6)$ cleared by initially putting the cops on the vertices $3, 4, 6$. The edge label in brackets, e.g. (35,36,X) correspond to the *clear* label of their endpoint. Here, X is meant to be the set 56, 57, 67 of edges and is used to simplify presentation. Finally, the edge labels with a grey background denote the *search* label of an edge.

Note that for each node $t \in V(T)$ the cop position $\mathit{cops}(t)$ in the strategy tree is implicitly defined by $\mathit{guard}(t)$ in the search tree. While every strategy tree corresponds to a search tree, not every search tree has a corresponding strategy tree. For instance, if there is an edge $e := (s,t) \in V(T)$ in the search tree such that $\mathit{search}(e) \cap \mathit{clear}(t) \neq E(G)$ then this means that initially the cops are $\mathit{guard}(s)$ with the robber being somewhere in $\mathit{search}(e)$ and from there the cops move to $\mathit{guard}(t)$. By doing so the cops decide to give up some part of what they have already searched and just consider $\mathit{clear}(t)$ to be free of the robber. Everything else is handed over to the robber and will be searched later. However, the corresponding move from $\mathit{guard}(s)$ to $\mathit{guard}(t)$ may not be possible in the cops and robber game in the sense that if the cops were to make this move the robber might have the chance to run to vertices inside $\mathit{clear}(t)$. Intuitively, the move from $\mathit{guard}(s)$ to $\mathit{guard}(t)$ corresponds


35,56,57,67
34, 36, 46
12,13,24
(35,3
∅

Definition 1.19 Let E be a set and $\phi : \mathrm{Pow}(E) \to \mathbb{N}$ be a function.

1 ϕ is **symmetric** if $\phi(X) = \phi(E \setminus X)$ for all $X \subseteq E$.
2 ϕ is **sub-modular** if $\phi(X)+\phi(Y) \geq \phi(X \cap Y)+\phi(X \cup Y)$ for all $X, Y \subseteq E$.

A symmetric and sub-modular function is called a **connectivity function**.

For our proof here we will work with an extension of sub-modularity to partitions of a set E.

Definition 1.20 If $P := \{X_1, \ldots, X_k\} \in \mathcal{P}(E)$ is a partition of a set E and $F \subseteq E$ then we define $P_{X_i \uparrow F}$ as

$$P_{X_i \uparrow F} := \{X_1 \cap F,\ ..., X_{i-1} \cap F,\ X_i \cup F^c,\ X_{i+1} \cap F, \ldots, X_k \cap F\},$$

where $F^c := E \setminus F$.

Definition 1.21 Let E be a set. A **partition function** is a function $\phi : \mathcal{P}(E) \to \mathbb{N}$. ϕ is **sub-modular** if for all $P := \{X_1, \ldots, X_k\} \in \mathcal{P}(E), Q := \{Y_1, \ldots, Y_s\} \in \mathcal{P}(E)$ and all i, j

$$\phi(P) + \phi(Q) \geq \phi(P_{X_i \uparrow Y_j}) + \phi(Q_{Y_j \uparrow X_i}).$$

It is worth pointing out that the definition of sub-modularity of partition functions indeed extends the usual definition of sub-modularity as defined above. For, if $P := \{X, X^c\}$ and $Q := \{Y, Y^c\}$ are bipartitions of a set E then

$$\begin{aligned}
\phi(P) + \phi(Q) &\geq \phi(P_{X \uparrow Y^c}) + \phi(Q_{Y^c \uparrow X}) \\
&= \phi(X \cup (Y^c)^c, X^c \cap Y^c) + \phi(Y \cap X, Y^c \cup X^c) \\
&= \phi(X \cup Y, X^c \cap Y^c) + \phi(Y \cap X, Y^c \cup X^c) \\
&= \phi(X \cup Y, (X \cup Y)^c) + \phi(Y \cap X, (Y \cap X)^c).
\end{aligned}$$

Hence, if we set $\Phi(X) := \phi(X, X^c)$ then this corresponds to the usual notion of sub-modularity of Φ as defined above.

We show next that the border function in Definition 1.16 is sub-modular.

Lemma 1.22 *Let G be a graph and $\phi(P) := |\delta(P)|$ for all partitions $P \in \mathcal{P}(E(G))$. Then ϕ is sub-modular.*

Proof Let $P := \{X_1, \ldots, X_r\}$ and $Q := \{Y_1, \ldots, Y_s\}$ be partitions of $E := E(G)$. Let $1 \leq i \leq r$ and $q \leq j \leq s$. By rearranging the sets P and Q we can assume w.l.o.g. that $i = j = 1$. We want to show that

$$\begin{aligned}
\phi(P) + \phi(Q) &\geq \phi(P_{X_1 \uparrow Y_1}) + \phi(Q_{Y_1 \uparrow X_1}) \\
&= |\delta(X_1 \cup Y_1^c, X_2 \cap Y_1, \ldots, X_r \cap Y_1\}| \ + \\
&\quad |\delta(Y_1 \cup X_1^c, Y_2 \cap X_1, \ldots, Y_S \cap X_1\}|.
\end{aligned}$$

We will prove the inequality by showing that if a vertex $v \in V(G)$ is contained in one of the sets $\delta(P_{X_1\uparrow Y_1}), \delta(Q_{Y_1\uparrow X_1})$ occurring on the right hand side, i.e. is contributing to a term on the right, then the vertex is also contributing to a term on the left. And if this vertex contributes to both terms on the right then it also contributes to both on the left.

Towards this aim, let $v \in V(G)$ be a vertex. Suppose first that v is contained in exactly one of $\delta(P_{X_1\uparrow Y_1})$ or $\delta(Q_{Y_1\uparrow X_1})$, i.e. contributes only to one term on the right hand side. W.l.o.g. we assume $v \in \delta(P_{X_1\uparrow Y_1})$. If there is $1 \leq i < j < r$ such that v is contained in an edge $e_1 \in X_i$ and $e_2 \in X_j$, then $v \in \delta(P)$. Otherwise, v must be incident to an edge $e \in Y_1^c$ and also to an edge $f \in X_j \cap Y_1$, for some $j > 1$. But then $v \in \delta(Q)$ as the edge f occurs in Y_1 and the edge e must be contained in one of the Y_l, $l > 1$.

Now, suppose $v \in \delta(P_{X_1\uparrow Y_1})$ and $v \in \delta(Q_{Y_1\uparrow X_1})$. But then, v is incident to an edge in $e \in X_i \cap Y_1$, for some $i > 1$, and also to an edge $f \in Y_j \cap X_1$, for some $j > 1$. Hence, $f \in X_1$ and $e \in X_i$ and therefore $v \in \delta(P)$ and, analogously, $e \in Y_1$ and $f \in Y_j$ and therefore $v \in \delta(Q)$. Hence, v contributes 2 to the left-hand side. This concludes the proof. □

We will primarily use the sub-modularity of ϕ in the following form.

Lemma 1.23 *Let G be a graph and $P := \{X_1, \ldots, X_k\} \in \mathcal{P}(E(G))$ be a partition of $E(G)$. Let $F \subseteq E(G)$ such that $F \cap X_1 = \varnothing$.*

$$\begin{array}{c} \textit{If } |\delta(F)| \leq |\delta(X_1)| \textit{ then } |\delta(P_{X_1\uparrow F})| \leq |\delta(P)| \\ \textit{If } |\delta(F)| < |\delta(E_1)| \textit{ then } |\delta(P_{X_1\uparrow F})| < |\delta(P)| \end{array}$$

Proof By sub-modularity of $\phi(P) := |\delta(P)|$ we know that

$$|\delta(P)| + |\delta(\{F, F^c\})| \geq |\delta(P_{X_1\uparrow F})| + |\delta(\{F \cup X_1^c, F^c \cap X_1)|.$$

But, as $F \cap X_1 = \varnothing$ we have $F \cup X_1^c = X_1^c$ and $F^c \cap X_1 = X_1$. Hence, we have

$$|\delta(P)| + |\delta(\{F, F^c\})| \geq |\delta(P_{X_1\uparrow F})| + |\delta(\{X_1, X_1^c)|$$

and therefore

$$|\delta(P)| \geq |\delta(P_{X_1\uparrow F})| + \big(|\delta(\{X_1, X_1^c)| - |\delta(\{F, F^c\})|\big)$$

from which the claim follows. □

Monotonicity of the Visible Cops and Robber Game

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

Theorem 1.24 *The visible cops and robber game on undirected graphs is monotone.*

As discussed above, the theorem follows immediately from the following lemma.

Lemma 1.25 *Let G be a graph and $\mathcal{T}$ be a search tree of G of width k. Then there is a monotone search tree of G of width k.*

Proof Let $m := |E(T)|$. We define the *weight* of a search tree $\mathcal{T} := (T, new, search, clear)$ as

$$weight(\mathcal{T}) := \sum_{t \in V(T)} |w(t)|$$

and its *badness* as

$$bn := \sum_{\substack{e \in E(T) \\ e \text{ non-monotone}}} m^{-dist(e)}$$

where the distance $dist(e)$ of an edge $e := (s, t)$ is defined as the distance of t from the root of T.

Given two search trees $\mathcal{T}_1, \mathcal{T}_2$ we say that $\mathcal{T}_1$ is *tighter* than $\mathcal{T}_2$ if $w(\mathcal{T}_1) < w(\mathcal{T}_2)$ or $w(\mathcal{T}_1) = w(\mathcal{T}_2)$ and $bn(\mathcal{T}_1) < bn(\mathcal{T}_2)$. Clearly, the tighter relation is a well-ordering.

Hence, to prove the lemma, we will show that if $\mathcal{T} := (T, new, search, clear)$ is a non-monotone search tree of G then there is tighter search tree of G of the same width as $\mathcal{T}$.

Towards this aim, let $e := (s, t) \in E(T)$ be a non-monotone edge in $\mathcal{T}$.
Case 1. Assume first that $|\delta(search(e))| \leq |\delta(clear(e))|$ and let $e_1, \ldots, e_r$ be the out-going edges of t. We define a new search tree $\mathcal{T}' := (T, new', search', clear')$ where $new'(v) := new(v), clear'(v) := clear(v)$ for all $v \neq t$ and $search'(f) = search(f)$ for all $f \neq e$ and

$$\begin{aligned} clear'(t) &:= E(G) \setminus search(e) \\ new'(t) &:= new(t) \cap search(e) \\ search'(e_i) &:= search(e_i) \cap search(e) \end{aligned}$$

By construction, $\{clear'(t), new'(t), search'(e_1), \ldots, search'(e_r)\}$ form a partition of $E(G)$. Furthermore, for all $f := (u, v) \in E(T)$ we still have $clear(v) \cap search(f) = \varnothing$ and therefore $\mathcal{T}'$ is a search-tree. We have to show that it is tighter than $\mathcal{T}$. Clearly, the weight of all nodes $v \neq t$ remains

unchanged. Furthermore, we get

$$
\begin{aligned}
|\mathit{guard}(t)| &= |\delta\big(\mathit{clear}(t), \mathit{search}(e_1), \ldots, \mathit{search}(e_r)\big) \cup V[\mathit{new}(t)]| && (1.1)\\
&= |\delta(\mathit{new}(t), \mathit{clear}(t), \mathit{search}(e_1), \ldots, \mathit{search}(e_r)\big) \cup \\
&\qquad \big(V[\mathit{new}(t)] \setminus \delta(\mathit{new}(t))\big)| && (1.2)\\
&= |\delta(\mathit{new}(t), \mathit{clear}(t), \mathit{search}(e_1), \ldots, \mathit{search}(e_r)\big)| \; + \\
&\quad |\big(V[\mathit{new}(t)] \setminus \delta(\mathit{new}(t))\big)| && (1.3)\\
&\geq |\mathit{new}'(t), \delta(\mathit{clear}'(t), \mathit{search}'(e_1), \ldots, \mathit{search}'(e_r)\big)| \; + \\
&\quad |\big(V[\mathit{new}(t)] \cap \mathit{search}(e) \setminus \delta(\mathit{new}(t)) \cap \mathit{search}(e)\big)| && (1.4)\\
&= |\delta(\mathit{clear}'(t), \mathit{search}'(e_1), \ldots, \mathit{search}'(e_r)\big) \cup \\
&\qquad \big(V[\mathit{new}'(t)] \cap \mathit{search}(e)\big)| \\
&= |\mathit{guard}'(t)|
\end{aligned}
$$

The equality between (1.1) and (1.2) follows from the fact that $V[\mathit{new}(t)] \cap \delta(\mathit{new}(t), \mathit{clear}(t), \mathit{search}(e_1), \ldots, \mathit{search}(e_r)\big) = \delta(\mathit{new}(t))$. The equality of (1.2) and (1.3) then follows as the two sets are disjoint by construction. The inequality in (1.4) follows from Lemma 1.23 above.

If $|\delta(\mathit{search}(e))| > |\delta(\mathit{clear}(e))|$ then the inequality in (1.4) is strict and therefore in this case we get $w_{\mathcal{T}'}(t) < w_{\mathcal{T}}(t)$ and therefore $\mathit{weight}(\mathcal{T}') < \mathit{weight}(\mathcal{T})$.

Otherwise, if $|\delta(\mathit{search}(e))| = |\delta(\mathit{clear}(e))|$ then the inequality in (1.4) may not be strict and we therefore only get that $w_{\mathcal{T}'}(t) \leq w_{\mathcal{T}}(t)$ and therefore $\mathit{weight}(\mathcal{T}') \leq \mathit{weight}(\mathcal{T})$. However, in this case the edge e is now monotone, by construction, and the only edges which may now have become non-monotone are $e_1, \ldots, e_r$ whose distance from the root is larger than the distance of e from the root. Therefore, the badness of $\mathcal{T}'$ is less than the badness of $\mathcal{T}$. This concludes the first case where $|\delta(\mathit{search}(e))| \leq |\delta(\mathit{clear}(e))|$.
Case 2. Now assume $|\delta(\mathit{search}(e))| > |\delta(\mathit{clear}(e))|$ and let $e_1, \ldots, e_r$ be the out-going edges of s other than e. We define a new search tree $\mathcal{T}' := (T, \mathit{new}', \mathit{search}')$ where $\mathit{new}'(v) := \mathit{new}(v), \mathit{clear}'(v) := \mathit{clear}(v)$ for all $v \neq t$ and $\mathit{search}'(f) = \mathit{search}(f)$ for all $f \neq e$ and

$$
\begin{aligned}
\mathit{search}'(e) &:= E(G) \setminus \mathit{clear}(t) \\
\mathit{new}'(s) &:= \mathit{new}(s) \cap \mathit{clear}(t) \\
\mathit{search}'(e_i) &:= \mathit{search}(e_i) \cap \mathit{clear}(t) \qquad \text{for all } 1 \leq i \leq r \\
\mathit{clear}'(s) &:= \mathit{clear}(s) \cap
\end{aligned}
$$

We demonstrate the construction in the proof by the search tree in Figure 1.5. Let s be the root of that tree, with $new(s) := \{34, 36, 46\}$ and t be the successor of s with $new(t) := \varnothing$. Let $e := (s, t)$. Thus, $search(e) := \{12, 13, 24\}$ and $clear(t) := \{35, 36, X\}$, where $X := \{56, 57, 67\}$.

Clearly, the edge e is non-monotone as

$$search(e) \cup clear(t) := \{12, 13, 24, 35, 36, 56, 57, 67\} \subsetneq E(G).$$

For instance the edge $34 \notin search(e) \cup clear(t)$.

Now, $\delta(search(e)) := \{3, 4\} \subseteq V(G)$ and $\delta(clear(t)) := \{3, 6\}$ and therefore we are in Case 1 of the proof above. Let e_1 be the edge from t to the node labelled $\{34, 46\}$ and let e_2 be the other out-going edge from t.

We construct the new search tree which is exactly as the old one except that now

$$\begin{aligned} clear'(t) &:= E(G) \setminus search(e) = \{34, 35, 36, 46, 56, 57, 67\} \\ new'(t) &:= new(t) \cap search(e) := \varnothing \\ search'(e_1) &:= search(e_1) \cap search(e) := \{24\} \\ search'(e_2) &:= search(e_2) \cap search(e) := \{12, 13\} \end{aligned}$$

The new search tree is shown in Figure 1.6. Note that $guard'(t)$ is now

$$\begin{aligned} guard'(t) &:= V[new'(t)] \cup \delta(clear'(e)) \cup \delta(search'(e_1)) \cup \delta(search'(e_2)) \\ &= \varnothing \cup \{3, 4\} \cup \{2, 4\} \cup \{2, 3\} \\ &= \{2, 3, 4\}. \end{aligned}$$

That is, in the new search tree the cops start on the vertices $3, 4, 6$ as before but now, if the robber moves into the component $\{1, 2\}$ then they go to $\{2, 3, 4\}$ as might be expected. Continuing in this way we would gradually turn the search tree into a monotone search tree corresponding to a monotone strategy.

Further Applications of Sub-Modularity

Sub-modularity has been used in numerous results establishing monotonicity of graph searching games. A very general application of this technique has been given by Fomin and Thilikos [2003] where it was shown that all invisible graph searching games defined by a sub-modular border function are monotone. The proof presented above has been given by Mazoit and Nisse [2008]. More recently, Amini et al. [2009], Lyaudet et al. [2009] and Adler

```
3
```

is approximately monotone. More formally, if G is a directed graph, then for all k, if k cops can catch the robber on G then $3k + 2$ cops can catch the robber with a monotone strategy.

The proof of this theorem relies on the concept of a *haven*, which is a representation of a winning strategy for the robber. Essentially, the proof idea is to iteratively construct a monotone winning strategy for $3k + 2$ cops, starting from some initial position. If at some point of the construction we can not extend the monotone winning strategy then this will give us enough information for constructing a haven of order k showing that the robber can win against k cops even in the non-monotone game.

Definition 1.28 Let G be a directed graph. A

safely remove all cops from the board except for those on $(Z \cup (V(C') \cap Y)$. But by construction of Z, $|V(C') \cap Y| \leq k$ and therefore there are at most $k+1+k = 2k+1$ cops on the board. Furthermore, $V(C) \subsetneq W$ as $Z \cap W \neq \varnothing$. Hence, the robber space has become strictly smaller. We can therefore continue in this way to define a monotone winning strategy for the cops unless at some point we have found a haven of order k. This concludes the proof of Theorem 1.27 as the existence of a haven of order k means that the robber wins against k cops.

Further Examples

Similar methods as in this example can be employed in a variety of cases. For instance, for the Robber and Marshal Game on hypergraphs presented in Section 1.3.6, Adler [2004] gave examples showing that these games are non-monotone. But again, using a very similar technique as in the previous proof, Adler et al. [2005] showed that if k Marshals have a winning strategy on a hypergraph then $3k+1$ Marshals have a monotone winning strategy.

Open Problems

We close this section by stating an open problem. Consider the visible directed reachability game on a directed graph G as defined in Section 1.3.5. The question whether this game is monotone has been open for a long time. Kreutzer and Ordyniak [2008] have exhibited examples of games where $3k-1$ cops have a wining strategy but at least $4k-2$ cops are needed for a monotone strategy, for all values of k. We have seen the example for the special case of $k = 2$ in Section 1.3.5 above.

Similarly, they give examples for the invisible inert directed reachability game where $6k$ cops have a winning strategy but no fewer than $7k$ cops have a monotone winning strategy, again for all values of k.

However, the problem whether these games are at least approximately monotone has so far been left unanswered.

Open Problem 1 *Are the directed visible reachability and the inert invisible directed reachability game approximately monotone?*

1.4.3 Games which are strongly non-monotone

We close this section by giving an example for a class of games which is not even approximately monotone. Recall the definition of *domination games* given in Section 1.3.7. Domination games are played on undirected graphs.

The searchers and the fugitive occupy vertices but a searcher not only controls the vertex it occupies but also all of its neighbours. Again we can study the visible and the invisible variant of the game.

Kreutzer and Ordyniak [2009] showed that there is a class $\mathcal{C}$ of graphs such that for all $G \in \mathcal{C}$ 2 searchers have a winning strategy on G but for every $k \in \mathbb{N}$ there is a graph $G_k \in \mathcal{C}$ such that no fewer than k searchers are needed for a monotone winning strategy. A similar result has also been shown for the visible case.

1.5 Obstructions

So far we have mostly studied strategies for the Searcher. However, if we want to show that k searchers have no winning strategy in a graph searching game $\mathcal{G}$, then we have to exhibit a winning strategy for the fugitive. The existence of a winning strategy for the fugitive on a graph searching game played on an undirected graph G gives a certificate that the graph is structurally fairly complex. Ideally, we would like to represent winning strategies for the fugitive in a simple way so that these strategies can be characterised by the existence of certain structures in the graph. Such structures have been studied intensively in the area of graph decompositions and have come to be known as *obstructions*.

In this section we will look at two very common types of obstructions, called *havens* and *brambles* which have been defined for numerous games. We will present these structures for the case of the visible cops and robber game played on an undirected graph.

We have already seen havens for the directed SCC game but here we will define them for the undirected case.

Definition 1.29 Let G be a graph. A haven of order k in G is a function $h : [V(G)]^{\leq k} \to \text{Pow}(V)$ such that for all $X \in [V(G)]^{\leq k}$ $f(X)$ is a component of $G - X$ and if $Y \subseteq X$ then $h(Y) \supseteq h(X)$.

It is easily seen that if there is a haven of order k in G then the robber wins against k cops on G.

Lemma 1.30 *If h is a haven bramble of order k in a graph G then the robber wins against k cops on G and conversely if the robber has a winning strategy against k cops then there is a haven of order k in G.*

An alternative way of formalising a robber winning strategy is to define

the strategy as a set of connected sub-graphs. This form of winning strategies is known as **bramble**.

Definition 1.31 Let G be a graph and $B, B' \subseteq V(G)$. B and B' **touch** if $B \cap B' \neq \varnothing$ or there is an edge $\{u, v\} \in E(G)$ with $u \in B$ and $v \in B'$.

A **bramble** in a graph G is set set $\mathcal{B} := \{B_1, \ldots, B_l\}$ of sets $B_i \subseteq V(G)$ such that

1 each B_i induces a connected sub-graph $G[B_i]$ and
2 for all i, j, B_i and B_j touch.

The **order** of $\mathcal{B}$ is

$$\min\{|X| : X \subseteq V(G) \text{ s.t.} X \cap B \neq \varnothing \text{ for all } B \in \mathcal{B}\}.$$

The **bramble width** $\mathrm{bw}(G)$ of G is the maximal order of a bramble of G.

We illustrate the definition by giving a bramble of order 3 for the graph depicted in Figure 1.1. In this graph, the set

$$\mathcal{B} := \big\{\{1,2,3\}, \quad \{7,8,9\}, \quad \{4,5,6\}\big\}$$

forms a bramble of order 3.

It is easily seen that the existence of a bramble of order k yields a winning strategy for the robber against k cops.

To give a more interesting example, consider the class of *grids*. A grid is a graph as indicated in Figure 1.7 depicting a 4×5-grid.

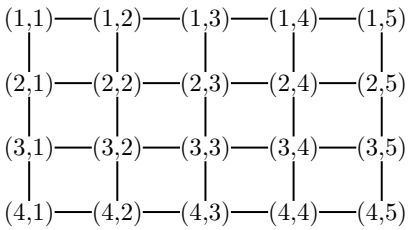


Figure 1.7 4×5-grid

More generally, a $n \times m$-grid is a graph with vertex set $\{(i, j) : 1 \leq i \leq n, 1 \leq j \leq m\}$ and edge set

$$\{\big(i, j), (i', j')\big) : |i - i'| + |j - j'| = 1\}.$$

If G is an $n \times m$-grid then its i-th row is defined as the vertices $\{(i, j) : 1 \leq j \leq m\}$ and its j-th column as $\{(i, j) : 1 \leq i \leq n\}$. A *cross* in a grid is the union of one row and one column. For any $n \times n$-grid we can define a bramble

$\mathcal{B}_n$ consisting of all crosses $\{(s,j) : 1 \leq j < n\} \cup \{(i,t) : 1 \leq i < n\}$, where $1 \leq s,t < n$, of the sub-grid induced by the vertices $\{(i,j) : 1 \leq i,j \leq n-1\}$ together with the sets $B := \{(n,j) : 1 \leq j \leq n\}$ and $R := \{(i,n) : 1 \leq i < n\}$ containing the bottom-most row and the right-most column except the last vertex of that column.

It is readily verified that this is a bramble. Clearly any pair of crosses shares a vertex and therefore touches. On the other hand, every cross touches the bottom row B and also the rightmost column R. Finally, B and L touch also.

The order of $\mathcal{B}_n$ is $n+1$. For, to cover every element of $\mathcal{B}_n$ we need two vertices to cover B and R, are they are disjoint and also disjoint from the other elements in $\mathcal{B}_n$. But to cover the crosses we need at least $n-1$ vertices as otherwise there would be a row and a column in the sub-grid of G_n without the bottom-row and right-most column from which no vertex would have been chosen. But then the corresponding cross would not be covered.

Grids therefore provide examples of graphs with very high bramble width. We will show now that this also implies that the number of cops needed to search the graph is very high. The following is the easy direction of the theorem below.

Lemma 1.32 *If $\mathcal{B}$ is a bramble of order $k+1$ in a graph G then the robber wins against k cops on G.*

Proof We describe a winning strategy for the robber against k cops. Let X be the initial position of the cops. As the order of $\mathcal{B}$ is $k+1$, there is at least one set $B \in \mathcal{B}$ not containing any cops and the robber can choose any vertex from this set. Now, suppose that after some steps, the cops are on X and the robber on a vertex in a set $B \in \mathcal{B}$ not containing any cop. Now suppose the cops go from X to X'. If X' does not contain a vertex from B then the robber does not move. Otherwise, there is a $B' \in \mathcal{B}$ not containing any vertex from X' and while the cops move from X to X', the robber can go from his current position in B to a new position in B' as B and B' are connected and touch. This defines a winning strategy for the robber. □

The converse of the previous result is also true but much more complicated to show.

Theorem 1.33 (Seymour and Thomas [1993]) *Let G be a graph and $k \geq 0$ be an integer. G contains a bramble of order $\geq k$ if, and only if, no fewer than k cops have a winning strategy in the visible Cops and Robber game on G if, and only if, no fewer than k cops have a monotone winning strategy in the visible Cops and Robber game on G.*

We refrain from giving the proof here and refer to Seymour and Thomas [1993] (where brambles were called screens) or the excellent survey by Reed [1997].

The previous result was stated in terms of tree-width rather than winning strategies for the cops and is often referred to as **tree-width duality theorem**. A very general way of establishing duality theorems of this form was studied by Amini et al. [2009], Adler [2009] and Lyaudet et al. [2009] and by Fomin and Thilikos [2003] for the case of an invisible robber.

1.6 An Application to Graph-Decompositions

As outlined in the introduction, graph searching games have found various applications in a number of areas in computer science. Among those, their application in structural graph theory has been a particularly driving force behind developments in graph searching. We demonstrate this by deriving a close connection between undirected cops and robber games and a graph structural concept called *tree-width*.

The concept of *tree-width* was developed by Robertson and Seymour [1982 –] as part of their celebrated graph minor project, even though concepts such as *partial k-trees*, which subsequently have been shown to be equivalent to tree-width, were known before.

Definition 1.34 Let G be a graph. A **tree-decomposition** of G is a pair $\mathcal{T} := (T, (B_t)_{t \in V(T)})$ where T is a tree and $B_t \subseteq V(G)$ for all $t \in V(T)$ such that

1 for all $v \in V(G)$ the set $\{t : v \in B_t\}$ induces a non-empty sub-tree of T and
2 for every edge $e := \{u, v\} \in E(G)$ there is a $t \in V(T)$ such that $\{u, v\} \subseteq B_t$.

The **width** $w(\mathcal{T})$ of $\mathcal{T}$ is

$$w(\mathcal{T}) := \max\{|B_t| : t \in V(T)\} - 1.$$

The **tree-width** of G is the minimal width of a tree-decomposition of G.

We will frequently use the following notation: if $S \subseteq T$ is a sub-tree of T then $B(S) := \{v : v \in B_l \text{ for some } l \in V(S)\}$.

From a graph structural point of view, the tree-width of a graph measures the similarity of a graph to being a tree. However, the concept also has immense algorithmic applications as from an algorithmic point of view a tree-

decomposition yields a recursive decomposition of a graph into small subgraphs and this allows to use the same dynamic programming approaches to solve problems on graphs of small tree-width that can be employed on trees. Determining the tree-width of a graph is NP-complete as shown by Arnborg et al. [1987], but there is an algorithm, due to Bodlaender [1996], which, given a graph G computes an optimal tree-decomposition in time $\mathcal{O}(2^{p(\mathrm{tw}(G))} \cdot |G|)$, for some polynomial p. Combining this with dynamic programming yields a powerful tool to solve NP-hard problems on graph classes of small tree-width. See Bodlaender [1997, 1998, 2005] for surveys including a wide range of algorithmic examples.

To help gaining some intuition about tree-decompositions we establish some simple properties and a normal form for tree-decompositions. We first agree on the following notation. From now on we will consider the tree T of a tree-decompositions to be a rooted tree, where the root can be chosen arbitrarily. If T is a rooted tree and $t \in V(T)$ then T_t is the sub-tree *rooted at* t, i.e. the sub-tree containing all vertices s such that t lies on the path from the root of T to s.

Lemma 1.35 *If G has a tree-decomposition of width k then it has a tree-decomposition $(T, (B_t)_{t \in V(T)})$ of width k so that if $\{s,t\} \in E(T)$ then $B_s \not\subseteq B_t$ and $B_t \not\subseteq B_s$.*

Proof Let $\mathcal{T} := (T, (B_t)_{t \in V(T)})$ be a tree-decomposition such that $B_s \subseteq B_t$ for some edge $\{s,t\} \in E(T)$. Then we can remove s from T and make all neighbours of s other than t neighbours of t. Repeating in this way we generate a tree-decomposition of the same width with the desired property. □

Definition 1.36 Let G be a graph. A separation of G is a triple (A, S, B) of non-empty sets such that $A \cup S \cup B = V(G)$ and there is no path in $G \setminus S$ from a vertex in A to a vertex in B.

Lemma 1.37 *Let $\mathcal{T} := (T, (B_t)_{t \in V(T)})$ be a tree-decomposition of a graph G and let $e := \{s,t\} \in E(T)$. Let T_s be the sub-tree of $T - e$ containing s and let T_t be the sub-tree of $T - e$ containing t. Finally, let $S := B_s \cap B_t$. Then $(B(T_t) \setminus S, S, B(T_s) \setminus S)$ is a separation in G.*

Exercise. Prove this lemma.

Definition 1.38 A tree-decomposition $\mathcal{T} := (T, (B_t)_{t \in V(T)})$ of a graph G is in *normal form* if whenever $t \in V(T)$ is a node and C is a component of $G \setminus B_t$ then there is exactly one successor t_C of t in T such that $V(C) = \bigcup_{s \in V(T_t)} B_s$.

Lemma 1.39 *If G has a tree-decomposition of width k then it also has a tree-decomposition of width k in normal form.*

Proof Let $\mathcal{T} := (T, (B_t)_{t \in V(T)})$ be a tree-decomposition of G. Let $t \in V(T)$. By Lemma 1.37, for every component C of $G \setminus B_t$ there is exactly one neighbour s of t such that $V(C) \subseteq B(T_s)$, where T_s is the component of $T - t$ containing s. So suppose that there are two components C, C' such that $C \cup C' \subseteq B(T_s)$ for some neighbour s of t. Let $\mathcal{T}'$ be the tree-decomposition obtained from $\mathcal{T}$ as follows. Take an isomorphic copy of T_s rooted at a vertex s' and add this as an additional neighbour of t. In the next step we replace every $B(l)$ by $B(l) \cap V(C)$ if $l \in V(T_s)$ and by $B(l) \setminus V(C)$ if $l \in V(T_{s'})$. We proceed in this way till we reach a tree-decomposition in normal form of the same width. □

The presence of a tree-decomposition of small width in a graph G is a witness that the graph has a rather simple structure and that its tree-width is small. But how would a certificate for large tree-width look like? If the tree-width of a graph is very large than there should be some structure in it that causes this high tree-width. Such structural reasons for width parameters to be high are usually referred to as *obstructions*. It turns out that using a graph searching game connection of tree-width, such obstructions can easily be identified as we can use the formalisations of winning strategies for the robber given in Section 1.5 above.

We aim next at establishing a game characterisation of tree-width in terms of the visible Cops and Robber game. It is not difficult to see that strategy trees for monotone winning strategies correspond to tree-decompositions.

Lemma 1.40 *Let G be an undirected graph of tree-width at most $k + 1$. Then k cops have a monotone winning strategy on G in the visible cops and robber game. Conversely, if $k + 1$ cops have a monotone winning strategy in the visible cops and robber game on G then the tree-width of G is at most k.*

Proof Assume first that $k + 1$ cops have a monotone winning strategy on G and let $\mathcal{T} := (T, \mathit{cops}, \mathit{robber})$ be a strategy tree witnessing this as defined in Definition 1.14. As $\mathcal{T}$ represents a monotone strategy, we can w.l.o.g. assume that for each node $t \in V(T)$, $\mathit{cops}(t)$ only contains vertices that can be reached by the robber. Formally, if $t \in V(T)$ and $t_1, \ldots, t_r$ are its out-neighbours, then if $v \in \mathit{cops}(t)$ there must exist an edge $\{v, u\} \in E(G)$ with $u \in \mathit{robber}((t, t_i))$ for at least one i. Clearly, it is never necessary to put a cop on a vertex that has no neighbour in the robber space as these cops cannot be reached by the robber. It is a simple exercise to show that under this assumption $(T, (\mathit{cops}(t))_{t \in V(T)})$ is a tree-decomposition of G.

Towards the converse, let $\mathcal{T} := (T, (B_t)_{t \in V(T)})$ be a tree-decomposition of G of width at most k. By Lemma 1.39, we can assume that $\mathcal{T}$ is in normal form. But then it is easily seen that $(T, \mathit{cops}, \mathit{robber})$ with $\mathit{cops}(t) := B_t$ and $\mathit{robber}((t,s)) := B(T_s) \setminus B_t$ is a monotone strategy tree, where T_s is the component of $T - (t,s)$ containing s. □

The previous lemma together with the monotonicity of the visible Cops and Robber game proved in Theorem 1.24 and Theorem 1.33 imply the following corollary. Note that it is the monotonicity of the game that brings the different concepts – winning strategies, tree-decompositions, obstructions – together to form a uniform characterisation of tree-width and search numbers. This is one of the reasons why monotonicity has been studied so intensively especially in structural graph theory.

Corollary 1.41 *For all graphs G:* $\mathrm{tw}(G) = \mathrm{bw}(G) = \mathrm{cw}(G) - 1$*, where* $\mathrm{bw}(G)$ *denotes the bramble width and* $\mathrm{cw}(G)$ *the minimal number of cops required to win the visible cops and robber game.*

A similar characterisation can be given for the invisible Cops and Robber game. A path-decomposition of a graph G is a tree-decomposition $(T, (B_t)_{t \in V(T)})$ of G where T is a simple path. The path-width of a graph is the minimal width of a path-decomposition of G. Similarly as above we can show that the path-width $\mathrm{pw}(G)$ of a graph is just one less than the minimal number of cops required to catch an invisible robber (with a monotone strategy) on G. The obstructions for path-width corresponding to brambles are called blockages. See Bienstock et al. [1991] for details.

1.7 Complexity of Graph Searching

In this section we study the complexity of computing the least number of searchers required to win a given graph searching game. As usual we will view this as a decision problem asking for a given game and a number k whether k searchers can catch a fugitive or whether they can even do so with a monotone strategy.

We have already stated a number of complexity results in Section 1.3. The aim of this section is to establish much more general results valid for almost all graph searching games within our framework.

Note that all variations of graph searching games described in this chapter – as games played on undirected, directed or hypergraphs, inert variants etc. – can all be described by suitably defining the relation $\mathcal{S}$ and the function $\mathcal{F}$ in a graph searching game $(V, \mathcal{S}, \mathcal{F}, c)$. The only exception is the distinction

between visible and invisible fugitives, which cannot be defined in the description of the game. We can therefore speak about the class $\mathcal{C}$ of the Cops and Robber games played on undirected graphs but have to say explicitly whether we mean the visible or invisible variant.

We will study the complexity questions both within classical complexity as well as parameterised complexity. But before we need to agree on the size of a graph searching game. For this we need to following restriction on games.

Definition 1.42 A class $\mathcal{C}$ of graph searching games is *concise* if

1 there is a polynomial $p(n)$ such that for every $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c) \in \mathcal{C}$ and all $X \in \text{Pow}(V)$, $c(X) \leq p(|X|)$ and
2 given $X, X' \subseteq V$ the relation $\mathcal{S}(X, X')$ can be decided in polynomial time and
3 given $X, X', R \subseteq V$ and $v \in V$ we can decide in polynomial time whether $v \in \mathcal{F}(X, R, X')$.

This condition rules out degenerated cases where, e.g., all Searcher positions have complexity 1. But it also disallows games where deciding whether a move is possible for any of the players is already computationally very complex. All graph searching games studied in this chapter are concise.

Definition 1.43 The size $|\mathcal{G}|$ of a graph searching game $\mathcal{G} := (V, \mathcal{S}, \mathcal{F}, c)$ is defined as $|V|$.

This definition is in line with the intuitive definition of size for, e.g., the visible cops and robber game where the input would only be the graph, and therefore the size would be the order or the size of the graph, whereas the rules of the game are given implicitly.

1.7.1 Classical Complexity Bounds for Graph Searching Games

In this section we present some general complexity bounds for graph searching games in the framework of classical complexity.

Definition 1.44 Let $\mathcal{C}$ be a concise class of graph searching games. The problem Vis-Search Width($\mathcal{C}$) is defined as

Vis Search Width($\mathcal{C}$)	
Input:	$\mathcal{G} \in \mathcal{C}$ and $k \in \mathbb{N}$
Problem:	Is there a winning strategy for k searchers on $\mathcal{G}$ against a visible fugitive?

We define Mon Vis Search Width($\mathcal{C}$) as the problem to decide whether k searchers have a monotone winning strategy on $\mathcal{G}$.

The corresponding problems Invis Search Width($\mathcal{C}$) and Mon Invis Search Width($\mathcal{C}$) for the invisible variant are defined analogously.

To simplify presentation we will refer to this problem simply as "the visible graph searching game on $\mathcal{C}$" and likewise for the invisible variant.

Games with a Visible Fugitive

We first consider the case of arbitrary, non-monotone strategies.

Lemma 1.45 *Let $\mathcal{C}$ be a consider class of graph searching games.*

1 *The visible graph searching game on $\mathcal{C}$ can be solved in exponential time.*
2 *The k-searcher visible graph searching game on $\mathcal{C}$ can be solved in polynomial time.*
3 *There are examples of visible graph searching games which are* Exptime*-complete.*

Proof Given $\mathcal{G} \in \mathcal{C}$ construct the game graph $\mathfrak{G}$ of the corresponding reachability game as defined in Section 1.2.6 above. As $\mathcal{C}$ is concise, this graph is of exponential size and can be constructed in exponential time. We can then use Lemma 1.10 to decide whether or not the Searcher has a winning strategy.

If the complexity is restricted to some fixed k, then the game graph is of polynomial size and therefore the game can be decided in polynomial time.

An example of a visible graph searching game which is complete for Exptime has been given by Goldstein and Reingold [1995], see Theorem 1.12. □

If we are only interested in the existence of monotone strategies, then we can prove slightly better complexity bounds.

Lemma 1.46 *Let $\mathcal{C}$ be a concise class of graph searching games.*

1 *The Searcher-monotone visible graph searching game on $\mathcal{C}$ can be solved in polynomial space.*
2 *The Searcher-monotone k-searcher visible graph searching game on $\mathcal{C}$ can be solved in polynomial time.*

Proof In a Searcher-monotone strategy the Searcher can only make at most polynomially many steps as he is never allowed to return to a vertex once vacated. As $\mathcal{C}$ is concise, this means that a complete play can be kept in polynomial space which immediately implies the result.

If, in addition, the number of searchers is restricted to a fixed k, we can use a straight forward alternating logarithmic space algorithm for it. □

For Fugitive-monotone strategies we can obtain a similar result if the searchers can move freely. In any Fugitive-monotone game the Fugitive-space can only decrease a linear number of times. However, a priori we have no guarantee that in between two positions where the fugitive-space does decrease, the searchers only need to make a linear number of steps. In particular, in game variants where the cops can only move along an edge or where their movement is restricted similar to the entanglement game, there might be variants where they need a large number of steps before the robber space shrinks again.

Games with an Invisible Fugitive

Lemma 1.47 *Let $\mathcal{C}$ be a concise class of graph searching games.*

1 *The invisible graph searching game on $\mathcal{C}$ can be solved in polynomial space.*
2 *The k-searcher invisible graph searching game on $\mathcal{C}$ can be solved in polynomial space.*
3 *There are examples of games which are* Pspace*-hard even in the case where k is fixed.*

Proof Recall that a winning strategy for the Searcher in an invisible graph searching game can be described by the sequence $X_0, \ldots, X_k$ of searcher positions. As $\mathcal{C}$ is concise, any such position only consumes polynomial space. We can therefore guess the individual moves of the searcher reusing space as soon as a move has been made. In this way we only need to store at most 2 Searcher positions and the fugitive space, which can all be done in polynomial space.

Clearly, Part 1 implies Part 2. Kreutzer and Ordyniak [2009] show that the invisible domination game is Pspace-complete even for 2 cops, which shows Part 3. □

Finally, we show that the complexity drops if we only consider monotone strategies in invisible graph searching games.

Lemma 1.48 *Let $\mathcal{C}$ be a concise class of graph searching games.*

1 *The Searcher-monotone invisible graph searching game on $\mathcal{C}$ can be solved in NP.*
2 *The Searcher-monotone k-searcher visible graph searching game on $\mathcal{C}$ can be solved in NP.*

3 There are examples of invisible graph searching games which are NP*-complete.*

Proof In a Searcher-monotone strategy the cop-player can only make at most polynomially many steps as he is never allowed to return to a vertex once vacated. As $\mathcal{C}$ is concise, this means that a complete strategy for the Searcher can be kept in polynomial space and therefore we can simply guess a strategy and then check that it is a winning strategy by playing it. This, clearly, can be done in polynomial time.

Megiddo et al. [1988] show that the invisible graph searching game is NP-complete and as this game is monotone Part 3 follows. □

The following table summarises the general results we can obtain.

	variant	*visible*	*invisible*
k free	non-monotone	EXPTIME	PSPACE
	monotone	PSPACE	NP
k-Searcher	non-monotone	PTIME	PSPACE
	monotone	PTIME	NP

1.7.2 Parameterised Complexity of Graph Searching

Due to their close connection to graph decompositions, graph searching games have been studied intensively with respect to parameterised complexity. We refer to Downey and Fellows [1998] and Flum and Grohe [2006] for an introduction to parameterised complexity.

Definition 1.49 Let $\mathcal{C}$ be a concise class of graph searching games. The problem p-VIS SEARCH WIDTH($\mathcal{C}$) is defined as

p-VIS SEARCH WIDTH($\mathcal{C}$)	
Input:	$\mathcal{G} \in \mathcal{C}$ and $k \in \mathbb{N}$
Parameter:	k
Problem:	Is there a winning strategy for k searchers on $\mathcal{G}$ in the visible fugitive game?

p-VIS SEARCH WIDTH($\mathcal{C}$) is **fixed-parameter tractable** (fpt) if there is a computable function $f : \mathbb{N} \to \mathbb{N}$, a polynomial $p(n)$ and an algorithm deciding the problem in time $f(k) \cdot p(|\mathcal{G}|)$.

The problem is in the complexity class **XP** if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm deciding the problem in time $|\mathcal{G}|^{f(k)}$.

Analogously we define p-MON VIS SEARCH WIDTH($\mathcal{C}$) as the problem to

decide whether k searchers have a monotone winning strategy on $\mathcal{G}$ and the corresponding invisible fugitive variants.

The correspondence between visible graph searching games and reachability games immediately implies the following theorem.

Theorem 1.50 *Let $\mathcal{C}$ be a concise class of abstract graph searching games. Then the visible graph searching game on $\mathcal{C}$ is in XP.*

This, however, fails for the case of invisible graph searching games. For instance, Kreutzer and Ordyniak [2009] show that deciding whether 2 searchers have a winning strategy in the invisible domination game is PSPACE-complete and therefore the problem cannot be in XP unless PSPACE=PTIME.

Much better results can be obtained for the visible and invisible Cops and Robber game on undirected graphs. Bodlaender [1996] presented a linear-time parameterised algorithm for deciding the tree-width and the path-width of a graph. As we have seen above, these parameters correspond to the visible and invisible Cops and Robber game on undirected graphs showing that the corresponding decision problems are fixed-parameter tractable.

The corresponding complexity questions for directed reachability games, on the other hand, are wide open.

1.8 Conclusion

The main objective of this chapter was to provide an introduction to the area of graph searching games and the main techniques used in this context. Graph searching has developed into a huge and very diverse area with many problems still left to be solved. Besides specific open problems such as the approximate monotonicity of directed reachability games in the visible and invisible inert variant, there is the general problem of finding unifying proofs for the various monotonicity and complexity results developed in the literature. Another active trend in graph searching is to extend the framework beyond graphs or hypergraphs to more general or abstract structures such as matroids.

Appendix A Notation

Our notation for graphs follows Diestel [2005] and we refer to this book for more information about graphs. This book also contains an excellent

introduction to structural graph theory and the theory of tree-width or graph decompositions in general.

If V is a set and $k \in \mathbb{N}$ we denote by $[V]^{\leq k}$ the set of all subsets of V of cardinality at most k. We write $\mathrm{Pow}(V)$ for the set of all subsets of V.

All structures and graphs in this section are finite. If G is a graph we denote its vertex set by $V(G)$ and its edge set by $E(G)$. The **size** of a graph is the number of edges in G and its **order** is the number of vertices.

If $e := \{u, v\} \in E(G)$ then we call u and v **adjacent** and u and e **incident**.

H is a **sub-graph** of G, denoted $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. If G is a graph and $X \subseteq V(G)$ we write $G[X]$ for the sub-graph of G **induced** by X, i.e. the graph (X, E') where $E' := \{\{u, v\} \in E(G) : u, v \in X\}$. We write $G \setminus X$ for the graph $G[V(G) \setminus X]$. If $e \in E(G)$ is a single edge we write $G - e$ for the graph obtained from G by deleting the edge e and analogously we write $G - v$, for some $v \in V(G)$, for the graph obtained from G by deleting v and all incident edges.

The **neighbourhood** $N_G(v)$ of a vertex $v \in V(G)$ in an undirected graph G is defined as $N_G(v) := \{u \in V(G) : \{u, v\} \in E(G)\}$.

A graph G is **connected** if G is non-empty and between any two $u, v \in V(G)$ there exists a path in G linking u and v. A **connected component** of a graph G is a maximal connected sub-graph of G.

A directed graph G is **strongly connected** if it is non-empty and between any two $u, v \in V(G)$ there is a directed path from u to v. A **strongly connected component**, or just **component**, of G is a maximal strongly connected sub-graph of G.

A **clique** is an undirected graph G such that $\{u, v\} \in E(G)$ for all $u, v \in V(G)$, $u \neq v$.

A **tree** is a connected acyclic undirected graph. A **directed tree** is a tree T such that there is one vertex $r \in V(T)$, the **root** of T, and every edge of T is oriented away from r.

Finally, a **hypergraph** is a pair $H := (V, E)$ where $E \subseteq \mathrm{Pow}(V)$ is a set of **hyperedges**, where each hyperedge is a set of vertices. We write $V(H)$ and $E(H)$ for the set of vertices and hyperedges of H.

Appendix References

I. Adler. Marshals, monotone marshals, and hypertree-width. *Journal of Graph Theory*, 47(4):275–296, 2004.

I. Adler. Directed tree-width examples. *J. Comb. Theory, Ser. B*, 97(5):718–725, 2007.

I. Adler. Games for width parameters and monotonicity. *available from arXiv.org as*, abs/0906.3857, 2009.

I. Adler, G. Gottlob, and M. Grohe. Hypertree-width and related hypergraph invariants. In *Proceedings of the 3rd European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB'05)*, DMTCS Proceedings Series, pages 5 – 10, 2005.

B. Alspach. Searching and sweeping graphs: a brief survey. *Matematiche (Catania)*, 59:5–37, 2006.

O. Amini, F. Mazoit, N. Nisse, and S. Thomassé. Submodular partition functions. *Discrete Mathematics*, 309(20):6000–6008, 2009.

S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8:277 – 284, 1987.

J. Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2):161–172, 2006.

D. Berwanger and E. Grädel. Entanglement - a measure for the complexity of directed graphs with applications to logic and games. In *LPAR*, pages 209–223, 2004.

D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. Dag-width and parity games. In *Symp. on Theoretical Aspects of Computer Science (STACS)*, 2006.

D. Bienstock and P. Seymour. Monotonicity in graph searching. *Journal of Algorithms*, 12(2):239–245, 1991.

D. Bienstock, N. Robertson, P. D. Seymour, and R. Thomas. Quickly excluding a forest. *J. Comb. Theory, Ser. B*, 52(2):274–283, 1991.

H. Bodlaender. A linear-time algorithm for finding tree-decompositions of small tree-width. *SIAM Journal on Computing*, 25:1305 – 1317, 1996.

H. Bodlaender. A partial k-aboretum of graphs with bounded tree-width. *Theoretical Computer Science*, 209:1 – 45, 1998.

H. L. Bodlaender. Discovering treewidth. In *SOFSEM*, pages 1–16, 2005.

H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proc. of Mathematical Foundations of Computer Science (MFCS)*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36, 1997.

N. Dendris, L. Kirousis, and D. Thilikos. Fugitive search games on graphs and related parameters. *Theoretical Computer Science*, 172(1–2):233 – 254, 1997.

R. Diestel. *Graph Theory*. Springer-Verlag, 3rd edition, 2005.

R. Downey and M. Fellows. *Parameterized Complexity*. Springer, 1998.

J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. ISBN 3-54-029952-1.

F. Fomin and D. Thilikos. On the monotonicity of games generated by symmetric submodular functions. *Discrete Applied Mathematics*, 131(2):323 – 335, 2003.

F. Fomin, D. Kratsch, and H. Müller. On the domination search number. *Discrete Applied Mathematics*, 127(3):565–580, 2003.

F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008.

M. K. Franklin, Z. Galil, and M. Yung. Eavesdropping games: a graph-theoretic approach to privacy in distributed systems. *J. ACM*, 47(2):225–243, 2000.

A. S. Goldstein and E. M. Reingold. The complexity of pursuit on a graph. *Theor. Comput. Sci.*, 143(1):93–112, 1995.

P. Golovach. Equivalence of two formalizations of a search problem on a graph. *Vestnik Leningrad Univ. Math.*, 22(1):13–19, 1989.

G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *Journal of Computer and Systems Science*, 66(4):775–808, 2003.

E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

L. J. Guibas, J.-c. Latombe, S. M. Lavalle, D. Lin, and R. Motwani. A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry and Applications*, 9:471–494, 1996.

P. Hunter. *Complexity and Infinite Games on Finite Graphs*. PhD thesis, Computer Laboratory, University of Cambridge, 2007.

P. Hunter and S. Kreutzer. Digraph measures: Kelly decompositions, games, and ordering. *Theoretical Computer Science (TCS)*, 399(3), 2008.

T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed tree-width. *J. Comb. Theory, Ser. B*, 82(1):138–154, 2001.

L. M. Kirousis and C. H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(3):205–218, 1986.

S. Kreutzer and S. Ordyniak. Digraph decompositions and monotonocity in digraph searching). In *34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, 2008.

S. Kreutzer and S. Ordyniak. Distance-d-domination games. In *34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, 2009.

A. S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40:224–245, 1993.

L. Lyaudet, F. Mazoit, and S. Thomassé. Partitions versus sets : a case of duality. *available at arXiv.org*, abs/0903.2100, 2009.

F. Mazoit and N. Nisse. Monotonicity of non-deterministic graph searching. *Theor. Comput. Sci.*, 399(3):169–178, 2008.

N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *J. ACM*, 35(1):18–44, 1988.

R. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Mathematics*, 43:235–239, 1983.

S. Ordyniak. *Complexity and Monotonicity in Graph Searching*. PhD thesis, Oxford University Computing Laboratory, 2009.

T. Parsons. Pursuit-evasion in a graph. *Theory and applications of graphs, Lecture Notes in Mathematics*, 642:426–441, 1978.

B. Reed. Tree width and tangles: A new connectivity measure and some applications. In R. Bailey, editor, *Surveys in Combinatorics*, pages 87–162. Cambridge University Press, 1997.

D. Richerby and D. M. Thilikos. Searching for a visible, lazy fugitive. In *Workshop on Graph-Theoretical Methods in Computer Science*, pages 348–359, 2008.

N. Robertson and P. Seymour. Graph minors I – XXIII, 1982 –. Appearing in Journal of Combinatorial Theory, Series B since 1982.

P. D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33 - 3 2 3 .

Index