Technical Report

UCAM-CL-TR-704
ISSN 1476-2986

Number 704

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Complexity and infinite games on finite graphs

Paul William Hunter

November 2007

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted July 2007 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Hughes Hall.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Abstract

This dissertation investigates the interplay between complexity, infinite games, and finite graphs. We present a general framework for considering two-player games on finite graphs which may have an infinite number of moves and we consider the computational complexity of important related problems. Such games are becoming increasingly important in the field of theoretical computer science, particularly as a tool for formal verification of non-terminating systems. The framework introduced enables us to simultaneously consider problems on many types of games easily, and this is demonstrated by establishing previously unknown complexity bounds on several types of games.

We also present a general framework which uses infinite games to define notions of structural complexity for directed graphs. Many important graph parameters, from both a graph theoretic and algorithmic perspective, can be defined in this system. By considering natural generalizations of these games to directed graphs, we obtain a novel feature of digraph complexity: directed connectivity. We show that directed connectivity is an algorithmically important measure of complexity by showing that when it is limited, many intractable problems can be efficiently solved. Whether it is structurally an important measure is yet to be seen, however this dissertation makes a preliminary investigation in this direction.

We conclude that infinite games on finite graphs play an important role in the area of complexity in theoretical computer science.

Acknowledgements

A body of work this large can rarely be completed without the assistance and support of many others. Any effort to try and acknowledge all of them would undoubtedly result in one or two being left out, so I am only able to thank those that feature most prominently in my mind at the moment.

The three people I am most indebted to fit nicely into the categories of my past, my present, and my future. Starting with my future (it always pays to look forwards), I am particularly grateful to Stephan Kreutzer. Working with you for a year in Berlin was a fantastic experience and I am looking forward to spending the next few years in the same city again. Thank you for all your support and guidance.

To Sarah, with your constant encouragement (some would say nagging), I am forever indebted. Without your care and support I might never have finished. And I would certainly not be where or who I am today without you.

Finally, the submission of this thesis ends my formal association with the person to whom I, and this dissertation, owe the most gratitude: my supervisor Anuj Dawar. Thank you for giving me the opportunity to work with you and for setting me on the path to my future. You have been an inspiration as a supervisor, I can only hope that when it is my turn to supervise PhD students I can live up to the example you have set me.

Contents

Chapter 1

Introduction

The aim of this dissertation is to investigate the interplay between infinite games, finite graphs, and complexity. In particular, we focus on two facets: the computational complexity of infinite games on finite graphs, and the use of infinite games to define the structural complexity of finite graphs. To present the motivation behind this investigation, we consider the three fundamental concepts of games, graphs and complexity.

What is a game?

Ask anyone what a game is and most people will respond with an example: chess, bridge, cricket, and so on. Almost everyone *understands* what a game is, but few people can immediately give a precise definition. Loosely speaking, a game involves interactions between a number of players (possibly only one) with some possible outcomes, though the outcome is not always the primary concern. The importance of games in many scientific fields arises from their usefulness as an informal description of systems with complex interactions; as most people understand games, a description in terms of a game can often provide a good intuition of the system. The prevalence of this application motivates the formal study of games, which results in the use of games to provide formal definitions. Such definitions can sometimes provide interpretations of concepts where traditional approaches are cumbersome or less than adequate. For example, the semantics of Hintikka's Independence Friendly logic [HS96] are readily expressed using games of imperfect information, but the traditional Tarski-style approaches are unwieldy.

Games in computer science

Mathematical games are playing an increasingly important role in computer science, both as informal descriptions and formal definitions. For example, tree-width, an algorithmically important graph parameter which we see frequently in this dissertation, can be intuitively presented as a game in which a number of cops attempt to capture a robber on a graph. Examples where games can provide formal definitions include interactive protocols and game semantics. An important example of an application of games, which motivates the games we consider, is the following game that arises when verifying if a system satisfies certain requirements.

Starting with the simple case of checking if a formula of propositional logic is satisfied by a truth assignment, consider the following game played by two players, Verifier and Falsifier, "on" the formula. The players recursively choose subformulas with Verifier choosing disjuncts and Falsifier choosing conjuncts until a literal is reached. If the truth value of that literal is

true then Verifier wins, otherwise Falsifier wins. The formula is satisfiable if, and only if, Verifier has a strategy to always win. This game is easily extended to the verification of first order formulas, with Verifier choosing elements bounded by existential quantifiers and Falsifier choosing elements bounded by universal quantifiers. Verifying a first order logic formula is very useful for checking properties of a static system, but often in computer science we are also interested in formally verifying properties of *reactive systems*, systems which interact with the environment and change over time. Requirements for such systems are often specified in richer logics such as Linear Time Logic (LTL), Computation Tree Logic (CTL) or the modal μ-calculus. This motivates the following extension of the Verifier-Falsifier game for verifying if a reactive system satisfies a given set of requirements. The game is played by two players, System and Environment, on the state space of the reactive system. The current state of the system changes, either as a consequence of some move effected by Environment, or some response by System. System takes the role of Verifier, trying to keep the system in a state which satisfies the requirements to be verified. Environment endeavours to demonstrate the system does not satisfy the requirements by trying to move the system into a state which does not satisfy the requirements.

The natural abstraction of these games is a game where two players move a token around a finite directed graph for a possibly infinite number of moves with the winner determined by some pre-defined condition. This abstraction encompasses many two-player, turn-based, zero-sum games of perfect information, and such games are found throughout computer science: in addition to the games associated with formal verification of reactive systems, examples of games which can be specified in this manner include Ehrenfeucht-Fraïssé games and the cops and robber game which characterizes tree-width. Unsurprisingly, these games have been extensively researched, particularly in the area of formal verification: see for example [BL69, Mul63, EJ88, Mos91, EJ91, IK02, DJW97]. Two important questions regarding the complexity of such games are left unresolved in the literature. These are the exact complexity of deciding *Muller games* and the exact complexity of deciding *parity games*. One of the goals of this dissertation is to address these questions with an investigation of the computational complexity of deciding the winner of these types of games.

What is a graph?

Graphs are some of the most important structures in discrete mathematics. Their ubiquity can be attributed to two observations. First, from a theoretical perspective, graphs are mathematically elegant. Even though a graph is a simple structure, consisting only of a set of vertices and a relation between pairs of vertices, graph theory is a rich and varied subject. This is partly due the fact that, in addition to being relational structures, graphs can also be seen as topological spaces, combinatorial objects, and many other mathematical structures. This leads to the second observation regarding the importance of graphs: many concepts can be abstractly represented by graphs, making them very useful from a practical viewpoint. From an algorithmic point of view, many problems can be abstracted to problems on graphs, making the study of graph algorithms a particularly fruitful line of research.

In computer science, many structures are more readily represented by *directed graphs*, for example: transition systems, communications networks, or the formal verification game we saw above. This means that the study of directed graphs and algorithms for directed graphs is particularly important to computer science. However, the increased descriptive power of directed

graphs comes at a cost: the loss of symmetry makes the mathematical theory more intricate. In this dissertation we explore both the algorithmic and mathematical aspects of directed graphs.

What is complexity?

Just as the definition of a game is difficult to pin down, the quality of "being complex" is best described by examples and synonyms. From an algorithmic perspective, a problem is more complex than another problem if the latter is easier to compute than the former. From a structural point of view, one structure is more complex than another if the first structure contains more intricacies. These are the two kinds of complexity relevant to this dissertation: *computational complexity* and *structural complexity*.

In the theory of algorithms, the notion of computational complexity is well defined. In model theory however, being structurally complex is very much a subjective notion, depending largely on the application one has in mind. For example, a graph with a large number of edges could be considered more complex than a graph with fewer edges. On the other hand, a graph with a small automorphism group could be considered more complex than a graph with a large automorphism group, as the second graph (which may well have more edges) contains a lot of repetition. As we are primarily interested in algorithmic applications in this dissertation, we focus on the structural aspects of graphs which influence the difficulty of solving problems. In Section 1.1.2 below, we loosely define this notion of graph structure by describing the fundamental concepts important in such a theory.

Having established what constitutes "structure", we turn to the problem of defining structural complexity. The most natural way is to define some sort of measure which gives an intuition for how "complex" a structure is. In Chapter 4, we discuss those properties that a good measure of structural complexity should have. But how do we find such measures in the first place? Also in Chapter 4 we present the notion of tree-width and argue that it is a good measure of complexity for undirected graphs. As we remarked above, tree-width has a characterization in terms of a two-player game, so it seems that investigating similar games would yield useful measures for structural complexity. Indeed this has been an active area of research for the past few years, for example: [KP86, LaP93, ST93, DKT97, JRST01, FT03, FFN05, BDHK06, HK07]. This line of research has recently started to trend away from showing game-theoretic characterizations of established structural complexity measures to defining important parameters from the definition of the game, an example of the transition from the use of games as an informal description to their use as a formal definition. Despite this activity, very little research has considered games on directed graphs. This is perhaps partly due to the lack, for some time, of a reasonable measure of structural complexity for directed graphs.
The second major goal of this dissertation is to use infinite games to define a notion of structural complexity for directed graphs which is algorithmically useful.

Organization of the thesis

In the remainder of this chapter we define the conventions we use throughout. Chapters 2 and 3 are primarily concerned with the analysis of the complexity of deciding the winner of infinite games on finite graphs. From Chapter 4 to Chapter 8 we investigate graph complexity measures defined by infinite games.

In Chapter 2 we formally define the games we are interested in. We introduce the notion

of a *winning condition type* and we establish a framework in which the expressiveness and succinctness of different types of winning conditions can be compared. We show that the problem of deciding the winner in Muller games is PSPACE-complete, and use this to show the non-emptiness and model-checking problems for Muller tree automata are also PSPACE-complete.

In Chapter 3 we analyse an algorithm for deciding parity games, the strategy improvement algorithm of [VJ00a]. We present the algorithm from a combinatorial perspective, showing how it relates to finding a global minimum on an acyclic unique sink oriented hypercube. We combine this with results from combinatorics to improve the bounds on the running time of the algorithm.

In Chapter 4 we discuss the problem of finding a reasonable notion of complexity for directed graphs. We present the definition of *tree-width*, arguably one of the most practical measures of complexity for undirected graphs, and we discuss the problem of extending the concept to directed graphs.

Building on the games defined in Chapter 2, in Chapter 5 we define the *graph searching game*. We show how we can use graph searching games to define robust measures of complexity for both undirected and directed graphs. This framework is general enough to include many examples from the literature, including tree-width.

In Chapters 6 and 7 we introduce two new measures of complexity for directed graphs: *DAG-width* and *Kelly-width*. Both arise from the work in Chapter 5, and both are generalizations of tree-width to directed graphs. While DAG-width is arguably the more natural generalization of the definition of tree-width, Kelly-width is equivalent to natural generalizations of other graph parameters equivalent to tree-width on undirected graphs, which we also introduce in Chapter 7. We show each measure is useful algorithmically by providing an algorithm for deciding parity games which runs in polynomial time on the class of directed graphs of bounded complexity. We compare both measures with other parameters defined in the literature such as tree-width, directed tree-width and directed path-width and show that these measures are markedly different to those already defined. Finally, in Chapter 7 we compare Kelly-width and DAG-width. We show that the two measures are closely related, but we also show that there are graphs on which the two measures differ.

In Chapter 8 we present some preliminary work towards a graph structure theory for directed graphs based on DAG-width and Kelly-width. We define generalizations of havens and brambles which seem to be appropriate structural features present in graphs of high complexity and absent in graphs of low complexity. We also consider the problem of generalizing the minor relation to directed graphs.

We conclude the dissertation in Chapter 9 by summarizing the results presented. We discuss the contribution made towards the stated research goals, and consider directions of future research arising from this body of work.

Notation and Conventions

We assume the reader is familiar with basic complexity theory, graph theory and discrete mathematics. We generally adopt the following conventions for naming objects.

- For elementary objects, or objects we wish to consider elementary, for example vertices or variables: $a, b, c, \ldots$
- For sets of elementary objects: $A, B, C, \ldots$

- For structures comprising several sets, including graphs and families of sets: $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$
- For more complex structures: $\mathbb{A}, \mathbb{B}, \mathbb{C}, \ldots$
- For sequences and simple functions: $\alpha, \beta, \gamma, \ldots$
- For more complex functions: $\mathfrak{A}, \mathfrak{B}, \mathfrak{C}, \ldots$

1.1.1 Sets and sequences

All sets and sequences we consider in this dissertation are countable. We use both $\mathbb{N}$ and ω to denote the natural numbers, using the latter when we require the linear order. We also assume that 0 is a natural number.

Let A be a set. We denote by $\mathcal{P}(A)$ the set of subsets of A. For a natural number k, $[A]^k$ denotes the set of subsets of A of size k, and $[A]^{\leq k}$ denotes the set of subsets of A of size $\leq k$. Given two sets A and B, $A \dot\cup B$ denotes their *disjoint union* and $A \triangle B$ denotes their *symmetric difference*. That is,

$$A \triangle B := (A \setminus B) \cup (B \setminus A).$$

For readability, we generally drop innermost parentheses or brackets when the intention is clear, particularly with functions. For example if $f : \mathcal{P}(A) \to B$, and $a \in A$, we write $f(a)$ for $f(\{a\})$.

We write sequences as words $a_1 a_2 \cdots$, using 0 as the first index when the first element of the sequence is especially significant. For a sequence π, $|\pi|$ denotes the length of π ($|\pi| = \omega$ if π is infinite). We denote sequence concatenation by $\cdot$. That is, if $\pi = a_1 a_2 \cdots a_n$ is a finite sequence and $\pi' = b_1 b_2 \cdots$ is a (possibly infinite) sequence, then $\pi \cdot \pi'$ is the sequence $a_1 a_2 \cdots a_n b_1 b_2 \cdots$. If $\pi = a_1 a_2 \cdots a_n$ is a finite sequence, π^ω is the infinite sequence $\pi \cdot \pi \cdot \pi \cdots$. Given a set A, the set A^* denotes the set of all finite sequences of elements of A, and the set A^ω denotes the set of all infinite sequences. We say a reflexive and transitive relation $\leq$ on A is a *well-quasi ordering* if for any infinite sequence $a_1 a_2 \cdots \in A^\omega$, there exists indices $i < j$ such that $x_i \leq x_j$.

Let $\pi = a_1 a_2 \cdots$ and $\pi' = b_1 b_2 \cdots$ be sequences of elements of A. We write $\pi \preceq \pi'$ if π is a *prefix* of π', that is, if there exists a sequence π'' such that $\pi' = \pi \cdot \pi''$. We write $\pi \leq \pi'$ if π is a *subsequence* of π', that is, there exists a sequence of natural numbers $n_1 < n_2 < \cdots$ such that $a_i = b_{n_i}$ for all $i \leq |\pi|$.

1.1.2 Graphs

The notation we use for the graph theoretical aspects of this dissertation generally follow Diestel [Die05], however rather than regarding directed graphs as undirected graphs with two maps Head and Tail from edges to vertices, we view directed graphs as relational structures. That is, a *directed graph*, or *digraph*, $\mathcal{G}$ consists of a set of *vertices*, denoted $V(\mathcal{G})$, and an *edge relation*, $E(\mathcal{G}) \subseteq V(\mathcal{G}) \times V(\mathcal{G})$. We use the definition in [Die05] for an *undirected graph*, that is $E(\mathcal{G})$ is a subset of $[V(\mathcal{G})]^2$. For an edge $e = (u, v)$ in a directed graph, the *head* of e is v and the *tail* is u, and we say e goes *from* u *to* v. To avoid ambiguities, we assume that the vertex and edge sets are disjoint. The *elements* of a graph $\mathcal{G}$, is the set defined as

$$\text{Elts}(\mathcal{G}) := V(\mathcal{G}) \cup E(\mathcal{G}).$$

We note that we could either adopt the policy of Diestel and view a directed graph as an undirected graph with some additional structural information, or alternatively we could view an undirected graph as a directed graph where the edge relation is symmetric and irreflexive. We reserve those interpretations for the following two maps between directed and undirected graphs. Let $\mathcal{D}$ be a directed graph. The *underlying undirected graph of* $\mathcal{D}$ is the undirected graph $\overline{\mathcal{D}}$ where:

- $V(\overline{\mathcal{D}}) = V(\mathcal{D})$, and
- $E(\overline{\mathcal{D}}) = \big\{\{u, v\} : (u, v) \in E(\mathcal{D})\}$.

Let $\mathcal{G}$ be an undirected graph. The *bidirected graph of* $\mathcal{G}$ is the directed graph $\overleftrightarrow{\mathcal{G}}$ where:

- $V(\overleftrightarrow{\mathcal{G}}) = V(\mathcal{G})$, and
- $E(\overleftrightarrow{\mathcal{G}}) = \big\{(u, v), (v, u) : \{u, v\} \in E(\mathcal{G})\}$.

We extend the definition of bidirection to parts of undirected graphs. For example a *bidirected cycle* is a subgraph of a directed graph which is a bidirected graph of a cycle. Regarding the pair of edges $\{(u, v), (v, u)\}$ arising from bidirecting an undirected edge, we call such a pair *anti-parallel*. For clarity when illustrating directed graphs, we use undirected edges to represent pairs of anti-parallel edges. For the remaining definitions, we use ordered pairs to describe edges in undirected graphs.

Let $\mathcal{G}$ be an undirected (directed) graph. A *(directed) path* in $\mathcal{G}$ is a sequence of vertices $\pi = v_1 v_2 \cdots$ such that for all i, $1 \leq i < |\pi|$, $(v_i, v_{i+1}) \in E(\mathcal{G})$. For a subset $X \subseteq V(\mathcal{G})$, the set of vertices *reachable* from X is defined as:

$$\mathit{Reach}_{\mathcal{G}}(X) := \{w \in V(\mathcal{G}) : \text{ there is a (directed) path to } w \text{ from some } v \in X\}.$$

For a subset $X \subseteq V(\mathcal{G})$ of the vertices, the subgraph of $\mathcal{G}$ *induced* by X is the undirected (directed) graph $\mathcal{G}[X]$ defined as:

- $V(\mathcal{G}[X]) = X$, and
- $E(\mathcal{G}[X]) = \{(u, v) \in E(\mathcal{G}) : u, v \in X\}$.

For convenience we write $\mathcal{G} \setminus X$ for the induced subgraph $\mathcal{G}[V(\mathcal{G}) \setminus X]$. Similarly, for a set E of edges, $\mathcal{G}[E]$ is the subgraph of $\mathcal{G}$ with vertex set equal to the set of endpoints of E, and edge set equal to E.

Let $v \in V(\mathcal{D})$ be a vertex of a directed graph $\mathcal{D}$. The *successors of* v are the vertices w such that $(v, w) \in E(\mathcal{D})$. The *predecessors of* v are the vertices u such that $(u, v) \in E(\mathcal{D})$. The successors and predecessors of v are the vertices *adjacent to* v. We say v is a *root (of* $\mathcal{D}$*)* if it has no predecessors, and a *leaf (of* $\mathcal{D}$*)* if it has no successors. The *outgoing edges of* v are all the edges from v to some successor of v, and the *incoming edges of* v are all the edges from a predecessor of v to v. The *outdegree of* v, $d_{\mathit{out}}(v)$ is the number of outgoing edges of v and the *indegree of* v, $d_{\mathit{in}}(v)$ is the number of incoming edges of v. Given a subset $V \subseteq V(\mathcal{G})$ of vertices, the *out-neighbourhood of* V, $N_{\mathit{out}}(V)$ is the set of successors of vertices of V not contained in V.

If $\mathcal{D}$ is a directed acyclic graph (DAG), we write $\preceq_{\mathcal{D}}$ for the reflexive, transitive closure of the edge relation. That is $v \preceq_{\mathcal{D}} w$ if, and only if, $w \in \mathit{Reach}_{\mathcal{D}}(v)$. If $v \preceq_{\mathcal{D}} w$, we say v is a *ancestor* of w and w is a *descendant* of v.

We denote by $\mathcal{D}^{op}$ the directed graph obtained by reversing the directions of the edges of $\mathcal{D}$. That is, $\mathcal{D}^{op}$ is the directed graph defined as:

- $V(\mathcal{D}^{op}) = V(\mathcal{D})$, and
- $E(\mathcal{D}^{op}) = (E(\mathcal{D}))^{-1} = \{(v,u) : (u,v) \in E(\mathcal{D})\}$.

In this dissertation we consider *transition systems* with a number of transition relations. That is, a transition system is a tuple $(S, s_I, E_1, E_2 \ldots)$ where S is the set of *states*, $s_I \in S$ is the *initial state*, and $E_i \subseteq S \times S$ are the transition relations. We observe that a transition system with one transition relation is equivalent to a directed graph with an identified vertex.

Structural relations

As we indicated earlier, the notion of *graph structure* is very much a qualitative concept. Just as the "structure" of universal algebra is best characterized by subalgebras, homomorphisms and products, the particular graph structure theory we are interested in is perhaps best characterized by the following "fundamental" relations: subgraphs, connected components and graph composition. As these concepts are frequently referenced, we include their definitions. First we have the subgraph relation.

Definition 1.1 (Subgraph)**.** Let $\mathcal{G}$ and $\mathcal{G}'$ be directed (undirected) graphs. We say $\mathcal{G}$ is a *subgraph* of $\mathcal{G}'$ if $V(\mathcal{G}) \subseteq V(\mathcal{G}')$ and $E(\mathcal{G}) \subseteq E(\mathcal{G}')$.

The next definition describes the building blocks of a graph, the *connected components*.

Definition 1.2 (Connected components)**.** Let $\mathcal{G}$ be an undirected graph. We say $\mathcal{G}$ is *connected* if for all $v, w \in V(\mathcal{G})$, $w \in \mathit{Reach}_{\mathcal{G}}(v)$. A *connected component of* $\mathcal{G}$ is a maximal connected subgraph.

It is easy to see that an undirected graph is the union of its connected components. That is, if $\mathcal{G}_1, \ldots, \mathcal{G}_m$ are the connected components of $\mathcal{G}$, then $V(\mathcal{G}) = \bigcup_{i=1}^{m} V(\mathcal{G}_i)$ and $E(\mathcal{G}) = \bigcup_{i=1}^{m} E(\mathcal{G}_i)$. From the maximality of a connected component, it follows that a connected component is an induced subgraph. Thus we often view a connected component as a set of vertices rather than a graph.

The final fundamental relation is *lexicographic product*, also known as *graph composition*.

Definition 1.3 (Lexicographic product)**.** Let $\mathcal{G}$ and $\mathcal{H}$ be directed (undirected) graphs. The *lexicographic product of* $\mathcal{G}$ *and* $\mathcal{H}$ is the directed (undirected) graph, $\mathcal{G} \bullet \mathcal{H}$, defined as follows:

- $V(\mathcal{G} \bullet \mathcal{H}) = V(\mathcal{G}) \times V(\mathcal{H})$, and
- $\big((v,w),(v',w')\big) \in E(\mathcal{G} \bullet \mathcal{H})$ if, and only if, $(v,v') \in E(\mathcal{G})$ or $v = v'$ and $(w,w') \in E(\mathcal{H})$.

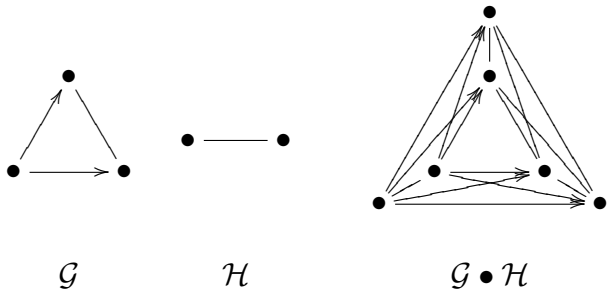


Figure 1.1: The lexicographic product of graphs $\mathcal{G}$ and $\mathcal{H}$

Intuitively, the graph $\mathcal{G} \bullet \mathcal{H}$ arises from replacing vertices in $\mathcal{G}$ with copies of $\mathcal{H}$, hence the name graph composition. Figure 1.1 illustrates an example of the lexicographic product of two graphs.

For directed graphs we have three more basic structural concepts: weakly connected components, strongly connected components and directed union. The first two are a refinement of connected components.

Definition 1.4 (Weakly/Strongly connected components)**.** Let $\mathcal{G}$ be a directed graph. We say $\mathcal{G}$ is *weakly connected* if $\overline{\mathcal{G}}$ is connected. We say $\mathcal{G}$ is *strongly connected* if for all $v, w \in V(\mathcal{G})$, $w \in \textit{Reach}_{\mathcal{G}}(v)$ and $v \in \textit{Reach}_{\mathcal{G}}(w)$. A *weakly (strongly) connected component of* $\mathcal{G}$ is a maximal weakly (strongly) connected subgraph.

We observe that a directed graph is the union of its weakly connected components. The union of the strongly connected components may not include all the edges of the graph. However, it is easy to see that if there is an edge from one strongly connected component to another, then there are no edges in the reverse direction. This leads to the third structural relation specific to directed graphs.

Definition 1.5 (Directed union)**.** Let $\mathcal{G}$, $\mathcal{G}_1$, and $\mathcal{G}_2$ be directed graphs. We say $\mathcal{G}$ is a *directed union of* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ if:

- $V(\mathcal{G}) = V(\mathcal{G}_1) \cup V(\mathcal{G}_2)$, and
- $E(\mathcal{G}) \subseteq E(\mathcal{G}_1) \cup E(\mathcal{G}_2) \cup (V(\mathcal{G}_1) \times V(\mathcal{G}_2))$.

It follows that a directed graph is a directed union of its strongly connected components.

1.1.3 Complexity

The computational complexity definitions of this dissertation follow [GJ79]. We consider polynomial time algorithms efficient, so we are primarily concerned with polynomial time reductions. We use standard *big-O notation* to describe asymptotically bounded classes of functions, particularly for describing complexity bounds.

Collaborations

The work in several chapters of this dissertation arose through collaborative work with others and we conclude this introduction by acknowledging these contributions. The work regarding winning conditions in Chapter 2 was joint work with Anuj Dawar and was presented at the 30th International Symposium on Mathematical Foundations of Computer Science [HD05]. Chapter 6 arose through collaboration with Dietmar Berwanger, Anuj Dawar and Stephan Kreutzer, and was presented at the 23rd International Symposium on Theoretical Aspects of Computer Science [BDHK06]. The concept and name *DAG-width* were also independently developed by Jan Obdržálek [Obd06]. Finally, the work in Chapter 7 arose through collaboration with Stephan Kreutzer and was presented at the 18th ACM-SIAM Symposium on Discrete Algorithms [HK07].

Chapter 2

Infinite games

In this chapter we formally define the games we use throughout this dissertation. The games we are interested in are played on finite or infinite graphs (whose vertices represent a state space) with two players moving a token along the edges of the graph. The (possibly) infinite sequence of vertices that is visited constitutes a play of the game, with the winner of a play being defined by some predetermined condition. As we discussed in the previous chapter, such games are becoming increasingly important in computer science as a means for modelling reactive systems; providing essential tools for the analysis, synthesis and verification of such systems.

It is known [Mar75] that under some fairly general assumptions, these games are determined. That is, for any game one player has a winning strategy. Furthermore, under the conditions we consider below, the games we consider are decidable: whichever player wins can be computed in finite time [BL69]. We are particularly interested in the computational complexity of deciding which player wins in these games. Indeed, this forms one of the underlying research themes of this dissertation.

As we are interested in the algorithmic aspects of these games, we need to restrict our attention to games that can be described in a finite fashion. This does not mean that the graph on which the game is played is necessarily finite as it is possible to finitely describe an infinite graph. Nor does having a finite game graph by itself guarantee that the game can be finitely described. Even with two nodes in a graph, the number of distinct plays can be uncountable and there are more possible winning conditions than one could possibly describe. Throughout this dissertation, we are concerned with *Muller games* played on finite graphs. These are games in which the graph is finite and the winner of a play is determined by the set of vertices of the graph that are visited infinitely often in the play (see Section 2.1 for formal definitions). This category of games is wide enough to include most kinds of game winning conditions that are considered in the literature, including Streett, Rabin, Büchi and parity games.

Since the complexity of a problem is measured as a function of the length of the description, the complexity of deciding which player wins a game depends on how exactly the game is described. In general, a Muller game is defined by a directed graph $\mathcal{A}$, and a winning condition $\mathcal{F} \subseteq \mathcal{P}(V(\mathcal{A}))$ consisting of a set of subsets of $V(\mathcal{A})$. One could specify $\mathcal{F}$ by listing all its elements explicitly (we call this an *explicit* presentation) but one could also adopt a formalism which allows one to specify $\mathcal{F}$ more succinctly. In this chapter we investigate the role the specification of the winning condition has in determining the complexity of deciding regular games. Examples of this line of research can be found throughout the literature, for instance the complexity of deciding Rabin games is known to be NP-complete [EJ88], for Streett games

it is known to be co-NP-complete. The complexity of deciding parity games is a central open question in the theory of regular games. It is known to be in NP∩co-NP [EJ91] and conjectured by some to be in PTIME. In Chapters 3, 6 and 7 we explore this problem in more detail. For Muller games, the exact complexity has not been fully investigated. In Section 2.3 we show that the complexity of deciding Muller games is PSPACE-complete for many types of presentation.

We also establish a framework in which the expressiveness and succinctness of different types of winning conditions can be compared. We introduce a notion of polynomial time *translatability* between formalisms which gives rise to a notion of game complexity stronger than that implied by polynomial time reductions of the corresponding decision problems. Informally, a specification is translatable into another if the representation of a game in the first can be transformed into a representation *of the same game* in the second.

The complexity results we establish for Muller games allow us to show two important problems related to Muller *automata* are also PSPACE-complete: the non-emptiness problem and the model-checking problem on regular trees.

The chapter is organised as follows. In Section 2.1 we present the formal definitions of arenas, games and strategies that we use throughout the remainder of the dissertation. In Section 2.2 we introduce the notion of a *winning condition type*, a formalization for specifying winning conditions. We provide examples from the literature and we consider the notion of translatability between condition types. In Section 2.3 we present some results regarding the complexity of deciding the games we consider here, including the PSPACE-completeness result for Muller games, and a co-NP-completeness result for two games we introduce. Finally, in Section 2.4 we show that the non-emptiness and model checking problems for Muller tree automata are also PSPACE-complete.

2.1 Preliminaries

In this section we present the definitions of arenas, games and strategies that we use throughout the dissertation. The definitions we use follow [GTW02]. In Section 2.1.4 we introduce a generalization of bisimulation appropriate for arenas and games, *game simulation*, and we show how it can be used to translate plays and strategies from one arena to another.

2.1.1 Arenas

Our first definition is a generalization of a transition system where two entities or *players* control the transitions.

Definition 2.1 (Arena)**.** An *arena* is a tuple $\mathcal{A} := (V, V_0, V_1, E, v_I)$ where:

- (V, E) is a directed graph,
- V_0, the set of *Player 0 vertices*, and V_1, the set of *Player 1 vertices*, form a partition of V, and
- $v_I \in V$ is the *initial vertex*.

Viewing arenas as directed graphs with some additional structure, we define the notions of *subarena* and *induced subarena* in the obvious way. Figure 2.1 illustrates an arena $\mathcal{A}$ with $V_0(\mathcal{A}) = \{v_4, v_5, v_6\}$ and $V_1(\mathcal{A}) = \{v_1, v_2, v_3, v_7, v_8, v_9\}$.

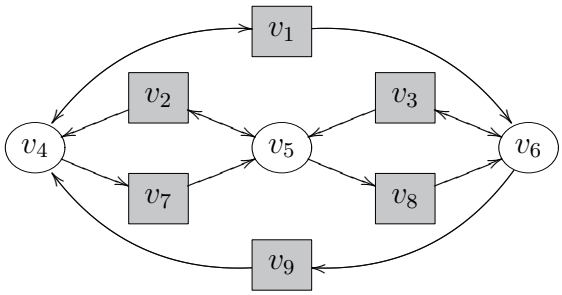


Figure 2.1: An example of an arena

Given an arena, $\mathcal{A}$, we consider the following set of interactions between two players: Player 0 and Player 1.[1] A token, or pebble, is placed on $v_I(\mathcal{A})$. Whenever the pebble is on a vertex $v \in V_0(\mathcal{A})$, Player 0 chooses a successor of v and moves the pebble to that vertex, and similarly when the pebble is on a vertex $v \in V_1(\mathcal{A})$, Player 1 chooses the move. This results in a (possibly infinite) sequence of vertices visited by the pebble. We call such a sequence a *play*. More formally,

Definition 2.2 (Play)**.** Given an arena $\mathcal{A}$ and $v \in V(\mathcal{A})$, a *play in $\mathcal{A}$ (from v)* is a (possibly infinite) sequence of vertices $v_1 v_2 \cdots$ such that $v_1 = v$ and for all $i \geq 1$, $(v_i, v_{i+1}) \in E(\mathcal{A})$. If v is not specified, we assume the play is from $v_I(\mathcal{A})$. The set of all plays in $\mathcal{A}$ from $v_I(\mathcal{A})$ is denoted by Plays$(\mathcal{A})$.

We observe that if $\mathcal{A}'$ is a subarena of $\mathcal{A}$ then Plays$(\mathcal{A}') \subseteq$ Plays$(\mathcal{A})$.

As an example, the infinite sequence $v_1 v_4 v_7 v_5 v_8 v_6 v_9 v_4 v_7 (v_5 v_2)^\omega$ is a play in the arena pictured in Figure 2.1, as is the finite sequence $v_1 v_4 v_7 v_5 v_8 v_6 v_9 v_4$.

When one of the players has no choice of move, we may assume that there is only one player as there is no meaningful interaction between the players.

Definition 2.3 (Single-player arena)**.** Let $\mathcal{A} = (V, V_0, V_1, E, v_I)$ be an arena. We say $\mathcal{A}$ is a *single-player arena* if for some $i \in \{0, 1\}$ and every $v \in V_i$, $d_{out}(v) \leq 1$.

An important concept relating to arenas and the games we consider is the notion of *duality*. In the dual situation, we interchange the roles of Player 0 and Player 1. This gives us the following definition of a *dual arena*.

Definition 2.4 (Dual arena)**.** Let $\mathcal{A} = (V, V_0, V_1, E, v_I)$ be an arena. The *dual arena of $\mathcal{A}$* is the arena defined by $\widetilde{\mathcal{A}} := (V, V_1, V_0, E, v_I)$.

We observe that for each arena $\mathcal{A}$, Plays$(\mathcal{A}) =$ Plays$(\widetilde{\mathcal{A}})$.

2.1.2 Games

Arenas and plays establish the interactions that we are concerned with. We now use these to define games by imposing outcomes for plays. The games we are interested in are zero-sum games, that is, if one player wins then the other player loses. We can therefore define a winning condition as a set of plays that are winning for one player, say Player 0, working on the premise that if a play is not in that set then it is winning for Player 1.

[1] For convenience we use the feminine pronoun for Player 0 and the masculine pronoun for Player 1

Definition 2.5 (Game)**.** A *game* is a pair $\mathbb{G} := (\mathcal{A}, \mathrm{Win})$ where $\mathcal{A}$ is an arena and $\mathrm{Win} \subseteq \mathrm{Plays}(\mathcal{A})$. For $\pi \in \mathrm{Plays}(\mathcal{A})$ if $\pi \in \mathrm{Win}$, we say π is *winning for Player 0*, otherwise π is *winning for Player 1*. A *single-player game* is a game $(\mathcal{A}, \mathrm{Win})$ where $\mathcal{A}$ is a single player arena.

As we mentioned earlier, to consider algorithmic aspects of these games we need to assume that they can be finitely presented. Muller games are an important example of a class of finitely presentable games. With a Muller game, if a player cannot move then he or she loses, otherwise the outcome of an infinite play is dependent on the set of vertices visited infinitely often.

Definition 2.6 (Muller game)**.** A game $\mathbb{G} = (\mathcal{A}, \mathrm{Win})$ is a *Muller game* if $\mathcal{A}$ is finite and there exists $\mathcal{F} \subseteq \mathcal{P}(V(\mathcal{A}))$ such that for all $\pi \in \mathrm{Plays}(\mathcal{A})$:

$$\pi \in \mathrm{Win} \iff \begin{cases} \pi \text{ is finite and ends with a vertex from } V_1(\mathcal{A}), \text{ or} \\ \pi \text{ is infinite and } \{v : v \text{ occurs infinitely often in } \pi\} \in \mathcal{F}. \end{cases}$$

If $\mathbb{G}$ is a Muller game, witnessed by $\mathcal{F} \subseteq \mathcal{P}(V(\mathcal{A}))$, we write $\mathbb{G} = (\mathcal{A}, \mathcal{F})$.

As an example, consider the arena $\mathcal{A}$ pictured in Figure 2.1. Let $\mathcal{F} = \big\{\{v_2, v_5\}\big\}$. Then $\mathbb{G} = (\mathcal{A}, \mathcal{F})$ is a Muller game. The play $v_1v_4v_7v_5v_8v_6v_9v_4v_7(v_5v_2)^\omega$ is winning for Player 0, but the play $v_1v_4v_7(v_5v_8v_6v_9v_4v_7)^\omega$ is winning for Player 1.

The games used in the literature in the study of logics and automata are generally Muller games. In these games, the set $\mathcal{F}$ is often not explicitly given but is specified by means of a *condition*. Different types of condition lead to various different types of games. We explore this in more detail in Section 2.2.

An important subclass of Muller games are the games where only one player wins any infinite play. Games such as Ehrenfeucht-Fraïssé games (on finite structures) [EF99] and the graph searching games we consider in Chapter 5 are examples of these types of games.

Definition 2.7 (Simple game)**.** A Muller game $\mathbb{G} = (\mathcal{A}, \mathcal{F})$ is a *simple game* if either $\mathcal{F} = \emptyset$, or $\mathcal{F} = \mathcal{P}(V(\mathcal{A}))$.

Two other important subclasses of Muller games which we consider in this chapter are union-closed and upward-closed games.

Definition 2.8 (Union-closed and Upward-closed games)**.** A Muller game $\mathbb{G} = (\mathcal{A}, \mathcal{F})$ is *union-closed* if for all $X, Y \in \mathcal{F}$, $X \cup Y \in \mathcal{F}$. $\mathbb{G}$ is *upward-closed* if for all $X \in \mathcal{F}$ and $Y \supseteq X$, $Y \in \mathcal{F}$.

Remark. Union-closed games are often called Streett-Rabin games in the literature, as Player 0's winning set can be specified by a set of Streett pairs (see Definition 2.38 below) and Player 1's winning set can be specified by a set of Rabin pairs (see Definition 2.37). However, to minimize confusion, we reserve the term *Streett game* for union-closed games with a condition presented as a set of Streett pairs, and the term *Rabin game* for the dual of a union-closed game (see below) with a condition presented as a set of Rabin pairs.

We conclude this section by considering dual games and subgames. In Definition 2.4 we defined the dual of an arena. The dual game is played on the dual arena, but we have to complement the winning condition in order to fully interchange the roles of the players. That is,

Definition 2.9 (Dual game)**.** Let $\mathbb{G} = (\mathcal{A}, \mathrm{Win})$ be a game. The game $\widetilde{\mathbb{G}} := (\widetilde{\mathcal{A}}, \overline{\mathrm{Win}})$ where $\widetilde{\mathcal{A}}$ is the dual arena of $\mathcal{A}$ and $\overline{\mathrm{Win}} = \mathrm{Plays}(\mathcal{A}) \setminus \mathrm{Win}$ is the *dual game of* $\mathbb{G}$.

Given a game on an arena $\mathcal{A}$ we can define a restricted game on a subarena $\mathcal{A}'$ by restricting the winning condition to valid plays in the subarena.

Definition 2.10 (Subgame)**.** Let $\mathbb{G} = (\mathcal{A}, \mathrm{Win})$ be a game, and $\mathcal{A}'$ a subarena of $\mathcal{A}$. The *subgame induced by* $\mathcal{A}'$ is the game $\mathbb{G}' = (\mathcal{A}', \mathrm{Win}')$ where $\mathrm{Win}' = \mathrm{Win} \cap \mathrm{Plays}(\mathcal{A}')$.

2.1.3 Strategies

As with most games we are less interested in outcomes of single plays in the game and more interested in the existence of strategies that ensure one player wins against any choice of moves from the other player.

Definition 2.11 (Strategy)**.** Let $\mathcal{A} = (V, V_0, V_1, E, v_I)$ be an arena. A *strategy (for Player i) in* $\mathcal{A}$ is a partial function $\sigma : V^*V_i \to V$ such that if $\sigma(v_1v_2\cdots v_n) = v'$ then $(v_n, v') \in E$. A play $\pi = v_1v_2\cdots$ is *consistent* with a strategy σ if for all $j < |\pi|$ such that $v_j \in V_i$, $\sigma(v_1v_2\cdots v_j) = v_{j+1}$.

Given a sequence of vertices visited, ending with a vertex in V_i, a strategy for Player i gives the vertex that Player i should then play to. We observe that given a strategy σ for Player 0 and a strategy τ for Player 1 from any vertex v there is a unique maximal play π^σ_τ from v consistent with σ and τ in the sense that any play consistent with both strategies is a prefix of π^σ_τ. We call this play the *play (from v) defined by strategies σ and τ*.

A useful class of strategies are those that can be defined from a fixed number of previously visited vertices.

Definition 2.12 (Strategy memory)**.** If a strategy σ has the property that for some fixed m, $\sigma(w) = \sigma(w')$ if w and w' agree on their last m letters, then we say that the strategy requires *finite memory* (of size $m-1$). If $m = 1$, we say the strategy is *memoryless* or *positional*.

Strategies extend to games in the obvious way.

Definition 2.13 (Game strategies)**.** Given a game $\mathbb{G} = (\mathcal{A}, \mathrm{Win})$, a *strategy for Player i in* $\mathbb{G}$ is a strategy for Player i in $\mathcal{A}$. A strategy σ for Player i is *winning* if all plays consistent with σ are winning for Player i. Player i *wins* $\mathbb{G}$ if Player i has a winning strategy from $v_I(\mathcal{A})$.

We observe that for any play $\pi = v_1v_2\cdots v_n$ in a Muller game, consistent with a winning strategy σ for Player i, if $v_n \in V_i(\mathcal{A})$ then $\sigma(\pi)$ is defined.
Earlier we alluded to the following important result of Büchi and Landweber [BL69].

Theorem 2.14 ([BL69])**.** *Let $\mathbb{G} = (\mathcal{A}, \mathcal{F})$ be a Muller game. One player has a winning strategy on $\mathbb{G}$ with finite memory of size at most $|V(\mathcal{A})|!$.*

An immediate corollary of this is that Muller games are decidable: we can check all possible strategies for both players that use at most $|V(\mathcal{A})|!$ memory, and see if the corresponding defined plays are winning. However, the complexity bounds on such an algorithm are enormous. In [McN93] McNaughton provided an algorithm with considerably better space and time bounds.

Theorem 2.15 ([McN93])**.** *Let $\mathbb{G} = (\mathcal{A}, \mathcal{F})$ be a Muller game with $\mathcal{A} = (V, V_0, V_1, E, v_I)$. Whether Player 0 has a winning strategy from v_I can be decided in time $O(|V|^2|E||V|!)$ and space $O(|V|^2)$.*

For union-closed games and their duals we can reduce the memory requirement for a winning strategy.

Theorem 2.16 ([Kla94])**.** *Let $\mathbb{G} = (\mathcal{A}, \mathcal{F})$ be a Muller game. If $\mathcal{F}$ is closed under unions and Player 1 has a winning strategy, then Player 1 has a memoryless winning strategy. Dually, if the complement of $\mathcal{F}$ is closed under union and Player 0 has a winning strategy, then Player 0 has a memoryless winning strategy.*

Two useful tools for constructing decidability algorithms are *force-sets* and *avoid-sets*.

Definition 2.17 (Force-set and Avoid-set)**.** Let $\mathcal{A}$ be an arena, and $X, Y \subseteq V(\mathcal{A})$. The set $\mathit{Force}^i_X(Y)$ is the set of vertices from which Player i has a strategy σ such that any play consistent with σ reaches some vertex in Y without leaving X. The set $\mathit{Avoid}^i_X(Y)$ is the set of vertices from which Player i has a strategy σ such that any play consistent with σ that remains in X avoids all vertices in Y.

We observe from the definitions that $\mathit{Force}^i_X(Y) = X \setminus \mathit{Avoid}^{1-i}_X(Y)$. We also observe that we may assume the strategies σ are memoryless: if Player i can force the play from v to some vertex of Y, the play to v is irrelevant.

Computing a force-set is an instance of the well-known alternating reachability problem, and in Algorithm 2.1 we present the standard algorithm for computing a force-set. Nerode, Remmel and Yakhnis [NRY96] provide an implementation of this algorithm which runs in time $O(|E(\mathcal{A})|)$, giving us the following:

Lemma 2.18. *Let $\mathcal{A}$ be an arena. For any sets $X, Y \subseteq V(\mathcal{A})$, $\mathit{Force}^0_X(Y)$ can be computed in time $O(|E(\mathcal{A})|)$*

Algorithm 2.1 $\textsc{Force}^0_X(Y)$

Returns: The set of vertices $v \in V(\mathcal{A})$ such that Player 0 has a strategy to force a play from v to some element of Y without visiting a vertex outside X.

let $R = \{v \in V_0(\mathcal{A}) \cap X : \text{ there exists } w \in Y \text{ with } (v, w) \in E(\mathcal{A})\}$.
let $S = \{v \in V_1(\mathcal{A}) \cap X : \text{ for all } w \text{ with } (v, w) \in E(\mathcal{A}),\ w \in Y\}$.
if $R \cup S \subseteq Y$ **then**
  return Y
else
  return $\textsc{Force}^0_X(R \cup S \cup Y)$.

2.1.4 Simulations

One of the most important concepts in transition systems is the notion of bisimulation. Two transition systems are bisimilar if each system can simulate the other. That is,

Definition 2.19 (Bisimulation)**.** Let $\mathcal{T} = (S, s_0, E)$ and $\mathcal{T}' = (S', s'_0, E')$ be transition systems. We say $\mathcal{T}$ and $\mathcal{T}'$ are *bisimilar* if there exists a relation $\sim \subseteq S \times S'$ such that:

- $s_0 \sim s'_0$,
- If $(s,t) \in E$ and $s \sim s'$ then there exists $t' \in S'$ such that $(s',t') \in E'$ and $t \sim t'$, and
- If $(s',t') \in E'$ and $s \sim s'$ then there exists $t \in S$ such that $(s,t) \in E$ and $t \sim t'$.

We now consider a generalization of bisimulation appropriate for arenas.

Definition 2.20 (Game simulation)**.** Let $\mathcal{A}$ and $\mathcal{A}'$ be arenas. A *game simulation from $\mathcal{A}$ to $\mathcal{A}'$* is a relation $\mathbf{S} \subseteq \big(V_0(\mathcal{A}) \times V_0(\mathcal{A}')\big) \cup \big(V_1(\mathcal{A}) \times V_1(\mathcal{A}')\big)$ such that:

(SIM-1) $v_I(\mathcal{A})\ \mathbf{S}\ v_I(\mathcal{A}')$,

(SIM-2) If $(u,v) \in E(\mathcal{A})$, $u \in V_0(\mathcal{A})$ and $u\ \mathbf{S}\ u'$, then there exists $v' \in V(\mathcal{A}')$ such that $(u',v') \in E(\mathcal{A}')$ and $v\ \mathbf{S}\ v'$, and

(SIM-3) If $(u',v') \in E(\mathcal{A}')$, $u' \in V_1(\mathcal{A}')$ and $u\ \mathbf{S}\ u'$, then there exists $v \in V(\mathcal{A})$ such that $(u,v) \in E(\mathcal{A})$ and $v\ \mathbf{S}\ v'$.

We write $\mathcal{A} \precsim \mathcal{A}'$ if there exists a game simulation from $\mathcal{A}$ to $\mathcal{A}'$.

We observe that $\precsim$ is reflexive and transitive and if $\mathcal{A} \precsim \mathcal{A}'$ then $\widetilde{\mathcal{A}'} \precsim \widetilde{\mathcal{A}}$. In Proposition 2.28 we show that it is also antisymmetric (up to bisimulation).

If $\mathcal{A} \precsim \mathcal{A}'$, then Player 0 can simulate plays on $\mathcal{A}'$ as plays on $\mathcal{A}$: every move made by Player 1 on $\mathcal{A}'$ can be translated to a move on $\mathcal{A}$, and for every response of Player 0 in $\mathcal{A}$, there is a corresponding response on $\mathcal{A}'$. Dually, Player 1 can simulate a play on $\mathcal{A}$ as a play on $\mathcal{A}'$. More precisely,

Lemma 2.21. *Let $\mathcal{A}$ and $\mathcal{A}'$ be arenas, and let $\mathbf{S}$ be a simulation from $\mathcal{A}$ to $\mathcal{A}'$. For any strategy σ for Player 0 in $\mathcal{A}$, and any strategy τ' for Player 1 in $\mathcal{A}'$, there exists a strategy σ' for Player 0 in $\mathcal{A}'$ and a strategy τ for Player 1 in $\mathcal{A}$ such that if $\pi = v_0 v_1 \cdots \in$ Plays$(\mathcal{A})$ is a play from $v_0 = v_I(\mathcal{A})$ consistent with σ and τ and $\pi' = v'_0 v'_1 \cdots \in$ Plays$(\mathcal{A}')$ is a play from $v'_0 = v_I(\mathcal{A}')$ consistent with σ' and τ', then $v_i\ \mathbf{S}\ v'_i$ for all i, $0 \le i \le \min\{|\pi|, |\pi'|\}$.*

Proof. We define σ' and τ as follows. Let $\pi = v_0 v_1 v_2 \cdots v_n$ and $\pi' = v'_0 v'_1 \cdots v'_n$ and suppose $v_i\ \mathbf{S}\ v'_i$ for all i, $0 \le i \le n$. Suppose first that $v_n \in V_0(\mathcal{A})$ (so $v'_n \in V_0(\mathcal{A}')$) and $\sigma(\pi) = v_{n+1}$. Since $(v_n, v_{n+1}) \in E(\mathcal{A})$ and $v_n\ \mathbf{S}\ v'_n$, from Condition (SIM-2) there exists v'_{n+1} such that $(v'_n, v'_{n+1}) \in E(\mathcal{A}')$ and $v_{n+1}\ \mathbf{S}\ v'_{n+1}$. Define $\sigma'(\pi') := v'_{n+1}$. Now suppose $v_n \in V_1(\mathcal{A})$ (so $v'_n \in V_1(\mathcal{A}')$). Let $\tau'(\pi') = v'_{n+1}$ and let v_{n+1} be the successor of v_n, such that $v_{n+1}\ \mathbf{S}\ v'_{n+1}$ guaranteed by Condition (SIM-3). Define $\tau(\pi) = v_{n+1}$. We observe that although σ' and τ are only defined for some plays, this definition is sufficient: as $v_0\ \mathbf{S}\ v'_0$, it follows by induction that for every play $\pi' = v'_0 v'_1 \cdots v'_n$ consistent with σ' and τ' there is a play $\pi = v_0 v_1 \cdots v_n$ (consistent with σ) such that $v_i\ \mathbf{S}\ v'_i$ for all i, $0 \le i \le n$. Thus if $v'_n \in V_0(\mathcal{A}')$, $\sigma(\pi')$ is well-defined. □

We observe that the strategies σ' and τ are independently derivable from τ' and σ respectively. That is, we can interchange the $\forall\tau'$ and $\exists\sigma'$ (or the $\forall\sigma$ and $\exists\tau$) quantifications to obtain:

Corollary 2.22. *Let $\mathcal{A}$ and $\mathcal{A}'$ be arenas, and let $\mathbf{S}$ be a game simulation from $\mathcal{A}$ to $\mathcal{A}'$. For every strategy σ for Player 0 in $\mathcal{A}$ there exists a strategy σ' for Player 0 in $\mathcal{A}'$ such that for every play $v_0'v_1'\cdots$ consistent with σ' there exists a play $v_0v_1\cdots$, consistent with σ such that $v_i \mathbf{S} v_i'$ for all i. Dually, for every strategy τ' for Player 1 in $\mathcal{A}'$ there exists a strategy τ for Player 1 in $\mathcal{A}$ such that for every play $v_0v_1\cdots$ consistent with τ there exists a play $v_0v_1\cdots$, consistent with τ' such that $v_i \mathbf{S} v_i'$ for all i.*

We call the strategies which we can derive in such a manner *simulated strategies*.

Definition 2.23 (Simulated search strategy)**.** Let $\mathcal{A}$, $\mathcal{A}'$, $\mathbf{S}$, σ, σ', τ and τ' be as above. We call σ' a $\mathbf{S}$-*simulated strategy of* σ, and τ a $\mathbf{S}$-*simulated strategy of* τ'.

We can use game simulations to translate winning strategies from one game into winning strategies in another. However, we require that a simulation respects the winning condition in some sense.

Definition 2.24 (Faithful simulation)**.** Let $\mathbb{G} = (\mathcal{A}, \mathrm{Win})$ and $\mathbb{G}' = (\mathcal{A}', \mathrm{Win}')$ be games. Let $\mathbf{S}$ be a game simulation from $\mathcal{A}$ to $\mathcal{A}'$, and let $\mathbf{S}$ also denote the pointwise extension of the relation to plays: $\pi \mathbf{S} \pi'$ if $|\pi| = |\pi'|$ and $v_i \mathbf{S} v_i'$ for all $v_i \in \pi$ and $v_i' \in \pi'$. We say $\mathbf{S}$ is $(\mathrm{Win}, \mathrm{Win}')$-*faithful* if for all $\pi \in \mathrm{Plays}(\mathcal{A})$ and all $\pi' \in \mathrm{Plays}(\mathcal{A}')$ such that $\pi \mathbf{S} \pi'$:

$$\pi \in \mathrm{Win} \Longrightarrow \pi' \in \mathrm{Win}'.$$

The next result follows immediately from the definitions.

Proposition 2.25. *Let $\mathbb{G} = (\mathcal{A}, \mathrm{Win})$ and $\mathbb{G}' = (\mathcal{A}', \mathrm{Win}')$ be games. Let $\mathbf{S}$ be a $(\mathrm{Win}, \mathrm{Win}')$-faithful game simulation from $\mathcal{A}$ to $\mathcal{A}'$. If σ is a winning strategy for Player 0 in $\mathbb{G}$ then any $\mathbf{S}$-simulated strategy is a winning strategy for Player 0 in $\mathbb{G}'$. Dually, if τ' is a winning strategy for Player 1 in $\mathbb{G}'$ then any $\mathbf{S}$-simulated strategy is a winning strategy for Player 1 in $\mathbb{G}$.*

For simple games checking if a game simulation is faithful is relatively easy. It follows from the definition of a game simulation that all finite plays automatically satisfy the criterion. Thus it suffices to check the infinite plays. But for simple games these are vacuously satisfied in two cases:

Lemma 2.26. *Let $\mathbb{G} = (\mathcal{A}, \mathcal{F})$ and $\mathbb{G}' = (\mathcal{A}', \mathcal{F}')$ be Muller games and let $\mathbf{S}$ be a simulation from $\mathcal{A}$ to $\mathcal{A}'$. If either $\mathcal{F} = \emptyset$ or $\mathcal{F}' = \mathcal{P}(V(\mathcal{A}'))$ then $\mathbf{S}$ is faithful.*

Corollary 2.27. *Let $\mathbb{G} = (\mathcal{A}, \mathcal{F})$ and $\mathbb{G}' = (\mathcal{A}', \mathcal{F}')$ be Muller games such that $\mathcal{F} = \emptyset$ or $\mathcal{F}' = \mathcal{P}(V(\mathcal{A}'))$. If $\mathcal{A} \precsim \mathcal{A}'$ and Player 0 wins $\mathbb{G}$, then Player 0 wins $\mathbb{G}'$. Dually, if $\mathcal{A} \precsim \mathcal{A}'$ and Player 1 wins $\mathbb{G}'$, then Player 1 wins $\mathbb{G}$.*

We conclude this section by showing how game simulations relate to bisimulation.

Proposition 2.28. *Let $\mathcal{A} = (V, V_0, V_1, E, v_I)$ and $\mathcal{A}' = (V', V_0', V_1', E', v_I')$ be arenas. If $\mathcal{A} \precsim \mathcal{A}'$ and $\mathcal{A}' \precsim \mathcal{A}$ then the transition systems (V, v_I, E) and (V', v_I', E') are bisimilar.*

Proof. Let $\mathbf{S}$ be a game simulation from $\mathcal{A}$ to $\mathcal{A}'$ and let $\mathbf{S}'$ be a game simulation from $\mathcal{A}'$ to $\mathcal{A}$. It follows from the definitions that the relation $\mathbf{S} \cup (\mathbf{S}')^{-1}$ is a bisimulation between the two transition systems. □

2.2 Winning condition presentations

As we discussed above, if we are interested in investigating the complexity of the problem of deciding Muller games, we need to consider the manner in which the winning condition is presented. As we see in Section 2.2.1, for many games that occur in the literature relating to logics and automata the winning condition can be expressed in a more efficient manner than simply listing the elements of $\mathcal{F}$. To formally describe such specifications, we introduce the concept of a *condition type*.

Definition 2.29 (Condition type)**.** A *condition type* is a function $\mathfrak{A}$ which maps an arena $\mathcal{A}$ to a pair $(\mathcal{I}^{\mathcal{A}}, \models^{\mathcal{A}})$ where $\mathcal{I}^{\mathcal{A}}$ is a set and $\models^{\mathcal{A}} \subseteq \text{Plays}(\mathcal{A}) \times \mathcal{I}^{\mathcal{A}}$ is the *acceptance relation*. We call elements of $\mathcal{I}^{\mathcal{A}}$ *condition types* (or simply, *conditions*). A *regular condition type* maps an arena $\mathcal{A}$ to a pair $(\mathcal{I}^{\mathcal{A}}, \models^{\mathcal{A}})$ where $\mathcal{I}^{\mathcal{A}}$ is a set of conditions and $\models^{\mathcal{A}} \subseteq \mathcal{P}(V(\mathcal{A})) \times \mathcal{I}^{\mathcal{A}}$.

Remark. In the sequel we will generally regard the relation $\models^{\mathcal{A}}$ as intrinsically defined, and associate $\mathfrak{A}(\mathcal{A})$ with the set $\mathcal{I}^{\mathcal{A}}$. That is, we will use $\Omega \in \mathfrak{A}(\mathcal{A})$ to indicate $\Omega \in \mathcal{I}^{\mathcal{A}}$.

A (regular) condition type defines a family of (Muller) games in the following manner. Let $\mathfrak{A}$ be a condition type, $\mathcal{A}$ an arena, and $\mathfrak{A}(\mathcal{A}) = (\mathcal{I}^{\mathcal{A}}, \models^{\mathcal{A}})$. For $\Omega \in \mathcal{I}^{\mathcal{A}}$, the game $(\mathcal{A}, \Omega)$ is the game $(\mathcal{A}, \text{Win})$ where $\text{Win} = \{\pi \in \text{Plays}(\mathcal{A}) : \pi \models^{\mathcal{A}} \Omega\}$. We generally call a game where the winning condition is specified by a condition of type $\mathfrak{A}$ an $\mathfrak{A}$*-game*, for example a *parity game* is a game where the winning condition is specified by a *parity condition* (see Definition 2.41 below). We can now state precisely the decision problem we are interested in.

$\mathfrak{A}$-GAME	
Instance:	A game $\mathbb{G} = (\mathcal{A}, \Omega)$ where $\Omega \in \mathfrak{A}(\mathcal{A})$.
Problem:	Does Player 0 have a winning strategy in $\mathbb{G}$?

The exploration of the complexity of this problem is one of the main research problems that this dissertation addresses.

Research aim. *Investigate the complexity of deciding* $\mathfrak{A}$-GAME *for various (regular) condition types* $\mathfrak{A}$*.*

2.2.1 Examples

We now give some examples of regular condition types that occur in the literature. First we observe that an instance $\Omega \in \mathfrak{A}(\mathcal{A})$ of a regular condition type $\mathfrak{A}$ defines a family of subsets of $V(\mathcal{A})$:

$$\mathcal{F}_\Omega := \{I \subseteq V(\mathcal{A}) : I \models^{\mathcal{A}} \Omega\}.$$

We call this the *set specified by the condition* Ω. In the examples below, we describe the set specified by a condition to define the acceptance relation $\models^{\mathcal{A}}$.

General purpose condition types

The first examples we consider are general purpose formalisms in that they may be used to specify any family of sets.

The most straightforward presentation of the winning condition of a Muller game $(\mathcal{A}, \mathcal{F})$ is given by explicitly listing all elements of $\mathcal{F}$. We call this an *explicit presentation*. We can view such a formalism in our framework as follows:

Definition 2.30 (Explicit condition type)**.** An instance of the *explicit condition type* is a set $\mathcal{F} \subseteq \mathcal{P}(V(\mathcal{A}))$. The set specified by an instance is the set which defines the instance.

In the literature an explicit presentation is sometimes called a *Muller condition*. However, we reserve that term for the more commonly used presentation for Muller games in terms of colours given next.

Definition 2.31 (Muller condition type)**.** An instance of the *Muller condition type* is a pair $(\chi, \mathcal{C})$ where, for some set C, $\chi : V(\mathcal{A}) \to C$ and $\mathcal{C} \subseteq \mathcal{P}(C)$. The set $\mathcal{F}_{(\chi,\mathcal{C})}$ specified by a Muller condition $(\chi, \mathcal{C})$ is the set $\{I \subseteq V(\mathcal{A}) : \chi(I) \in \mathcal{C}\}$.

To distinguish Muller games from games with a winning condition specified by a Muller condition, we explicitly state the nature of the presentation of the winning condition if it is critical.

From a more practical perspective, when considering applications of these types of games it may be the case that there are vertices whose appearance in any infinite run is irrelevant. This leads to the definition of a *win-set condition*.

Definition 2.32 (Win-set condition type)**.** An instance of the *win-set condition type* is a pair $(W, \mathcal{W})$ where $W \subseteq V(\mathcal{A})$ and $\mathcal{W} \subseteq \mathcal{P}(W)$. The set $\mathcal{F}_{(W,\mathcal{W})}$ specified by a win-set condition $(W, \mathcal{W})$ is the set $\{I \subseteq V(\mathcal{A}) : W \cap I \in \mathcal{W}\}$.

Another way to describe a winning condition is as a boolean formula. Such a formalism is somewhat closer in nature than the specifications we have so far considered to the motivating problem of verifying reactive systems: requirements of such systems are more readily expressed as logical formulas. Winning conditions of this kind were considered by Emerson and Lei [EL85].

Definition 2.33 (Emerson-Lei condition type)**.** An instance of the *Emerson-Lei condition type* is a boolean formula φ with variables from the set $V(\mathcal{A})$. The set $\mathcal{F}_\varphi$ specified by an Emerson-Lei condition φ is the collection of sets $I \subseteq V(\mathcal{A})$ such that the truth assignment that maps each element of I to true and each element of $V(\mathcal{A}) \setminus I$ to false satisfies φ.

A boolean formula can contain a lot of repetition, so it may be more efficient to consider *boolean circuits* rather than formulas. This motivates one of the most succinct types of winning condition we consider.

Definition 2.34 (Circuit condition type)**.** An instance of the *circuit condition type* is a boolean circuit C with input nodes from the set $V(\mathcal{A})$ and one output node. The set $\mathcal{F}_C$ specified by a circuit condition C is the collection of sets $I \subseteq V(\mathcal{A})$ such that C outputs true when each input corresponding to a vertex in I is set to true and all other inputs are set to false.

The final general purpose formalisms we consider are somewhat more exotic. In [Zie98], Zielonka introduced a representation for a family of subsets of a set V, $\mathcal{F} \subseteq \mathcal{P}(V)$, in terms of a labelled tree where the labels on the nodes are subsets of V.

Definition 2.35 (Zielonka tree and Zielonka DAG)**.** Let V be a set and $\mathcal{F} \subseteq \mathcal{P}(V)$. The *Zielonka tree* (also called a *split tree* of the set $\mathcal{F}$, $\mathcal{Z}_{\mathcal{F},V}$, is defined inductively as:

1. If $V \notin \mathcal{F}$ then $\mathcal{Z}_{\mathcal{F},V} = \mathcal{Z}_{\overline{\mathcal{F}},V}$, where $\overline{\mathcal{F}} = \mathcal{P}(V) \setminus \mathcal{F}$.

2. If $V \in \mathcal{F}$ then the root of $\mathcal{Z}_{\mathcal{F},V}$ is labelled with V. Let $M_1, M_2, \ldots, M_k$ be the $\subseteq$-maximal sets in $\overline{\mathcal{F}}$, and let $\mathcal{F}|_{M_i} = \mathcal{F} \cap \mathcal{P}(M_i)$. The successors of the root are the subtrees $\mathcal{Z}_{\mathcal{F}|_{M_i},M_i}$, for $1 \leq i \leq k$.

A *Zielonka DAG* is constructed as a Zielonka tree except nodes labelled by the same set are identified, making it a directed acyclic graph. Nodes of $\mathcal{Z}_{\mathcal{F},V}$ labelled by elements of $\mathcal{F}$ are called *0-level nodes*, and other nodes are *1-level nodes*.

Zielonka trees are intimately related to Muller games. In particular they identify the size of memory required for a winning strategy: the "amount" of branching of 0-level nodes indicates the maximum amount of memory required for a winning strategy for Player 0, and similarly for 1-level nodes and Player 1 [DJW97]. For example, the 1-level nodes of a Zielonka tree of a union-closed family of sets have at most one successor, indicating that if Player 1 has a winning strategy then he has a memoryless winning strategy. Thus we also consider games where the winning condition is specified as a Zielonka tree (or the more succinct Zielonka DAG).

Definition 2.36 (Zielonka tree and Zielonka DAG condition types)**.** An instance of the *Zielonka tree (DAG) condition type* is a Zielonka tree (DAG) $\mathcal{Z}_{\mathcal{F},V(\mathcal{A})}$ for some $\mathcal{F} \subseteq \mathcal{P}(V(\mathcal{A}))$. The set specified by an instance is the set $\mathcal{F}$ used to define the instance.

Other condition types

We now consider formalisms that can only specify restricted families of sets such as union-closed or upward-closed families. The first formalism we consider is a well-known specification, introduced by Rabin in [Rab72] as an acceptance condition for infinite automata.

Definition 2.37 (Rabin condition type)**.** An instance of the *Rabin condition type* is a set of pairs $\Omega = \{(L_i, R_i) : 1 \leq i \leq m\}$. The set $\mathcal{F}_\Omega$ specified by a Rabin condition Ω is the collection of sets $I \subseteq V(\mathcal{A})$ such that there exists an i, $1 \leq i \leq m$, such that $I \cap L_i \neq \emptyset$ and $I \cap R_i = \emptyset$.

The remaining formalisms we consider can only be used to specify families of sets that are closed under union. The first of these, the Streett condition type, introduced in [Str82], is similar to the Rabin condition type.

Definition 2.38 (Streett condition type)**.** An instance of the *Streett condition type* is a set of pairs $\Omega = \{(L_i, R_i) : 1 \leq i \leq m\}$. The set $\mathcal{F}_\Omega$ specified by a Streett condition Ω is the collection of sets $I \subseteq V(\mathcal{A})$ such that for all i, $1 \leq i \leq m$, either $I \cap L_i \neq \emptyset$ or $I \cap R_i = \emptyset$.

The Streett condition type is useful for describing fairness conditions such as those considered in [EL85]. An example of a fairness condition for infinite computations is: "every process enabled infinitely often is executed infinitely often". Viewing vertices of an arena as states of an infinite computation system where some processes are executed and some are enabled, this is equivalent to saying "for every process, either the set of states which enable the process is visited finitely often or the set of states which execute the process is visited infinitely often", which we see is easily interpreted as a Streett condition.

The Streett and Rabin condition types are dual in the following sense: for any set $\mathcal{F} \subseteq \mathcal{P}(V(\mathcal{A}))$ which can be specified by a Streett condition, there is a Rabin condition which specifies $\mathcal{P}(V(\mathcal{A})) \setminus \mathcal{F}$, and conversely. Indeed, if $\Omega = \{(L_i, R_i) : 1 \leq i \leq m\}$ is a Streett condition, then for the Rabin condition $\widetilde{\Omega} = \{(R_i, L_i) : 1 \leq i \leq m\}$ we have $\mathcal{F}_{\widetilde{\Omega}} = \mathcal{P}(V(\mathcal{A})) \setminus \mathcal{F}_\Omega$. This

implies that the dual of a Streett game can be expressed as a Rabin game, and conversely the dual of a Rabin game can be expressed as a Streett game.

If we are interested in specifying union-closed families of sets efficiently, we can consider the closure under union of a given set. This motivates the following definition:

Definition 2.39 (Basis condition type)**.** An instance of the *basis condition type* is a set $\mathcal{B} \subseteq \mathcal{P}(V(\mathcal{A}))$. The set $\mathcal{F}_{\mathcal{B}}$ specified by a basis condition $\mathcal{B}$ is the collection of sets $I \subseteq V(\mathcal{A})$ such that there are $B_1, \ldots, B_n \in \mathcal{B}$ with $I = \bigcup_{1 \leq i \leq n} B_i$.

In a similar manner to the basis condition type, if we are interested in efficiently specifying an upward-closed family of sets, we can explicitly list the $\subseteq$-minimal elements of the family. This gives us the *superset condition type*, also called a *superset Muller condition* in [LTMN02].

Definition 2.40 (Superset condition type)**.** An instance of the *superset condition type* is a set $\mathcal{M} \subseteq \mathcal{P}(V(\mathcal{A}))$. The set $\mathcal{F}_{\mathcal{M}}$ specified by a superset condition $\mathcal{M}$ is the set $\{I \subseteq V(\mathcal{A}) : M \subseteq I \text{ for some } M \in \mathcal{M}\}$.

The final formalism we consider is one of the most important and interesting Muller condition types, the *parity condition type*.

Definition 2.41 (Parity condition type)**.** An instance of the *parity condition type* is a function $\chi : V(\mathcal{A}) \to \mathsf{P}$ where $\mathsf{P} \subseteq \omega$ is a set of *priorities*. The set $\mathcal{F}_\chi$ specified by a parity condition χ is the collection of sets $I \subseteq V(\mathcal{A})$ such that $\max\{\chi(v) : v \in I\}$ is even.

Remark. We have technically defined here the *max-parity condition*. There is an equivalent formalism sometimes considered where the parity of the *minimum* priority visited infinitely often determines the winner, called the *min-parity condition*. Throughout this dissertation we only consider the max-parity condition.

It is not difficult to show that the set specified by a parity condition is closed under union as is the complement of the set specified. Therefore, from Theorem 2.16 we have the following:

Theorem 2.42 (Memoryless determinacy of parity games [EJ91, Mos91])**.** *Let $\mathbb{G} = (\mathcal{A}, \chi)$ be a parity game. The player with a winning strategy has a winning strategy which is memoryless.*

Indeed, any union-closed set with a union-closed complement can be specified by a parity condition, implying that the parity condition is one of the most expressive conditions where memoryless strategies are sufficient for both players. This result is very useful in the study of infinite games and automata: one approach to showing that Muller automata are closed under complementation is to reduce the problem to a parity game, and utilise the fact that if Player 1 has a winning strategy then he has a memoryless strategy to construct an automaton which accepts the complementary language [EJ91].

One of the reasons why parity games are an interesting class of games to study is that the exact complexity of the problem of deciding the winner remains elusive. In Chapter 3 we discuss this and other reasons why parity games are important in more detail.

2.2.2 Translations

We now present a framework in which we can compare the expressiveness and succinctness of condition types by considering transformations between games which keep the arena the same. More precisely, we define what it means for a condition type to be *translatable* to another condition type as follows.

Definition 2.43 (Translatable)**.** Given two condition types $\mathfrak{A}$ and $\mathfrak{B}$, we say that $\mathfrak{A}$ is *polynomially translatable* to $\mathfrak{B}$ if for any arena $\mathcal{A}$, with $\mathfrak{A}(\mathcal{A}) = (\mathcal{I}^{\mathcal{A}}_{\mathfrak{A}}, \models^{\mathcal{A}}_{\mathfrak{A}})$ and $\mathfrak{B}(\mathcal{A}) = (\mathcal{I}^{\mathcal{A}}_{\mathfrak{B}}, \models^{\mathcal{A}}_{\mathfrak{B}})$, there is a function $f : \mathcal{I}^{\mathcal{A}}_{\mathfrak{A}} \to \mathcal{I}^{\mathcal{A}}_{\mathfrak{B}}$ such that for all $\Omega \in \mathcal{I}^{\mathcal{A}}_{\mathfrak{A}}$:

- $f(\Omega)$ is computed in time polynomial in $|\mathcal{A}| + |\Omega|$, and
- For all $\pi \in \text{Plays}(\mathcal{A})$, $\pi \models^{\mathcal{A}}_{\mathfrak{A}} \Omega \iff \pi \models^{\mathcal{A}}_{\mathfrak{B}} f(\Omega)$.

As we are only interested in polynomial translations, we simply say $\mathfrak{A}$ is *translatable* to $\mathfrak{B}$ to mean that it is polynomially translatable. Clearly, if condition type $\mathfrak{A}$ is translatable to $\mathfrak{B}$ then the problem of deciding the winner for games of type $\mathfrak{A}$ is reducible in polynomial time to the corresponding problem for games of type $\mathfrak{B}$. That is,

Lemma 2.44. *Let $\mathfrak{A}$ and $\mathfrak{B}$ be condition types such that $\mathfrak{A}$ is translatable to $\mathfrak{B}$. Then there is a polynomial time reduction from* $\mathfrak{A}$-GAME *to* $\mathfrak{B}$-GAME.

If condition type $\mathfrak{A}$ is not translatable to $\mathfrak{B}$ this may be for one of three reasons. Either $\mathfrak{A}$ is more expressive than $\mathfrak{B}$ in that there are sets $\mathcal{F}$ that can be expressed using conditions from $\mathfrak{A}$ but no condition from $\mathfrak{B}$ can specify $\mathcal{F}$; or there are some sets for which the representation of type $\mathfrak{A}$ is necessarily more succinct; or the translation, while not size-increasing, can not be computed in polynomial time. We are primarily interested in the second situation. Formally, we say

Definition 2.45 (Succinctness)**.** $\mathfrak{A}$ is *more succinct* than $\mathfrak{B}$ if $\mathfrak{B}$ is translatable to $\mathfrak{A}$ but $\mathfrak{A}$ is not translatable to $\mathfrak{B}$.

We now consider translations between some of the condition types we defined in Section 2.2.1.

Translations between general purpose condition types

It is straightforward to show that win-set conditions are more succinct than explicit presentations. To translate an explicitly presented game $(\mathcal{A}, \mathcal{F})$ to a win-set condition, simply take $W = V(\mathcal{A})$ and $\mathcal{W} = \mathcal{F}$. To show that win-set conditions are not translatable to explicit presentations, consider a game where $W = \emptyset$ and $\mathcal{W} = \{\emptyset\}$. The set $\mathcal{F}_{(W,\mathcal{W})}$ specified by this condition consists of all subsets of $V(\mathcal{A})$ and thus an explicit presentation must be exponential in length.

Proposition 2.46. *The win-set condition type is more succinct than an explicit presentation.*

Similarly, there is a trivial translation from the Emerson-Lei condition type to the circuit condition type. However, the question of whether there is a translation in the other direction is an important open problem in the field of circuit complexity [Pap95].

Open problem 2.47. *Is the circuit condition type more succinct than the Emerson-Lei condition type?*

We now show, through the next theorems, that circuit presentations are more succinct than Zielonka DAG presentations, which, along with Emerson-Lei presentations, are more succinct than Muller presentations, which are in turn more succinct than win-set presentations.

Theorem 2.48. *The Muller condition type is more succinct than the win-set condition type.*

Proof. Given a win-set game $\big(\mathcal{A}, (W, \mathcal{W})\big)$, we construct a Muller condition describing the same set of subsets as $(W, \mathcal{W})$. For the set of colours we use $C = W \cup \{c\}$, where c is distinct from any element of W. The colouring function $\chi : V(\mathcal{A}) \to C$ is then defined as:

- $\chi(w) = w$ for $w \in W$,
- $\chi(v) = c$ for $v \notin W$.

The family $\mathcal{C}$ of subsets of C is the set $\big\{X, X \cup \{c\} : X \in \mathcal{W}\big\}$. For $I \subseteq V$, if $I \subseteq W$, then $\chi(I) = I$ otherwise $\chi(I) = \{c\} \cup I$. Either way, $I \cap W$ is in $\mathcal{W}$ if, and only if, $\chi(I) \in \mathcal{C}$.

To show that there is no translation in the other direction, consider a Muller game on $\mathcal{A}$, where half of $V(\mathcal{A})$, V_r, is coloured red, the other half coloured blue, and the family of sets of colours is $\mathcal{C} = \big\{\{\mathsf{red}\}\big\}$. The family $\mathcal{F}$ described by this condition consists of the $2^{|V(\mathcal{A})|/2} - 1$ non-empty subsets of V_r. Now consider trying to describe this family using a win-set condition. In general, for the set $\mathcal{F}'$ specified by the win-set condition $(W, \mathcal{W})$, any $v \notin W$, and $X \subseteq V(\mathcal{A})$ we have $\{v\} \cup X \in \mathcal{F}' \Leftrightarrow X \in \mathcal{F}'$. Observe that in our game no vertex has this latter property: if $v \in V_r$, then $\{v\} \in \mathcal{F}$, but $\emptyset \notin \mathcal{F}$; and if $v \notin V_r$ then $\{v\} \cup V_r \notin \mathcal{F}$, but $V_r \in \mathcal{F}$. Thus our win-set, W must be equal to $V(\mathcal{A})$, and $\mathcal{W}$ is the explicit listing of the $2^{|V(\mathcal{A})|/2} - 1$ subsets of V_r. Thus $(W, \mathcal{W})$ cannot be produced in polynomial time. □

Theorem 2.49. *The Zielonka DAG condition type is more succinct than the Muller condition type.*

Proof. Given a Muller game consisting of an arena $\mathcal{A} = (V, V_0, V_1, E, v_I)$, a colouring $\chi : V \to C$ and a family $\mathcal{C}$ of subsets of C, we construct a Zielonka DAG $\mathcal{Z}_{\mathcal{F},V}$ which describes the same set of subsets of $V(\mathcal{A})$ as the Muller condition $(\chi, \mathcal{C})$. Consider the Zielonka DAG $\mathcal{Z}_{\mathcal{C},C}$, whose nodes are labelled by sets of colours. If we replace a label $L \subseteq C$ in this tree with the set $\{v \in V : \chi(v) \in L\}$ then we obtain a Zielonka DAG $\mathcal{Z}_{\mathcal{F},V}$ over the set of vertices. We argue that $\mathcal{F}$ is, in fact, the set specified by the Muller condition $(\chi, \mathcal{C})$ and then show that $\mathcal{Z}_{\mathcal{C},C}$ can be constructed in polynomial time. Since the translation from $\mathcal{Z}_{\mathcal{C},C}$ to $\mathcal{Z}_{\mathcal{F},V}$ involves an increase in size by at most a factor of $|V|$, this establishes that Muller games are translatable to Zielonka DAGs.

Let $I \subseteq V$ be a set of vertices. If $I \in \mathcal{F}$ then, by the definition of Zielonka DAGs, I is a subset of a label X of a 0-level node t of $\mathcal{Z}_{\mathcal{F},V}$ and is not contained in any of the labels of the 1-level successors of t. That is, for each 1-level successor u of t, there is a vertex $v \in I$ such that $\chi(v) \notin \chi(L_u)$ where L_u is the label of u. Moreover, $\chi(I) \subseteq \chi(X)$. Now $\chi(X)$ is, by construction, the label of a 0-level node of $\mathcal{Z}_{\mathcal{C},C}$ and we have established that $\chi(I)$ is contained in this label and is not contained in any of the labels of the 1-level successors of that node. Therefore, $\chi(I) \in \mathcal{C}$. Similarly, by interchanging 0-level and 1-level nodes, $\chi(I) \notin \mathcal{C}$ if $I \notin \mathcal{F}$.

To show that we can construct $\mathcal{Z}_{\mathcal{C},C}$ in polynomial time, observe first that every subset $X \subseteq C$ has at most $|C|$ maximal subsets. Note further that the label of any node in $\mathcal{Z}_{\mathcal{C},C}$ is either C, some element of $\mathcal{C}$ or a maximal (proper) subset of an element of $\mathcal{C}$. Thus, $\mathcal{Z}_{\mathcal{C},C}$ is no larger than $1 + |\mathcal{C}| + |C||\mathcal{C}|$. This bound on the size of the DAG is easily turned into a bound on the time required to construct it, using the inductive definition of Zielonka trees. Thus, we have shown that the Muller condition type is translatable into the Zielonka DAG condition type.

To show there is no translation in the other direction, consider the family $\mathcal{F}$ of subsets of $V(\mathcal{A})$ which consist of 2 or more elements. The Zielonka DAG which describes this family

consists of $|V(\mathcal{A})| + 1$ nodes – one 0-level node labelled by $V(\mathcal{A})$, and $|V(\mathcal{A})|$ 1-level nodes labelled by the singleton subsets of $V(\mathcal{A})$. However, to express this as a Muller condition, each vertex must have a distinct colour since for any pair of vertices there is a set in $\mathcal{F}$ that contains one but not the other. Thus, $|\mathcal{C}| = |\mathcal{F}| = 2^{|V(\mathcal{A})|} - |V(\mathcal{A})| - 1$. It follows that the translation from Zielonka DAGs to Muller conditions cannot be done in polynomial time. □

To show the remaining results, we use the following observation:

Lemma 2.50. *There is no translation from the Emerson-Lei condition type to the Zielonka DAG condition type.*

Proof. Let $V(\mathcal{A}) = V = \{x_1, \ldots, x_{2k}\}$, and consider the family of sets $\mathcal{F}$ described by the formula

$$\varphi := \bigvee_{1 \leq i \leq k} (x_{2i-1} \wedge x_{2i}).$$

Clearly $|\varphi| = O(|V(\mathcal{A})|)$. Now consider the Zielonka DAG $\mathcal{Z}_{\mathcal{F},V}$ describing $\mathcal{F}$. As $V \in \mathcal{F}$, the root of $\mathcal{Z}_{\mathcal{F},V}$ is a 0-level node labelled by V. The maximal subsets of V not in $\mathcal{F}$ are the 2^k subsets containing exactly one of $\{x_{2i-1}, x_{2i}\}$ for $1 \leq i \leq k$. Thus $\mathcal{Z}_{\mathcal{F},V}$ must have at least this number of nodes, and is therefore not constructible in polynomial time. □

Theorem 2.51. *The Emerson-Lei condition type is more succinct than the Muller condition type.*

Proof. Given a Muller game consisting of an arena $\mathcal{A}$, a colouring $\chi : V(\mathcal{A}) \to C$ and a family $\mathcal{C}$ of subsets of C, let φ be the boolean formula defined as:

$$\varphi := \bigvee_{X \in \mathcal{C}} \Big(\bigwedge_{c \in X} \big(\bigvee_{\chi(v) = c} v \big) \wedge \bigwedge_{c \notin X} \big(\bigwedge_{\chi(v) = c} \neg v \big) \Big).$$

It is easy to see that a subset $I \subseteq V(\mathcal{A})$ satisfies φ if, and only if, there is some set $X \in \mathcal{C}$ such that for all colours $c \in X$ there is some $v \in I$ such that $\chi(v) = c$ and for all colours $c' \notin X$ there is no $v \in I$ such that $\chi(v) = c'$. Since φ can clearly be constructed in time polynomial in $|\mathcal{C}| + |V(\mathcal{A})|$, it follows that there is a translation from the Muller condition type to the Emerson-Lei condition type.

For the reverse direction, we observe that as there is a translation from the Muller condition type to the Zielonka DAG condition type, if there were a translation from the Emerson-Lei condition type to the Muller condition type, this would contradict Lemma 2.50 as "translatability" is transitive. □

Theorem 2.52. *The circuit condition type is more succinct than the Zielonka DAG condition type.*

Proof. Given a Zielonka DAG game $(\mathcal{A}, \mathcal{Z}_{\mathcal{F},V})$ where $V = V(\mathcal{A})$, we define, for each node t in $\mathcal{Z}_{\mathcal{F},V}$ a boolean circuit C_t. This circuit is defined by induction on the height of t. For convenience, we associate each circuit with its output node. Suppose the label of t is X. We have the following cases:

(i) t is a 0-level ($X \in \mathcal{F}$) leaf: In this case, let $C_t = \bigwedge_{x \notin X} \neg x$.

(ii) t is a 1-level ($X \notin \mathcal{F}$) leaf: In this case, let $C_t = \bigvee_{x \notin X} x$.

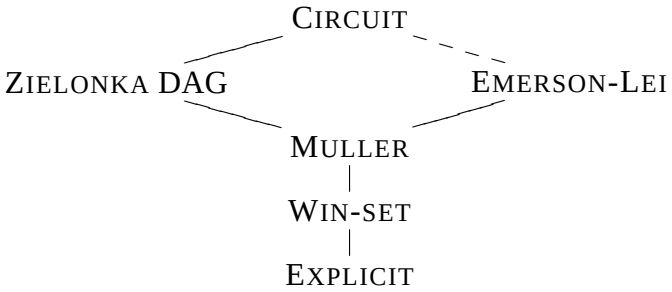


Figure 2.2: Summary of the succinctness results

(iii) t is a 0-level node with k successors $t_1, \ldots, t_k$: In this case, let $C_t = \bigwedge_{x \notin X} \neg x \wedge \bigwedge_{i=1}^{k} C_{t_i}$.

(iv) t is a 1-level node with k successors $t_1, \ldots, t_k$: In this case, let $C_t = \bigvee_{x \notin X} x \vee \bigvee_{i=1}^{k} C_{t_i}$.

We claim that the condition $\mathcal{F}$ is specified by the circuit C_r where r is the root of $\mathcal{Z}_{\mathcal{F},V}$. This formula has size at most $|V(\mathcal{A})||\mathcal{Z}_{\mathcal{F},V}|$ and is constructed in polynomial time. To show its correctness we argue by induction on the height of any node t with label X that C_t defines the restriction of $\mathcal{F}$ to X. We consider the following cases:

(i) t is a 0-level leaf. In this case any subset of X is in $\mathcal{F}$. $I \subseteq V(\mathcal{A})$ satisfies C_t if, and only if, no variable that is not in X appears in I, that is $I \subseteq X$.

(ii) t is a 1-level leaf. In this case any subset of X is not in $\mathcal{F}$. Here $I \subseteq V(\mathcal{A})$ satisfies C_t if, and only if, there is some element in I which is not in X, that is $I \not\subseteq X$.

(iii) t is a 0-level node with k successors labelled by $X_1, \ldots, X_k$. In this case any subset of X is in $\mathcal{F}$ unless it is a subset of X_i for some i, in which case whether it is in $\mathcal{F}$ is determined by nodes lower in the DAG. Here $I \subseteq V(\mathcal{A})$ satisfies C_t if, and only if, I is a subset of X and I satisfies C_{t_i} for all successors.

(iv) t is a 1-level node with k successors labelled by $X_1, \ldots, X_k$. In this case any subset of X is not in $\mathcal{F}$ unless it is a subset of X_i for some i. Here $I \subseteq V$ satisfies C_t if, and only if, either I is not contained in X, or there is some successor t_i such that I satisfies C_{t_i}.

We observe that as there is a translation from the Emerson-Lei condition type to the circuit condition type, Lemma 2.50 implies there is no translation from the circuit condition type to the Zielonka DAG condition type. □

Figure 2.2 summarizes the succinctness results we have so far shown, with the more succinct types towards the top. The dashed edge indicates that there is a translation but it is not known whether there is a translation in the opposite direction.

Translations between union-closed condition types

Turning to union-closed condition types, we observe that the basis condition type is a succinct way of describing union-closed sets. It is not even known if it is translatable to the circuit condition type, the most succinct type considered above. In Section 2.3.2 we show that the problem of deciding basis games is co-NP-complete. It follows from the NP-completeness of

Rabin games [EJ88], and duality that the problem of deciding Streett games is co-NP-complete. The following result implies that we cannot use translatability to obtain upper or lower bounds on the complexity of basis games based on the known bounds for Streett games.

Theorem 2.53. *The basis and Streett condition types are incomparable with respect to translatability. That is, neither is translatable to the other.*

Proof. To show there is no translation from Streett games to basis games, let $V(\mathcal{A}) = \{x_1, \ldots, x_{2k}\}$, and consider the Streett game with winning condition described by the pairs $\big\{(L_i, \emptyset) : 1 \leq i \leq k\big\}$, where $L_i = \{x_{2i-1}, x_{2i}\}$. Note that the family of sets described by this condition is $\mathcal{F} = \big\{X \subseteq V(\mathcal{A}) : \forall i \; X \not\subseteq V(\mathcal{A}) \setminus L_i\big\}$. Any basis for $\mathcal{F}$ must include the minimal elements of $\mathcal{F}$. However, the minimal elements include

$$\mathcal{M} = \big\{\{v_1, \ldots, v_k\} : v_i \in \{x_{2i-1}, x_{2i}\}\big\},$$

and $|\mathcal{M}| = 2^k$. Thus $\mathcal{F}$ cannot be represented by a basis constructible in polynomial time.

To show there is no translation in the other direction, let $V(\mathcal{A}) = \{x_1, \ldots, x_{2k}\}$, and consider the family $\mathcal{F}$ of sets formed by closing

$$\mathcal{B} = \big\{\{x_{2i-1}, x_{2i}\} : 1 \leq i \leq k\big\}$$

under union. Note that this is the same construction as for the proof of Theorem 2.52. Observe that $\mathcal{F}$ contains $2^k - 1$ sets, each with an even number of elements. Any Streett condition which describes the same family must contain at least this number of pairs in order to exclude the sets of odd cardinality. Thus $\mathcal{F}$ cannot be represented by a Streett condition which is constructible in polynomial time. □

It should be clear that the superset condition type is translatable to the basis condition type. We include the result for completeness.

Proposition 2.54. *The superset condition type is translatable to the basis condition type.*

We conclude these results with the following two observations regarding translations between explicit presentations and the basis and superset condition types.

Proposition 2.55. *The superset condition type is more succinct than an explicit presentation of an upward-closed set.*

Proof. Given an explicitly presented upward-closed game $(\mathcal{A}, \mathcal{F})$, the set $\mathcal{F}$, viewed as a superset condition, clearly describes the same set of subsets of $V(\mathcal{A})$. Conversely, for the superset game $\Big(\mathcal{A}, \big\{\{v\} : v \in V(\mathcal{A})\big\}\Big)$, the set described by the winning condition is of size $2^{|V(\mathcal{A})|} - 1$, and therefore cannot be explicitly presented in polynomial time. □

Corollary 2.56. *The basis condition type is more succinct than an explicit presentation of a union-closed set.*

Proof. The fact that the basis condition type is not translatable to an explicit presentation follows from Proposition 2.55 and Proposition 2.54 as "translatable" is transitive. The other direction is straightforward, the explicit presentation itself suffices as a basis. □

2.2.3 Extendibility

We now introduce a property of condition types that allows us to make simplifying assumptions about the arena. We say a regular condition type is *extendible* if it can "ignore" a set of added vertices. More precisely,

Definition 2.57 (Extendible condition type)**.** Let $\mathfrak{A}$ be a regular condition type. We say $\mathfrak{A}$ is *extendible* if for any arenas $\mathcal{A}$ and $\mathcal{A}'$ such that $V(\mathcal{A}) \subseteq V(\mathcal{A}')$, and any instance $\Omega \in \mathfrak{A}(\mathcal{A})$, there is an instance $\Omega' \in \mathfrak{A}(\mathcal{A}')$, computable in time polynomial in $|\Omega| + |V(\mathcal{A}')|$, such that $\mathcal{F}_{\Omega'} = \{I \subseteq V(\mathcal{A}') : I \cap V(\mathcal{A}) \in \mathcal{F}_\Omega\}$.

We observe that if $|V(\mathcal{A}')| - |V(\mathcal{A})| = m$, then $|\mathcal{F}_{\Omega'}| = 2^m|\mathcal{F}_\Omega|$, so in particular, an explicit presentation is not extendible. However, all the other condition types we have so far considered are extendible.

Proposition 2.58. *The following condition types are extendible: Muller, circuit, Emerson-Lei, Zielonka tree/DAG, win-set, parity, Rabin, Streett, basis, and superset.*

Proof. Let us fix arenas $\mathcal{A}$ and $\mathcal{A}'$ such that $V(\mathcal{A}) \subseteq V(\mathcal{A}')$. We show for each condition type above how to compute the required instance Ω' from a given Ω. It follows from the definitions that for the circuit, Emerson-Lei, win-set, Rabin, Streett and superset conditions taking $\Omega' = \Omega$ suffices. So let us consider the other condition types.

Suppose $\Omega = (\chi, \mathcal{C})$ is a Muller condition instance with $\chi : V(\mathcal{A}) \to C$. We define $\Omega' = (\chi', \mathcal{C}')$ as follows. Let $C' = C \cup \{c\}$ where c is not an element of C. We define

$$\chi'(v) := \begin{cases} \chi(v) \text{ if } v \in V(\mathcal{A}) \\ c \text{ otherwise} \end{cases}$$

and we define $\mathcal{C}' := \mathcal{C} \cup \{I \cup \{c\} : I \in \mathcal{C}\}$. $(\chi', \mathcal{C}')$ is clearly computable in time polynomial in $|\Omega| + |V(\mathcal{A}')|$, and for every $I \subseteq V(\mathcal{A}')$ we have $\chi'(I) \in \mathcal{C}'$ if, and only if, $\chi(I \cap V(\mathcal{A})) \in \mathcal{C}$. Thus Ω' is as required.

Similarly, if $\Omega = (\chi, \mathsf{P})$ is a parity condition, we let $\mathsf{P}' = \mathsf{P} \cup \{p\}$ for some odd $p < \min\{\chi(v) : v \in V(\mathcal{A})\}$ and define $\chi'(v) = p$ for $v \notin V(\mathcal{A})$, and $\chi(v) = v$ otherwise. For any set $I \subseteq V(\mathcal{A}')$, if $I \cap V(\mathcal{A}) \neq \emptyset$ then $\max\{\chi'(v) : v \in I\} = \max\{\chi(v) : v \in I \cap V(\mathcal{A})\}$, so $I \in \mathcal{F}_{\Omega'}$ if, and only if, $I \cap V(\mathcal{A}) \in \mathcal{F}_\Omega$. Otherwise, if $I \cap V(\mathcal{A}) = \emptyset$, then $\min\{\chi'(v) : v \in I\} = p$, and as $\emptyset \notin \mathcal{F}_\Omega$ and p is odd, we have $I \notin \mathcal{F}_{\Omega'}$ and $I \cap V(\mathcal{A}) \notin \mathcal{F}_\Omega$. Thus Ω' is as required.

Given a Zielonka structure $\mathcal{Z}_{\mathcal{F},V}$ where $V = V(\mathcal{A})$, consider the Zielonka structure $\Omega' = \mathcal{Z}_{\mathcal{F}',V'}$, where $V' = V(\mathcal{A}')$, defined by adding $V(\mathcal{A}') \setminus V(\mathcal{A})$ to each label. That is, if t is a node in $\mathcal{Z}_{\mathcal{F},V}$, labelled by $X \subseteq V$, then t is a node in $\mathcal{Z}_{\mathcal{F}',V'}$ labelled by $X \cup (V(\mathcal{A}') \setminus V(\mathcal{A}))$. Now consider $I \in \mathcal{F}'$. From the definition of a Zielonka structure, I is a subset of a label of a 0-level node t and not a subset of a label of any of the successors of t. Suppose t is labelled, in $\mathcal{Z}_{\mathcal{F},V}$, by X, so $I \subseteq X \cup (V' \setminus V)$. Thus $I \cap V(\mathcal{A}) \subseteq X$. Now suppose $I \cap V(\mathcal{A})$ is a subset of Y, a label (in $\mathcal{Z}_{\mathcal{F},V}$) of a successor of t. It follows that $I \subseteq Y \cup (V' \setminus V)$, and so I is a subset of a label (in $\mathcal{Z}_{\mathcal{F}',V'}$) of a successor of t, contradicting the choice of t. So $I \cap V(\mathcal{A}) \in \mathcal{F}$. Interchanging the roles of 0-level nodes and 1-level nodes establishes that if $I \notin \mathcal{F}'$ then $I \cap V(\mathcal{A}) \notin \mathcal{F}$. Thus Ω' is as required.

Finally, given an instance of a basis condition type $\Omega = \mathcal{B}$, we define $\Omega' = \mathcal{B}'$ as follows:

$$\mathcal{B}' = \mathcal{B} \cup \big\{\{v\} : v \in V(\mathcal{A}') \setminus V(\mathcal{A})\big\}.$$

Suppose $I = \bigcup_{i=1}^{n} B_i$ for sets $B_1, \ldots, B_n \in \mathcal{B}'$, where for some $m \leq n$, $B_i \in \mathcal{B}$ for $i \leq m$. From the definition of $\mathcal{B}'$, it follows that $I \cap V(\mathcal{A}) = \bigcup_{i=1}^{m} B_i$, so $I \cap V(\mathcal{A}) \in \mathcal{F}_\Omega$. Conversely, if $I \cap V(\mathcal{A}) \in \mathcal{F}_\Omega$, let $I \cap V(\mathcal{A}) = \bigcup_{i=1}^{m} B_i$. From the definition of $\mathcal{B}'$, there exists $B_{m+1}, \ldots, B_n \in \mathcal{B}'$ such that $I \setminus V(\mathcal{A}) = \bigcup_{i=m+1}^{n} B_i$. So $I = \bigcup_{i=1}^{n} B_i$ for $B_1, \ldots, B_n \in \mathcal{B}'$ and hence $I \in \mathcal{F}_{\Omega'}$. □

Given a game with a winning condition specified by an extendible condition type, we can add vertices to the arena without significantly changing the size of the instance. This enables us to assume that the arena has a very simple structure.

Theorem 2.59. *Let $\mathfrak{A}$ be an extendible regular condition type and $\mathbb{G} = (\mathcal{A}, \Omega)$ be a Muller game with $\Omega \in \mathfrak{A}(\mathcal{A})$. Then there exists a Muller game $(\mathcal{A}', \Omega')$ with $\Omega' \in \mathfrak{A}(\mathcal{A}')$, computable in time polynomial in $||\mathbb{G}||$, such that:*

(i) $\mathcal{A}'$ is a bipartite graph with $E(\mathcal{A}') \subseteq (V_0(\mathcal{A}') \times V_1(\mathcal{A}')) \cup (V_1(\mathcal{A}') \times V_0(\mathcal{A}'))$,

(ii) All vertices in $V_0(\mathcal{A}')$ have out-degree at most 2, and

(iii) Player 0 wins $\mathbb{G}$ if, and only if, she wins $\mathbb{G}'$.

Proof. We construct $\mathcal{A}'$ from $\mathcal{A}$ in a series of stages by adding vertices and adding and replacing edges, so $V(\mathcal{A}) \subseteq V(\mathcal{A}')$. We observe that the resulting arena has size polynomial in $|\mathcal{A}|$, so it can be constructed in polynomial time. We then use the definition of extendible condition type to obtain the winning condition Ω' from Ω. Since the size of $\mathcal{A}'$ is polynomial in the size of $\mathcal{A}$, we can compute Ω' in time polynomial in $|\Omega| + |\mathcal{A}|$. It is clear from the definition of extendible condition types that in the resulting game Player 0 wins from $v_I(\mathcal{A})$ if, and only if, she wins from $v_I(\mathcal{A}')$. Thus it remains to show the first two conditions may be met with at most a polynomial increase in the size of the arena.

First we ensure all vertices in $V_0(\mathcal{A}')$ have out-degree at most 2. If $v \in V_0(\mathcal{A})$ has out-degree $m > 2$, we replace the m outgoing edges from v with a binary branching tree, rooted at v, with m leaves – the successors of v. We observe that this requires adding at most m vertices and m edges. Each of the newly added vertices are added to $V_1(\mathcal{A})$. After repeating this for all vertices in $V_0(\mathcal{A})$, the resulting arena $\mathcal{A}'$ has at most $|V(\mathcal{A})| + |E(\mathcal{A})|$ vertices, and $2|E(\mathcal{A})|$ edges, and every vertex in $V_0(\mathcal{A}')$ has out-degree at most 2.

Now suppose all vertices in $\mathcal{A}$ have out-degree at most 2. For each edge $e = (u, v) \in E(\mathcal{A})$ such that $u, v \in V_0(\mathcal{A})$ ($u, v \in V_1(\mathcal{A})$), add a vertex v_e to $V_1(\mathcal{A})$ ($V_0(\mathcal{A})$) and replace the edge e with edges (u, v_e) and (v_e, v). After repeating this for all edges in $E(\mathcal{A})$, the resulting arena $\mathcal{A}'$ has at most $|V(\mathcal{A})| + |E(\mathcal{A})|$ vertices, and $2|E(\mathcal{A})|$ edges, and $E(\mathcal{A}') \subseteq V_0(\mathcal{A}') \times V_1(\mathcal{A}') \cup V_1(\mathcal{A}') \times V_0(\mathcal{A}')$. □

2.3 Complexity results

In this section we consider the complexity of deciding whether Player 0 has a winning strategy in a Muller game when the winning condition is specified using some of the formalisms we

have considered. We show that the problem of deciding Muller games in which the winning condition is specified by a win-set condition is PSPACE-complete. It follows from our results on translatability that the decision problems for Muller games with winning condition specified by a Muller condition, Zielonka DAG or an Emerson-Lei condition are all also PSPACE-complete. We also show that the decision problems for basis and superset games are co-NP-complete.

We first consider some upper bounds. A well-known result is that simple games can be decided in linear time.

Theorem 2.60. *Let $\mathbb{G} = (\mathcal{A}, \mathcal{F})$ be a simple game. Whether Player 0 wins $\mathbb{G}$ can be decided in time $O(|E(\mathcal{A})|)$.*

Proof. Suppose $\mathcal{F} = \emptyset$, the case when $\mathcal{F} = \mathcal{P}(V(\mathcal{A}))$ is dual. Let $W \subseteq V_1(\mathcal{A})$ be the set of vertices in $V_1(\mathcal{A})$ with no outgoing edges. We observe that Player 0 wins from $v_I(\mathcal{A})$ if, and only if, Player 0 can force the play to a vertex $v \in W$. Thus, Player 0 has a winning strategy if, and only if, $v_I(\mathcal{A}) \in \mathit{Force}^0_{\mathcal{A}}(W)$. The required complexity bound then follows from Lemma 2.18. □

In [IK02], Ishihara and Khoussainov considered the following restriction on explicitly presented Muller games:

Definition 2.61 (Fully Separated game)**.** Let $\mathbb{G} = (\mathcal{A}, \mathcal{F})$ be an explicitly presented Muller game. We say $\mathbb{G}$ is *fully separated* if for each $X \in \mathcal{F}$ there exists $v_X \in X$ such that $v_X \notin Y$ for all $Y \in \mathcal{F}, Y \neq X$.

Khoussainov showed that the winner of a fully separated game can be decided in time $O(|V(\mathcal{A})|^2|E(\mathcal{A})|)$. We now prove a generalization of this result by showing that explicitly presented Muller games can be decided in polynomial time if the winning condition is an anti-chain with respect to the subset relation.

Theorem 2.62. *Let $\mathbb{G} = (\mathcal{A}, \mathcal{F})$ be an explicitly presented Muller game such that $\mathcal{F}$ is an anti-chain, that is, $X \not\subseteq Y$ for all $X, Y \in \mathcal{F}$. Whether Player 0 wins $\mathbb{G}$ can be decided in time $O(|\mathcal{F}||V(\mathcal{A})|^2|E(\mathcal{A})|)$.*

Proof. Consider the algorithm ANTICHAIN$(\mathcal{A}, \mathcal{F})$ in Algorithm 2.2. We show that it is correct and returns in time $O(|\mathcal{F}||V(\mathcal{A})|^2|E(\mathcal{A})|)$.

We first show that ANTICHAIN$(\mathcal{A}, \mathcal{F})$ returns true if, and only if, Player 0 has a winning strategy in $\mathbb{G} = (\mathcal{A}, \mathcal{F})$. Let us suppose N has been computed as above. We consider three cases:

(i) $v_I(\mathcal{A}) \in N$. From the definition of N, there exists $v \in V(\mathcal{A})$ and $X \in \mathcal{F}$ such that Player 0 can force the play to v from $v_I(\mathcal{A})$ and Player 0 has a winning strategy from v which visits every vertex in X, and only vertices in X, infinitely often. The winning strategy for Player 0 is then to force the play to v and play this strategy. Since $X \in \mathcal{F}$, this is a winning strategy.

(ii) $N = \emptyset$. In this case, for every $X \in \mathcal{F}$, Player 1 has a strategy τ_X from every vertex in $\mathcal{A}$ which can ensure either not all vertices of X are visited infinitely often, or some vertices not in X are visited infinitely often. The strategy for Player 1 on $(\mathcal{A}, \mathcal{F})$ is as follows. Play anything until the play enters some $X \in \mathcal{F}$, then play the strategy τ_X until the play leaves X. Clearly if there is no $X \in \mathcal{F}$ such that the play remains forever in X, Player 1

Algorithm 2.2 ANTICHAIN$(\mathcal{A}, \mathcal{F})$

Returns: true if, and only if, Player 0 has a winning strategy from $v_I(\mathcal{A})$ in $(\mathcal{A}, \mathcal{F})$ when $\mathcal{F}$ is an anti-chain.

```
for each X ∈ F do
    let N_X = {v : Player 0 has a winning strategy from v in the game (A, {X})}
let N = Force^0_A(⋃_{X∈F} N_X)
if v_I(A) ∈ N then
    return true
else if N = ∅ then
    return false
else
    let F' = {X ∈ F : X ∩ N = ∅}
    return ANTICHAIN(A \ N, F')
```

wins the play. So let us suppose the play remains indefinitely in X for some $X \in \mathcal{F}$. From the definition of τ_X, the set I of vertices visited infinitely often is properly contained in X. Since $\mathcal{F}$ is an anti-chain, it follows that $I \notin \mathcal{F}$. Thus Player 1 wins the play.

(iii) $N \neq \emptyset$ and $v_I(\mathcal{A}) \notin N$. In this case, Player 1 can force the play to remain in $\mathcal{A} \setminus N$ and it follows from case (i) above that Player 0 has a winning strategy from every vertex in N. Clearly, if Player 0 has a winning strategy in $(\mathcal{A} \setminus N, \mathcal{F}')$ then she has a winning strategy in the larger game: if Player 1 chooses to keep the play in $\mathcal{A} \setminus N$ then Player 0 can play her winning strategy on the subgame, otherwise if Player 1 chooses to move to a vertex in N, Player 0 can play her winning strategy from N. Conversely, if Player 1 has a winning strategy in $(\mathcal{A} \setminus N, \mathcal{F}')$ then, as he can force the play to remain in $\mathcal{A} \setminus N$, he can play his winning strategy on the subgame.

Thus, ANTICHAIN$(\mathcal{A}, \mathcal{F})$ returns true if, and only if, Player 0 has a winning strategy in $\mathbb{G} = (\mathcal{A}, \mathcal{F})$.

To show the algorithm returns in time $O(|\mathcal{F}||V(\mathcal{A})|^2|E(\mathcal{A})|)$, we require the following result from [IK02]:

Lemma 2.63 ([IK02])**.** *Let $\mathbb{G} = (\mathcal{A}, \mathcal{F})$ be an explicitly presented Muller game with $\mathcal{F} = \{X\}$. Whether Player 0 has a winning strategy from a vertex $v \in V(\mathcal{A})$ can be decided in time $O(|V(\mathcal{A})||E(\mathcal{A})|)$.*

It follows that at each stage of the recursion, it takes $O(|\mathcal{F}||V(\mathcal{A})||E(\mathcal{A})|)$ time to compute N. Furthermore, since $|N| \geq 1$ whenever ANTICHAIN$(\mathcal{A}, \mathcal{F})$ is recursively called, it follows that the algorithm has recursion depth at most $|V(\mathcal{A})|$. Thus the algorithm runs in time $O(|\mathcal{F}||V(\mathcal{A})|^2|E(\mathcal{A})|)$ as required. □

2.3.1 PSPACE-completeness

As we saw in Theorem 2.15, McNaughton [McN93] presented an algorithm for deciding Muller games in space $O(|V(\mathcal{A})|^2)$. In fact, the games he considered were win-set games. However, the algorithm is easily adapted to the case where the winning condition is presented explicitly, or as a Muller condition, a Zielonka DAG, an Emerson-Lei condition, or a circuit condition

without significant increase in the space requirements. Thus, each of these classes of games is decidable in PSPACE.

We now show corresponding lower bounds. By the results of the previous section, it suffices to establish the hardness result for the win-set condition type.

Theorem 2.64. *Deciding win-set games is* PSPACE*-complete.*

Proof. By the above comments, we only need to show PSPACE-hardness. For this, we reduce the problem of QSAT (satisfiability of a quantified boolean formula [QBF]) to the problem of deciding the winner of a win-set game.

We assume, without loss of generality a QBF, $\Phi = Q_{k-1}x_{k-1} \ldots \forall x_1 \exists x_0 \varphi$ is given in which quantifiers are strictly alternating and φ is in disjunctive normal form with 3 literals per clause. We then define a win-set game $\mathbb{G}_\Phi = (\mathcal{A}, \Omega)$, where $\Omega = (W, \mathcal{W})$, as follows:

- $V_0(\mathcal{A}) = \{\varphi\} \cup \{x, \neg x : \text{for all variables } x\}$,
- $V_1(\mathcal{A}) = \{C_0, \ldots, C_{m-1}\}$, the set of clauses in φ,
- $E(\mathcal{A})$ given by:
  - $(\varphi, C_j) \in E(\mathcal{A})$ for $0 \leq j < m$;
  - If $C_j = (l_0 \wedge l_1 \wedge l_2)$, then $(C_j, l_0), (C_j, l_1), (C_j, l_2) \in E(\mathcal{A})$;
  - $(x_i, x_{i-1}), (x_i, \neg x_{i-1}) \in E(\mathcal{A})$ for $0 < i < k$;
  - $(\neg x_i, x_{i-1}), (\neg x_i, \neg x_{i-1}) \in E(\mathcal{A})$ for $0 < i < k$; and
  - $(x_0, \varphi), (\neg x_0, \varphi) \in E(\mathcal{A})$,
- $v_I(\mathcal{A}) = \varphi$,
- $W = V_0(\mathcal{A}) \setminus \{\varphi\}$, and $\mathcal{W}$ is

$$\mathcal{W} = \big\{S_i, S_i \cup \{x_i\}, S_i \cup \{\neg x_i\} : 0 \leq i < k,\ i \text{ even}\big\}$$

  where $S_0 = \emptyset$ and for $i > 0$, $S_i = \{x_j, \neg x_j : 0 \leq j < i\}$.

Figure 2.3 illustrates how the arena of $\mathbb{G}_\Phi$ would look if φ contained the clauses $(x_0 \wedge x_{k-1} \wedge \neg x_k)$ and $(\neg x_0 \wedge x_{k-1} \wedge x_k)$.

Note that as this is a win-set game, we are only interested in vertices of W that are visited infinitely often. Observe that the winning condition ensures that Player 0 can win if, and only if, the minimum i such that at most one of x_i and $\neg x_i$ is visited infinitely often is even. The idea behind the strategy for Player 0 is to perpetually verify φ. The choice of strategies by both players then dictates the choices of the truth values for each of the variables, and the winning condition guarantees a winning strategy for Player 0 if, and only if, Φ is true. To formally show that Player 0 has a winning strategy if, and only if, Φ is true, we proceed by induction on k, the number of quantifiers of Φ.

Base case: $k = 1$ By the idempotence of $\wedge$ and $\vee$ and assuming Φ is closed, Φ is logically equivalent to one of the following forms.

- $\Phi = \exists x_0.x_0$ or $\exists x_0.\neg x_0$. In this case the arena consists of four vertices, $\{\varphi, C_0, x_0, \neg x_0\}$. Player 0 wins by always returning to φ from whichever of x_0 and $\neg x_0$ Player 1 is forced to play to, and Φ is clearly true.

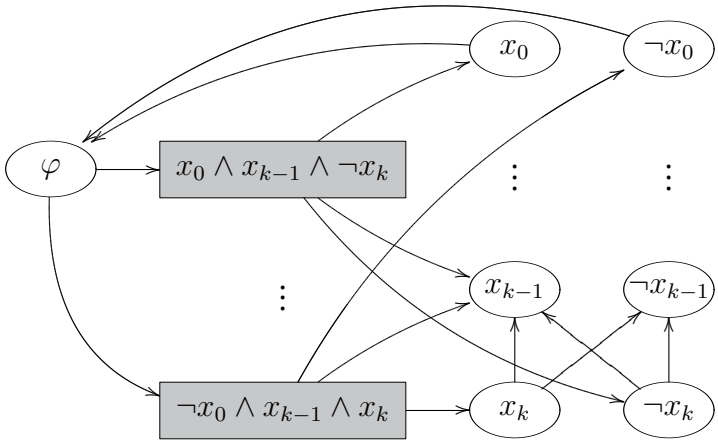


Figure 2.3: Arena of $\mathbb{G}_\Phi$ for $\varphi = (x_0 \wedge x_{k-1} \wedge \neg x_k) \vee \ldots \vee (\neg x_0 \wedge x_{k-1} \wedge x_k)$

- $\Phi = \exists x_0.(x_0 \vee \neg x_0)$. Here Φ is also true. The arena consists of five vertices $\{\varphi, C_0, C_1, x_0, \neg x_0\}$ and Player 0 has the only choice (at φ and x_0). A winning strategy is to always play from φ to C_0, and to return immediately to φ from x_0.
- $\Phi = \exists x_0.(x_0 \wedge \neg x_0)$. Here Φ is false. The arena consists of four vertices $\{\varphi, C_0, x_0, \neg x_0\}$ and Player 1 can force the play to visit both x_0 and $\neg x_0$ infinitely often by alternately choosing each from C_0. Note that this strategy requires memory to remember which vertex was visited last time.

Note that if x_0 does not appear in φ, we can add the clause $(x_0 \wedge \neg x_0)$ without changing the truth value of Φ.

Inductive case: The inductive hypothesis asserts that if Φ has $k-1$ quantifiers and is closed, then Player 0 has a winning strategy if, and only if, Φ is true. To show that this implies the case for k quantifiers, we use the following lemma which shows how subgames correspond to restricted subformulas. First we introduce some notation. If x is free in φ and v is either true or false, we write $\varphi[x \mapsto \mathsf{v}]$ to denote the formula obtained by substituting v for x in φ and simplifying. Note that if $\varphi[x \mapsto \mathsf{true}]$ simplifies to true then φ must have at least one clause containing the single literal x, and if it simplifies to false, then all clauses contain $\neg x$. The crucial lemma can now be stated as

Lemma 2.65. *If $\Phi = Qx\varphi$ ($Q \in \{\exists, \forall\}$) and $\varphi[x \mapsto \mathsf{true}]$ does not simplify to true or false, then $\mathbb{G}_{\varphi[x \mapsto \mathsf{true}]}$ is isomorphic to the subgame of $\mathbb{G}_\Phi = (\mathcal{A}, \Omega)$ induced by the set $\mathit{Avoid}^1_{\mathit{Avoid}^0_{\mathcal{A}}(\neg x)}(x)$. Dually, if $\varphi[x \mapsto \mathsf{false}]$ does not simplify to true or false, then $\mathbb{G}_{\varphi[x \mapsto \mathsf{false}]}$ is isomorphic to the subgame of $\mathbb{G}_\Phi$ induced by the set $\mathit{Avoid}^1_{\mathit{Avoid}^0_{\mathcal{A}}(x)}(\neg x)$.*

Proof. $\varphi[x \mapsto \mathsf{true}]$ consists of the clauses of φ that do not contain $\neg x$, with all occurrences of x removed. The assumption that $\varphi[x \mapsto \mathsf{true}]$ does not simplify to true or false implies that there is at least one such clause. The arena for the game $\mathbb{G}_{\varphi[x \mapsto \mathsf{true}]}$ thus consists of vertices for $\varphi[x \mapsto \mathsf{true}]$, the clauses, and the variables (and their negations) of φ, excluding x and $\neg x$. The edges are the same as those for $\mathbb{G}_\Phi$ restricted to this vertex set. We show that the subarena of $\mathbb{G}_\Phi$ induced by $\mathit{Avoid}^1_{\mathit{Avoid}^0_{\mathcal{A}}(\neg x)}(x)$ is identical. As the winning condition only depends on vertices corresponding to variables, it follows that the winning conditions are also identical.

In $\mathbb{G}_\Phi = (\mathcal{A}, \Omega)$, the set $\mathit{Avoid}^0_{\mathcal{A}}(\neg x)$ consists of the vertices from which Player 0 can avoid $\neg x$. As Player 1 chooses the play from vertices corresponding to clauses, the set of vertices

from which Player 1 can reach $\neg x$ is $\{\neg x\} \cup \{C : \neg x \in C\}$. As there is at least one clause that does not contain $\neg x$, Player 0 can play to that clause to avoid $\neg x$ from φ. The only other vertex from which it is possible to reach $\neg x$ is x (as x is the outermost variable in Φ), and from there Player 0 can play to either y (for the next outermost variable y) or φ (if no such variable exists). Thus

$$\mathit{Avoid}^0_{\mathcal{A}}(\neg x) = V(\mathcal{A}) \setminus \big(\{\neg x\} \cup \{C : \neg x \in C\}\big).$$

Next we consider $\mathit{Avoid}^1_{V'}(x)$ for $V' = \mathit{Avoid}^0_{\mathcal{A}}(\neg x)$. As φ does not contain a clause containing x by itself, Player 0 cannot force the play to x from φ, as Player 1 can always choose to play to another literal. Furthermore, as x is the outermost variable in Φ, the only edges to x are from vertices associated with clauses. Thus x is the only vertex from which Player 0 can force the play to visit x, so

$$\mathit{Avoid}^1_{V'}(x) = V' \setminus \{x\}.$$

Thus $\mathit{Avoid}^1_{\mathit{Avoid}^0_{\mathcal{A}}(\neg x)}(x) = V(\mathcal{A}) \setminus \big(\{x, \neg x\} \cup \{C : \neg x \in C\}\big)$, which is precisely the vertex set of $\mathbb{G}_{\varphi[x \mapsto \mathsf{true}]}$. The edges for both arenas are those of $\mathbb{G}_\Phi$ restricted to these vertices, as are the winning conditions. Thus the two games are identical. ⊣

To complete the inductive step, we consider two cases.

- $\Phi = \exists x_{k-1}.\varphi$. If Φ is true, then there is a truth value v such that $\varphi[x_{k-1} \mapsto \mathsf{v}]$ is true. Assume that $\mathsf{v} = \mathsf{true}$, the case for $\mathsf{v} = \mathsf{false}$ being similar. The winning strategy for Player 0 is then to avoid $\neg x_{k-1}$ and try to play to x_{k-1}, playing through each vertex in S_{k-1} when the latter vertex is reached. Note that to play through each vertex in S_{k-1} requires at least two visits to x_{k-1} – Player 0 must remember (the parity of) the number of times she has visited that vertex. If $\varphi[x_{k-1} \mapsto \mathsf{v}]$ simplifies to true, then Player 0 can force the play to visit x_{k-1}, by playing to the clause that only contains x_{k-1}. Otherwise Player 1 can play to avoid x_{k-1}, restricting the play to $\mathit{Avoid}^1_{\mathit{Avoid}^0_{\mathcal{A}}(\neg x_{k-1})}(x_{k-1})$. From the above lemma, this subgame is equivalent to $\mathbb{G}_{\varphi[x_{k-1} \mapsto \mathsf{true}]}$, and from the inductive hypothesis, Player 0 has a winning strategy on this game. Thus the strategy of Player 0 is to play her winning strategy on the smaller game. If Φ is false, then Player 1 plays a strategy similar to the strategy of Player 0 in the case below.

- $\Phi = \forall x_{k-1}.\varphi$. In this case, if Φ is true, then for both choices of truth value $\mathsf{v} \in \{\mathsf{true}, \mathsf{false}\}$, $\varphi[x_{k-1} \mapsto \mathsf{v}]$ is true. The winning strategy for Player 0 is to alternately attempt to play to each of x_{k-1} and $\neg x_{k-1}$ (and then through all vertices in S_{k-1}), avoiding the other at the same time. If, at any point, Player 1 plays to avoid the vertex Player 0 is attempting to reach, Player 0 plays her winning strategy on the reduced game (which exists from the lemma and the inductive hypothesis). Again, if Φ is false, Player 1 plays a strategy similar to the strategy of Player 0 in the previous case. Note that in this case Player 0 cannot force the play to visit both x_{k-1} and $\neg x_{k-1}$.

□

From our work on translatability in Section 2.2 and our observation regarding the PSPACE solvability of these games, we obtain completeness results for Muller games when the winning condition is presented as a Muller condition, Zielonka DAG, Emerson-Lei condition or a circuit condition.

Corollary 2.66. *The following problems are* PSPACE*-complete: Deciding Muller games with winning condition specified by a Muller condition, deciding Zielonka DAG games, deciding Emerson-Lei games, and deciding circuit games.*

It can be verified that an explicit presentation of the winning condition constructed in the proof of Theorem 2.64 would be exponentially larger than the presentation using a win-set. Thus, the proof cannot be used to provide a PSPACE-hardness result for the explicitly presented games. The exact complexity of deciding the winner of such games remains open. Indeed, it is conceivable (though it appears unlikely) that the problem is in PTIME.

Open problem 2.67. *Determine the precise complexity of deciding explicitly presented Muller games.*

Bounded tree-width arenas

In Chapter 4 we present a graph parameter known as *tree-width*. Tree-width is a measure of how closely a graph resembles a tree. It has proved useful in the design of algorithms as many problems that are intractable on general graphs are known to have polynomial time solutions when restricted to graphs of bounded tree-width. In the context of Muller games, Obdržálek [Obd03] exhibited a polynomial-time algorithm for deciding the winner in parity games on arenas of bounded tree-width. We show that this is not the case for Muller games (and neither, therefore, for Zielonka DAG games, Emerson-Lei games, and circuit games). The proof of Theorem 2.64 can be modified so that the arenas constructed all have tree-width two provided we allow ourselves to specify the winning condition as a Muller condition rather than a win-set.

Theorem 2.68. *Deciding Muller games specified by a Muller condition on arenas of tree-width 2 is* PSPACE*-complete.*

Proof. Membership of PSPACE follows from the fact that deciding general Muller games specified by a Muller condition is in PSPACE.

The construction to show PSPACE-hardness is similar to that of Theorem 2.64. The reduction is also from QSAT, and the proof that it is in fact a reduction is similar. Given a QBF $\Phi = Q_{k-1}x_{k-1} \ldots \forall x_1 \exists x_0 \varphi$ where φ is in DNF with three literals per clause, the Muller game we construct is:

- $V_1(\mathcal{A}) = D$ where D is the set of clauses.
- $V_0(\mathcal{A}) = \{\varphi\} \cup \big(D \times \{1,2,3\} \times \{x, \neg x : x \text{ is a variable}\}\big)$.
- We have the following edges in $E(\mathcal{A})$ for all $c \in D$:
  - (φ, c),
  - $\big(c, (c,n,l)\big)$ if l is the n-th literal in c,
  - $\big((c,n,x_i), (c,n,x_{i-1})\big)$ if the n-th literal of c is x_i ($i > 0$)
  - $\big((c,n,x_0), \varphi\big)$ if the n-th literal of c is x_0
  - $\big((c,n,x_i), (c,n,\neg x_i)\big)$ for all i less than the index of the n-th literal of c
  - $\big((c,n,\neg x_i), (c,n,x_{i-1})\big)$ for all i less than or equal to the index of the n-th literal of c

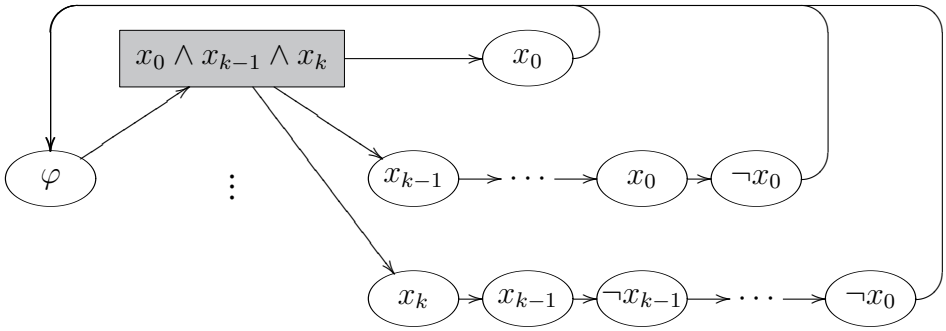


Figure 2.4: Arena with bounded tree-width

- $\big((c, n, \neg x_0), \varphi\big)$ for all n.

- $C = \{\varphi\} \cup \{x, \neg x : x \text{ is a variable}\}$ is the set of colours,

- $\chi : V(\mathcal{A}) \to C$ defined as:
  - $\chi(\varphi) = \chi(c) = \varphi$ for all $c \in D$
  - $\chi\big((c, n, l)\big) = l$.

- $\mathcal{C} = \big\{S_i, S_i \cup \{x_i\}, S_i \cup \{\neg x_i\} : 0 \leq i < k, \, i \text{ even}\big\}$ where $S_0 = \{\varphi\}$ and for $i > 0$, $S_i = \{\varphi\} \cup \{x_j, \neg x_j : 0 \leq j < i\}$.

Figure 2.4 illustrates how this arena differs from that of Theorem 2.64.

The resulting arena has tree-width 2, and the proof that Player 0 has a winning strategy if, and only if, Φ is true is similar to that of Theorem 2.64. □

2.3.2 Complexity of union-closed games

We now turn our attention to Muller games where the winning condition $\mathcal{F}$ is a union-closed set. Among games studied in the literature, Streett games and parity games are examples of condition types that can only specify union-closed games. Union-closed games were also studied as a class in [IK02]. One consideration that makes them an interesting case to study is that they admit memoryless strategies for Player 1 [Kla94]. That is, on a game with a union-closed winning condition, if Player 1 has a winning strategy then he has a strategy which is a function only of the current position. One consequence of this fact is that, for explicitly presented union-closed games, the problem of deciding whether Player 0 wins such a game is in co-NP. This is because once a memoryless strategy for Player 1 is fixed, the problem of deciding whether Player 0 wins against that fixed strategy is in PTIME. Indeed, it is a version of a simple game. Thus, to decide whether Player 1 has a winning strategy we can nondeterministically guess such a strategy and then verify that Player 0 cannot defeat it. Hence, determining whether Player 1 wins is in NP and therefore deciding whether Player 0 wins is in co-NP. In this section, we aim to establish a corresponding lower bound for two condition types that can only represent union-closed games, namely the basis and superset condition types.

We saw with Theorem 2.53 that we cannot use the known complexity bounds on Streett games to easily establish similar bounds for basis games. Nevertheless, deciding basis games is still in co-NP.

Proposition 2.69. *Deciding basis games is in* co-NP.

Proof. From the comments above, it suffices to show that if we fix a memoryless strategy for Player 1 then we can decide the resulting single player basis game in polynomial time.

The algorithm is as follows. Let $\mathcal{B}$ be the basis for the winning condition. Initially let $\mathcal{B}_0 = \mathcal{B}$, and repeat the following:

1. Let $X_i = \bigcup_{B \in \mathcal{B}_i} B$.
2. Partition X_i into strongly connected components (SCCs).
3. Remove any element of $\mathcal{B}_i$ which is not wholly contained in a SCC to obtain $\mathcal{B}_{i+1}$,

until $\mathcal{B}_i = \mathcal{B}_{i-1}$, at which point, let $X = X_i$. This takes at most $O\big(|\mathcal{B}|(|V(\mathcal{A})| + |E(\mathcal{A})|)\big)$ time using a standard SCC-partitioning algorithm. At this point, every SCC of X is a union of basis elements – all x in X are members of basis elements, and any basis elements not contained in any SCC of X is removed at step 3. Furthermore, any strongly connected set of $V(\mathcal{A})$ which is a union of basis elements is a subset (of an SCC) of X, because the algorithm preserves such sets. Thus, Player 0 can win from any node from which she can reach X (play to X and then visit every node within an SCC of X forever); and Player 0 cannot win if she cannot reach X (there is no union of basis elements for which Player 0 can visit every vertex infinitely often). Thus the set of nodes from which Player 0 wins can be computed in $O\big(|\mathcal{B}|(|V| + |E|) + |E|\big)$ time. □

We now obtain the lower bounds we seek on superset games.

Theorem 2.70. *Deciding superset games is* co-NP*-complete.*

Proof. Membership of co-NP follows from Propositions 2.54 and 2.69. To show co-NP-hardness, we use a reduction from validity of DNF formulas.

Given a formula $\varphi(x_0, x_1, \ldots, x_{k-1})$ in DNF, consider the superset game defined as follows:

- for every variable x_i we include three vertices, $x_i, \neg x_i \in V_0(\mathcal{A})$ and $x'_i \in V_1(\mathcal{A})$;
- for each i we have the edges $(x'_i, x_i), (x'_i, \neg x_i), (x_i, x'_{i+1}), (\neg x_i, x'_{i+1})$, where addition is taken modulo k;
- $v_I(\mathcal{A}) = x_0$; and
- the winning condition is specified by the set

$$\mathcal{M} = \big\{\{l_i \in V_0(\mathcal{A}) : l_i \text{ is a literal of } C\} \text{ for every clause } C \text{ of } \varphi\big\},$$

As the superset condition is closed under union, if Player 1 has a winning strategy he has a memoryless winning strategy. Note that any memoryless strategy for Player 1 effectively chooses a truth value for each variable. The set of vertices visited infinitely often is a superset of an element of $\mathcal{M}$ if, and only if, the truth assignment chosen by Player 1 makes one clause of φ (and hence φ) true. Thus Player 0 wins this game if, and only if, there is no truth assignment which makes φ false. □

Corollary 2.71. *Deciding basis games is* co-NP*-complete.*

We note in conclusion that the exact complexity of deciding union-closed games when they are explicitly presented remains an open problem. It is clearly in co-NP but the above arguments do not establish lower bounds for it.

Open problem 2.72. *Determine the precise complexity of deciding explicitly presented union-closed games.*

2.4 Infinite tree automata

One of the original motivations for studying Muller and related games was to establish decidability results for problems such as non-emptiness and model checking for infinite tree automata [McN66]. A reduction to non-emptiness of infinite tree automata is used in some of the most effective algorithms for deciding satisfiability of formulas in logics such as $S2S$, μ-calculus, CTL*, and other logics useful for reasoning about non-terminating, branching computation. Furthermore, determining if a structure satisfies a formula in any of these logics reduces to determining if a certain automaton accepts a particular tree. In this section we show that the non-emptiness and model-checking problems (for regular trees) are PSPACE-complete for Muller automata. We first present the definitions of infinite trees and infinite tree automata.

Definition 2.73 (Infinite tree)**.** For $k \in \mathbb{N}$, let $[k] = \{1, 2, \ldots, k\}$. An infinite, k-ary branching tree labelled by elements of Σ is a function $t : [k]^* \to \Sigma$. *Nodes* of an infinite tree are elements of its domain, the *root* of an infinite tree is the empty string.

Definition 2.74 (Regular tree)**.** A *subtree* of tree t rooted at $u \in [k]^*$ is the tree t_u defined as $t_u(v) = t(u \cdot v)$ for all $v \in [k]^*$. A tree t is *regular* if it has finitely many distinct subtrees, or equivalently, if there are finitely many equivalence classes under the equivalence relation

$$u \sim v \iff t(u \cdot w) = t(v \cdot w) \quad \forall w \in [k]^*.$$

Note that if a tree is regular it can be represented by a finite transition system, with the equivalence classes of the above equivalence relation as states, the equivalence class containing the root as the initial vertex, and k distinct transition relations.

Definition 2.75 (Infinite tree automaton)**.** An infinite (Muller) (k-ary) tree automaton is a tuple $\mathbb{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$ where

- Q is a finite set of states
- Σ is a finite alphabet
- $\delta \subseteq Q \times \Sigma \times Q^k$ is a transition relation
- q_0 is the initial state
- $\mathcal{F} \subseteq \mathcal{P}(Q)$ is the acceptance condition.

Given an infinite, k-ary branching tree t labelled by elements of Σ, a run of $\mathbb{A}$ on t is an infinite, k-ary branching tree r labelled by elements of Q satisfying the following two conditions.

- The root of r is labelled by q_0 ($r(\epsilon) = q_0$).

- For all $w \in [k]^*$, if $r(w) = q$, $r(w \cdot 1) = q_1$, $r(w \cdot 2) = q_2$, ..., $r(w \cdot k) = q_k$, and $t(w) = a$, then $(q, a, q_1, q_2, \ldots, q_k) \in \delta$.

We say a run r is successful if for every (infinite) path, the set I of states visited infinitely often is an element of $\mathcal{F}$. We say $\mathbb{A}$ *accepts* t if there is a successful run of $\mathbb{A}$ on t. Given an automaton $\mathbb{A}$, the *language* of $\mathbb{A}$ is the set of trees

$$\mathcal{L}(\mathbb{A}) := \{t : \mathbb{A} \text{ accepts } t\}.$$

Two important decision problems in automata theory are non-emptiness and model-checking.

NON-EMPTINESS OF MULLER TREE AUTOMATA	
Instance:	A Muller automaton $\mathbb{A}$
Problem:	Is $\mathcal{L}(\mathbb{A}) \neq \emptyset$?

MODEL-CHECKING FOR MULLER TREE AUTOMATA	
Instance:	A Muller automaton $\mathbb{A}$, and a regular infinite tree t
Problem:	Is $t \in \mathcal{L}(\mathbb{A})$?

The close connection between automata and games can be established by considering the game where the moves of Player 0 consist of choosing a transition in δ to make from a current state, and the moves of Player 1 consist of choosing which branch of the tree to descend. With this translation in mind, the non-emptiness problem reduces to the problem of finding the winner in the win-set game $\big(\mathcal{A}, (W, \mathcal{W})\big)$ with

- $V_0(\mathcal{A}) = W = Q$,
- $V_1(\mathcal{A}) = Q^k$,
- $\mathcal{W} = \mathcal{F}$,
- edges from $V_0(\mathcal{A})$ to $V_1(\mathcal{A})$ determined by δ: an edge from q to $(q_1, q_2, \ldots, q_k)$ if there is $a \in \Sigma$ such that $(q, a, q_1, \ldots q_k) \in \delta$, and
- edges from $V_1(\mathcal{A})$ to $V_0(\mathcal{A})$ being projections: an edge from $(q_1, \ldots, q_k)$ to q_i for all $i \in [k]$.

Clearly if Player 0 has a winning strategy in this game, it is possible to construct a tree which the automaton accepts. Conversely, if Player 1 has a winning strategy, no such tree exists.

By adapting the proof of Theorem 2.64 we are able to show that the non-emptiness problem for Muller automata as well as the problem of determining whether a given automaton accepts a given regular tree are both PSPACE-complete.

Theorem 2.76. *The non-emptiness problem for Muller tree automata is* PSPACE*-complete.*

Proof. Membership in PSPACE is established by the above polynomial time reduction from the non-emptiness problem of Muller automata to win-set games. Here we show PSPACE hardness through a reduction from QSAT (satisfiability of a quantified boolean formula [QBF]).

Given a QBF $\Phi = Q_{k-1}x_{k-1} \ldots \forall x_1 \exists x_0 \varphi$, where φ is in disjunctive normal form with 3 literals per clause, we construct the following Muller automaton $\mathbb{A}_\Phi = (Q, \Sigma, q_I, \delta, \mathcal{F})$ that accepts infinite ternary trees:

- $Q = \{q_\varphi\} \cup \{q_x, q_{\neg x} : \text{for all variables } x\}$
- $\Sigma = \{a\}$ [2]
- $q_I = q_\varphi$
- $\delta \subseteq Q \times Q^3$ given by:
  - for each clause $(l_0 \wedge l_1 \wedge l_2) \in \varphi$, $(q_\varphi, q_{l_0}, q_{l_1}, q_{l_2}) \in \delta$;
  - $(q_{x_i}, q_{x_{i-1}}, q_{x_{i-1}}, q_{x_{i-1}}) \in \delta$ for $0 < i < k$;
  - $(q_{\neg x_i}, q_{x_{i-1}}, q_{x_{i-1}}, q_{x_{i-1}}) \in \delta$ for $0 < i < k$;
  - $(q_{x_0}, q_\varphi, q_\varphi, q_\varphi) \in \delta$; and
  - $(q_{\neg x_0}, q_\varphi, q_\varphi, q_\varphi) \in \delta$.
- $\mathcal{F} = \big\{S_i, S_i \cup \{q_{x_i}\}, S_i \cup \{q_{\neg x_i}\} : 0 \leq i < k,\ i \text{ even}\big\}$ where $S_i = \{q_\varphi\} \cup \{q_{x_j}, q_{\neg x_j} : 0 \leq j < i\}$.

Now by using the reduction to win-set games outlined above, asking if $\mathbb{A}_\Phi$ accepts any tree is equivalent to asking if Player 0 has a winning strategy (from q_φ) on the win-set game used in Theorem 2.64. □

The model checking problem also reduces to deciding which player wins an infinite game. However, depending on how the tree is presented, the resulting arena may be of infinite size. If the tree is presented as a finite transition system, a game with finite arena can be constructed, and we can apply Theorem 2.76 to obtain the following corollary.

Corollary 2.77. *Given a regular, infinite, k-ary branching tree t (represented as a transition system) and a Muller automaton $\mathbb{A} = (Q, \Sigma, \delta, q_I, \mathcal{F})$, asking if $\mathbb{A}$ accepts t is* PSPACE*-complete.*

Proof. PSPACE hardness follows from the proof of Theorem 2.76, as the automata constructed there accept at most one tree – the ternary branching tree with all nodes labelled by a.

To show that the problem is in PSPACE, we reduce it to the problem of deciding a Muller game. Let $(S, s_0, t_1, \ldots, t_k)$ denote the transition system representing the tree t. The required Muller game, $\big(\mathcal{A}, (\chi, \mathcal{C})\big)$, is given by the following.

- $V_0(\mathcal{A}) = Q \times S$.
- $V_1(\mathcal{A}) = Q \times S \times Q^k$.
- There is an edge from $(q, s) \in V_0(\mathcal{A})$ to $(q, s, q_1, \ldots q_k) \in V_1(\mathcal{A})$ whenever $(q, a, q_1, \ldots, q_k) \in \delta$ where a is the label of s.
- There is an edge from $(q, s, q_1, \ldots, q_k) \in V_1(\mathcal{A})$ to $(q_i, t_i(s)) \in V_0(\mathcal{A})$ for $1 \leq i \leq k$.
- $v_I(\mathcal{A}) = (q_I, s_0)$,
- Q is the set of colours,
- $\chi : V(\mathcal{A}) \to Q$ is defined by taking the first component of the vertex.

[2] as Σ is a singleton, for ease of reading we omit a from the description of δ

- $\mathcal{C} = \mathcal{F}$.

It is clear from the definitions that Player 0 has a winning strategy from (q_I, s_0) in this game if, and only if, $\mathbb{A}$ accepts t. □

Chapter 3

Strategy Improvement for Parity Games

In Chapter 2 we introduced parity games and briefly remarked on the significance of determining the complexity of deciding them. One factor contributing to the importance of the analysis of parity games is that deciding the winner of a parity game is polynomial-time equivalent to the model-checking problem of modal μ-calculus, a highly expressive fragment of monadic second order logic [EJS01]. Indeed, the modal μ-calculus is the bisimulation invariant fragment of monadic second order logic, and therefore includes logics useful for verification such as the branching time temporal logic CTL* [Dam94].

Another interesting aspect of parity games is that the complexity of deciding the winner remains tantalizingly elusive. In Section 2.3 we observed that when we can restrict one player to memoryless strategies we can nondeterministically guess the strategy and if we can check in polynomial time if that strategy is winning, we have demonstrated an NP algorithm (if Player 0 has a memoryless winning strategy) or a co-NP algorithm (if Player 1 has a memoryless strategy). So, from Theorem 2.42 we obtain the following corollary:

Corollary 3.1. *Deciding the winner of a parity game is in* NP $\cap$ co-NP.

It is believed by some that parity games are decidable in polynomial time, however the problem has so far resisted attempts to find tractable algorithms, giving us the following well-researched open problem:

Open problem 3.2. *Determine the exact complexity of deciding parity games.*

In this chapter, we analyse one of the best candidates for a tractable algorithm for parity games: the strategy improvement algorithm. In Chapters 6 and 7 we define a large class (indeed, the largest class so far known) of graphs on which parity games can be solved in polynomial time.

Currently, the best known algorithm for deciding a parity game on general arenas runs in time $n^{O(\sqrt{n/\log n})}$ where n is the number of vertices of the arena [JPZ06]. If the number of priorities, p, is small compared to the size of the arena, say $p = o(\sqrt{n/\log n})$, we can slightly improve on this with an algorithm that runs in time $O\Big(dm \cdot \big(\frac{n}{\lfloor p/2 \rfloor}\big)^{\lfloor p/2 \rfloor}\Big)$ where m is the number of edges of the arena [Jur00]. However, in [VJ00a], Vöge and Jurdziński introduced a strategy improvement algorithm which appears to do quite well in practice, even when the number of priorities is large. To date, the best known upper bound for its running time is $O\big(mn \prod_{v \in V_0(\mathcal{A})} d_{out}(v)\big)$, which is in general exponential in the number of vertices. However, no family of examples has yet been found that runs in worse than linear time. In this chapter

we analyse the structure of this algorithm and use combinatorial results to improve the known upper and lower bounds. The analysis we use is primarily taken from [VJ00b].

3.1 The strategy improvement algorithm

The idea behind the strategy improvement algorithm is to define a measure dependent on the strategy of Player 0. Then, starting with an arbitrary strategy for Player 0, to make local adjustments based on this measure to obtain a new strategy which is in some sense improved. This process is then repeated until no further improvements can be made. At this point, with a judicious choice of measure, the strategy is the optimal play for Player 0, and the winning sets for each player can easily be computed. This procedure is readily extended to any strategy that requires finite memory, so from Theorem 2.14 we see that it can be used for games other than parity games. However, with parity games we can restrict ourselves to memoryless strategies and then at each stage both the measure and the local improvements can be efficiently computed.

In order to fully describe the algorithm, we need to introduce some concepts. Using the notation of Chapter 2, let us fix a parity game $\mathbb{G} = (\mathcal{A}, \chi)$ where $\chi : V(\mathcal{A}) \to \mathsf{P}$. For convenience we assume no vertex in $\mathcal{A}$ has out-degree 0. For the remainder of this chapter, we assume all strategies are positional.

To be able to evaluate strategies, we first identify the characteristics of a play which are important. A *play profile* is a triple (l, P, e) where $l \in \mathsf{P}$, $P \subseteq \mathsf{P}$ and $e \in \omega$. Given an infinite play $\pi = v_1 v_2 \cdots$ in $\mathbb{G}$, we associate with π a play profile, $\Theta(\pi) := (l, P, e)$, as follows. We define l to be the maximum priority occurring infinitely often in $\chi(\pi)$, so the parity of l determines the winner of the play. We define P to be the set of priorities greater than l that occur in $\chi(\pi)$, and e to be the minimal index such that $\chi(v_e) = l$ and $\chi(v_{e'}) \leq l$ for all $e' \geq e$. A *valuation* is a mapping from each vertex $v \in V(\mathcal{A})$ to a play profile of an infinite play from v.

We next define an ordering that compares play profiles by how beneficial they are to each player. We begin by defining a useful linear order on the set of priorities. The *reward order*, $\sqsubseteq$, is defined as follows: for $i, j \in \mathsf{P}$, $i \sqsubseteq j$ if either

(i) i is odd and j is even, or
(ii) i and j are even and $i \leq j$, or
(iii) i and j are odd and $i \geq j$.

Intuitively, $i \sqsubseteq j$ if j is "better" for Player 0 than i. We extend $\sqsubseteq$ to play profiles by defining $(l, P, e) \sqsubset (m, Q, f)$ if either

(i) $l \sqsubset m$; or
(ii) $l = m$ and $\max_{\leq}(P \triangle Q)$ is odd and in P, or even and in Q; or
(iii) $l = m$, $P = Q$, and either l is odd and $e < f$, or l is even and $e > f$.

The measure we use to implement the strategy improvement algorithm is a valuation that gives the $\sqsubseteq$-minimal play profile amongst all plays consistent with the current strategy for Player 0. More precisely, let σ be a strategy for Player 0, and for $v \in V(\mathcal{A})$ let $\mathrm{Plays}_\sigma(v)$ be the set of all infinite plays starting from v consistent with σ. We define the valuation φ_σ by:

$$\varphi_\sigma(v) := \min{}_{\sqsubseteq}\{\Theta(\pi) : \pi \in \mathrm{Plays}_\sigma(v)\}.$$

The next proposition, taken from [VJ00b], helps give an intuitive understanding of φ_σ. Given a strategy σ for Player 0 and a strategy τ for Player 1, we observe there is precisely one infinite play $\pi_{\sigma\tau}(v)$ consistent with σ and τ from each vertex $v \in V(\mathcal{A})$. We write $\Theta_{\sigma\tau}$ for the valuation defined by:

$$\Theta_{\sigma\tau}(v) := \Theta\big(\pi_{\sigma\tau}(v)\big).$$

If we further extend $\sqsubseteq$ to a partial order on valuations, $\trianglelefteq$, in a pointwise manner then Proposition 5.1 of [VJ00b] can be stated as:

Proposition 3.3. *The set $\{\Theta_{\sigma\tau} : \tau$ is a strategy for Player 1$\}$ has a $\trianglelefteq$-minimal element and it is equal to φ_σ.*

Intuitively, this means that φ_σ is equivalent to the valuation defined by σ and the best counter-strategy for Player 1 against σ. Consequently, φ_σ can be efficiently computed by fixing the strategy of Player 0 and considering the strategies of Player 1 in the resulting single player game.

After computing φ_σ, the algorithm makes local *improvements* to the strategy σ by switching (if necessary) $\sigma(v)$ to the successor of v with the $\sqsubseteq$-maximal φ_σ value. The resulting strategy σ' is improved in the sense that $\varphi_\sigma \trianglelefteq \varphi_{\sigma'}$. This is then repeated until no further improvements can be made. At this point the strategy σ is optimal for Player 0, that is, Player 0 can win from a vertex $v \in V(\mathcal{A})$ against any strategy for Player 1 if, and only if, she can win playing σ from v against any strategy. We can then compute the winning sets by fixing Player 0's strategy and finding the winning sets for Player 1 in the single player game. Algorithm 3.1 provides a detailed description of the critical part of the strategy improvement algorithm.

Algorithm 3.1 Strategy optimization

Returns: An optimal strategy for Player 0
  select a strategy σ for Player 0 at random
  repeat
    let $\sigma = \sigma'$ {Store current strategy}
    Compute φ_σ
    for each $v \in V_0$ **do** {Improve σ locally according to φ_σ}
      select w such that $(v, w) \in E(\mathcal{A})$ and
        $\varphi_\sigma(w) = \max_\sqsubseteq\{\varphi_\sigma(v') : (v, v') \in E(\mathcal{A})\}$
      if $\varphi_\sigma\big(\sigma(v)\big) \sqsubset \varphi_\sigma(w)$ **then**
        let $\sigma'(v) = w$
  until $\sigma = \sigma'$
  return σ

As an example, let us consider the parity game pictured in Figure 3.1. Let σ be the strategy for Player 0 defined by $\sigma(a) = a_0$, $\sigma(b) = b_0$ and $\sigma(c) = c_1$. We will compute φ_σ for the vertices a_0, b_0 and b_1. Against σ, Player 1 has a choice of strategies at a_0: either he can play to c, resulting in an infinite play with maximum priority 4, or he can play to a, resulting in an infinite play with maximum priority 1. As $1 \sqsubset 4$, the latter is the $\sqsubseteq$-minimal choice and so $\varphi_\sigma(a_0) = (1, \emptyset, 0)$. At b_0, Player 1's choice appears to depend on the strategy at a_0: if he plays to a and the strategy at a_0 is to play to a then the resulting play has maximum priority 1, otherwise if the strategy at a_0 is to play to c the resulting play has maximum priority 4. However $3 \sqsubset 1$, so the $\sqsubseteq$-minimal play in either case is going to be to play to b, resulting in $\varphi_\sigma(b_0) = (3, \emptyset, 0)$.

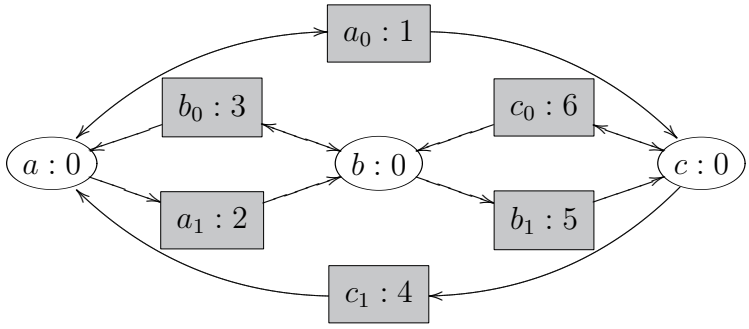


Figure 3.1: A parity game

σ	i	$\varphi_\sigma(a_i)$	$\varphi_\sigma(b_i)$	$\varphi_\sigma(c_i)$	σ'	VID
000	0	$(3, \{6\}, 4)$	$(3, \emptyset, 0)$	$(3, \{6\}, 2)$	**011**	**011**
	1	$(3, \emptyset, 2)$	$(3, \{5, 6\}, 4)$	$(3, \{4, 6\}, 6)$		
001	0	$(1, \emptyset, 0)$	$(3, \emptyset, 0)$	$(3, \{6\}, 2)$	**011**	**010**
	1	$(3, \emptyset, 2)$	$(1, \{4, 5\}, 4)$	$(1, \{4\}, 2)$		
010	0	$(1, \emptyset, 0)$	$(1, \{3\}, 2)$	$(6, \emptyset, 0)$	**110**	**100**
	1	$(6, \emptyset, 4)$	$(6, \emptyset, 2)$	$(1, \{4\}, 2)$		
011	0	$(1, \emptyset, 0)$	$(1, \{3, 4, 5\}, 6)$	$(1, \{4, 5, 6\}, 6)$	**010**	**001**
	1	$(1, \{2, 4, 5\}, 6)$	$(1, \{4, 5\}, 4)$	$(1, \{4\}, 2)$		
100	0	$(3, \emptyset, 4)$	$(3, \emptyset, 0)$	$(3, \{6\}, 2)$	**010**	**110**
	1	$(3, \emptyset, 2)$	$(3, \{5, 6\}, 4)$	$(3, \{4\}, 4)$		
101	0	$(3, \emptyset, 4)$	$(3, \emptyset, 0)$	$(3, \{6\}, 2)$	**000**	**101**
	1	$(3, \emptyset, 2)$	$(3, \{4, 5\}, 6)$	$(3, \{4\}, 4)$		
110	0	$(6, \emptyset, 6)$	$(6, \emptyset, 6)$	$(6, \emptyset, 0)$	**110**	**000**
	1	$(6, \emptyset, 4)$	$(6, \emptyset, 2)$	$(6, \emptyset, 6)$		
111	0	$(5, \emptyset, 4)$	$(5, \emptyset, 2)$	$(5, \{6\}, 2)$	**000**	**111**
	1	$(5, \emptyset, 2)$	$(5, \emptyset, 0)$	$(5, \emptyset, 4)$		

Table 3.1: Table of valuations, next strategy and improvement vectors for all strategies

The valuation at b_1 is only dependent on the choice of strategy at a_0, so $\varphi_\sigma(b_1) = (1, \{4, 5\}, 4)$. Turning to the subsequent, improved strategy σ', we have $(3, \emptyset, 0) \sqsubset (1, \{4, 5\}, 4)$. Therefore, switching σ at b will be an improvement for Player 0, and hence $\sigma'(b) = b_1$.

Using **ijk** as shorthand for the strategy which maps a to a_i, b to b_j, and c to c_k, the full table of relevant valuations and subsequent strategies for each strategy is presented in Table 3.1. Also included in this table is the *vector of improving directions (VID)*, indicating which elements of σ had improvements. Not only does this help identify **110** as the optimal strategy, but it is worth observing that each entry in the VID column is unique. As we see in the next section, this is not a coincidence.

Figure 3.2: AUSOs of the 2-cube (l) and the orientations which are not AUSOs (r)

3.2 A combinatorial perspective

In this section we show how the strategy improvement algorithm can be viewed as an optimization problem on a well-studied combinatorial structure. We will introduce the concepts of *acyclic unique sink oriented hypercubes* and the *bottom-antipodal sink-finding algorithm* and we will prove the following result:

Theorem 3.4. *The strategy improvement algorithm is a bottom-antipodal sink-finding algorithm on an acyclic unique sink orientation of the strategy hypercube.*

Although this result appears in [BSV03], we present an alternative proof that utilises results from [VJ00b].

First we recall some definitions relating to hypercubes. A d*-dimensional hypercube* is an undirected graph $\mathcal{H}_d$ such that $V(\mathcal{H}_d) = \{0,1\}^d$, and there is an edge between $(a_1,\ldots,a_d)$ and $(b_1,\ldots,b_d)$ if for some $i \leq d$, $a_i \neq b_i$ and $a_j = b_j$ for all $j \neq i$. We call a_i the i*-th component* of a vertex $(a_1,\ldots,a_d)$ in a d-dimensional hypercube. A *subcube* is a subgraph induced by a set of vertices which agree on some set of components. We observe that a subcube of a d-dimensional hypercube is a d'-dimensional hypercube for some $d' \leq d$, and we can specify a subcube by a single vertex together with a set of adjacent edges. Given a set $I \subseteq \{1,\ldots,d\}$ of natural numbers and a vertex $v = (a_1,\ldots,a_d)$ of a d-dimensional hypercube, we denote by $\text{Switch}_I(v)$ the vertex $v' = (b_1,\ldots,b_d)$ obtained by switching the components in I of v. That is, $b_i = a_i$ if, and only if, $i \notin I$. Given a vertex v in a d-dimensional hypercube, the vertex *antipodal* to v is the vertex $\text{Switch}_{\{1,\ldots,d\}}(v)$.

Given a parity game $(\mathcal{A},\chi)$, we assume that every vertex in $V_0(\mathcal{A})$ has out-degree two. From Theorem 2.59, we can always transform a parity game into one for which every vertex in $V_0(\mathcal{A})$ has out-degree at most two. We can assume there are no vertices of out-degree 0, as we can use force-sets to determine if either player can force the play to one of these vertices. We can also change any vertex in $V_0(\mathcal{A})$ with out-degree 1 to be a vertex in $V_1(\mathcal{A})$ as this does not affect the outcome of the game. As this can all be done in polynomial time, this assumption is not too restrictive. If we fix an order on $V_0(\mathcal{A}) = \{v_1,\ldots,v_d\}$, and write v_i^0 and v_i^1 for the two successors of $v_i \in V_0(\mathcal{A})$, then each vector $(b_1,\ldots,b_d) \in \{0,1\}^n$ defines a strategy for Player 0 by mapping v_i to $v_i^{b_i}$, and conversely each strategy defines a unique vector. Therefore, the space of all Player 0's strategies is equivalent to vertex set of the d-dimensional hypercube. For convenience, we will simply refer to the strategy space as the *strategy hypercube*. We now introduce some additional concepts to help establish Theorem 3.4.

An *orientation* of a d-dimensional hypercube is a directed graph with a d-dimensional hypercube as an underlying undirected graph and at most one edge between any pair of vertices. We say an orientation is an *acyclic unique sink orientation (AUSO)* if it is acyclic and every subgraph induced by a subcube has a unique sink (or, equivalently, a unique source). Figure 3.2 shows the two AUSOs for the 2-cube (left), together with the two orientations of the 2-cube which are not AUSOs (right).

Acyclic unique sink orientations of hypercubes are very important combinatorial structures, particularly as a generalization of linear programming optimization problems. For example, a *pseudo-boolean function (PBF)* is a function from a hypercube to $\mathbb{R}$, and a common optimization problem is to find the vertex which attains the maximum (or minimum) value of a PBF. In [HSLdW88], a hierarchy of classes of PBFs was introduced, and one of these classes was the *completely unimodal pseudo-boolean functions*: functions such that every subcube has a unique local minimum. Clearly, a completely unimodal PBF induces an AUSO, and conversely any function to $\mathbb{R}$ which respects an AUSO will be completely unimodal.

One useful concept associated with AUSOs is the *vector of improving directions*. Let VID : $\{0,1\}^n \to \{0,1\}^n$ be the function that maps each vertex of a hypercube with an AUSO to the vector which indicates which edges are outgoing from that vertex. That is, if there is an edge from v to v' where v and v' differ in the i-th component, then the i-th component of VID(v) is 1 and the i-th component of VID(v') is 0.

An important class of problems for AUSOs and similar structures are *polynomial local search* problems (PLS). These are optimization problems where the cost of a solution and "neighbouring" solutions can be efficiently computed, with the overall goal being to find a locally optimal solution – one which is better than all its neighbours. For example, if computing the directions of edges incident with a vertex can be done in polynomial time, then finding the unique global sink of an acyclic unique sink oriented hypercube is a problem in PLS. Clearly, given a hypercube we could iterate through all vertices to find the sink, but as is usually the case for interesting problems in PLS, iterating through all possible solutions is considered infeasible. For the sink-finding problem a more interesting question is: can we find the global sink in time polynomial in the dimension of the hypercube? In fact, for acyclic unique sink oriented hypercubes, this is an important open problem.

Open problem 3.5. *Given an n-dimensional hypercube with an AUSO, is there a polynomial p such that the global sink can be found with at most $p(n)$ vertex queries?*

One reason for the importance of this question is that there are interesting structural results for AUSOs that suggest this question can be answered in the affirmative. Firstly, an n-dimensional hypercube with an AUSO satisfies the Hirsch conjecture [WH88], which means that from each vertex there is a directed path of length at most n to the global sink. Secondly, we have the following observation from Williamson Hoke [WH88] which shows that the vector of improving direction takes a very special form:

Theorem 3.6 ([WH88])**.** *VID is a bijection.*

However, despite these results, an efficient sink-finding algorithm on hypercubes with AUSOs remains elusive.

The connection between AUSOs and the strategy improvement algorithm is summarized in the following theorem:

Theorem 3.7. *The valuation φ_σ induces an AUSO on the strategy hypercube.*

In order to prove this, we must first indicate how φ_σ induces an orientation. Let $<$ be any linear ordering on the set of Player 0's strategies. We extend $\trianglelefteq$ to a partial order on strategies by saying $\sigma \lhd \sigma'$ if either

(i) $\varphi_\sigma \lhd \varphi_{\sigma'}$, or

(ii) $\varphi_\sigma = \varphi_{\sigma'}$ and $\sigma < \sigma'$.

This gives us an orientation on the strategy hypercube, as we see with the following result:

Lemma 3.8. *Let σ and σ' be strategies for Player 0 such that $\sigma(v) = \sigma'(v)$ for all but one $v \in V_0(\mathcal{A})$. Then either $\sigma \lhd \sigma'$, or $\sigma' \lhd \sigma$.*

The proof of this result follows directly from the following two results from [VJ00b].

Lemma 5.7 of [VJ00b]. *Let $I \subseteq \{1, \ldots, d\}$ be a set of natural numbers, and let σ be a strategy for Player 0. If, for each $i \in I$, $\varphi_\sigma\big(\sigma(v_i)\big) \sqsubset \varphi_\sigma(v_i')$ where v_i' is the successor of v_i not equal to $\sigma(v_i)$, then $\sigma \trianglelefteq \mathit{Switch}_I(\sigma)$.*

Claim 7.2 of [VJ00b]. *Let $I \subseteq \{1, \ldots, d\}$ be a set of natural numbers, and let σ be a strategy for Player 0. If, for each $i \in I$, $\varphi_\sigma\big(\sigma(v_i)\big) \not\sqsubset \varphi_\sigma(v_i')$ where v_i' is the successor of v_i not equal to $\sigma(v_i)$, then $\mathit{Switch}_I(\sigma) \trianglelefteq \sigma$.*

The orientation is then obtained by adding an edge from σ to σ' if $\sigma(v) = \sigma'(v)$ for all but one $v \in V_0(\mathcal{A})$ and $\sigma \lhd \sigma'$. We now need to show that this orientation is an AUSO. To do this, we use the fact that the strategy improvement algorithm terminates.

Theorem 3.1 of [VJ00b]. *The strategy improvement algorithm correctly computes the winner of a parity game.*

Since $\trianglelefteq$ is a partial order it is clear that this orientation is acyclic. In order to show that it is an AUSO, we use the following result about unique sink orientations.

Proposition 3.9 ([WH88])**.** *A hypercube orientation is a unique sink orientation if, and only if, every 2-dimensional subcube has a unique sink.*

Next we observe that every subcube of the strategy hypercube induces a subgame of the original parity game: by definition, there is a set $V \subseteq V_0(\mathcal{A})$ on which all strategies of the subcube agree. The induced subgame is obtained by fixing Player 0's choices on V to agree with all the strategies of the subcube. Furthermore, in these subgames φ_σ takes the same values as in the original parity game. Thus the resulting strategy hypercube of the subgame is a subcube of the strategy hypercube of the original game. Therefore, if any 2-dimensional subcube of the strategy hypercube does not have a unique sink, we can produce a parity game with a 2-dimensional strategy hypercube with the same orientation. The only acyclic orientation of a 2-cube without a unique sink is one with antipodal sinks and sources (see Figure 3.2). In Lemma 3.10 we describe how the strategy improvement algorithm works on an oriented hypercube, and from this we see that if the algorithm begins at a source of this 2-dimensional hypercube, then the subsequent strategy will always be the other source. Thus, on this orientation, the algorithm never terminates. Since Theorem 3.1 of [VJ00b] ensures that the strategy improvement algorithm always terminates, every 2-dimensional subcube has a unique sink, and we have therefore shown that the orientation defined by $\lhd$ is an AUSO. This completes the proof of Theorem 3.7.

Returning to the example parity game from the previous section, we can read the orientation of the strategy hypercube directly from Table 3.1. For example, consider the strategy $\sigma = \{\mathbf{001}\}$. Since $\varphi_\sigma(a_1) \sqsubset \varphi_\sigma(a_0)$, it follows that $\mathbf{101} \trianglelefteq \mathbf{001}$, thus there is an edge from $\mathbf{101}$ to $\mathbf{001}$. Figure 3.3 shows the resulting oriented strategy hypercube.

Having established that the set of strategies for Player 0 forms a hypercube oriented by $\lhd$, we can investigate how the strategy improvement algorithm operates on this cube. From Algorithm 3.1, we see that a strategy σ switches at each point where $\varphi_\sigma\big(\sigma(v)\big)$ is not $\sqsubseteq$-maximal.

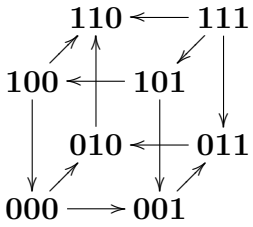


Figure 3.3: Oriented strategy hypercube for the parity game in Figure 3.1

If this is adjusted so that when there is a choice of strategies with $\sqsubseteq$-maximal φ_σ values, we choose the $<$-largest strategy, then from Lemma 3.8 we see that we are switching σ at the vertices corresponding to the outgoing edges in the strategy hypercube. That is,

Lemma 3.10. *Let σ be a strategy for Player 0 and C_σ be the subcube of the oriented strategy hypercube defined by σ and the outgoing edges from σ. Then the subsequent strategy σ' in the strategy improvement algorithm is the vertex antipodal to σ on C_σ.*

This is a well-known sink-finding procedure for AUSO hypercubes called BOTTOM-ANTIPODAL [SS05], described in Algorithm 3.2. It is clear that on an AUSO hypercube, BOTTOM-ANTIPODAL ter-

Algorithm 3.2 BOTTOM-ANTIPODAL

Returns: Global sink of an AUSO hypercube
  select a vertex v at random
  repeat
    Compute VID(v)
    let $v = v \oplus \text{VID}(v)$ {XOR v and VID(v)}
  until VID$(v) = \mathbf{0}$
  return v

minates with the global sink: at each stage we are jumping from the unique source of the subcube defined by the improving directions to some other vertex in that subcube, so we are always reducing the minimal distance to the global sink. Combining Lemma 3.10 with Theorem 3.7 gives us the main result:

Theorem 3.4. *The strategy improvement algorithm is a bottom-antipodal sink-finding algorithm on an acyclic unique sink orientation of the strategy hypercube.*

3.3 Improving the known complexity bounds

The upper bound of $O\big(mn \prod_{v \in V_0} d_{out}(v)\big)$ for the running time of the strategy improvement algorithm arises from the observations that it takes $O(mn)$ time to compute φ_σ and there are $\big(\prod_{v \in V_0(\mathcal{A})} d_{out}(v)\big)$ different strategies for Player 0 [VJ00a]. The results of Section 3.2 enable us to improve the trivial upper bound obtained by naïvely running through all possible strategies. Mansour and Singh [MS99] showed that a BOTTOM-ANTIPODAL sink-finding algorithm will visit at most $O\big(\frac{2^d}{d}\big)$ vertices of a d-dimensional hypercube. However, we can improve this upper bound further by using results from combinatorics. Instead of using the BOTTOM-ANTIPODAL algorithm, we can use other sink-finding algorithms such as the FIBONACCI SEE-SAW of Szabó

and Welzl [SW01], described in Algorithm 3.3. This algorithm utilises structural results of AUSOs such as Theorem 3.6 and has the best-known running time upper bound, $O(1.61^d)$, amongst sink-finding algorithms.

Algorithm 3.3 FIBONACCI SEE-SAW

Returns: Global sink of an AUSO hypercube
  select a vertex m at random
  let w be the vertex antipodal to m
  let $C_m = \{m\}$ and $C_w = \{w\}$ {Antipodal i-dimensional subcubes}
  for $i = 0$ to n **do**
    Compute VID$(m) = (m_0, m_1, \ldots)$ and VID$(w) = (w_0, w_1, \ldots)$
    let $d = \min\{j : m_j \neq w_j\}$
    let C'_m be the i-dimensional subcube parallel to C_m in direction d from m
    let C'_w be the i-dimensional subcube parallel to C_w in direction d from w
    if $m_d = 1$ **then** {m is the minimal vertex of an $(i+1)$-dimensional subcube}
      Compute $w =$ FIBONACCI SEE-SAW(C'_w)
    else {w is the minimal vertex of an $(i+1)$-dimensional subcube}
      Compute $m =$ FIBONACCI SEE-SAW(C'_m)
    let $C_m = C_m \cup C'_m$ and $C_w = C_w \cup C'_w$
  return m

These results give us the following improved upper bounds for the strategy improvement algorithm:

Proposition 3.11. *Assuming each vertex in $V_0(\mathcal{A})$ has out-degree two:*

(i) The strategy improvement algorithm runs in time $O(mn \cdot 2^{n_0}/n_0)$.

(ii) The Fibonacci strategy improvement algorithm runs in time $O(mn \cdot 1.61^{n_0})$.

Where $m = |E(\mathcal{A})|$, $n = |V(\mathcal{A})|$ and $n_0 = |V_0(\mathcal{A})|$.

Turning to lower bounds, natural questions to consider are completeness results. In particular, is strategy improvement or finding the sink of an AUSO hypercube PLS-complete? Björklund et al. [BSV03] show that this is not the case.

Theorem 3.12 ([BSV03])**.** *The problem of finding optimal strategies in parity games is not* PLS*-complete with respect to tight* PLS*-reductions.*

Because PLS-complete problems have exponentially long improvement paths [Yan97], the fact that strategy improvement is not PLS-complete gives further support to the hypothesis that it may only require polynomially many iterations.

However, we can also ask if there are examples of parity games which require an exponential number of strategies to be considered by the strategy improvement algorithm. As a first step towards this, Schurr and Szabó [SS05] generated a family of oriented hypercubes for which BOTTOM-ANTIPODAL visits $2^{d/2}$ vertices. It remains an open problem whether there is a family of parity games with these hypercubes as their strategy hypercubes. In fact, this can be generalized to a more interesting open problem:

Open problem 3.13. *Given a hypercube with an AUSO, can a parity game be constructed in polynomial time with that hypercube as its strategy hypercube?*

A positive answer to this question would not only give an exponential worst case for the strategy improvement algorithm, but it would also relate Open Problems 3.2 and 3.5: a polynomial time algorithm for finding the sink on an AUSO would give a polynomial time algorithm for solving parity games and vice versa. On the other hand a negative answer to this question would give a smaller class of AUSOs for which finding a polynomial time sink-finding procedure is an interesting and important problem.

This leads to another interesting question: Can we classify the AUSO hypercubes that correspond to parity games? As we mentioned previously, Hammer et al. [HSLdW88] introduced a hierarchy of pseudo-boolean functions including completely unimodal functions. It seems plausible that the class of PBFs corresponding to parity games might lie within one of the more restrictive families they considered. For example, viewing a d-dimensional hypercube $\mathcal{H}_d$ as a polytope in $\mathbb{R}^d$, a PBF φ on $\mathcal{H}_d$ is *linearly separable* if for all $r \in \mathbb{R}$ there exists a hyperplane separating the vertices v with $\varphi(v) \geq r$ from the vertices v' with $\varphi(v') < r$. It is easily seen that a divide-and-conquer algorithm can find the sink of a linearly separable hypercube in time linear in the dimension, so if the hypercube orientations associated with parity games are linearly separable then the strategy improvement algorithm would run in polynomial time. However, as the next result shows, the hierarchy of [HSLdW88] is not fine enough to separate parity games and completely unimodal functions. We say a pseudo-boolean function $f : \{0,1\}^n \to \mathbb{R}$ is *pseudomodular* if for all $v, w \in \{0,1\}^n$:

(i) $\min\{f(v), f(w)\} \leq \max\{f(v \wedge w), f(v \vee w)\}$, and
(ii) $\min\{f(v \wedge w), f(v \vee w)\} \leq \max\{f(v), f(w)\}$.

In [HSLdW88], the class of pseudomodular functions was the least restrictive class of PBFs included in completely unimodal functions. However,

Proposition 3.14. *There exists a parity game with an oriented strategy hypercube that cannot be induced by a pseudomodular function.*

Proof. Consider the parity game from Figure 3.1. Its oriented strategy hypercube can be seen in Figure 3.3. We see that

$$\mathbf{111} \lhd \mathbf{000} \lhd \mathbf{001} \lhd \mathbf{110}.$$

Now taking $v = \mathbf{001}$ and $w = \mathbf{110}$ we see that there is no function $f : \{0,1\}^3 \to \mathbb{R}$ that can simultaneously respect $\lhd$ and satisfy both pseudomodular axioms above. □

This result is not surprising, there is no obvious reason why the joins and meets of strategies should satisfy the pseudomodular conditions. However, it does imply for instance that there are strategy hypercubes which are not linearly separable.

Chapter 4

Complexity measures for digraphs

In the last few chapters we examined the computational complexity of some graph-based games. We saw how the winning condition influences the difficulty of the problem of finding a winner of such games. We now turn our attention to the other aspect of such games, the arena. The aim of the next few chapters is to investigate measures of graph complexity, in particular measures for directed graphs. As we will see, such metrics give insight into the structure theory of graphs and help identify those characteristics that act as a barrier to finding efficient solutions of various important problems (for example, finding the winner of a parity game, or finding a Hamiltonian path). Consistent with the overall theme of this dissertation, the complexity measures we define will be based on games.

So what makes a good complexity measure? First we have to consider what it is we are aiming to measure. This of course depends largely on the application one has in mind. For instance, a group theorist may be interested in graph automorphisms and so a useful measure might reflect the size of the automorphism group. A topologist might be interested in a measure that indicates how many edges must cross in a drawing of the graph on a surface, or how many paths there are between any pair of vertices. We are interested in algorithmic aspects, so a practical measure might indicate the difference between tractable and intractable instances of many NP-complete problems. A good measure of complexity may even encompass more than one such aim. So one desirable property is *soundness*: the measure can be defined in equivalent ways for different applications. Another desirable property is *robustness*: the measure should be "well-behaved". For example, if we simplify the graph, then the measure should not increase. Again, the concept of simplification is dependent on the application. For the group theorist, a simple graph is one in which all vertices have similar structure, for example, a clique. For the topologist a simple graph might be an acyclic graph. From the algorithmic perspective, simplifying would include operations that likely reduce the complexity of many problems, for instance taking subgraphs. In this case simple graphs would be a class on which many NP-complete problems have polynomial time solutions – again, acyclic graphs are a good example. Dually, if we complicate the graph the measure should not decrease, and if this complication is in some way uniform, we would expect the measure to increase uniformly. One final desirable feature, particularly for algorithmic purposes, is that the measure should somehow encompass large classes of graphs. For example, acyclicity is a sound and robust measure, but it only takes two values, a graph is either acyclic or it is not. So although acyclicity provides a boundary between tractable and intractable instances of many NP-complete problems, we cannot use it to find larger classes of graphs which may admit efficient solutions. This suggests that a generalization of acyclicity,

perhaps indicating how acyclic a graph is, would be an ideal candidate for a good complexity measure. This is precisely the type of measure we consider in this and the following chapters.

In this chapter we introduce an important and well-known measure for undirected graphs called tree-width. We show how it matches the criteria outlined above, and we discuss the problem with its extension to directed graphs, providing motivation for subsequent chapters.

4.1 Tree-width

Tree-width can be seen as a measure of graph complexity for both topological and algorithmic purposes. That it serves both purposes is not surprising as it is often the complexity of the structure of the graph that makes problems difficult to solve; many NP-complete problems can be solved in polynomial time on the topologically simple class of acyclic graphs. As the name suggests, the tree-width of a graph indicates how close that graph is to being a tree. For example, trees have tree-width 1, simple cycles have tree-width 2, and highly connected graphs such as cliques have tree-width one less than the number of vertices in the graph.

Although Robertson and Seymour coined the name tree-width [RS84], the parameter had been around for many years prior to this, testament to the importance of tree-width as a measure of graph complexity. Rose and Tarjan [RT75] considered a symbolic approach to Gaussian elimination on matrices which amounts to vertex elimination on graphs. They introduced several parameters which reflect how "difficult" it is to perform a sequence of eliminations: for example the *width* of an elimination reflects the maximum number of operations required at any stage of the elimination. The minimum width over all vertex eliminations is a graph measure equivalent to tree-width. Halin [Hal76] considered S-functions: mappings from graphs to integers satisfying certain formal conditions, a class of functions which includes graph parameters such as the chromatic number, the vertex-connectivity and the homomorphism-degree. Halin showed that there is a maximal S-function under the natural point-wise partial ordering of S-functions, and this function turns out to be the tree-width of the graph. Arnborg [Arn85] was one of the first to show the algorithmic importance of tree-width, by finding efficient solutions to many NP-complete problems on partial k-trees, a characterization of the class of graphs with tree-width bounded by k. We will revisit some of these alternative characterizations of tree-width in Chapter 7.

To formally define tree-width, we must first introduce the notion of a tree decomposition. A tree decomposition of a graph $\mathcal{G}$ is an arrangement of subgraphs of $\mathcal{G}$ in a tree-like manner so that all paths in the graph respect this arrangement. More precisely,

Definition 4.1 (Tree decompositions and tree-width)**.** Let $\mathcal{G}$ be an undirected graph. A *tree decomposition* of $\mathcal{G}$ is a pair $(\mathcal{T}, \mathcal{X})$ where $\mathcal{T}$ is a tree and $\mathcal{X} = (X_t)_{t \in V(\mathcal{T})}$ is a family of subsets of $V(\mathcal{G})$ such that:

(T1) $\mathcal{X}$ is a cover of $V(\mathcal{G})$, that is, $\bigcup_{X \in \mathcal{X}} X = V(\mathcal{G})$,

(T2) For each vertex $v \in V(\mathcal{G})$ the subgraph of $\mathcal{T}$ induced by the set $\{t : v \in X_t\}$ is a connected subtree, and

(T3) For each edge $\{u, v\} \in E(\mathcal{G})$ there exists $t \in V(\mathcal{T})$ such that $\{u, v\} \subseteq X_t$.

The *width* of a decomposition $(\mathcal{T}, \mathcal{X})$ is $\max\{|X_t| : t \in V(\mathcal{T})\} - 1$. The *tree-width* of a graph $\mathcal{G}$, Tree-width$(\mathcal{G})$ is the minimum width over all tree decompositions of $\mathcal{G}$.

To see how this definition corresponds with our informal description above, let $\mathcal{G}$ be an undirected graph and $(\mathcal{T}, \mathcal{X})$ be a pair such that $\mathcal{T}$ is a tree and $\mathcal{X} = (X_t)_{t \in V(\mathcal{T})}$ is a cover of $V(\mathcal{G})$. For an arc[1] $e = \{s, t\} \in E(\mathcal{T})$, we observe that the removal of e from $\mathcal{T}$ gives two subtrees: one, $\mathcal{T}_s$, containing the node s, the other, $\mathcal{T}_t$ containing the node t. Let $V_s = \bigcup_{t' \in \mathcal{T}_s} X_{t'}$ and $V_t = \bigcup_{t' \in \mathcal{T}_t} X_{t'}$. We define the following condition:

(T4) For each arc $\{s, t\} \in E(\mathcal{T})$, every path from V_s to V_t contains at least one vertex in $X_s \cap X_t$.

Condition (T4) can be used as an alternative to conditions (T2) and (T3) as we see in the following lemma.

Lemma 4.2. *Let $\mathcal{G}$ be an undirected graph, and $(\mathcal{T}, \mathcal{X})$ a pair such that $\mathcal{T}$ is a tree and $\mathcal{X} = (X_t)_{t \in V(\mathcal{T})}$ is a cover of $V(\mathcal{G})$. Then (T4) holds if, and only if, both (T2) and (T3) hold.*

Proof. Suppose (T4) holds. For each vertex $v \in V(\mathcal{G})$, let $\mathcal{T}[v]$ be the subgraph of $\mathcal{T}$ induced by the set $\{t \in V(\mathcal{T}) : v \in X_t\}$. Suppose $\mathcal{T}[v]$ is not connected. Let C_1 and C_2 be two distinct components of $\mathcal{T}[v]$. Since $\mathcal{T}$ is a tree, there is a unique path in $\mathcal{T}$ from C_1 to C_2. Let (s, s') be the first arc in that path. Since C_1 and C_2 are distinct components, we have $s \in C_1$ and $s' \notin V(\mathcal{T}[v])$, so $v \in X_s \subseteq V_s$, but $v \notin X_{s'}$, so $v \notin X_s \cap X_{s'}$. However, $C_2 \subseteq \mathcal{T}_{s'}$, so $v \in V_{s'}$. As the path (of length 0) from v to itself does not go through $X_s \cap X_{s'}$ we have a contradiction. Thus (T2) holds. Now let $e = \{u, v\}$ be an edge of $\mathcal{G}$ and suppose $\mathcal{T}[u]$ and $\mathcal{T}[v]$ have no nodes in common. Let (s, s') be the first arc in the unique path from $\mathcal{T}[u]$ to $\mathcal{T}[v]$ in $\mathcal{T}$. We observe that $u \in \mathcal{T}_s$, $u \notin \mathcal{T}_{s'}$, $v \notin \mathcal{T}_s$ and $v \in \mathcal{T}_{s'}$. But then no vertex on the (length 1) path from u to v along e is contained in $X_s \cap X_{s'}$, a contradiction. Therefore, (T3) holds.

Now suppose (T2) and (T3) hold. Let $\{s, s'\}$ be an arc of $\mathcal{T}$. Let $(v_1, \ldots, v_n)$ be a path from $v_1 \in V_s$ to $v_n \in V_{s'}$. We show that there must be some i such that $v_i \in X_s \cap X_{s'}$. If $v_i \in V_s \cap V_{s'}$ for any i, $1 \leq i \leq n$, then it follows from (T2) that $v_i \in X_s \cap X_{s'}$ and we are done. So assume that there is no i such that $v_i \in V_s \cap V_{s'}$. Since $v_1 \in V_s$ and $v_n \in V_{s'}$, it follows that there is some j, $1 < j \leq n$ such that $v_i \in V_s$ for all $1 \leq i < j$, and $v_j \in V_{s'}$. But there is an edge from v_{j-1} to v_j so from (T3) there exists $t \in V(\mathcal{T})$ such that $\{v_{j-1}, v_j\} \subseteq X_t$. Now $V(\mathcal{T}_s) \cup V(\mathcal{T}_{s'}) = V(\mathcal{T})$, so either $t \in V(\mathcal{T}_s)$ or $t \in V(\mathcal{T}_{s'})$. In the first case it follows that $v_j \in V_s$, and in the second it follows that $v_{j-1} \in V_{s'}$, both of which are contradictions. Therefore (T4) holds. □

Path-width

Path-width, also introduced by Robertson and Seymour [RS83], is a measure of complexity for undirected graphs closely related to tree-width. Just as tree-width indicates how close a graph is to being a tree, path-width indicates how close a graph is to being a path. Indeed, a path decomposition is a tree decomposition in which the underlying tree is a path. More precisely,

Definition 4.3 (Path decomposition and path-width)**.** Let $\mathcal{G}$ be an undirected graph. A *path decomposition* of $\mathcal{G}$ is a sequence $X_1, \ldots, X_n$ of subsets of $V(\mathcal{G})$ such that:

(P1) $\bigcup_{i=1}^{n} X_i = V(\mathcal{G})$,

(P2) If $i \leq j \leq k$ then $X_i \cap X_k \subseteq X_j$, and

[1] To assist with descriptions, we use the terms *nodes* and *arcs* when referring to $\mathcal{T}$, and the terms *vertices* and *edges* for $\mathcal{G}$.

(P3) For each $e = \{u, v\} \in E(\mathcal{G})$, there exists $i \leq n$ such that $\{u, v\} \subseteq X_i$.

The *width* of a path decomposition, $X_1, \ldots, X_n$, is $\max\{|X_i| : 1 \leq i \leq k\} - 1$. The *path-width* of $\mathcal{G}$ is the smallest width of any path decomposition of $\mathcal{G}$.

It is worth observing that if $X_1, \ldots, X_n$ is a path decomposition of a graph $\mathcal{G}$, then so is $X_n, \ldots, X_1$. Thus a path decomposition is not completely dependent on the linear order imposed by the fact that it is a sequence.

Because a path decomposition is also a tree decomposition, path-width is a weaker notion of graph complexity than tree-width. That is, if a graph has path-width k, then the graph has tree-width $\leq k$. The difference between the two can be arbitrarily large: the class of trees has tree-width 1, but unbounded path-width. However, as argued in [DK05], path-width can be seen as a first approximation of tree-width, and many interesting structural results can be established with the measure. For example, we have the following result of Bienstock, Robertson, Seymour and Thomas:

Theorem 4.4 ([BRST91])**.** *For every forest $\mathcal{T}$, every graph of path-width $\geq |V(\mathcal{T})| - 1$ has a minor isomorphic to $\mathcal{T}$.*

4.1.1 Structural importance of tree-width

Lemma 4.2 gives us a good insight into what graph properties tree-width measures. If we take the given definition of a tree decomposition, we see that tree-width is essentially a measure indicating how much structure we need to ignore before the graph becomes acyclic. In this way, tree-width measures the cyclicity of a graph. On the other hand, if we define tree decompositions using (T1) and (T4) we see that tree-width measures how well separate parts of the graph are linked. In other words, tree-width also measures the connectedness of a graph. Lemma 4.2 asserts that on undirected graphs cyclicity and connectedness generalize to the same measure. As we will see, this distinction is important, because on directed graphs cyclicity and connectedness are significantly different, giving us a variety of complexity measures to consider.

In Chapter 1, we indicated that the concept of "graph structure" that we are interested in investigating is algorithmically motivated. As we have suggested, cyclicity and connectedness are important algorithmic structural properties, so this suggests that tree-width is a useful measure for graph structure.

An important relation for the theory of graph structure that we are investigating is the *minor relation*. Intuitively the minor relation relates two graphs if one is structurally "more complex" than the other. We formally define the concept in Chapter 8. It is not surprising that tree-width and the minor relation are closely connected. Indeed, tree-width was an important tool in the proof by Robertson and Seymour [RS04] of the Graph Minor Theorem (see Theorem 8.42), described by Diestel as "among the deepest results mathematics has to offer" [Die05]. In addition many other structural measures have been shown to be intimately related to tree-width. For instance a *feedback vertex set* is a set of vertices whose removal result in an acyclic graph. It is easy to show that if a graph has a feedback vertex set of size k, then it has tree-width at most $k + 1$. Two other important structural measures are havens and brambles.

Definition 4.5 (Haven)**.** Let $\mathcal{G}$ be an undirected graph and $k \in \mathbb{N}$. A *haven of order k* in $\mathcal{G}$ is a function $\beta : [V(\mathcal{G})]^{<k} \to \mathcal{P}(V(\mathcal{G}))$ such that for all $X \subseteq V(\mathcal{G})$ with $|X| < k$:

(H1) $\beta(X)$ is a non-empty connected component of $\mathcal{G} \setminus X$, and

(H2) If $Y \subseteq X$, then $\beta(Y) \supseteq \beta(X)$.

Definition 4.6 (Bramble)**.** Let $\mathcal{G}$ be an undirected graph. A *bramble* in $\mathcal{G}$ is a set $\mathcal{B}$ of connected subsets of $V(\mathcal{G})$ such that for all pairs $B, B' \in \mathcal{B}$ either $B \cap B' \neq \emptyset$, or there exists $\{u, v\} \in E(\mathcal{G})$ with $u \in B$ and $v \in B'$. The *width* of a bramble $\mathcal{B}$ is the minimum size of a set which has a non-empty intersection with every element of $\mathcal{B}$.

Seymour and Thomas [ST93] demonstrated the relation between havens, brambles and tree-width with the following theorem:

Theorem 4.7 ([ST93])**.** *Let $\mathcal{G}$ be an undirected graph. The following are equivalent:*

1. *$\mathcal{G}$ has tree-width $\geq k - 1$*
2. *$\mathcal{G}$ has a haven of order k.*
3. *$\mathcal{G}$ has a bramble of width k.*

This theorem asserts that the smallest width of all tree decompositions is always equal to the largest width of all brambles. Since the width of tree decompositions is a maximizing measure and the width of brambles is a minimizing measure, Theorem 4.7 is a minimax theorem. We explore this aspect of tree-width further in Chapter 8.

The importance of tree-width as a measure of structural complexity suggests that tree-width is robust under various structural transformations, particularly those, such as taking subgraphs, which may affect the complexity of problems. Indeed, this can be verified by examining the definition of tree decompositions, but is perhaps best illustrated by Theorem 5.37, which we present in the next chapter.

4.1.2 Algorithmic importance of tree-width

The nature of tree decompositions further supports the algorithmic significance of tree-width, as the structure of a decomposition lends itself well to dynamic programming techniques [Bod88]. When we restrict to a class of graphs of bounded tree-width, we bound the size of the tree decompositions and many algorithms based on dynamic programming will run in polynomial time. Thus restricting to classes of graphs of bounded tree-width can provide large classes of tractable instances for many NP-complete problems. This was best illustrated by Arnborg and Proskurowski [AP89], when they provided efficient algorithms for many well-known NP-complete problems on graphs of bounded tree-width. This was further extended by Courcelle's elegant characterization of a large class of problems which can be efficiently solved with dynamic programming:

Theorem 4.8 ([Cou90])**.** *Any problem which can be formulated in Monadic Second Order logic can be solved in linear time on any class of graphs of bounded tree-width.*

Of course the applicability of these results depends largely on the complexity of the following decision problem:

TREE-WIDTH	
Instance:	An undirected graph $\mathcal{G}$, and a natural number k
Problem:	Is the tree-width of $\mathcal{G}$ at most k?

While this problem is NP-complete [ACP87], for a fixed value k determining if a graph has tree-width k and indeed, computing a tree decomposition of width k if one exists, can be performed in linear time [Bod96]. This means that finding the tree-width of a graph is fixed parameter tractable, and so it is not surprising that tree-width has also played a major role in advancing the field of parameterized complexity.

As we mentioned earlier many important graph parameters are closely related to tree-width, so a common technique for finding fixed parameter tractable algorithms for parameterized problems is to use tree-width to separate instances into those which can be trivially solved and those which can be solved using bounded tree-width techniques. For example, consider the parameterized problem of finding a feedback vertex set of size k. We can use the fixed parameter tractable algorithm for computing tree-width to compute a tree decomposition of width $k+1$. If no such decomposition exists then there cannot be a feedback vertex set of size k. Otherwise, since the feedback vertex set problem can be formulated in MSO, Courcelle's theorem implies there exists an algorithm to solve the problem in linear time, giving us a fixed parameter tractable algorithm for finding a feedback vertex set of size k.

4.1.3 Extending tree-width to other structures

The above discussion indicates that tree-width is a practical, sound and robust complexity measure for undirected graphs. We now consider other structures such as directed graphs or hypergraphs. One key to the success of tree-width is that tree decompositions are readily extendable to arbitrary relational structures. If, in Definition 4.1, we replace "vertices" with "elements of the universe", and condition (T3) with:

(T3′) For each relation R and each tuple $(a_1, a_2, \ldots)$ in the interpretation of R there exists $t \in V(\mathcal{T})$ such that $\{a_1, a_2, \ldots\} \subseteq X_t$,

then we obtain a definition of tree-width for general relational structures. Consequently, we can benefit from the algorithmic advantages of tree-width, such as a structure well-suited to dynamic programming, and obtain large classes of tractable instances of problems outside graph problems. But how good is tree-width as a measure of complexity on these structures? It is easy to see that the tree-width of a structure is precisely the tree-width of the Gaifman graph of that structure: the graph with vertex set equal to the universe of the structure and an edge between any two elements that occur in a tuple of a relation. The main drawback of this approach is that by considering the Gaifman graph, we lose information about the structure, and in some cases this information loss may be crucial. For example, the Gaifman graph of a directed graph is the undirected graph obtained by ignoring the orientation of the edges, so the tree-width of a directed graph is the tree-width of the underlying undirected graph. This means that directed acyclic graphs (DAGs) can have arbitrary tree-width as any graph can be the underlying graph of a DAG. However, many interesting problems based on directed graphs are greatly simplified when restricted to DAGs, so we would expect DAGs to have low complexity. This suggests that tree-width is not a good complexity measure of directed graphs, especially for algorithmic purposes.

This leads to the following research problem, the investigation of which forms the core of the remaining chapters.

Research aim. *Find a complexity measure for directed graphs which generalizes tree-width.*

Before we give an overview of the current status of this problem, we discuss what exactly "generalizes tree-width" entails. First, we are interested in measures which generalize tree-width as a measure. This has two aspects. As tree-width is defined for directed graphs, we are not interested in measures that may be "worse than" tree-width. In other words, we are searching for measures that are bounded above by tree-width. On the other hand, we can view undirected graphs as directed graphs by interpreting an undirected edge as a pair of anti-parallel edges – recall the definition of bidirection in Section 1.1.2. So we can look for a measure which matches tree-width on undirected graphs by using this transformation to directed graphs.

The second property of tree-width we are interested in generalizing is the structural aspect. Many structural properties of graphs have natural extensions to directed graphs, for example acyclicity or connectivity. A good generalization of tree-width to directed graphs would reflect the behaviour of tree-width with regard to these properties. In particular we expect structurally simple directed graphs such as DAGs and directed cycles to have low complexity, but structurally complex directed graphs such as cliques to have high complexity, just as trees and cycles have small tree-width and cliques have large tree-width. Similarly, we expect that a reasonable measure would be robust under the structural relations for directed graphs we considered in Section 1.1.2. For example, we expect that the measure would not increase under the taking of subgraphs, or that it would be possible to compute the measure on a graph from its strongly or weakly connected components, or more generally from a pair of subgraphs which comprise a directed union. This last property was considered in [JRST01] as an important property for the generalization of tree-width to directed graphs.

Finally, we are also interested in generalizing tree-width in the algorithmic sense. We are particularly interested in being able to find efficient algorithms for interesting problems on directed graphs of bounded complexity. Having some sort of decomposition which generalizes tree decompositions might be one way to achieve this.

4.2 Directed tree-width

In [JRST01], Johnson, Robertson, Seymour and Thomas introduced an extension of tree-width to directed graphs known as directed tree-width. Informally, directed tree-width is based on a decomposition, known as an arboreal decomposition, which is defined by generalizing Condition (T4). Formally, to define directed tree-width, we require the following definition:

Definition 4.9 (Z-normal)**.** Given two disjoint subsets Z and S of vertices of a digraph $\mathcal{G}$, we say S is Z-normal if for every directed path, $v_1 \cdots v_n$, in $\mathcal{G}$ such that $v_1, v_n \in S$, either $v_i \in S$ for all $1 \leq i \leq n$, or there exists $j \leq n$ such that $v_j \in Z$.

Also, given a directed tree $\mathcal{T}$ with edges oriented away from a unique vertex $r \in V(\mathcal{T})$ (called the *root*), we write $t > e$ for $t \in V(\mathcal{T})$ and $e \in E(\mathcal{T})$ if e occurs on the unique directed path from r to t, and $e \sim t$ if e is incident with t. The following concepts were introduced in [JRST01].

Definition 4.10 (Arboreal decompositions [JRST02])**.** An *arboreal decomposition* of a digraph $\mathcal{G}$ is a tuple $(\mathcal{T}, \mathcal{B}, \mathcal{W})$ where $\mathcal{T}$ is a directed tree with a unique root, and $\mathcal{B} = (B_t)_{t \in V(\mathcal{T})}$ and $\mathcal{W} = (W_e)_{e \in E(\mathcal{T})}$ are families of subsets of $V(\mathcal{G})$ that satisfy:

(R1) $\mathcal{B}$ is a partition of $V(\mathcal{G})$ into non-empty sets, and

(R2) If $e \in E(\mathcal{T})$, then $B_{\geq e} := \bigcup\{B_t | t > e\}$ is W_e-normal.

The *width* of an arboreal decomposition $(\mathcal{T}, \mathcal{B}, \mathcal{W})$ is the minimum k such that for all $t \in V(\mathcal{T})$, $|B_t \cup \bigcup_{e \sim t} W_e| \leq k + 1$. The *directed tree-width* of a digraph $\mathcal{G}$, dtw$(\mathcal{G})$, is the minimal width of all its arboreal decompositions.

It follows from this definition that directed tree-width does generalize tree-width as a measure in the sense described above.

Towards showing that directed tree-width is also a structural generalization, Johnson et al. considered the natural generalization of havens (using strongly connected components rather than connected components) and proved the following analogue of Theorem 4.7:

Theorem 4.11 ([JRST01])**.** *Let $\mathcal{G}$ be a directed graph.*

1. *If $\mathcal{G}$ has a haven of order k then $\mathcal{G}$ has directed tree-width $\geq k - 1$.*
2. *If $\mathcal{G}$ has no haven of order k then $\mathcal{G}$ has directed tree-width $\leq 3k - 2$.*

Johnson et al. conjectured that the bound in the second item could be reduced to $\leq k - 1$, showing an equivalence between havens and directed tree-width. However Adler [Adl05] has shown that this is not the case. Safari [Saf05] showed that natural generalization of brambles (using strongly connected sets rather than connected sets), can also be related to havens and directed tree-width.

Theorem 4.12 ([Saf05])**.** *For a directed graph $\mathcal{G}$ let $H(\mathcal{G})$ be the largest order of a haven in $\mathcal{G}$, and $B(\mathcal{G})$ the largest width of any bramble in $\mathcal{G}$. Then*

$$H(\mathcal{G}) \leq 2B(\mathcal{G}) \leq 2H(\mathcal{G}),$$

and there exist graphs for which equality holds in either inequality.

Johnson et al. also demonstrated the algorithmic potential of directed tree-width, firstly by providing a general algorithm scheme for finding efficient algorithms on digraphs of bounded directed tree-width, and secondly by using this scheme to produce an algorithm which solves the following problem in polynomial time on graphs of bounded directed tree-width:

k-DISJOINT PATHS	
Instance:	A directed graph $\mathcal{G}$, and a set of k pairs of (not necessarily disjoint) vertices $\{(s_1, t_1), \ldots (s_k, t_k)\}$
Problem:	Are there k vertex disjoint paths $P_1, \ldots, P_k$ in $\mathcal{G}$ such that for each i, P_i is a path from s_i to t_i?

A corollary of this result is that many other important NP-complete problems, such as the Hamiltonian path and cycle problems, can be solved efficiently on graphs of bounded directed tree-width.

Theorem 4.13 ([JRST01])**.** *The following problems can be solved in polynomial time on any class of directed graphs with bounded directed tree-width: Hamiltonian cycle, Hamiltonian path, k-Disjoint paths, Hamiltonian path with prescribed endpoints, Even cycle through a given vertex.*

In terms of parameterized complexity, directed tree-width is also quite useful. Although there is no known algorithm for computing the exact directed tree-width of a graph apart from a brute-force search, generalizing the approach used to compute tree-width in fixed parameter linear time gives us a fixed parameter tractable algorithm for computing an approximation of directed tree-width. This means that we can use directed tree-width in a similar role as tree-width for finding fixed parameter tractable algorithms for problems on directed graphs.

Johnson et al. conclude their paper by observing that several other more natural extensions of tree decompositions to directed graphs are not appropriate as they are not robust under simple graph operations. They highlight that one of the major problems with defining a notion of tree-width for directed graphs is that on directed graphs many other structural measures are not as closely linked as they are in the undirected case, as we saw in Theorem 4.12.

4.3 Beyond directed tree-width

So with a seemingly appropriate complexity measure defined, why is the generalization of tree-width to directed graphs still an interesting research problem? The answer is that directed tree-width does not seem to complete the whole picture. For a start, unlike with tree-width the definition is awkward, as is the given algorithm scheme, and it is difficult to gain an intuitive understanding. The structure of arboreal decompositions is not as flexible as tree decompositions, which means we cannot provide alternative forms of the decomposition which may be useful algorithmically (see, for example, Theorem 6.28). This makes it challenging to develop algorithms outside of those provided in [JRST01], suggesting directed tree-width is not as practical as it first appears.

In addition, contrary to the claims made in [JRST01], directed tree-width is not robust under some very simple graph operations. Adler [Adl05] has shown that directed tree-width may increase under the taking of butterfly minors (see Definition 8.28), and it appears that this can be extended to showing that directed tree-width may increase under the taking of subgraphs. However, it follows from Theorem 4.11, that this increase can only be by a constant factor, as havens are robust under these operations. While this means that algorithmically directed tree-width is still a useful measure of complexity, it lessens the importance of directed tree-width as a structural measure. This was further shown by Adler, with the following result which shows that havens are distinct from directed tree-width.

Theorem 4.14 ([Adl05])**.** *There exists a directed graph $\mathcal{G}$ with no haven of order 4 and directed tree-width 4.*

This implies that we cannot reduce the bound in the second part of Theorem 4.11 to obtain an equivalence between havens and directed tree-width.

Nevertheless, in the next chapter we show that Theorem 4.11 implies that directed tree-width at least approximates a good complexity measure for directed graphs. But the picture is still not complete. The problem is that on directed graphs there is a difference between connectivity and reachability – if there is a path from u to v it does not necessarily follow that u and v are in the same strongly connected component, and similarly, if u and v are in the same weakly connected component, there may not be a path from u to v. The tree-width of a directed graph can be seen as a measure of its weak connectivity, as tree-width is a connectivity measure that, on directed graphs, ignores edge direction. Likewise, the definitions of directed tree-width and its alternative characterizations suggest that directed tree-width is a measure of

the strong connectivity of a graph. So the question can be asked, "What, if anything, measures the reachability, or *directed connectivity*, of a directed graph?" In Chapters 6 and 7 we address this question, introducing two distinct, but closely related measures which seem to indicate directed connectivity. As strong connectedness implies reachability, and reachability implies weak connectedness, it is not surprising that these measures lie between tree-width and directed tree-width. We argue that as these measures are closer to tree-width than directed tree-width is, they are more practical as a complexity measure for directed graphs. In Chapter 8 we consider the structural implications of the question, endeavouring to find generalizations of havens, brambles and minors that correspond to our measures.

An interesting follow-up question is "Should a good complexity measure for directed graphs be invariant under edge reversal?" As many important structural features such as cycles or strongly connected sets are preserved under reversing edges, it would seem that a good structural measure would be invariant under this operation. However, from an algorithmic point of view edge direction is much more critical. Consider the problem of trying to find a path between two vertices when it is not easy to compute the edge relation, but it is relatively easy to compute the successors of a vertex. Such a problem might arise for instance if we were considering the computations of a Turing machine. On a tree where all edges are oriented away from a single vertex, finding such a path could involve a lot of back-tracking, but with all edges oriented towards a single vertex, the problem becomes significantly easier. Unlike directed tree-width, the measures we introduce in Chapters 6 and 7 are not invariant under the edge reversal operation, providing further evidence that they are more suitable extensions of tree-width from a practical point of view.

Chapter 5

Graph searching games

With a view to finding good complexity measures for directed graphs, we now turn our attention to a means of developing robust measures of graph complexity. We introduce a game played between two players, one controlling a fugitive located on the graph, and the other controlling a set of searchers whose purpose is to locate the fugitive. Such games are useful for describing problems such as trying to locate a virus in a network, or locate someone in a cave system. They can also be used to define measures of graph complexity: we obtain various complexity measures by considering variants of the game and the resources required by the searchers to locate the fugitive. Indeed, the tree-width of a graph can be characterized by the minimum number of searchers required to locate the fugitive in some of the variants we consider.

We first define a very general form of the game which encompasses many games considered in the literature, for example [ST93, KP86, BG04, DKT97, FFN05, GLS01, GM06]. This enables us to define some important concepts we use throughout the next few chapters: plays, searches, strategies and monotonicity. After demonstrating how this game includes other games considered in the literature, we introduce a general framework for developing measures of graph complexity. In Section 5.4, we show how these measures are robust under some basic graph operations such as taking subgraphs. Finally, we conclude the chapter by considering the complexity of the problem of determining these graph parameters.

5.1 Definitions

The definitions we present in this chapter are applicable to both directed and undirected graphs, though it is often necessary to assume we are working within only one of these classes. Thus we use the term *graph* to refer to a structure with a single, binary edge relation which may or may not be symmetric.

We recall from Definition 2.7, the definition of a *simple game*. The game we are interested in is a simple game played on an arena defined by the graph to be searched. That is,

Definition 5.1 (Graph searching game)**.** A *graph searching game type* is a function Γ which maps a graph $\mathcal{G}$ to a triple $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$ where $\mathcal{L}_s$ and $\mathcal{L}_f$ are sets of subsets of $\mathrm{Elts}(\mathcal{G})$ and $\mathcal{A}$ is an arena which satisfy:

- $\emptyset \in \mathcal{L}_s$,
- $\emptyset \notin \mathcal{L}_f$, and $\mathcal{L}_f$ has a unique $\subseteq$-maximal element $R_{\max}$,

- $V_0(\mathcal{A}) \subseteq \mathcal{L}_s \times \mathcal{L}_f$ consists of pairs (X, R) where $X \cap R = \emptyset$,
- $V_1(\mathcal{A}) \subseteq \mathcal{L}_s \times \mathcal{L}_s \times \mathcal{L}_f$ consists of triples of the form (X, X', R) where $X \cap R = \emptyset$,
- $v_I(\mathcal{A}) = (\emptyset, R_{\max})$,
- If $\big((X, R), (X', X'', R')\big) \in E(\mathcal{A})$ then $X = X'$ and $R = R'$,
- If $\big((X, X', R), (X'', R')\big) \in E(\mathcal{A})$ then $X' = X''$ and for all $r' \in R'$ there is $r \in R$ such that r and r' are in the same (weakly) connected component of $\mathcal{G} \setminus (X \cap X')$, and
- If $S \subseteq R$, then for all S' such that $\big((X, X', S), (X', S')\big) \in E(\mathcal{A})$, there exists $R' \supseteq S'$ such that $\big((X, X', R), (X', R')\big) \in E(\mathcal{A})$.

Given a graph searching game type Γ, and a graph $\mathcal{G}$, with $\Gamma(\mathcal{G}) = (\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$ the *graph searching game on $\mathcal{G}$ (defined by $\Gamma(\mathcal{G})$)* is the simple game $\mathbb{G}^{\Gamma}_{\mathcal{G}} := (\mathcal{A}, \mathcal{F})$, where $\mathcal{F} = \emptyset$, so Player 1 wins all infinite plays. In a graph searching game elements of $V_0(\mathcal{A})$ are called *positions (of the game)*, elements of $V_1(\mathcal{A})$ are called *intermediate positions*, and we call Player 0 the *searchers* and Player 1 the *fugitive*.

Intuitively, the game works as follows. A graph searching game on $\mathcal{G}$ is a game played by a number of co-operating searchers against an omniscient fugitive. All entities occupy elements of $\mathcal{G}$, however, while the locations of the searchers are known to everyone, the location of the fugitive is not necessarily known, so the fugitive "occupies" a set of potential locations. When the game is at the position (X, R), $X \in \mathcal{L}_s$ represents the location of the searchers, and $R \in \mathcal{L}_f$ represents the set of potential fugitive locations. The initial position, $(\emptyset, R_{\max})$, thus indicates that at the beginning there are no searchers on $\mathcal{G}$ and the fugitive may be anywhere on $R_{\max}$. The searchers and fugitive move around $\mathcal{G}$, but, as indicated by the edge relation of the arena, only the fugitive is necessarily constrained by the topology of $\mathcal{G}$.

From position (X, R), the searchers, if possible, choose a new set of locations X'. If this is not possible then the fugitive has escaped and he wins. Otherwise, the game proceeds to the intermediate position (X, X', R). For ease of later descriptions, we say the searchers on $X \setminus X'$ have been *removed* while the searchers on $X \cap X'$ remain *stationary* and the searchers on $X' \setminus X$ will be *placed* after the fugitive has completed his move.

The fugitive responds to the move of the searchers at each of his potential locations, but he is not permitted to pass through any stationary searchers. However, he is omniscient and is aware of the impending occupation of $X' \setminus X$ by the searchers that will be placed, and can modify his response accordingly. The final condition in the definition of the arena of a graph searching game asserts that the responses of the fugitive at each of his potential locations are somewhat independent: if the set of potential locations is increased, then so are the sets of his potential responses. Some information about the response of the fugitive may be available to the searchers, resulting in a (visible) choice for the fugitive about the next set, R', of his potential locations. If he has no such choice and no possible location to move to ($R' = \emptyset$), then he has been captured and the searchers win. Otherwise, the game proceeds to the position (X', R'). This whole process is represented in the graph searching game by moving the token from (X, R) to the vertex (X, X', R), and then to (X', R'). If the fugitive can avoid capture forever, then again he has escaped and he wins.

From this we can see that an arena of a graph searching game on $\mathcal{G}$ can be described by defining the set of positions and a set of legal transitions between positions, essentially "ignoring" non-terminal intermediate positions. It follows that all plays ending with a move from the fugitive can be fully described as a sequence of positions:

$$(X_0, R_0)(X_1, R_1) \cdots (X_n, R_n)$$

where $(X_0, R_0) = (\emptyset, R_{\max})$ and for $0 \leq i < n$ and for all $r' \in R_{i+1}$ there is $r \in R_i$ such that r and r' are in the same connected component of $\mathcal{G} \setminus (X_i \cap X_{i+1})$. We extend this to include plays that are winning for the searchers by using $R_n = \emptyset$ to indicate that the play ended at (X_{n-1}, X_n, R_{n-1}). This motivates the following definition:

Definition 5.2 (Search)**.** Let $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ be a graph searching game on $\mathcal{G}$ defined by $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$, and let $(X_1, R_1) \in V_0(\mathcal{A})$. A *proper search from* (X_1, R_1) in $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ is a (possibly infinite) sequence, $(X_1, R_1)(X_2, R_2) \cdots$, such that for all $i \geq 1$:

- $(X_i, R_i) \in V_0(\mathcal{A})$,
- $\big((X_i, R_i), (X_i, X_{i+1}, R_i)\big) \in E(\mathcal{A})$, and
- $\big((X_i, X_{i+1}, R_i), (X_{i+1}, R_{i+1})\big) \in E(\mathcal{A})$.

A *complete search from* (X_1, R_1) in $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ is a finite sequence $(X_1, R_1) \cdots (X_n, R_n)$ such that

- $(X_1, R_1) \cdots (X_{n-1}, R_{n-1})$ is a proper search from (X_1, R_1) in $\mathbb{G}^{\Gamma}_{\mathcal{G}}$,
- $\big((X_{n-1}, R_{n-1}), (X_{n-1}, X_n, R_{n-1})\big) \in E(\mathcal{A})$,
- $(X_{n-1}, X_n, R_{n-1})\big) \in V_1(\mathcal{A})$ has no outgoing edges, and
- $R_n = \emptyset$.

A *search* in $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ is a sequence which is either a proper or a complete search. A search π can be *extended* to a search π', if π is a prefix of π'. A search from $v_I(\mathcal{A})$ is *winning for the searchers* if it can be extended to a complete search, otherwise it is *winning for the fugitive*.

In the sequel we will generally adopt this representation of plays as we are primarily concerned with the game from the perspective of the searchers.

Variants of graph searching games are obtained by restricting the moves available to the searchers and the fugitive, in other words, by placing restrictions on the arena on which the game is played. Before we consider some examples, we introduce some definitions and results relating to strategies.

5.1.1 Strategies

Since a graph searching game is a simple game, it follows that the winner is determined by reachability, and therefore if either the fugitive or the searchers have a winning strategy, they have a memoryless strategy. However, in this chapter we are interested in *resource bounded* winning strategies, and in this case memoryless strategies, indeed, even finite memory strategies may no longer be sufficient. However, to ensure that computing such strategies remains decidable, we impose restrictions on the resource measures we consider so that searches consistent with strategies are only ever simple paths in the arena. This motivates the definition of a *history-dependent strategy*.

Definition 5.3 (History-dependent strategy)**.** Let $\mathcal{G}$ be a graph, and $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ a graph searching game on $\mathcal{G}$ defined by $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$. Given a set Σ, a *history-dependent strategy for the searchers* is a partial function $\sigma : \Sigma^* \times \mathcal{L}_s \times \mathcal{L}_f \to \Sigma \times \mathcal{L}_s$ such that:

- $\sigma(\epsilon, X_0, R_0)$ is defined for the empty word ϵ, and $(X_0, R_0) = v_I(\mathcal{A})$,
- If $\sigma(w, X, R) = (a, X')$ for $(X, R) \in V_0(\mathcal{A})$, then
  - $(X, X', R) \in V_1(\mathcal{A})$,
  - there is an edge in $E(\mathcal{A})$ from (X, R) to (X, X', R), and
  - if there is an edge in $E(\mathcal{A})$ from (X, X', R) to $(X', R') \in V_0(\mathcal{A})$ then $\sigma(w{\cdot}a, X', R')$ is defined.

We say a search $\pi = (X_0, R_0)(X_1, R_1)\cdots$ is *consistent* with σ if there exists a word $w = a_1 a_2 \cdots \in \Sigma^* \cup \Sigma^\omega$ such that for all $i \geq 0$, $X_{i+1} = \sigma(a_1 \cdots a_i, X_i, R_i)$. We call w the *history consistent* with π.

Remark. In the sequel we will usually define history-dependent strategies inductively, often omitting the associated history when it is clear from the context what the play to a given position should be.

Nevertheless, we show in Section 5.3 that the resource bounded strategies we are primarily concerned with are equivalent to winning strategies in a graph searching game. For this reason, we reserve the definition of *strategies* for positional strategies.

Definition 5.4 (Strategy)**.** Let $\mathcal{G}$ be a graph, and $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ a graph searching game on $\mathcal{G}$ defined by $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$. A *strategy for the searchers* is a partial function, $\sigma : \mathcal{L}_s \times \mathcal{L}_f \to \mathcal{L}_s$, such that if $\sigma(X, R)$ is defined there is an edge in $E(\mathcal{A})$ from (X, R) to $(X, \sigma(X, R), R)$. If $\pi = (X_0, R_0)(X_1, R_1)\cdots$ is a search in $\mathbb{G}^{\Gamma}_{\mathcal{G}}$, we say π is *consistent* with σ if for all $i \geq 0$, $X_{i+1} = \sigma(X_i, R_i)$. We say σ is *winning (for the searchers)* if every search from $v_I(\mathcal{A})$ consistent with σ is winning for the searchers.

A *strategy for the fugitive* is a partial function $\rho : \mathcal{L}_s \times \mathcal{L}_s \times \mathcal{L}_f$ such that if $(X, X', R) \in V_1(\mathcal{A})$, there is an edge in $E(\mathcal{A})$ from (X, X', R) to $(X', \rho(X, X', R))$. If $\pi = (X_0, R_0)(X_1, R_1)\cdots$ is a search in $\mathbb{G}^{\Gamma}_{\mathcal{G}}$, we say π is *consistent* with ρ if for all $i \geq 0$, $R_{i+1} = \sigma(X_i, X_{i+1}, R_i)$. We say ρ is *winning (for the fugitive)* if every search from $v_I(\mathcal{A})$ consistent with ρ is winning for the fugitive.

Given a strategy σ for the searchers and a strategy ρ for the fugitive, the unique maximal search consistent with σ and ρ is the *search defined by σ and ρ*

We now use strategies to define a structure that will prove useful in the next few chapters. Given a strategy σ for the searchers in a graph searching game $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ defined by $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$, we see that σ induces a subgraph of $\mathcal{A}$ in the following way. Let $V \subseteq V(\mathcal{A})$ be the set of positions and intermediate positions reached by some play from $v_I(\mathcal{A})$ consistent with σ. Considering for the moment positional strategies, it follows that from each position $(X, R) \in V$ there is precisely one successor $(X, X', R) \in V$, namely the element of $V_1(\mathcal{A})$ with $X' = \sigma(X, R)$. The structure we are interested in is a slight variation of this subgraph where, just as with our policy for describing searches, the intermediate positions are ignored.

Definition 5.5 (Strategy digraph)**.** Let $\mathcal{G}$ be a graph and $\mathbb{G}_{\mathcal{G}}^{\Gamma}$ a graph searching game on $\mathcal{G}$ defined by $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$. Let σ be a strategy for the searchers. The *strategy digraph defined by* σ, $\mathcal{D}_\sigma$, is the directed graph defined as:

- $V(\mathcal{D}_\sigma)$ is the set of all pairs (X, R), including "positions" of the form $(X, \emptyset)$, such that there is some search in $\mathbb{G}_{\mathcal{G}}^{\Gamma}$, $(X_0, R_0)(X_1, R_1)\cdots$, from $v_I(\mathcal{A}) = (X_0, R_0)$ and consistent with σ, with $(X, R) = (X_i, R_i)$ for some i.
- There is an edge from (X, R) to (X', R') in $E(\mathcal{D}_\sigma)$ if $X' = \sigma(X, R)$ and either there is an edge from (X, X', R) to (X', R') in $E(\mathcal{A})$, or there are no edges from (X, X', R) in $E(\mathcal{A})$ and $R' = \emptyset$.

Remark. Sometimes it may be convenient to assume that nodes of the form $(X', \emptyset)$ of a strategy digraph are duplicated so that each such position actually corresponds to a vertex (X, X', R) in $V_1(\mathcal{A})$. When this is the case, we see that every leaf of the form $(X, \emptyset)$ has a unique predecessor: if $(X', \emptyset)$ is associated with (X, X', R) then (X, R) is the unique predecessor of $(X', \emptyset)$. We observe that after these duplications, we still have $|V(\mathcal{D}_\sigma)| \leq |V(\mathcal{A})|$.

An observation that will prove useful concerns the form the strategy digraph takes for winning strategies.

Lemma 5.6. *Let $\mathcal{G}$ be a graph and $\mathbb{G}_{\mathcal{G}}^{\Gamma}$ a graph searching game on $\mathcal{G}$ defined by $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$. If σ is a winning strategy for the searchers then $\mathcal{D}_\sigma$ is a directed acyclic graph and all leaves of $\mathcal{D}_\sigma$ are of the form $(X, \emptyset)$.*

Proof. We observe that from the definition, there is a path from $v_I(\mathcal{A}) = (X_0, R_0)$ to every node $(X, R) \in V(\mathcal{D}_\sigma)$. We also observe that every path $(X_0, R_0)(X_1, R_1)\cdots$ in $\mathcal{D}_\sigma$ from $v_I(\mathcal{A})$ corresponds to a search in $\mathbb{G}_{\mathcal{G}}^{\Gamma}$ consistent with σ, and if (X, R) is a leaf then there is no search consistent with σ extending any consistent search which ends at (X, R). Since σ is a winning strategy for the searchers, all searches consistent with σ can be extended to a complete search. Thus, if (X, R) is a leaf, it follows that all searches from (X_0, R_0) which end at (X, R) must be complete, so $R = \emptyset$. To show acyclicity, it suffices to show that if $\mathcal{D}_\sigma$ is not acyclic, then σ is not a winning strategy for the searchers. Suppose $(Y_1, S_1)\cdots(Y_m, S_m)$ is a cycle in $\mathcal{D}_\sigma$. By our earlier observation, $\pi = (Y_1, S_1)\cdots(Y_m, S_m)(Y_1, S_1)$ is a search from (Y_1, S_1) consistent with σ. Now from the definition of $V(\mathcal{D}_\sigma)$, there exists a search $\pi' = (X_0, R_0)\cdots(X_k, R_k)$, where $(X_k, R_k) = (Y_1, S_1)$ consistent with σ from $(X_0, R_0) = v_I(\mathcal{A})$. Therefore, the infinite search

$$\pi' \cdot \pi \cdot \pi \cdots = (X_0, R_0)\cdots(Y_1, S_1)\cdots(Y_m, S_m), (Y_1, S_1)\cdots$$

is a search from $v_I(\mathcal{A})$ consistent with σ. As this cannot possibly be extended to a finite search and the fugitive wins all infinite plays, it follows that σ is not a winning strategy for the searchers. □

Definition 5.7 (Strategy DAG)**.** Let $\mathcal{G}$ be a graph and $\mathbb{G}_{\mathcal{G}}^{\Gamma}$ a graph searching game on $\mathcal{G}$. If σ is a winning strategy for the searchers then we call $\mathcal{D}_\sigma$ the *strategy DAG* defined by σ.

One important property of plays, searches and strategies that we are interested in is the concept of monotonicity. In particular, we concentrate on two types of monotonicity: fugitive-monotonicity, where the set of potential fugitive locations is always non-increasing, and searcher-monotonicity, where no location vacated by a searcher is ever re-occupied.

Definition 5.8 (Fugitive and Searcher Monotonicity)**.** Let $\mathcal{G}$ be a graph and let $\pi = (X_0, R_0)(X_1, R_1)\cdots$ be a search in a graph searching game on $\mathcal{G}$. We say π is

- *fugitive-monotone* if $R_i \supseteq R_{i+1}$ for all $i \geq 0$, and
- *searcher-monotone* if $X_i \cap X_k \subseteq X_j$ for $0 \leq i \leq j \leq k$.

A strategy, σ, for the searchers in a graph searching game on $\mathcal{G}$ is *fugitive-monotone* (*searcher-monotone*) if every search consistent with σ is fugitive-monotone (searcher-monotone).

Our next result concerning strategies in the general graph searching game is a useful observation regarding monotone strategies. We show that, under some simple assumptions, a searcher-monotone winning strategy must also be fugitive-monotone. Let us say that a graph searching game *permits idling* if the fugitive is able to remain at any location which is not about to be occupied by a searcher. Furthermore, let us say that a graph searching game is *vacating sensitive* if, whenever some location becomes available to the fugitive, there must be some location, previously occupied by a searcher, that the fugitive can now occupy. More precisely,

Definition 5.9 (Permits idling)**.** Let $\mathcal{G}$ be a graph and $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ a graph searching game on $\mathcal{G}$ defined by $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$. We say $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ *permits idling* if for all $(X, X', R) \in V_1(\mathcal{A})$ and all $r \in R \setminus X'$, there exists $R' \subseteq \text{Elts}(\mathcal{G})$ such that $r \in R'$ and there is an edge in $E(\mathcal{A})$ from (X, X', R) to (X', R').

Definition 5.10 (Vacating sensitive)**.** Let $\mathcal{G}$ be a graph and $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ a graph searching game on $\mathcal{G}$ defined by $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$. We say that $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ is *vacating sensitive* if, whenever there is an edge in $E(\mathcal{A})$ from (X, X', R) to (X', R') with $R' \not\subseteq R$, then $X \cap R' \neq \emptyset$.

Lemma 5.11. *Let $\mathcal{G}$ be a graph and $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ a graph searching game on $\mathcal{G}$ which permits idling and is vacating sensitive. If σ is a searcher-monotone winning strategy for the searchers on $\mathbb{G}^{\Gamma}_{\mathcal{G}}$, then σ is fugitive-monotone.*

Proof. Suppose $\pi = (X_0, R_0)(X_1, R_1)\cdots$ is a search consistent with σ which is not fugitive-monotone. Let i be the least index such that $R_i \not\supseteq R_{i+1}$. Since $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ is vacating sensitive, there exists $r \in X_i \cap R_{i+1}$. But then, as $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ permits idling, the fugitive can always choose a response which includes r until it is occupied by a searcher. That is, there is a search $\pi' = (X'_0, R'_0)(X'_1, R'_1)\cdots$, consistent with σ, which agrees with π up to (X_{i+1}, R_{i+1}) and either there is some k such that $r \in R'_j$ for all j with $i+1 \leq j < k$ and $r \in X_k$, or $r \in R'_j$ for all $j \geq i+1$. In the first case, we have $r \in X'_i \cap X'_k$ but as $r \in R'_{i+1}$, we also have $r \notin X'_{i+1}$, contradicting the fact that σ is searcher-monotone. In the second case, since $R'_j \neq \emptyset$ for all j, it follows that π' is an infinite search, contradicting the fact that σ is a winning strategy for the searchers. □

Remark. Earlier, we asserted that variations of graph searching games are obtained by imposing restrictions on the arena. In this way, we see that questions relating to fugitive-monotone strategies can be viewed as questions in a restricted version of the game: the game defined in the same way with the restriction that we do not allow the searchers to make any move which enables the fugitive to make a non-monotone move (a move for which the set of potential fugitive locations is not non-increasing). That is, if $\mathcal{A}$ is the arena of a graph searching game, let $\mathcal{A}'$ be the arena obtained by removing edges from (X, R) to (X, X', R) if there is an edge from (X, X', R) to (X', R') where $R' \not\subseteq R$. Now a strategy for the searchers on $\mathcal{A}'$ is a fugitive-monotone strategy

for the searchers on $\mathcal{A}$. On the other hand, searcher-monotonicity is a more dynamic restriction – the moves available to the searchers are dependent on the play to that point. Lemma 5.11 illustrates how, in some cases, the strategy restrictions imposed by searcher-monotonicity can also be interpreted as restrictions on the game.

5.1.2 Simulations

In Definition 2.20, we saw the idea of a *game simulation*. We now introduce a refinement of this suitable for graph searching games.

Definition 5.12 (Searching simulation)**.** Let $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ be a graph searching game on $\mathcal{G}$ defined by $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$, and $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ be a graph searching game on $\mathcal{G}'$ defined by $(\mathcal{L}'_s, \mathcal{L}'_f, \mathcal{A}')$. A *searching simulation* from $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ to $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ is a pair of relations (R_s, R_f) such that:

- $R_s \subseteq \mathcal{L}_s \times \mathcal{L}'_s$, $R_f \subseteq \mathcal{L}_f \times \mathcal{L}'_f$, and
- The relation $\mathbf{S}$ on $V(\mathcal{A}) \times V(\mathcal{A}')$ defined by
  - $(X, R)\ \mathbf{S}\ (Y, R')$ if $(X, Y) \in R_s$ and $(R, R') \in R_f$, and
  - $(X, X', R)\ \mathbf{S}\ (Y, Y', R')$ if $(X, Y), (X', Y') \in R_s$ and $(R, R') \in R_f$,

  is a game simulation from $\mathcal{A}$ to $\mathcal{A}'$.

As a searching simulation is a restricted game simulation, and searches correspond to plays in the arena, the next result follows immediately from Lemma 2.21.

Lemma 5.13. *Let $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ be a graph searching game on $\mathcal{G}$ defined by $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$, and $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ be a graph searching game on $\mathcal{G}'$ defined by $(\mathcal{L}'_s, \mathcal{L}'_f, \mathcal{A}')$. Let (R_s, R_f) be a searching simulation from $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ to $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ with $(\emptyset, \emptyset) \in R_f$. For all searcher strategies σ on $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ and all fugitive strategies ρ' on $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$, there exists a searcher strategy σ' on $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ and a fugitive strategy ρ on $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ such that if $\pi_{(\sigma,\rho)} = (X_1, R_1)(X_2, R_2)\cdots$ is the search in $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ defined by σ and ρ, and $\pi_{(\sigma',\rho')} = (X'_1, R'_1)(X'_2, R'_2)\cdots$ is the search in $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ defined by σ' and ρ', then $|\pi_{(\sigma,\rho)}| = |\pi_{(\sigma',\rho')}|$, and $(X_i, X'_i) \in R_s$ and $(R_i, R'_i) \in R_f$ for all $i \leq |\pi_{(\sigma,\rho)}|$.*

As with game simulations, we observe that the definition of the strategy σ' is independent of the choice of ρ. This gives us the following analogue to Corollary 2.22:

Corollary 5.14. *Let $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ be a graph searching game on $\mathcal{G}$, and $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ be a graph searching game on $\mathcal{G}'$. Let (R_s, R_f) be a searching simulation from $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ to $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ with $(\emptyset, \emptyset) \in R_f$, and let σ be a strategy for the searchers on $\mathbb{G}^{\Gamma}_{\mathcal{G}}$. Then there exists a strategy σ' for the searchers on $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ such that for every search $(X'_1, R'_1)(X'_2, R'_2)\cdots$ consistent with σ' there exists a search $(X_1, R_1)(X_2, R_2)\cdots$ consistent with σ with $(X_i, X'_i) \in R_s$ and $(R_i, R'_i) \in R_f$ for all $i \geq 1$.*

As with game simulations, we call the strategies which we can derive from a simulation *simulated strategies*.

Definition 5.15 (Simulated search strategy)**.** The strategy σ' in Corollary 5.14 is called a *strategy (R_s, R_f)-simulated by σ*.

This enables us to state the following consequence of Corollary 2.27.

Lemma 5.16. *Let $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ be a graph searching game on $\mathcal{G}$ and $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ a graph searching game on $\mathcal{G}'$. Let (R_s, R_f) be a searching simulation from $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ to $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$, and let σ be a strategy for the searchers on $\mathbb{G}^{\Gamma}_{\mathcal{G}}$. If σ' is a strategy (R_s, R_f)-simulated by σ on $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$, then:*

1. *If σ is a winning strategy, then σ' is a winning strategy, and*
2. *If $(X, X') \in R_s$ and $(R, R') \in R_f$, then $\big(\sigma(X, R), \sigma'(X', R')\big) \in R_s$.*

With some straightforward assumptions about the relations which comprise a searching simulation, we can show that strategies simulated by monotone strategies are also monotone. First we recall two definitions regarding relations of sets.

Definition 5.17 (Monotone and $\cap$-compatible relation)**.** Let X and Y be sets, and let $R \subseteq \mathcal{P}(X) \times \mathcal{P}(Y)$ be a relation between subsets of X and subsets of Y. We say R is *monotone* if for all $(A, A'), (B, B') \in R$ with $A \subseteq B$, we have $A' \subseteq B'$. We say R is $\cap$*-compatible* if for all $(A, A'), (B, B') \in R$, $(A \cap B, A' \cap B') \in R$.

Lemma 5.18. *Let $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ be a graph searching game on $\mathcal{G}$ and $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ a graph searching game on $\mathcal{G}'$. Let (R_s, R_f) be a searching simulation from $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ to $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$, and let σ is a strategy for the searchers on $\mathbb{G}^{\Gamma}_{\mathcal{G}}$. If σ' is a strategy (R_s, R_f)-simulated by σ on $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$, then:*

1. *If R_f is monotone and σ is fugitive-monotone, then σ' is fugitive-monotone, and*
2. *If R_s is monotone and $\cap$-compatible and σ is searcher-monotone, then σ' is searcher-monotone.*

Proof. Let $\pi' = (X'_1, R'_1)(X'_2, R'_2)\cdots$ be a search consistent with σ'. By the definition of simulated strategies, there exists a search $(X_1, R_1)\cdots$ consistent with σ such that $(X_i, X'_i) \in R_s$ and $(R_i, R'_i) \in R_f$ for all $i \geq 1$.

1: If σ is fugitive-monotone, then $R_i \supseteq R_{i+1}$ for all $i \geq 1$, so if R_f is monotone, it follows that $R'_i \supseteq R'_{i+1}$ for all $i \geq 1$. Thus π' is fugitive monotone, and as π' was arbitrary, it follows that σ' is fugitive-monotone.

2: If σ is searcher-monotone, then for all $i \leq j \leq k$, we have $X_i \cap X_k \subseteq X_j$. If R_s is $\cap$-compatible, then $(X_i \cap X_k, X'_i \cap X'_k) \in R_s$, and so if R_s is also monotone, then $X'_i \cap X'_k \subseteq X'_j$. Thus π' is searcher-monotone, and as π' was arbitrary, it follows that σ' is searcher-monotone. □

We now introduce some concepts that will prove useful later when we establish robustness results for graph searching games.

Definition 5.19 (Quasi-simulation family)**.** A *quasi-simulation family* is a partial function $\mathfrak{R}$ which assigns to a pair of graphs $(\mathcal{G}, \mathcal{G}')$ a pair of relations (R'_s, R'_f) with $R'_s, R'_f \subseteq \mathcal{P}(\text{Elts}(\mathcal{G})) \times \mathcal{P}(\text{Elts}(\mathcal{G}'))$.

Often it is easier to define a quasi-simulation family as a pair of partial functions $(\mathfrak{R}_s, \mathfrak{R}_f)$, each of which takes a pair of graphs $(\mathcal{G}, \mathcal{G}')$ to a relation from $\mathcal{P}(\text{Elts}(\mathcal{G}))$ to $\mathcal{P}(\text{Elts}(\mathcal{G}'))$

Definition 5.20 ($\mathfrak{R}$-closure)**.** Let $\mathfrak{R}$ be a quasi-simulation family, and Γ a graph searching type. We say Γ is $\mathfrak{R}$-*closed* if for any pair of graphs $\mathcal{G}$ and $\mathcal{G}'$ with $\mathfrak{R}(\mathcal{G}, \mathcal{G}') = (R'_s, R'_f)$, $\Gamma(\mathcal{G}) = (\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$ and $\Gamma(\mathcal{G}') = (\mathcal{L}'_s, \mathcal{L}'_f, \mathcal{A}')$; (R_s, R_f) is a searching simulation from $\mathbb{G}^{\Gamma}_{\mathcal{G}'}$ to $\mathbb{G}^{\Gamma}_{\mathcal{G}}$, where $R_s = R'_s \cap (\mathcal{L}'_s \times \mathcal{L}_s)$ and $R_f = R'_f \cap (\mathcal{L}'_f \times \mathcal{L}_f)$.

To help gain an intuition, we provide an example of $\mathfrak{R}$-closure. Consider the following property of graph searching game types.

Definition 5.21 (Respects restriction)**.** Let Γ be a graph searching game type. We say Γ *respects restriction* if for any graphs $\mathcal{G}$ and $\mathcal{G}'$ such that $\mathcal{G}$ is a subgraph of $\mathcal{G}'$, if $\Gamma(\mathcal{G}) = (\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$ and $\Gamma(\mathcal{G}') = (\mathcal{L}'_s, \mathcal{L}'_f, \mathcal{A}')$, then

- If $R_{\max}$ is the $\subseteq$-maximal element of $\mathcal{L}_f$, and $R'_{\max}$ is the $\subseteq$-maximal element of $\mathcal{L}'_f$, then $R_{\max} = R'_{\max} \cap \text{Elts}(\mathcal{G})$.
- If there is an edge from (X, R) to (X, X', R) in $E(\mathcal{A}')$ and $v = (X \cap \text{Elts}(\mathcal{G}), R \cap \text{Elts}(\mathcal{G})) \in V(\mathcal{A})$, then there is an edge from v to $(X \cap \text{Elts}(\mathcal{G}), X' \cap \text{Elts}(\mathcal{G}), R \cap \text{Elts}(\mathcal{G}))$ in $E(\mathcal{A})$, and
- If there is an edge from (Y, Y', S) to (Y', S') in $E(\mathcal{A})$ then for all X, X', R such that $(X, X', R) \in V_1(\mathcal{A}')$, $Y = X \cap \text{Elts}(\mathcal{G})$, $Y' = X' \cap \text{Elts}(\mathcal{G})$, and $S = R \cap \text{Elts}(\mathcal{G})$, there exists R' such that $S' = R' \cap \text{Elts}(\mathcal{G})$ and there is an edge from (X, X', R) to (X', R') in $E(\mathcal{A}')$.

Intuitively, if a graph searching game type respects restriction, then if $\mathcal{G}$ is a subgraph of $\mathcal{G}'$, a strategy for the searchers in $\mathcal{G}'$ is also a strategy in $\mathcal{G}$ when we disregard the elements of $\mathcal{G}'$ which are not part of $\mathcal{G}$. In other words, a restriction of a search strategy is a search strategy of a restriction. In Section 5.4 we introduce the dual notion, restriction reflection, in which a search strategy of a graph can be viewed as a search strategy in any larger graph. We now show that this property corresponds to an $\mathfrak{R}$-closure for a quasi-simulation family $\mathfrak{R}$ of relations similar to the superset relation.

Definition 5.22 ($\supset$)**.** For each pair of graphs $(\mathcal{G}', \mathcal{G})$, with $\mathcal{G}$ a subgraph of $\mathcal{G}'$, we define $\supset_{\mathcal{G}}^{\mathcal{G}'} \subseteq \mathcal{P}(\text{Elts}(\mathcal{G}')) \times \mathcal{P}(\text{Elts}(\mathcal{G}))$ as follows. For $A \subseteq \text{Elts}(\mathcal{G}')$ and $B \subseteq \text{Elts}(\mathcal{G})$ we say $A \supset_{\mathcal{G}}^{\mathcal{G}'} B$ if $B = A \cap \text{Elts}(\mathcal{G})$. Let $\supset$ denote the function which assigns to each pair of graphs $(\mathcal{G}', \mathcal{G})$, with $\mathcal{G}$ a subgraph of $\mathcal{G}'$, the pair of relations $(\supset_{\mathcal{G}}^{\mathcal{G}'}, \supset_{\mathcal{G}}^{\mathcal{G}'})$.

Lemma 5.23. *Let Γ be a graph searching game type. Then Γ respects restriction if, and only if, Γ is $\supset$-closed.*

Proof. Let $\mathcal{G}$ and $\mathcal{G}'$ be graphs. We observe that if neither $\mathcal{G}$ is a subgraph of $\mathcal{G}'$ nor $\mathcal{G}'$ is a subgraph of $\mathcal{G}$ then nothing can be said about whether Γ respects restriction or whether Γ is $\supset$-closed. Thus we assume without loss of generality that $\mathcal{G}$ is a subgraph of $\mathcal{G}'$. Let $\Gamma(\mathcal{G}) = (\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$ and $\Gamma(\mathcal{G}') = (\mathcal{L}'_s, \mathcal{L}'_f, \mathcal{A}')$. For convenience we will drop the subscript and superscript and use $\supset$ to denote the relation $\supset_{\mathcal{G}}^{\mathcal{G}'}$.

First let us assume Γ respects restriction. From the definition of $\supset$, we have $\text{Elts}(\mathcal{G}') \supset \text{Elts}(\mathcal{G})$, thus we must show that $(\supset, \supset)$ is a searching simulation from $\mathbb{G}^{\Gamma}_{\mathcal{G}'}$ to $\mathbb{G}^{\Gamma}_{\mathcal{G}}$. In the definition of $\mathfrak{R}$-closure, we assume $\supset$ is restricted to be a relation on the appropriate sets, so it suffices to show that the relation defined by pointwise application of $\supset$ is a game simulation from $\mathcal{A}'$ to $\mathcal{A}$. For convenience we will also denote the pointwise relation by $\supset$. Clearly, since $\emptyset \cap \text{Elts}(\mathcal{G}) = \emptyset$, we have $\emptyset \supset \emptyset$. Furthermore, if $R_{\max}$ is the $\subseteq$-maximal element of $\mathcal{L}_f$ and $R'_{\max}$ is the $\subseteq$-maximal element of $\mathcal{L}'_f$, then as Γ respects restriction, $R_{\max} = R'_{\max} \cap \text{Elts}(\mathcal{G})$. Thus $R'_{\max} \supset R_{\max}$, and $(\emptyset, R'_{\max}) \supset (\emptyset, R_{\max})$. Thus $(\supset, \supset)$ satisfies (SIM-1). Now suppose there is an edge from (X, R) to (X, X', R) in $\mathcal{A}'$ and $(X, R) \supset (Y, S)$. From the definition

of $\supset$, $Y = X \cap \text{Elts}(\mathcal{G})$ and $S = R \cap \text{Elts}(\mathcal{G})$, so by the definition of respecting restriction, there is an edge from (Y, S) to $(Y, X' \cap \text{Elts}(\mathcal{G}), S)$ in $\mathcal{A}$. Since clearly $X' \supset (X' \cap \text{Elts}(\mathcal{G}))$, it follows that (SIM-2) is satisfied. Finally suppose there is an edge in $\mathcal{A}$ from (Y, Y', S) to (Y', S') and $(Y, Y', S) \supset (X, X', R)$. From the definition of $\supset$, we have $Y = X \cap \text{Elts}(\mathcal{G})$, $Y' = X' \cap \text{Elts}(\mathcal{G})$ and $S = R \cap \text{Elts}(\mathcal{G})$. Thus, as Γ respects restriction, there exists $R' \in \mathcal{L}'_f$ such that $S' = R' \cap \text{Elts}(\mathcal{G})$ and there is an edge in $\mathcal{A}$ from (X, X', R) to (X', R'). Since $X' \supset Y'$ and $R' \supset S'$, it follows that $(X', R') \supset (Y', S')$, thus (SIM-3) is satisfied. Therefore, $(\supset, \supset)$ is a searching simulation from $\mathbb{G}^{\Gamma}_{\mathcal{G}'}$ to $\mathbb{G}^{\Gamma}_{\mathcal{G}}$. Since $\mathcal{G}$ and $\mathcal{G}'$ were arbitrary, it follows that Γ is $\supset$-closed.

Now suppose Γ is $\supset$-closed. Since the relation defined by pointwise application of $\supset$ is a game simulation from $\mathcal{A}'$ to $\mathcal{A}$, $v_I(\mathcal{A}) = (\emptyset, R_{\max})$, and $v_I(\mathcal{A}') = (\emptyset, R'_{\max})$, it follows from (SIM-1) that $\emptyset \supset \emptyset$ and $R'_{\max} \supset R_{\max}$. From the definition of $\supset$, it follows that $R_{\max} = R'_{\max} \cap \text{Elts}(\mathcal{G})$. Now suppose there is an edge from (X, R) to (X, X', R) in $\mathcal{A}'$, and $(Y, S) \in V(\mathcal{A})$ where $Y = X \cap \text{Elts}(\mathcal{G})$ and $S = R \cap \text{Elts}(\mathcal{G})$. From the definition of $\supset$, it follows that $X \supset Y$ and $R \supset S$, thus as $(\supset, \supset)$ is a game simulation, it follows from (SIM-2) that there exists v' such that there is an edge from (Y, S) to v' and v' is related to (X, X', R) by the pointwise application of $\supset$. By the definition of graph searching games, $v' = (Y, Y', S)$ for some $Y' \in \mathcal{L}_s$, and by the definition of searching simulation $X' \supset Y'$. Thus $Y' = X' \cap \text{Elts}(\mathcal{G})$. Finally suppose there is an edge from (Y, Y', S) to (Y', S') and X, X', R are such that $Y = X \cap \text{Elts}(\mathcal{G})$, $Y' = X' \cap \text{Elts}(\mathcal{G})$ and $S = R \cap \text{Elts}(\mathcal{G})$. From the definition of $\supset$, $X \supset Y$, $X' \supset Y'$ and $R \supset S$. Thus, from (SIM-3), there exists $v \in V_0(\mathcal{A}')$ such that there is an edge from (X, X', R) to v and v is related to (Y', S'). From the definition of graph searching games, $v = (X', R')$ for some R', and by the definition of searching simulation, $R' \supset S'$. Thus $S' = R' \cap \text{Elts}(\mathcal{G})$. Therefore, all conditions necessary for respecting restriction are satisfied. Since $\mathcal{G}$ and $\mathcal{G}'$ were arbitrary, it follows that Γ respects restriction. □

5.2 Examples

We now look at some examples of graph searching game types which occur in the literature. Many of these examples were introduced to provide an intuitive understanding of some of the graph parameters we discussed in the previous chapter. We show how each of these games can be described using the framework we have introduced, thereby motivating the use of graph searching games to formally define measures of graph complexity.

5.2.1 Cops and visible robber

The *cops and visible robber game* was introduced in [ST93] to provide a characterization of tree-width. We can define it as a graph searching game played on an undirected graph $\mathcal{G}$, as follows.

Definition 5.24 (Cops and visible robber game)**.** Let $\mathcal{G}$ be an undirected graph. The *cops and visible robber game on* $\mathcal{G}$ is a graph searching game on $\mathcal{G}$ defined by the triple $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$ where:

- $\mathcal{L}_s = \mathcal{P}(V(\mathcal{G}))$, $\mathcal{L}_f = \{R \subseteq V(\mathcal{G}) : R \text{ is non-empty and connected}\} \cup \{V(\mathcal{G})\}$,
- $(X, R) \in V_0(\mathcal{A})$ if R is a connected component of $\mathcal{G} \setminus X$,

- $(X, X', R) \in V_1(\mathcal{A})$ if $(X, R) \in V_0(\mathcal{A})$ and $X' \in \mathcal{L}_s$,
- $\big((X, R), (X, X', R)\big) \in E(\mathcal{A})$ for all $(X, R) \in V_0(\mathcal{A})$,
- $\big((X, X', R), (X', R')\big) \in E(\mathcal{A})$ if $R \cup R'$ is contained in a connected component of $\mathcal{G} \setminus (X \cap X')$.

Intuitively, the cops (searchers) and robber (fugitive) occupy vertices of the graph. There is no constraint on the cops, they can be removed and placed on any set of vertices. The robber is constrained to move along paths of any length in the graph, provided he does not pass through a stationary cop. The robber's location in the graph is known to the cops, but because he is able to move infinitely fast, we view his set of potential locations as a connected component of the subgraph obtained by removing vertices occupied by cops. A move consists of some cops being removed from the graph, and announcing vertices that are about to be occupied. The robber is then able to move to any vertex he can reach, and then cops are placed on the announced vertices. If the robber is located on a vertex which has become occupied, then he is captured and the cops win. If he can avoid capture forever, then he wins.

We observe that the cops and visible robber game permits idling: given an intermediate position (X, X', R) and $r \in R \setminus X'$, let R' be the connected component of $\mathcal{G} \setminus X'$ which contains r. Then $R \cup R'$ is contained in a connected component as they are connected sets with a non-empty intersection. Thus there is an edge from (X, X', R) to (X', R'). Furthermore, the game is vacating sensitive: if it is possible to move from (X, X', R) to (X', R') where $R' \not\subseteq R$ then there exists $r \in R' \setminus R$ such that r is adjacent to some vertex in R. Now $R \cup \{r\}$ is connected, so if $r \notin X$, then R is not a connected component of $\mathcal{G} \setminus X$. Hence $r \in X$, so $X \cap R' \neq \emptyset$. Thus we can apply Lemma 5.11 to obtain:

Lemma 5.25. *A cop-monotone winning strategy in the cops and visible robber game is robber-monotone.*

There are some interesting variants of the cops and visible robber game obtained by restricting the movements of the cops. For example, cops are either removed or placed so (X, X', R) is an intermediate position only if either $X' \subseteq X$, or $X \subseteq X'$; at most one cop is moved, so (X, X', R) is an intermediate position only if $|X' \mathbin{\triangle} X| \leq 1$; or at most one cop is placed, so (X, X', R) is an intermediate position only if $|X' \setminus X| \leq 1$. Another variation is the following parameterized class of games, in which we bound the number of cops trying to capture the robber:

Definition 5.26 (k-cops and visible robber game)**.** Let $\mathcal{G}$ be an undirected graph. The *k-cops and visible robber game on $\mathcal{G}$* is defined as the cops and visible robber game, except $\mathcal{L}_s = [V(\mathcal{G})]^{\leq k}$.

In Section 5.3 we show that strategies in these games are equivalent to resource-bounded strategies in the unrestricted game, where the resource we are concerned with is the maximum number of cops occupying the graph at any stage. While this may seem obvious, the observation is quite useful when we consider the complexity of the problem of determining the existence of resource-bounded winning strategies.

We also show in Section 5.3 how this game, particularly this last variant, is closely connected to tree-width. So it would seem that extending this game to directed graphs would be a useful way to generalize tree-width to directed graphs. There are two obvious ways to extend this

game: we could extend the informal description, constraining the robber to move along directed paths of any length; or we could extend the formal description, having positions (X, R) where R is a strongly connected component of $\mathcal{G} \setminus X$, and a transition from (X, R) to (X', R') if $R \cup R'$ is contained in a strongly connected component of $\mathcal{G} \setminus (X \cap X')$. The game corresponding to the latter extension seems less intuitive: it corresponds to restricting the robber to being able to move along directed paths to any vertex from which he has a directed cop-free path back to his starting vertex. This game, which we call the *strongly connected visible robber game*, or more simply the *strong visible robber game*, was considered in [JRST01], and later in this chapter we discuss its relationship with directed tree-width. We investigate the other, arguably more natural, generalization in Chapter 6.

5.2.2 Cops and invisible robber

The *cops and invisible robber game*, also known as the *node searching game*, or *vertex decontamination* has been well-studied in the context of graph theory [KP86, BS91, LaP93]. In our framework, the definition is as follows.

Definition 5.27 (Cops and invisible robber game)**.** Let $\mathcal{G}$ be an undirected graph. The *cops and invisible robber game on* $\mathcal{G}$ is a graph searching game on $\mathcal{G}$ defined by the triple $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$ where:

- $\mathcal{L}_s = \mathcal{P}(V(\mathcal{G}))$, $\mathcal{L}_f = \mathcal{P}(V(\mathcal{G})) \setminus \{\emptyset\}$,
- $(X, R) \in V_0(\mathcal{A})$ if R is a union of non-empty connected components of $\mathcal{G} \setminus X$,
- $(X, X', R) \in V_1(\mathcal{A})$ if $(X, R) \in V_0(\mathcal{A})$ and $X' \in \mathcal{L}_s$,
- $\big((X, R), (X, X', R)\big) \in E(\mathcal{A})$ for all $(X, R) \in V_0(\mathcal{A})$,
- $\big((X, X', R), (X', R')\big) \in E(\mathcal{A})$ if $R' = \mathit{Reach}_{\mathcal{G} \setminus (X \cap X')}(R) \setminus X'$.

The game is played on an undirected graph $\mathcal{G}$ in the same way as the cops and visible robber: the cops are free to move anywhere on $\mathcal{G}$, and the robber can run at great speed along cop-free paths in the graph. In this game however, the location of the robber is not known to the cops – they are only aware of the vertices the robber cannot be at: either because those vertices are currently occupied by cops, or there is no possibility that the robber could not have reached those vertices from when they were vacated by cops. So positions in this game are pairs (X, R) where $X, R \subseteq V(\mathcal{G})$ and R is a union of connected components of $\mathcal{G} \setminus X$, and a search in this game ending at (X, R) can be extended to a search ending at (X', R') if, and only if, $R' = \mathit{Reach}_{\mathcal{G} \setminus (X \cap X')}(R)$. We observe that since R' is uniquely determined from X, X' and R, the robber has no choice from the intermediate position (X, X', R), so this game is effectively a single player game.

In the literature, this game is often viewed as the problem of trying to clean a contaminated graph. Vertices where the robber could be are "contaminated", vertices where the robber cannot be are "cleared", and occupation of a vertex by a cop "clears" that vertex.

5.2.3 Cave searching

The next game we consider is an example of a searching game motivated by a real-life problem. In [Bre67], in a publication for the spelunking community, Breisch considered the problem of finding a lost person in a cave system. In response to a question posed by some cavers about whether existing search techniques could be improved, Parsons [Par78] reformulated the problem as a graph-theoretical problem and investigated games known as *graph sweeping games*. These can be defined as graph searching games as follows.

Definition 5.28 (Graph sweeping game)**.** Let $\mathcal{G}$ be an undirected graph. The *graph sweeping game on* $\mathcal{G}$ is the graph searching game on $\mathcal{G}$ defined by the triple $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$, where:

- $X \in \mathcal{L}_s$ if, and only if, $X = V \cup E$, where $V \subseteq V(\mathcal{G})$, $E \subseteq E(\mathcal{G})$, $|E| \leq 1$, and if $E = \{e\}$ then $e \cap V \neq \emptyset$,
- $\mathcal{L}_f = \mathcal{P}(\text{Elts}(\mathcal{G})) \setminus \{\emptyset\}$,
- $(X, R) \in V_0(\mathcal{A})$ if, and only if, $X \cap R = \emptyset$,
- $(X, X', R) \in V_1(\mathcal{A})$ if, and only if, $X = V \cup E$, $X' = V' \cup E'$, with $V, V' \subseteq V(\mathcal{G})$ and $E, E' \subseteq E(\mathcal{G})$, and either $E' = \emptyset$ and $V' \setminus V = \emptyset$, or if $E' = \big\{\{u, v\}\big\}$ with $v \in V'$ then $u \in V$.
- If $(X, R) \in V_0(\mathcal{A})$ and $(X, X', R) \in V_1(\mathcal{A})$ then $\big((X, R), (X, X', R)\big) \in E(\mathcal{A})$, and
- There is an edge from (X, X', R) to (X', R') if, and only if, R' consists of all elements $x \in \text{Elts}(\mathcal{G}) \setminus X'$ such that if C is the connected component of $\mathcal{G} \setminus \big(X' \cap (X \cup E(\mathcal{G}))\big)$ which contains x, then $C \cap R \neq \emptyset$.

In this game, the graph represents the cave system, with edges representing traversable paths. The fugitive, or lost caver, is located somewhere in the cave system – represented in this game by having sets of elements of $\mathcal{G}$ for the locations of the fugitive. The searchers move through the graph by moving from one vertex to an adjacent vertex along an edge connecting them.

5.2.4 Detectives and robber

The next game was introduced by Berwanger and Grädel [BG04] to define a measure of complexity for directed graphs known as *entanglement*. We can present their definition in terms of graph searching games as follows.

Definition 5.29 (Detectives and robber game)**.** Let $\mathcal{G}$ be a directed graph. The *detectives and robber game on* $\mathcal{G}$ is a graph searching game defined by the triple $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$ where:

- $\mathcal{L}_s = \mathcal{P}(V(\mathcal{G}))$, $\mathcal{L}_f = \big\{\{r\} : r \in V(\mathcal{G})\} \cup \{V(\mathcal{G})\}$,
- $V_0(\mathcal{A}) = \{(\emptyset, V(\mathcal{G}))\} \cup \{(X, \{r\}) : r \notin X\}$,
- $V_1(\mathcal{A}) = \{(\emptyset, \emptyset, V(\mathcal{G}))\} \cup \{(X, X', \{r\}) : (X, \{r\}) \in V_0(\mathcal{A}) \text{ and } X' \subseteq X \cup \{r\}\}$,
- If $(X, R) \in V_0(\mathcal{A})$ and $(X, X', R) \in V_1(\mathcal{A})$ then $\big((X, R), (X, X', R)\big) \in E(\mathcal{A})$,

- There is an edge from $(\emptyset, \emptyset, V(\mathcal{G}))$ to $(\emptyset, \{r\})$ for all $r \in V(\mathcal{G})$,
- For all $(r, r') \in E(\mathcal{G})$ and $(X, X', \{r\}) \in V_1(\mathcal{A})$ with $r' \notin X'$, there is an edge in $E(\mathcal{A})$ from $(X, X', \{r\})$ to (X', r'), and
- There are no other edges in $E(\mathcal{A})$.

In this game, the detectives and robber occupy vertices in the graph. The robber has to move to a successor of his current location and the detectives can only move to the last position of the robber or remain where they are.

5.2.5 Cops and inert robber

As with the cops and visible robber game defined in Definition 5.24, the final game we consider is also a game played on an undirected graph closely related to tree-width. Introduced by Dendris, Kirousis and Thilikos [DKT97], the *cops and inert robber game* can also be viewed as a graph searching game in the following manner.

Definition 5.30 (Cops and inert robber)**.** Let $\mathcal{G}$ be an undirected graph. The *cops and inert robber game on* $\mathcal{G}$ is the graph searching game on $\mathcal{G}$ defined by the triple $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$, where:

- $\mathcal{L}_s = \mathcal{P}(V(\mathcal{G}))$, $\mathcal{L}_f = \mathcal{P}(V(\mathcal{G})) \setminus \{\emptyset\}$,
- $(X, R) \in V_0(\mathcal{A})$ if R is a union of non-empty connected components of $\mathcal{G} \setminus X$,
- $(X, X', R) \in V_1(\mathcal{A})$ if $(X, R) \in V_0(\mathcal{A})$ and $X' \in \mathcal{L}_s$,
- $\big((X, R), (X, X', R)\big) \in E(\mathcal{A})$ for all $(X, R) \in V_0(\mathcal{A})$,
- $\big((X, X', R), (X', R')\big) \in E(\mathcal{A})$ if $R' = \big(R \cup \mathit{Reach}_{\mathcal{G}\setminus(X\cap X')}(R \cap X')\big) \setminus X'$.

As with the cops and invisible robber game defined in Definition 5.27, in this game the cops and robber occupy vertices of the graph, the cops are free to move anywhere in the graph, and the robber may run at great speed along paths in the graph. Furthermore, the location of the robber is unknown to the cops. However we impose the restriction that he is only able to move from his position if it is about to be occupied by a cop. Thus at position (X, R), X represents the location of the cops and R represents the set of potential locations. Now if the cops move to X', then the resulting potential locations for the robber consist of his current set of locations together with any vertex v for which there is a path from a vertex in $R \cap X'$ to v, excluding any vertex now occupied by a cop. Thus R', the new set of potential locations, can be defined as:

$$R' = \big(R \cup \mathit{Reach}_{\mathcal{G}\setminus(X\cap X')}(R \cap X')\big) \setminus X'.$$

In the next section we see that this game is also closely connected to tree-width, suggesting that the generalization of this game to directed graphs would be a practical way to develop complexity measures which extend tree-width. In Chapter 7 we consider such a generalization.

5.2.6 Cops and robber games

Examples 5.2.1, 5.2.2, and 5.2.5 highlight one of the most important and simple variants of the graph searching game, the cops and robber game. In this game the cops (searchers) and the robber (fugitive) only occupy vertices of graph, with the robber being able to start at any vertex of the graph.

Definition 5.31 (Cops and robber game)**.** Let $\mathcal{G}$ be a graph and $\mathbb{G}_{\mathcal{G}}^{\Gamma}$ be a graph searching game on $\mathcal{G}$ defined by a triple $(\mathcal{L}_c, \mathcal{L}_r, \mathcal{A})$. We say $\mathbb{G}_{\mathcal{G}}^{\Gamma}$ is a *cops and robber game* if $\mathcal{L}_c \subseteq \mathcal{P}(V(\mathcal{G}))$, $\mathcal{L}_r \subseteq \mathcal{P}(V(\mathcal{G}))$ and $V(\mathcal{G}) \in \mathcal{L}_r$. We call the searchers of a cops and robber game the *cops*, and the fugitive is called the *robber*. Likewise, searcher-monotone searches and strategies are *cop-monotone* and fugitive-monotone searches and strategies are *robber-monotone*. A graph searching game type Γ is a *cops and robber game type* if for all graphs $\mathcal{G}$, $\mathbb{G}_{\mathcal{G}}^{\Gamma}$ is a cops and robber game.

One advantage of the restriction of the searchers and fugitive to vertices of the graph is that the resulting games are less dependent on the edges of the graph. In particular, it is often the case that the presence of multiple edges or loops does not affect the game – the arena is the same as the arena for the graph searching game on the graph with all loops removed and all multiple edges replaced with a single edge. In the sequel we assume all cops and robber games are played on simple graphs, unless otherwise stated.

5.3 Complexity measures

Unlike the games we considered in Chapter 2, we are not solely concerned with which player wins a graph searching game. In most of the examples above, it is clear that the searchers can always find the fugitive by (eventually) occupying all of the graph, so as it stands the question is not interesting – the searchers always have a winning strategy. One exception to this is the parameterized class of games, the k-cops and visible robber games defined in Definition 5.24. This suggests that it may be more fruitful to consider resource-bounded strategies. For instance, for a cops and robber game, we can ask "Given $k \in \mathbb{N}$, can the cops capture the robber while at any time occupying at most k vertices?". Consistent with viewing the cops as physical entities, this can be viewed as asking if there is a winning strategy for k cops, defined more precisely as:

Definition 5.32 (Winning strategy for k cops)**.** Let $\mathbb{G}_{\mathcal{G}}^{\Gamma}$ be a cops and robber game, σ a strategy for the cops, and $k \in \mathbb{N}$. We say that σ is a *winning strategy for k cops* if σ is a winning strategy, and for any search $(X_0, R_0)(X_1, R_1)\cdots$ consistent with σ, $|X_i| \leq k$ for all i.

From this we can derive a complexity measure, in this particular case, the minimum number of cops required to capture the robber. In the following chapters this is the measure we are interested in, but for the remainder of this chapter we consider a more general framework which encompasses many other important graph parameters. For this we introduce the concept of a *resource measure* that can be used to restrict plays and, by association, strategies in a graph searching game. First, we introduce two partial orders on the class of sequences of sets.

Definition 5.33. Let $\pi = X_1X_2\cdots$ and $\pi' = Y_1Y_2\cdots$ be two (possibly infinite) sequences of sets. We write $\pi' \leq \pi$ if π' is a subsequence of π. That is, there exists an increasing sequence of indices $n_1 < n_2 < \cdots \leq |\pi|$ such that $Y_i = X_{n_i}$ for all $i \leq |\pi'|$. We write $\pi' \subseteq \pi$ if $|\pi'| \leq |\pi|$ and for all $i \leq |\pi'|$, $Y_i \subseteq X_i$.

Definition 5.34 (Resource measure)**.** A *resource measure* is a function φ which maps sequences of finite sets to elements of $\omega \cup \{\omega\}$, with $\varphi(\pi) = \omega$ only if π is infinite. We say φ is *order-preserving (order-reversing)* if for all $\pi, \pi' \in \text{dom}(\varphi)$, $\pi' \leq \pi \Rightarrow \varphi(\pi') \leq \varphi(\pi)$ ($\pi' \leq \pi \Rightarrow \varphi(\pi') \geq \varphi(\pi)$). We say φ is *monotone (anti-monotone)* if for all $\pi, \pi' \in \text{dom}(\varphi)$, $\pi' \subseteq \pi \Rightarrow \varphi(\pi') \leq \varphi(\pi)$ ($\pi' \subseteq \pi \Rightarrow \varphi(\pi') \geq \varphi(\pi)$).

The resource measure which motivated the above discussion is an example of a monotone, order-preserving resource measure:

Definition 5.35 ($\varphi_{\max}$)**.** The resource measure $\varphi_{\max}$ is defined as follows. If $\pi = X_1 X_2 \cdots$ is a sequence of finite sets, then

$$\varphi_{\max}(\pi) = \max_{i \geq 1}\{|X_i|\}.$$

A resource measure φ defines a measure on a search $\pi = (X_0, R_0)(X_1, R_1)\cdots$ in the following way: let $\pi_1 = X_0 X_1 \cdots$ be the sequence of first components of elements of π, and define $\varphi(\pi) := \varphi(\pi_1)$. We only consider the sequence of searcher locations because we are primarily interested in the resource usage of the searchers. It follows that requiring a resource measure to be bounded imposes a restriction on the searches, and consequently, the strategies available in a graph searching game. So asking if the searchers have a winning strategy is no longer a trivial problem. Indeed, it would seem that interesting metrics for graphs could be derived from the "optimal" bounds of resource measures for which the searchers still have a winning strategy. This leads to the following definition of a very general measure of graph complexity defined by graph searching games.

Definition 5.36 (Graph searching width)**.** Let Γ be a graph searching game type, and φ an order-preserving (order-reversing) resource measure. Let $\mathcal{G}$ be a graph. The (Γ, φ)-*width* of $\mathcal{G}$, $w_{(\Gamma,\varphi)}(\mathcal{G})$, is the minimum (maximum) k such that in $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ there exists a winning strategy for the searchers, σ, so that for any search, π, consistent with σ, we have $\varphi(\pi) \leq k$ ($\varphi(\pi) \geq k$). Likewise, if we restrict to fugitive-monotone or searcher-monotone winning strategies in $\mathbb{G}^{\Gamma}_{\mathcal{G}}$, we obtain the *fugitive-monotone* or *searcher-monotone* (Γ, φ)-*width* of $\mathcal{G}$.

Remark. As we are interested in minimizing (maximizing) an order-preserving (order-reversing) measure, it suffices to consider searches that are simple paths in the arena – any loops are only going to increase (decrease) the resource requirements. Consequently, we only need to consider strategies that require finite memory to determine if the searchers have a resource bounded winning strategy. Thus, the requirement that the resource measure is order-preserving (or order-reversing) ensures that the restriction of searches obtained by bounding the resource measure does not affect the decidability of determining if the searchers have a winning strategy. In particular, the requirement maintains our maxim that strategies with finite memory are sufficient, especially for the resource bounded game.

Many practical measures of graph complexity can be defined using this framework, as we see with the following examples.

5.3.1 Example: Cops and visible robber

We recall the cops and visible robber game defined in Example 5.2.1. In [ST93] when this game was first considered, Seymour and Thomas showed that it could be used to characterize tree-width by observing that the number of cops required to capture the robber was equal to one more than the tree-width of the graph being searched. More precisely, they proved:

Theorem 5.37 ([ST93])**.** *Let $\mathcal{G}$ be an undirected graph. The following are equivalent:*

1. *$\mathcal{G}$ has tree-width $\leq k-1$.*
2. *k cops have a cop-monotone winning strategy in the cops and visible robber game.*
3. *k cops have a robber-monotone winning strategy in the cops and visible robber game.*
4. *k cops have a winning strategy in the cops and visible robber game.*

Recalling the definition of $\varphi_{\max}$ in Definition 5.35, we can rephrase this theorem as:

Corollary 5.38. *Let Γ be the cops and visible robber game type defined in Definition 5.24, and let $\mathcal{G}$ be an undirected graph. Then*

$$\textit{Tree-width}(\mathcal{G}) = w_{(\Gamma,\varphi_{\max})}(\mathcal{G}).$$

We remarked in Example 5.2.1 that there were several variants of the cops and visible robber depending on various restrictions placed on the movement of the cops. It is easy to see informally that the number of cops required to catch the robber in each of these games is the same. We now provide a formal proof of this often glossed-over point.

Proposition 5.39. *Let Γ_0 be the cops and visible robber game type defined in Definition 5.24. Let Γ_1 be the cops and visible robber game type where cops are either placed or removed. Let Γ_2 be the cops and visible robber game type where at most one cop is placed, and let Γ_3 be the cops and visible robber game type where at most one cop is moved at a time. Let $\mathcal{G}$ be an undirected graph. Then the following are equivalent:*

(i) *k cops have a winning strategy in $\mathbb{G}_{\mathcal{G}}^{\Gamma_0}$.*

(ii) *k cops have a winning strategy in $\mathbb{G}_{\mathcal{G}}^{\Gamma_1}$.*

(iii) *k cops have a winning strategy in $\mathbb{G}_{\mathcal{G}}^{\Gamma_2}$.*

(iv) *k cops have a winning strategy in $\mathbb{G}_{\mathcal{G}}^{\Gamma_3}$.*

Proof. From the definitions provided in Example 5.2.1, it follows easily that a strategy for the searchers in $\mathbb{G}_{\mathcal{G}}^{\Gamma_3}$ is a strategy in $\mathbb{G}_{\mathcal{G}}^{\Gamma_2}$ and also a strategy in $\mathbb{G}_{\mathcal{G}}^{\Gamma_1}$; a strategy for the searchers in $\mathbb{G}_{\mathcal{G}}^{\Gamma_2}$ is a strategy in $\mathbb{G}_{\mathcal{G}}^{\Gamma_0}$; and a strategy in $\mathbb{G}_{\mathcal{G}}^{\Gamma_1}$ is also a strategy in $\mathbb{G}_{\mathcal{G}}^{\Gamma_0}$. Thus (iv)⇒(iii)⇒(i) and (iv)⇒(ii)⇒(i). We now show that (i)⇒(iv).

Suppose k cops have a winning strategy σ in $\mathbb{G}_{\mathcal{G}}^{\Gamma_0}$. Let $\Gamma_0(\mathcal{G}) = (\mathcal{L}_c, \mathcal{L}_r, \mathcal{A})$, and $\Gamma_3(\mathcal{G}) = (\mathcal{L}'_c, \mathcal{L}'_r, \mathcal{A}')$. Note that by the definition of Γ_3, $\mathcal{L}'_c = \mathcal{L}_c$, $\mathcal{L}'_r = \mathcal{L}_r$, $V_0(\mathcal{A}) = V_0(\mathcal{A}')$ and $V_1(\mathcal{A}) \supseteq V_1(\mathcal{A}')$. We show how to define a strategy σ' for k cops such that for all $(X,R) \in V_0(\mathcal{A}')$, $|\sigma'(X,R) \mathbin{\triangle} X| \leq 1$. The idea is that we replace each move of σ which involves moving more than one cop with a sequence of moves: removing one cop at a time from X until cops remain on $X \cap \sigma(X,R)$, and then adding cops one at a time until they occupy $\sigma(X,R)$. More formally, let $\Sigma = V_0(\mathcal{A})$. We define a history-dependent strategy σ' as follows. Let $\sigma'(\epsilon, X_0, R_0) = \big((X_0,R_0),\emptyset\big)$ where $(X_0,R_0) = v_I(\mathcal{A})$. Now suppose $w \in \Sigma^*$, $w \neq \epsilon$, and the last symbol of w is $(X,R) \in V_0(\mathcal{A})$. Define $\sigma'(w,X',R')$ as follows. If $X \cap \sigma(X,R) \subset X' \subseteq X$, let $X'' = X' \setminus \{v\}$ for some $v \in X' \setminus \sigma(X,R)$, and define $\sigma'(w,X',R') := \big((X,R),X''\big)$.

Otherwise, if $X \cap \sigma(X,R) \subseteq X' \subset \sigma(X,R)$, let $X'' = X' \cup \{v\}$ for some $v \in \sigma(X,R) \setminus X'$, and define $\sigma'(w, X', R') := \big((X,R), X''\big)$. Finally, if $X' = \sigma(X,R)$ define $\sigma'(w, X', R') = \big((X', R'), X'\big)$. Clearly σ' is a strategy for at most k cops which involves placing or removing at most one cop at each step. We now show that it is a winning strategy.

Let $\pi = (X'_0, R'_0)(X'_1, R'_1)\cdots$ be a search consistent with σ'. Let $w' \in \Sigma^* \cup \Sigma^\omega$ be the history consistent with π, and let w be the word obtained by replacing repeated symbols in w' with single occurrences. We observe that these repetitions arise where we have replaced a single multiple-cop move with a finite sequence of single-cop moves so w is infinite if, and only if, w' is infinite. We also observe that by the definition of σ', w is a subsequence of π. We make the following claim:

Claim. The search defined by w is a search consistent with σ.

Proof of claim. Let $w = (X_1, R_1)(X_2, R_2)\cdots$. From the definition of σ' we have $X_{i+1} = \sigma(X_i, R_i)$ for all $i \geq 1$, so it suffices to show that for all $i \geq 1$ there is an edge in $\mathcal{A}$ from (X_i, X_{i+1}, R_i) to (X_{i+1}, R_{i+1}). That is, each possible set of locations for the robber available after the sequence of single-cop moves is available after a single multiple-cop move. Let m and n be such that $(X_i, R_i) = (X'_m, R'_{\max})$ and $(X_{i+1}, R_{i+1}) = (X'_n, R'_n)$ and let q be such that $m \leq q \leq n$ and $X'_q = X_i \cap X_{i+1}$. We prove by induction that for all j, with $m \leq j \leq n$, $R'_j \cup R'_{\max}$ is contained in a connected component of $\mathcal{G} \setminus (X'_m \cap X'_j)$. Clearly this is true for $j = m$. Now suppose for some $j \geq m$, $R'_j \cup R'_{\max}$ is contained in a connected component of $\mathcal{G} \setminus (X'_m \cap X'_j)$, and consider R'_{j+1}. By the definition of the cops and visible robber game, $R'_j \cup R_{j+1}$ is contained in a connected component of $\mathcal{G} \setminus (X'_j \cap X_{j+1})$. We consider the following two cases. If $j < q$, then $X_{j+1} \subseteq X_j \subseteq X_m$ and $R_j \supseteq R_{\max}$. Thus the connected component R of $\mathcal{G} \setminus X'_{j+1}$ which contains $R_{\max}$ is the only component contained in the same connected component of $\mathcal{G} \setminus (X'_{j+1} \cap X'_j)$ as R_j. Thus $R'_{j+1} = R$. Since $R \cup R'_{\max} = R$ is a connected component of $\mathcal{G} \setminus X'_{j+1} = \mathcal{G} \setminus (X'_{j+1} \cup X'_m)$, our hypothesis holds for $j+1$. Now suppose $j \geq q$. Then $X'_m \cap X_j = X_i \cap X_{i+1}$, and $X'_{j+1} \supseteq X'_j$. Thus if R'_{j+1} is in the same connected component of $\mathcal{G} \setminus (X'_j \cap X'_{j+1}) = \mathcal{G} \setminus X'_j$ as R'_j, it follows that $R'_j \supseteq R'_{j+1}$. By the inductive hypothesis, R'_j is in the same connected component of $\mathcal{G} \setminus (X'_m \cap X'_j)$ as $R'_{\max}$. But as $\mathcal{G} \setminus (X'_m \cap X'_j) = \mathcal{G} \setminus (X_i \cap X_{i+1}) = \mathcal{G} \setminus (X'_m \cap X'_{j+1})$, it follows that R'_{j+1} is in the same connected component of $\mathcal{G} \setminus (X'_m \cap X'_{j+1})$ as $R'_{\max}$. This completes the inductive step and the proof of the claim. ⊣

Next we observe that as there is always a move available to the cops, π is winning for the robber if, and only if, it is infinite. But this is the case if, and only if, w is infinite. As σ is a winning strategy, there are no infinite searches consistent with σ, thus π must be finite and therefore winning for the searchers. □

Our final observation regarding the cops and visible robber game and the number of cops required to capture the robber is a straightforward result which relates the game and the resource measure with the parameterized class of games we also introduced in Example 5.2.1.

Lemma 5.40. *Let $\mathcal{G}$ be an undirected graph. The cops have a winning strategy in the k-cops and visible robber game if, and only if, k cops have a winning strategy in the cops and visible robber game.*

Proof. Clearly a winning strategy σ for k cops in the cops and visible robber game is a winning strategy for the cops in the k-cops and robber game: since $|\sigma(X,R)| \leq k$ for all positions

(X, R) in the cops and visible robber game, it follows that $\sigma(X, R) \in [V(\mathcal{G})]^{\leq k}$ for all positions (X, R) in the k-cops and visible robber game.

For the converse, let σ be a winning strategy for the cops in the k-cops and robber game. Let us extend σ to a strategy in the cops and visible robber game by defining $\sigma(X, R) = \emptyset$ for all $X \subseteq V(\mathcal{G})$ with $|X| > k$. Then, since $|\sigma(X, R)| \leq k$ for all positions (X, R), σ is a strategy for k cops. Since any search in the cops and visible robber game consistent with σ is also a search in the k-cops and visible robber game consistent with σ, it follows that σ is a winning strategy in the cops and visible robber game. □

Remark. This example shows that with the resource measure $\varphi_{\max}$ we can view resource bounded strategies as winning strategies in a parameterized family of graph searching games. As such games are simple, if either the fugitive or the searchers have a winning strategy, then they have a memoryless winning strategy. This justifies our use of positional strategies in subsequent chapters.

Theorem 5.37 motivates the nomenclature used for Theorem 4.7: a haven is, as the name suggests, a characterization of a winning strategy for the robber. Carrying this reasoning to the definition of haven used in [JRST01], we see that Theorem 4.11 can be restated as the following characterization of directed tree-width in terms of graph searching games. We recall the strongly connected visible robber game defined in Example 5.2.1.

Lemma 5.41. *Let $\mathcal{G}$ be a digraph. Either $\mathcal{G}$ has directed tree-width $\leq 3k + 1$ or k cops do not have a winning strategy in the strong visible robber game on $\mathcal{G}$.*

5.3.2 Example: Cops and invisible robber

We now consider the resource measure $\varphi_{\max}$ applied to the cops and invisible robber game. Kirousis and Papadimitriou [KP86] showed that the number of cops required to capture the robber in this game is equivalent to one more than the path-width of the graph.

Theorem 5.42 ([KP86])**.** *Let $\mathcal{G}$ be an undirected graph. The following are equivalent:*

1. *$\mathcal{G}$ has path-width $\leq k - 1$.*
2. *k cops have a cop-monotone winning strategy in the cops and invisible robber game.*
3. *k cops have a robber-monotone winning strategy in the cops and invisible robber game.*
4. *k cops have a winning strategy in the cops and invisible robber game.*

Together with Theorem 5.37, this theorem shows how we can view the relationship between path-width and tree-width via graph searching games. As an example of the consequence of this, Fomin, Fraigniaud and Nisse [FFN05] considered a parameterized family of cops and robber games where the robber is invisible, but the cops are allowed q queries of the location of the robber during a search. The resulting family of measures corresponding to the number of cops required in each game gives a parameterization which lies between path-width ($q = 0$) and tree-width ($q = \infty$). Because such parameterized measures can be seen as a generalization of both path-width and tree-width, they are particularly useful for investigating the structural complexity of graphs.

5.3.3 Example: Cops and inert robber

We again consider the $\varphi_{\max}$ resource measure, but this time with the cops and inert robber game. Dendris, Kirousis and Thilikos [DKT97] showed that the number of cops required to capture an invisible, inert robber is another measure equivalent to one more than tree-width.

Theorem 5.43 ([DKT97])**.** *Let $\mathcal{G}$ be an undirected graph. The following are equivalent:*

1. *$\mathcal{G}$ has tree-width $\leq k-1$.*
2. *k cops have a robber-monotone winning strategy in the cops and inert robber game.*
3. *k cops have a winning strategy in the cops and inert robber game.*

Combining this with Theorem 5.37, we see that the number of cops required to capture a robber in the cops and visible robber game is equal to the number of cops required to capture a robber in the cops and inert robber game. In Chapter 7, where we consider the generalization of the cops and inert robber game to directed graphs, we show that this is not the case for the generalizations of the games to digraphs.

Dendris et al. also showed that the cop-monotone version of the cops and inert robber game may require more cops than the robber-monotone version. In Chapter 7, we show that the number of cops required in the cop-monotone version of the natural extension of this game to directed graphs is equivalent to the extension of path-width to digraphs.

5.3.4 Example: Other resource measures

We now consider some graph parameters which can be characterized by the invisible and inert robber games, but with other resource measures. In [FG00], Fomin and Golovach considered the following resource measure which intuitively represents the "cost" of a search.

Definition 5.44 (φ_{cost})**.** The resource measure φ_{cost} is defined as follows. If $\pi = X_1 X_2 \cdots$ is a sequence of finite sets, then

$$\varphi_{\text{cost}}(\pi) = \sum_{i \geq 1} |X_i|.$$

In [FG00] it was shown that the minimum cost of a search in a cops and invisible robber game on a graph $\mathcal{G}$ is equivalent to the *profile* of $\mathcal{G}$: the minimal number of edges of an interval supergraph of $\mathcal{G}$. In [FHT04] it was shown that the minimum cost of a search in a cops and inert robber game on $\mathcal{G}$ is equivalent to the fill-in of $\mathcal{G}$: the minimum number of edges which need to be added to make $\mathcal{G}$ chordal. Summarizing these results in our framework:

Theorem 5.45 ([FG00, FHT04])**.** *Let Γ_0 be the cops and invisible robber game type defined in Definition 5.27 and let Γ_1 be the cops and inert robber game type defined in Definition 5.30. Let $\mathcal{G}$ be an undirected graph. Then*

1. *The profile of $\mathcal{G}$ is equal to $w_{(\Gamma_0, \varphi_{cost})}(\mathcal{G})$.*
2. *The fill-in of $\mathcal{G}$ is equal to $w_{(\Gamma_1, \varphi_{cost})}(\mathcal{G})$.*

In [RS82] Rosenberg and Sudborough considered a pebbling game which Fomin et al. [FHT04] observed can be seen as a version of the cops and invisible robber game. Rosenberg and Sudborough showed that minimizing the resource measure defined by the maximum life-time of a pebble on the graph is equivalent to finding the bandwidth of the graph: the minimum, over all linear layouts of the vertices of the graph, of the maximum distance between any pair of adjacent vertices. Fomin et al. [FHT04] viewed this resource measure in the setting of graph searching games, to define the following measure which indicates the "occupation time" of a search.

Definition 5.46 (φ_{ot})**.** Let $\pi = X_1X_2\cdots$ be a sequence of finite subsets of a set V. For each $i \geq 1$ let $\chi_i : V \to \{0,1\}$ be the characteristic function of X_i, so that $\chi_i(v) = 1$ if, and only if, $v \in X_i$. Then φ_{ot} is defined as follows:

$$\varphi_{\text{ot}}(\pi) = \max_{v \in V} \sum_{i \geq 1} \chi_i(v).$$

Remark. In order for this measure to be non-trivial, we assume that we are working with version of the cops and robber game where at most one cop is moved at a time.

The result of Rosenberg and Sudborough can then be summarized thus:

Theorem 5.47 ([RS82])**.** *Let Γ be the cops and invisible robber game type defined in Definition 5.27 where at most one cop is moved at a time, and let $\mathcal{G}$ be an undirected graph. Then the bandwidth of $\mathcal{G}$ is equal to $w_{(\Gamma,\varphi_{ot})}(\mathcal{G})$.*

Fomin et al. [FHT04] used Theorem 5.47 to generate a generalization of bandwidth, called *treespan*, by considering the resource measure φ_{ot} on the cops and inert robber game.

Theorem 5.48 ([FHT04])**.** *Let Γ be the cops and inert robber game type defined in Definition 5.30 where at most one cop is moved at a time, and let $\mathcal{G}$ be an undirected graph. Then the treespan of $\mathcal{G}$ is equal to $w_{(\Gamma,\varphi_{ot})}(\mathcal{G})$.*

5.3.5 Monotonicity

Theorems 5.37, 5.42 and 5.43 all indicate an interesting property of some of the graph searching games we have considered: the restriction imposed by bounding the resources supercedes the restriction imposed by monotonicity. This provides an explanation as to why measures like tree-width are good complexity measures from a practical and structural perspective: winning strategies which are not necessarily monotone indicate the existence of various structural properties such as havens or brambles (as we see in Chapter 8); on the other hand, monotone winning strategies are very useful algorithmically. As we saw with Lemma 5.11, monotone strategies can be represented as restrictions on the arena, so it is often easier to compute monotone winning strategies. Furthermore, as we see in the next few chapters, monotone strategies often lend themselves to decompositions with properties that make them very useful for practical purposes. Thus it is important to identify games where monotonicity is not too great a restriction, as these games will provide measures that are good indicators of algorithmic and structural complexity. This leads to the question, "For which graph searching game types and resource measures is monotonicity sufficient?" More precisely,

Open problem 5.49. *For which graph searching game types Γ and resource measures φ does (Γ, φ)-width give a bound on fugitive-monotone or searcher-monotone $(\mathbb{G}, \varphi)$-width?*

Remark. Allowing approximate equivalence gives some flexibility in the above question: while it may not be the case that a winning strategy implies the existence of a monotone winning strategy with the same resource bounds, it might still be possible that the resource requirements for a monotone strategy can be deduced from those of a winning strategy.

5.4 Robustness results

We now use the framework we have developed to show that the complexity measures we have defined are well-behaved under some simple graph operations, thus indicating their significance as a robust measure of graph complexity. In particular we show that, under some reasonable assumptions, the width measure defined by a graph searching game and a resource measure does not increase under the simplification operation of taking subgraphs. We also show that the complexity measure we have defined can be determined from the connected components of the graph. Finally, we consider the cops and robber game. We show that the restriction of having the searchers and the fugitive located on vertices enables us to show that the width measure defined by the number of cops required in a cops and robber game suitably increases under a graph operation which can be seen as a uniform complication, namely graph composition.

For convenience, we only consider width measures defined by order-preserving resource measures. Thus for each of the following results, there is a dual result obtained by replacing order-preserving with order-reversing, monotone with anti-monotone, and $\leq$ with $\geq$.

5.4.1 Subgraphs

In Definition 5.21 we introduced a restriction on graph searching game types, *respecting restriction*, which asserted that searching strategies in a graph $\mathcal{G}$ can be restricted to be searching strategies in subgraphs of $\mathcal{G}$. It turns out that imposing this restriction on the graph searching game type and the monotonicity restriction on the resource measure is sufficient to show that graph searching width is well-behaved with respect to subgraphs.

Theorem 5.50. *Let Γ be a graph searching game type which respects restriction. Let φ be a monotone, order-preserving resource measure. For any two graphs $\mathcal{G}, \mathcal{G}'$ such that $\mathcal{G}'$ is a subgraph of $\mathcal{G}$:*

$$w_{(\Gamma,\varphi)}(\mathcal{G}') \leq w_{(\Gamma,\varphi)}(\mathcal{G}).$$

Proof. Let $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ and $\mathbb{G}^{\Gamma}_{\mathcal{G}'}$ be the graph searching games on $\mathcal{G}$ and $\mathcal{G}'$ defined by $\Gamma(\mathcal{G})$ and $\Gamma(\mathcal{G}')$ respectively. Since Γ respects restriction, it follows from Lemma 5.23 that $(\supset, \supset)$ is a searching simulation from $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ to $\mathbb{G}^{\Gamma}_{\mathcal{G}'}$. Let σ be a winning searcher strategy in $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ such that for any search π consistent with σ, $\varphi(\pi) \leq w_{(\Gamma,\varphi)}(\mathcal{G})$. Let σ' be a searching strategy in $\mathbb{G}^{\Gamma}_{\mathcal{G}'}$ $(\supset, \supset)$-simulated by σ. It follows from Lemma 5.16 that σ' is a winning strategy for the searchers. Furthermore, by the definition of σ', for any search $\pi' = (X'_0, R'_0)(X'_1, R'_1)\cdots$ consistent with σ' there exists a search $\pi = (X_0, R_0)(X_1, R_1)\cdots$ consistent with σ such that $X_i \supset X'_i$ for all i. Thus, $X'_i = X_i \cap \mathrm{Elts}(\mathcal{G}') \subseteq X_i$. Since φ is monotone, it follows that $\varphi(\pi') \leq \varphi(\pi) \leq w_{(\Gamma,\varphi)}(\mathcal{G})$, and this holds for any search π'. Thus, from the definition of $w_{(\Gamma,\varphi)}(\mathcal{G}')$, we have $w_{(\Gamma,\varphi)}(\mathcal{G}') \leq w_{(\Gamma,\varphi)}(\mathcal{G})$ as required. □

In Lemma 5.18 we observed properties sufficient for a simulation to respect fugitive and searcher-monotonicity. We now show that $\supset$ satisfies these properties, implying that Theorem 5.50 can be extended to fugitive-monotone and searcher-monotone width.

Lemma 5.51. *Let $\mathcal{G}$ and $\mathcal{G}'$ be graphs with $\mathcal{G}$ a subgraph of $\mathcal{G}'$. The relation $\supset_{\mathcal{G}}^{\mathcal{G}'}$ is monotone and $\cap$-compatible.*

Proof. Take $X', Y' \subseteq \text{Elts}(\mathcal{G}')$ and $X, Y \subseteq \text{Elts}(\mathcal{G})$ such that $X' \supset_{\mathcal{G}}^{\mathcal{G}'} X$ and $Y' \supset_{\mathcal{G}}^{\mathcal{G}'} Y$. From the definition of $\supset$, it follows that $X = X' \cap \text{Elts}(\mathcal{G})$ and $Y = Y' \cap \text{Elts}(\mathcal{G})$. Thus, if $X' \subseteq Y'$, $X = X' \cap \text{Elts}(\mathcal{G}) \subseteq Y' \cap \text{Elts}(\mathcal{G}) = Y$, so $\supset_{\mathcal{G}}^{\mathcal{G}'}$ is monotone. Furthermore, $(X' \cap Y') \cap \text{Elts}(\mathcal{G}) = (X' \cap \text{Elts}(\mathcal{G})) \cap (Y' \cap \text{Elts}(\mathcal{G})) = X \cap Y$, so $(X' \cap Y') \supset_{\mathcal{G}}^{\mathcal{G}'} (X \cap Y)$, and therefore $\supset_{\mathcal{G}}^{\mathcal{G}'}$ is $\cap$-compatible. □

Corollary 5.52. *Let Γ be a graph searching game type which respects restriction. Let φ be a monotone, order-preserving resource measure. For any two graphs $\mathcal{G}, \mathcal{G}'$ such that $\mathcal{G}'$ is a subgraph of $\mathcal{G}$:*

1. *The fugitive-monotone (Γ, φ)-width of $\mathcal{G}$ is at most the fugitive-monotone (Γ, φ)-width of $\mathcal{G}'$, and*

2. *The searcher-monotone (Γ, φ)-width of $\mathcal{G}$ is at most the searcher-monotone (Γ, φ)-width of $\mathcal{G}'$.*

5.4.2 Connected components

We now show how the widths of the connected components of a graph can be used to compute the width of the graph. First we need to introduce a notion which is dual to restriction respecting.

Definition 5.53 (Reflects restriction)**.** Let Γ be a graph searching game type. We say Γ *reflects restriction* if for any graphs $\mathcal{G}$ and $\mathcal{G}'$ such that $\mathcal{G}$ is a subgraph of $\mathcal{G}'$, $\Gamma(\mathcal{G}) = (\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$, and $\Gamma(\mathcal{G}') = (\mathcal{L}'_s, \mathcal{L}'_f, \mathcal{A}')$, then

- If $R_{\max}$ is the $\subseteq$-maximal element of $\mathcal{L}_f$, and $R'_{\max}$ is the $\subseteq$-maximal element of $\mathcal{L}'_f$, then $R_{\max} = R'_{\max} \cap \text{Elts}(\mathcal{G})$.

- If there is an edge from (Y, S) to (Y, Y', S) in $E(\mathcal{A})$ then for all $(X, R) \in V_0(\mathcal{A}')$ and $(X, X', R) \in V_1(\mathcal{A}')$ such that $Y = X \cap \text{Elts}(\mathcal{G})$, $S = R \cap \text{Elts}(\mathcal{G})$ and $Y' = X' \cap \text{Elts}(\mathcal{G})$, there is an edge in $E(\mathcal{A}')$ from (X, R) to (X, X', R), and

- If there is an edge from (X, X', R) to (X', R') in $E(\mathcal{A}')$ and $(Y, Y', S) \in V(\mathcal{A})$ for $Y = X \cap \text{Elts}(\mathcal{G})$,$Y' = X' \cap \text{Elts}(\mathcal{G})$ and $S = R \cap \text{Elts}(\mathcal{G})$, then either $R' \cap \text{Elts}(\mathcal{G}) = \emptyset$ or there is an edge from (Y, Y', S) to $(Y', R' \cap \text{Elts}(\mathcal{G}))$ in $E(\mathcal{A})$.

Just as respecting restriction can be viewed as $\supset$-closure, it would appear that restriction reflection should also be equivalent to $\mathfrak{R}$-closure for some quasi-simulation family $\mathfrak{R}$ similar to $\supset$. However, the last condition in the definition is problematic for the game simulation: the fugitive may be able to move in the larger graph ($R' \neq \emptyset$), but because $R' \cap \text{Elts}(\mathcal{G}) = \emptyset$, there is no response on the smaller graph. Nevertheless, we are able to obtain a result, similar to Lemma 5.13, sufficient for our purposes.

Lemma 5.54. *Let Γ be a graph searching game type which reflects restriction and let $\mathcal{G}$ and $\mathcal{G}'$ be graphs such that $\mathcal{G}$ is a subgraph of $\mathcal{G}'$. Let $\Gamma(\mathcal{G}) = (\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$, $\Gamma(\mathcal{G}') = (\mathcal{L}'_s, \mathcal{L}'_f, \mathcal{A}')$, and take $(X'_0, R'_0) \in V(\mathcal{A}')$ such that $X'_0 \cap Elts(\mathcal{G}) = \emptyset$ and $R'_0 \cap Elts(\mathcal{G})$ is either $\emptyset$ or the $\subseteq$-maximal element of $\mathcal{L}_f$. If σ is a winning strategy for the searchers in $\mathbb{G}^\Gamma_\mathcal{G}$, then there exists a strategy $\widetilde{\sigma}$ for the searchers on $\mathbb{G}^\Gamma_{\mathcal{G}'}$ such that any search from (X'_0, R'_0) consistent with $\widetilde{\sigma}$ can be extended to a search $(X'_0, R'_0)(X'_1, R'_1)\cdots$ consistent with $\widetilde{\sigma}$ so that there exists $n \geq 0$ with $R'_n \cap Elts(\mathcal{G}) = \emptyset$, and for all i, $1 \leq i \leq n$, $X'_i = \sigma(X_{i-1}, R_{i-1})$ for some $(X_{i-1}, R_{i-1}) \in V_0(\mathcal{A})$.*

Proof. For $(X', R') \in V(\mathcal{A}')$ with $(X, R) \in V(\mathcal{A})$ where $X = X' \cap \text{Elts}(\mathcal{G})$ and $R = R' \cap \text{Elts}(\mathcal{G})$, define $\widetilde{\sigma}(X', R') := \sigma(X, R)$. From the second condition of restriction reflection, this is a well-defined (partial) strategy: $(X', \sigma(X', R'), R')$ is a successor of (X', R'). We now show that $\widetilde{\sigma}$ is sufficiently defined to satisfy the requirements of the lemma.

Let $\pi' = (X'_0, R'_0)(X'_1, R'_1)\cdots(X'_n, R'_n)$ be a search from (X'_0, R'_0) consistent with $\widetilde{\sigma}$. For $i \geq 0$, let $X_i = X'_i \cap \text{Elts}(\mathcal{G})$ and $R_i = R'_i \cap \text{Elts}(\mathcal{G})$. By the definition of $\widetilde{\sigma}$, $X'_i = \sigma(X_{i-1}, R_{i-1})$ for all i such that $R_{i-1} \neq \emptyset$. Thus if we take n to be the minimum index such that $R_n = \emptyset$, we are done. So suppose there is no n such that $R_n = \emptyset$. We claim:

Claim. $\pi = (X_0, R_0)(X_1, R_1)\cdots$ is a search from $v_I(\mathcal{A})$ consistent with σ.

Proof of claim. We prove this by induction on i, the length of π consistent with σ. From the definition of (X'_0, R'_0), and since $R'_0 \cap \text{Elts}(\mathcal{G}) \neq \emptyset$, $(X_0, R_0) = v_I(\mathcal{A})$, so the claim is true for $i = 0$. Now suppose $(X_0, R_0)\cdots(X_i, R_i)$ is consistent with σ. From the definition of σ', $X_{i+1} = X'_{i+1} = \sigma(X_i, R_i)$. As $(X'_0, R'_0)\cdots(X'_{i+1}, R'_{i+1})$ is consistent with $\widetilde{\sigma}$, and $R'_{i+1} \cap \text{Elts}(\mathcal{G}) \neq \emptyset$, it follows that there is an edge in $E(\mathcal{A}')$ from (X'_i, X'_{i+1}, R'_i) to (X'_{i+1}, R'_{i+1}). Thus, from the third condition of restriction reflection, there is an edge from (X_i, X_{i+1}, R_i) to (X_{i+1}, R_{i+1}). Therefore, $(X_0, R_0)\cdots(X_{i+1}, R_{i+1})$ is consistent with σ as $X_{i+1} = \sigma(X_i, R_i)$ and there is an edge from (X_i, X_{i+1}, R_i) to (X_{i+1}, R_{i+1}). ⊣

Now, since σ is a winning strategy for the searchers, every search from $v_I(\mathcal{A})$ consistent with σ can be extended to a complete search. However, $R_i \neq \emptyset$ for all $i \geq 0$, so π cannot be extended to a complete search. Thus there exists n such that $R_n = \emptyset$, contradicting the assumption that there is no such n. □

We also need to assume that our graph searching games satisfy the following property: if the searchers have a winning strategy from (X, R) then the searchers can play the same strategy and win from (X, S) for any $S \subseteq R$. To be more precise, we require the graph searching game type to be $(\text{id}, \supseteq)$-closed where id is the quasi-simulation family which assigns to each pair of graphs $(\mathcal{G}, \mathcal{G}')$ with $\mathcal{G} = \mathcal{G}'$ the identity relation, and $\supseteq$ is the quasi-simulation family which assigns to each pair of graphs $(\mathcal{G}, \mathcal{G}')$ with $\mathcal{G} = \mathcal{G}'$ the superset relation. Given such a graph searching game type, we can apply Lemma 5.13 to obtain the following:

Lemma 5.55. *Let Γ be a graph searching type which is $(id, \supseteq)$-closed, and let $\mathcal{G}$ be a graph with $\Gamma(\mathcal{G}) = (\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$. For any $(X_1, R_1), (X'_1, R'_1) \in V(\mathcal{A})$ with $X_1 = X'_1$ and $R_1 \supseteq R'_1$ and any strategy σ for the searchers on $\mathbb{G}^\Gamma_\mathcal{G}$, there exists a strategy for the searchers σ' on $\mathbb{G}^\Gamma_\mathcal{G}$ such that for every search $(X'_1, R'_1)(X'_2, R'_2)\cdots$ consistent with σ', there exists a search $(X_1, R_1)(X_2, R_2)\cdots$ consistent with σ with $X_i = X'_i$ and $R_i \supseteq R'_i$ for all i.*

To compute the width of a graph from the widths of its connected components, we need to be able to combine the widths of the components. To do this we require some sort of operation, $\oplus$,

on ω which reflects how our resource measure is computed. For example, if we are interested in the number of searchers required to capture a fugitive, then the function $\max$ is the combining operation we are interested in, the number of searchers required in the whole graph is at most the maximum number of any of its components. In fact, we can use any operation $\oplus$ for which our resource measure is "well-behaved", in the following sense:

Definition 5.56 ($\oplus$-morphism)**.** Let φ be a resource measure and $\oplus : \omega \times \omega \to \omega$ an operation on ω. We say φ is a *$\oplus$-morphism* if $\varphi(\pi \cdot \pi') = \varphi(\pi) \oplus \varphi(\pi')$ for all sequences π and π'.

Remark. We note that if φ is a $\oplus$-morphism, then (on the image of φ) the operation $\oplus$ is uniquely defined. That is, for any resource measure φ, there is at most one possible operation $\oplus$ such that φ is a $\oplus$-morphism. However, we also observe that given any monoid structure $(\mathrm{id}, \oplus)$ on ω and a function f from finite sets to ω, we can define a $\oplus$-morphism $\varphi_\oplus$ as follows:

$$\begin{aligned} \varphi_\oplus(\epsilon) &= \mathrm{id}, \\ \varphi_\oplus(X_1 \cdots X_n) &= f(X_1) \oplus \cdots \oplus f(X_n), \text{ and} \\ \varphi_\oplus(\pi) &= \omega \text{ if } \pi \text{ is infinite.} \end{aligned}$$

We also note that if φ is a $\oplus$-morphism, then, due to the associativity of concatenation, $\oplus$ is necessarily associative. That is, if $a = \varphi(\pi_a)$, $b = \varphi(\pi_b)$, and $c = \varphi(\pi_c)$, then we have:

$$\begin{aligned} (a \oplus b) \oplus c &= \big(\varphi(\pi_a) \oplus \varphi(\pi_b)\big) \oplus \varphi(\pi_c) \\ &= \varphi\big((\pi_a \cdot \pi_b) \cdot \pi_c\big) \\ &= \varphi\big(\pi_a \cdot (\pi_b \cdot \pi_c)\big) \\ &= \varphi(\pi_a) \oplus \big(\varphi(\pi_b) \oplus \varphi(\pi_c)\big) = a \oplus (b \oplus c). \end{aligned}$$

Our next observation is that if we combine the restrictions we have just introduced, then the combination of the widths of the components of a graph provides an upper bound on the width of the graph.

Lemma 5.57. *Let Γ be a graph searching game type which reflects restriction and is $(id, \supseteq)$-closed. Let φ be an order-preserving $\oplus$-morphism. If $\mathcal{G}$ is a graph with (weakly) connected components $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n$, then:*

$$w_{(\Gamma,\varphi)}(\mathcal{G}) \leq \bigoplus_{j=1}^{n} w_{(\Gamma,\varphi)}(\mathcal{G}_j).$$

Proof. Let $\Gamma(\mathcal{G}) = (\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$ and for $1 \leq j \leq n$, let $\Gamma(\mathcal{G}_j) = (\mathcal{L}_s^j, \mathcal{L}_f^j, \mathcal{A}^j)$. For convenience, for each set $X \subseteq \mathrm{Elts}(\mathcal{G})$, let $X^j = X \cap \mathrm{Elts}(\mathcal{G}_j)$. Note that since Γ reflects restriction, if $R_{\max}$ is the $\subseteq$-maximal element of $\mathcal{L}_f$, then $R^j_{\max}$ is the $\subseteq$-maximal element of $\mathcal{L}_f^j$. For each j, $1 \leq j \leq n$, let σ_j be a winning strategy for the searchers such that for every search π_j in $\mathbb{G}^\Gamma_{\mathcal{G}_j}$ consistent with σ_j, $\varphi(\pi_j) \leq w_{(\Gamma,\varphi)}(\mathcal{G}_j)$. The idea is that the strategy defined by playing each of the strategies σ_j sequentially is a winning strategy which has a resource requirement of at most $\bigoplus_{j=1}^n w_{(\Gamma,\varphi)}(\mathcal{G}_j)$. Before we formally define the strategy, we make the following observation.

Claim. Let $(X_1, R_1)(X_2, R_2) \cdots$ be a search in $\mathbb{G}^\Gamma_{\mathcal{G}}$. For any j, $1 \leq j \leq n$, if there exists $n \geq 0$ such that $R^j_n = \emptyset$, then $R^j_i = \emptyset$ for all $i \geq n$.

Proof of claim. Fix j, and suppose n is such that $R^j_n = \emptyset$. Suppose there exists $i > n$ such that $R^j_i \neq \emptyset$. Let k be the minimal index such that $R^j_k \neq \emptyset$, and take $r' \in R^j_k$. From the definition of a graph searching game, there exists $r \in R_{k-1}$ such that r and r' are in the same (weakly) connected component of $\mathcal{G} \setminus (X_{k-1} \cap X_k)$. Thus, as $r' \in \mathrm{Elts}(\mathcal{G}_j)$, it follows that $r \in \mathrm{Elts}(\mathcal{G}_j)$. Thus $r \in R^j_{k-1}$, contradicting the minimality of k. Therefore $R^j_i = \emptyset$ for all $i \geq n$. ⊣

We define σ inductively as follows. If $\mathcal{G}$ has one connected component, let $\sigma = \sigma_1$. Clearly σ is a winning strategy on $\mathcal{G}$, and for any search π consistent with σ we have $\varphi(\pi) \leq w_{(\Gamma,\varphi)}(\mathcal{G}_1)$. Now consider the subgraph $\mathcal{G}' = \bigcup_{j=2}^n \mathcal{G}_j$. Let $\Gamma(\mathcal{G}') = (\mathcal{L}'_s, \mathcal{L}'_f, \mathcal{A}')$. Suppose there exists a winning strategy σ_0 on $\mathbb{G}^\Gamma_{\mathcal{G}'}$ such that for any search π consistent with σ_0 we have $\varphi(\pi) \leq \bigoplus_{j=2}^n w_{(\Gamma,\varphi)}(\mathcal{G}_j)$. Using the notation from Lemma 5.54, let $\widetilde{\sigma_0}$ be the strategy on $\mathbb{G}^\Gamma_{\mathcal{G}}$ defined by σ_0, and let $\widetilde{\sigma_1}$ be the strategy on $\mathbb{G}^\Gamma_{\mathcal{G}}$ defined by σ_1. The strategy σ is as follows: from $(\emptyset, R_{\max})$, play $\widetilde{\sigma_1}$ until a position (X, R) is reached where $R \cap V(\mathcal{G}_1) = \emptyset$. That is, until $R' \cap V(\mathcal{G}') = \emptyset$, let $\sigma(X', R') = \widetilde{\sigma_1}(X', R')$. From Lemma 5.54, we have $X \subseteq V(\mathcal{G}_1)$, so $X \cap V(\mathcal{G}') = \emptyset$, and since $R \subseteq V(\mathcal{G}')$, $R \cap V(\mathcal{G}') \subseteq R'_{\max}$ where $R'_{\max}$ is the $\subseteq$-maximal element of $\mathcal{L}'_f$. Thus $(X, R'_{\max})$ is $(\mathrm{id}, \supseteq)$-related to (X, R). Since $X \cap V(\mathcal{G}') = \emptyset$, it follows that $\widetilde{\sigma_0}(X, R'_{\max})$ is defined. Let σ'_0 be a $(\mathrm{id}, \supseteq)$-simulated strategy of $\widetilde{\sigma_0}$, which, from Lemma , plays from (X, R) when $\widetilde{\sigma_0}$ plays from $(X, R_{\max})$. For all subsequent positions (X', R') reached, including (X, R), define $\sigma(X', R') = \sigma'_0(X', R')$. From the earlier claim, as $R' \cap V(\mathcal{G}_1) = \emptyset$, it follows from the definition of simulated strategies that σ is well-defined for all subsequent positions. As σ_1 and σ' are winning strategies, it also follows that σ is a winning strategy.

Let us now consider the resources required by σ. Let $\pi = (X_0, R_0)(X_1, R_1)\cdots$ be a search consistent with σ. From the definition of σ, it follows that $\pi = \pi_1 \cdot \pi'$ where π_1 is a search consistent with $\widetilde{\sigma_1}$ and π' is a search consistent with σ'_0. Therefore, from Lemmas 5.54 and 5.55, it follows that the sequence $\overline{\pi} = X_0 X_1 \cdots$ is equal to $\overline{\pi_1} \cdot \overline{\pi'}$ where $\overline{\pi_1}$ is the sequence of first components of a search consistent with σ_1 and $\overline{\pi'}$ is the sequence of first components of a search consistent with σ'. Thus

$$\begin{aligned}\varphi(\pi) &= \varphi(\pi_1 \cdot \pi') = \varphi(\pi_1) \oplus \varphi(\pi') \\ &\leq w_{(\Gamma,\varphi)}(\mathcal{G}_1) \oplus \bigoplus_{j=2}^n w_{(\Gamma,\varphi)}(\mathcal{G}_j) = \bigoplus_{j=1}^n w_{(\Gamma,\varphi)}(\mathcal{G}_j).\end{aligned}$$

As this holds for any play consistent with σ, and σ is a winning strategy, it follows that $w_{(\Gamma,\varphi)}(\mathcal{G}) \leq \bigoplus_{j=1}^n w_{(\Gamma,\varphi)}(\mathcal{G}_j)$. □

If we impose some further restrictions on the operation $\oplus$, and suitable restrictions on Γ and φ, we can use Theorem 5.50 to obtain equality in the above result.

Definition 5.58. Let $\oplus : \omega \times \omega \to \omega$ be an operation on ω. We say $\oplus$ is *monotone* if for all $a, b, c, d \in \omega$ with $a \leq b$ and $c \leq d$, $a \oplus c \leq b \oplus d$. We say $\oplus$ is *deflationary* if for all $a \in \omega$, $a \geq a \oplus a$.

Theorem 5.59. *Let Γ be a graph searching game type which respects and reflects restriction and is $(\mathrm{id}, \supseteq)$-closed. Let $\oplus : \omega \times \omega \to \omega$ be an associative, monotone, and deflationary operation on ω. Let φ be a monotone, order-preserving $\oplus$-morphism. If $\mathcal{G}$ is a graph and*

$\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n$ are the (weakly) connected components of $\mathcal{G}$, then,

$$w_{(\Gamma,\varphi)}(\mathcal{G}) = \bigoplus_{j=1}^{n} w_{(\Gamma,\varphi)}(\mathcal{G}_j).$$

Proof. From Lemma 5.57, we have $w_{(\Gamma,\varphi)}(\mathcal{G}) \leq \bigoplus_{j=1}^{n} w_{(\Gamma,\varphi)}(\mathcal{G}_j)$. For the reverse inequality, we observe that as $\mathcal{G}_j$ is a subgraph of $\mathcal{G}$ for all j, we have from Theorem 5.50, $w_{(\Gamma,\varphi)}(\mathcal{G}_j) \leq w_{(\Gamma,\varphi)}(\mathcal{G})$ for all j. Thus, as $\oplus$ is deflationary and monotone:

$$w_{(\Gamma,\varphi)}(\mathcal{G}) \geq \bigoplus_{j=1}^{n} w_{(\Gamma,\varphi)}(\mathcal{G}) \geq \bigoplus_{j=1}^{n} w_{(\Gamma,\varphi)}(\mathcal{G}_j).$$

□

5.4.3 Lexicographic product

We now consider the cops and robber game with the resource measure that indicates the maximum number of cops used by a strategy, $\varphi_{\max}$. We show that, under some simple assumptions, if we replace vertices in a graph with copies of a complete graph with n vertices, the number of cops required to capture the robber increases by a factor of n. We recall from Section 1.1.2 the definition of the lexicographic product. We now introduce some useful relations between a graph and its lexicographic factors. Although these definitions are quite technical, later in the section we introduce some more intuitive properties which we show are sufficient to establish the robustness results we are interested in.

Definition 5.60 ($\mathbf{M}_{\mathcal{H}}$, $\mathbf{D}_{\mathcal{H}}$ and $\mathbf{P}_{\mathcal{H}}$). Let $\mathcal{G}$ and $\mathcal{H}$ be graphs and let $\mathcal{G}' = \mathcal{G} \bullet \mathcal{H}$. We define $\mathbf{M}^{\mathcal{G}}_{\mathcal{H}} \subseteq \mathcal{P}(V(\mathcal{G})) \times \mathcal{P}(V(\mathcal{G}'))$ and $\mathbf{D}^{\mathcal{G}}_{\mathcal{H}}, \mathbf{P}^{\mathcal{G}}_{\mathcal{H}} \subseteq \mathcal{P}(V(\mathcal{G}')) \times \mathcal{P}(V(\mathcal{G}))$ as follows. If $A \subseteq V(\mathcal{G})$ and $B \subseteq V(\mathcal{G}')$, then

- $A\ \mathbf{M}^{\mathcal{G}}_{\mathcal{H}}\ B$ if $B = A \times V(\mathcal{H})$,
- $B\ \mathbf{D}^{\mathcal{G}}_{\mathcal{H}}\ A$ if $A = \{u : (u, v) \in B \text{ for all } v \in V(\mathcal{H})\}$,
- $B\ \mathbf{P}^{\mathcal{G}}_{\mathcal{H}}\ A$ if $A = \{u : (u, v) \in B \text{ for some } v \in V(\mathcal{H})\}$.

The following results follow immediately from Lemma 5.16 and provide an idea of the results we are interested in.

Lemma 5.61. *Let $\mathcal{G}$ and $\mathcal{H}$ be graphs and let $\mathcal{G}' = \mathcal{G} \bullet \mathcal{H}$. Let $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ be a cops and robber game on $\mathcal{G}$ and $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ be a cops and robber game on $\mathcal{G}'$. If $(\mathbf{M}^{\mathcal{G}}_{\mathcal{H}}, \mathbf{M}^{\mathcal{G}}_{\mathcal{H}})$ is a searching simulation from $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ to $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ and k cops have a winning strategy on $\mathbb{G}^{\Gamma}_{\mathcal{G}}$, then $k \cdot |V(\mathcal{H})|$ cops have a winning strategy on $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$.*

Proof. Let σ be a winning strategy for the cops on $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ which uses at most k cops. Let σ' be a strategy for the cops on $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ $(\mathbf{M}^{\mathcal{G}}_{\mathcal{H}}, \mathbf{M}^{\mathcal{G}}_{\mathcal{H}})$-simulated by σ. From Lemma 5.16, σ' is a winning strategy for the cops. From the definition of $\mathbf{M}^{\mathcal{G}}_{\mathcal{H}}$, for each position (X', R') of $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ we have $\sigma'(X', R') = \sigma(X, R) \times V(\mathcal{H})$ for some position (X, R) of $\mathbb{G}^{\Gamma}_{\mathcal{G}}$. So $|\sigma'(X', R')| \leq k \cdot |V(\mathcal{H})|$, and therefore σ' is a winning strategy for at most $k \cdot |V(\mathcal{H})|$ cops. □

Lemma 5.62. *Let $\mathcal{G}$ and $\mathcal{H}$ be graphs and let $\mathcal{G}' = \mathcal{G} \bullet \mathcal{H}$. Let $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ be a cops and robber game on $\mathcal{G}$ and $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ be a cops and robber game on $\mathcal{G}'$. If $(\mathbf{D}^{\mathcal{G}}_{\mathcal{H}}, \mathbf{P}^{\mathcal{G}}_{\mathcal{H}})$ is a searching simulation from $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ to $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ and the robber can defeat $k-1$ cops on $\mathbb{G}^{\Gamma}_{\mathcal{G}}$, then the robber can defeat $k \cdot |V(\mathcal{H})| - 1$ cops on $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$.*

Proof. We consider the contrapositive: suppose $k \cdot |V(\mathcal{H})| - 1$ cops have a winning strategy σ' on $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$. We show that $k-1$ cops have a winning strategy on $\mathbb{G}^{\Gamma}_{\mathcal{G}}$. Let σ be a strategy $(\mathbf{D}^{\mathcal{G}}_{\mathcal{H}}, \mathbf{P}^{\mathcal{G}}_{\mathcal{H}})$-simulated by σ'. From Lemma 5.16, σ is a winning strategy for the cops. Suppose $|\sigma(X,R)| \geq k$ for some position (X,R). From the definition of σ, there exists a position (X',R') of $\mathbb{G}^{\Gamma'}_{\mathcal{G}'}$ such that $\sigma'(X',R')\ \mathbf{D}^{\mathcal{G}}_{\mathcal{H}}\ \sigma(X,R)$. But then, as $|\sigma(X,R)| \geq k$, $|\sigma'(X',R')| \geq k \cdot |V(\mathcal{H})|$, contradicting the assumption that σ' was a strategy for $k \cdot |V(\mathcal{H})| - 1$ cops. Thus σ' is a winning strategy for $k-1$ cops. □

With these two results in mind, we introduce two quasi-simulation families which we use to define the restriction on graph searching game types that we require for games to be well-behaved under lexicographic product.

Definition 5.63 (Composition-expanding)**.** Let $\mathfrak{M}$ be the quasi-simulation family which assigns to each pair of graphs $(\mathcal{G}, \mathcal{G}')$, where $\mathcal{G}' = \mathcal{G} \bullet \mathcal{K}$ for some complete graph $\mathcal{K}$, the pair of relations $(\mathbf{M}^{\mathcal{G}}_{\mathcal{K}}, \mathbf{M}^{\mathcal{G}}_{\mathcal{K}})$. Let $\mathfrak{D}$ be the quasi-simulation family which assigns to each pair of graphs $(\mathcal{G}', \mathcal{G})$, where $\mathcal{G}' = \mathcal{G} \bullet \mathcal{K}$ for some complete graph $\mathcal{K}$, the pair of relations $(\mathbf{D}^{\mathcal{G}}_{\mathcal{K}}, \mathbf{P}^{\mathcal{G}}_{\mathcal{K}})$. Let Γ be a cops and robber game type. We say Γ is *composition-expanding* if it is $\mathfrak{M}$-closed and $\mathfrak{D}$-closed.

Using Lemmas 5.61 and 5.62, we obtain:

Theorem 5.64. *Let Γ be a composition-expanding cops and robber game type. Let $\mathcal{G}$ be a graph, and let $\mathcal{K}_n$ be the complete graph on n vertices. Then*

$$n \cdot w_{(\Gamma,\varphi_{\max})}(\mathcal{G}) = w_{(\Gamma,\varphi_{\max})}(\mathcal{G} \bullet \mathcal{K}_n).$$

Proof. Let $w_{(\Gamma,\varphi_{\max})}(\mathcal{G}) = k$ and $w_{(\Gamma,\varphi_{\max})}(\mathcal{G} \bullet \mathcal{K}_n) = m$. From Lemma 5.61, we have $m \leq n \cdot k$, so suppose $m = n \cdot k - r$. But if $r \geq 1$, then by Lemma 5.62, $w_{(\Gamma,\varphi_{\max})}(\mathcal{G}) \leq k-1$. Thus $r = 0$ and the result follows. □

To help identify cops and robber game types which are composition-expanding, we now present an alternative characterization of composition-expanding, similar to the definition of restriction respecting in Definition 5.21. Just as with Lemma 5.23, the proof follows directly from the definitions, and is therefore omitted.

Lemma 5.65. *Let Γ be a cops and robber game type such that for all graphs $\mathcal{G}$ and all complete graphs $\mathcal{K}$, where $\Gamma(\mathcal{G}) = (\mathcal{L}_c, \mathcal{L}_r, \mathcal{A})$, $\Gamma(\mathcal{G} \bullet \mathcal{K}) = (\mathcal{L}'_c, \mathcal{L}'_r, \mathcal{A}')$ and:*

(I) If there is an edge in $E(\mathcal{A})$ from (Y,S) to (Y,Y',S) and $(X,R) \in V_0(\mathcal{A}')$ for $X = Y \times V(\mathcal{K})$ and $R = S \times V(\mathcal{K})$, then there is an edge in $E(\mathcal{A}')$ from (X,R) to (X,X',R) where $X' = Y' \times V(\mathcal{K})$;

(II) If there is an edge in $E(\mathcal{A}')$ from $(X,R) \in V_0(\mathcal{A}')$ to (X,X',R) and $(Y,S) \in V_0(\mathcal{A})$ for $Y = \{u : (u,v) \in X \text{ for all } v \in V(\mathcal{K})\}$ and $S = \{u : (u,v) \in R \text{ for some } v \in V(\mathcal{K})\}$, then there is an edge in $E(\mathcal{A})$ from (Y,S) to (Y,Y',S) where $Y' = \{u : (u,v) \in X' \text{ for all } v \in V(\mathcal{K})\}$;

(III) If there is an edge in $E(\mathcal{A}')$ from (X, X', R) to (X', R') and $(Y, Y', S) \in V_1(\mathcal{A})$ where $X = Y \times V(\mathcal{K})$, $X' = Y' \times V(\mathcal{K})$, and $R = S \times V(\mathcal{K})$; and then $R' = S' \times V(\mathcal{K})$ for some S' and there is an edge in $E(\mathcal{A})$ from (Y, Y', S) to (Y', S')

(IV) If there is an edge in $E(\mathcal{A})$ from (Y, Y', S) to (Y', S') and $(X, X', R) \in V_1(\mathcal{A}')$ where $Y = \{u : (u, v) \in X \text{ for all } v \in V(\mathcal{K})\}$, $Y' = \{u : (u, v) \in X' \text{ for all } v \in V(\mathcal{K})\}$, and $S = \{u : (u, v) \in R \text{ for some } v \in V(\mathcal{K})\}$, then there is an edge in $E(\mathcal{A}')$ from (X, X', R) to (X', R') for some R' such that $S' = \{u : (u, v) \in R' \text{ for some } v \in V(\mathcal{K})\}$,

then Γ is composition-expanding.

We observed in Lemma 5.51 that the $\supset$ relation satisfied the necessary conditions for $(\supset, \supset)$-simulation to respect fugitive and searcher-monotonicity. We now show that the relations $\mathbf{M}$, $\mathbf{D}$, and $\mathbf{P}$ also satisfy similar conditions implying that Theorem 5.64 holds for robber-monotone and cop-monotone width.

Lemma 5.66. *Let $\mathcal{G}$ be a graph and $\mathcal{K}$ a complete graph.*

1. *The relation $\mathbf{M}_{\mathcal{K}}^{\mathcal{G}}$ is monotone and $\cap$-compatible.*
2. *The relation $\mathbf{D}_{\mathcal{K}}^{\mathcal{G}}$ is monotone and $\cap$-compatible.*
3. *The relation $\mathbf{P}_{\mathcal{K}}^{\mathcal{G}}$ is monotone.*

Proof. 1: Take $X, Y \subseteq V(\mathcal{G})$ and $X', Y' \subseteq V(\mathcal{G} \bullet \mathcal{K})$ such that $X \ \mathbf{M}_{\mathcal{K}}^{\mathcal{G}} \ X'$ and $Y \ \mathbf{M}_{\mathcal{K}}^{\mathcal{G}} \ Y'$. By the definition of $\mathbf{M}_{\mathcal{K}}^{\mathcal{G}}$, it follows that $X' = X \times V(\mathcal{K})$ and $Y' = Y \times V(\mathcal{K})$. So if $X \subseteq Y$, $X' \subseteq Y'$, and so $\mathbf{M}_{\mathcal{K}}^{\mathcal{G}}$ is monotone. Furthermore, since $(X \cap Y) \times V(\mathcal{K}) = (X \times V(\mathcal{K})) \cap (Y \times V(\mathcal{K}))$, it follows that $\mathbf{M}_{\mathcal{K}}^{\mathcal{G}}$ is $\cap$-compatible.

2: Take $X, Y \subseteq V(\mathcal{G})$ and $X', Y' \subseteq V(\mathcal{G} \bullet \mathcal{K})$ such that $X' \ \mathbf{D}_{\mathcal{K}}^{\mathcal{G}} \ X$ and $Y' \ \mathbf{D}_{\mathcal{K}}^{\mathcal{G}} \ Y$. By the definition of $\mathbf{D}_{\mathcal{K}}^{\mathcal{G}}$, it follows that $X = \{u : (u, v) \in X' \text{ for all } v \in V(\mathcal{K})\}$ and $Y = \{u : (u, v) \in Y' \text{ for all } v \in V(\mathcal{K})\}$. Now if $X' \subseteq Y'$, it follows that $X = \{u : (u, v) \in X' \text{ for all } v \in V(\mathcal{K})\} \subseteq \{u : (u, v) \in Y' \text{ for all } v \in V(\mathcal{K})\} = Y$. Thus $\mathbf{D}_{\mathcal{K}}^{\mathcal{G}}$ is monotone. Furthermore, $\{u : (u, v) \in X' \cap Y' \text{ for all } v \in V(\mathcal{K})\} = \{u : (u, v) \in X' \text{ for all } v \in V(\mathcal{K})\} \cap \{u : (u, v) \in Y' \text{ for all } v \in V(\mathcal{K})\}$, so $(X' \cap Y') \ \mathbf{D}_{\mathcal{K}}^{\mathcal{G}} \ X \cap Y$, and hence $\mathbf{D}_{\mathcal{K}}^{\mathcal{G}}$ is $\cap$-compatible.

3: Take $X, Y \subseteq V(\mathcal{G})$ and $X', Y' \subseteq V(\mathcal{G} \bullet \mathcal{K})$ such that $X' \ \mathbf{P}_{\mathcal{K}}^{\mathcal{G}} \ X$ and $Y' \ \mathbf{P}_{\mathcal{K}}^{\mathcal{G}} \ Y$. By the definition of $\mathbf{P}_{\mathcal{K}}^{\mathcal{G}}$, it follows that $X = \{u : (u, v) \in X' \text{ for some } v \in V(\mathcal{K})\}$ and $Y = \{u : (u, v) \in Y' \text{ for some } v \in V(\mathcal{K})\}$. Now if $X' \subseteq Y'$, it follows that $X = \{u : (u, v) \in X' \text{ for some } v \in V(\mathcal{K})\} \subseteq \{u : (u, v) \in Y' \text{ for some } v \in V(\mathcal{K})\} = Y$. Thus $\mathbf{P}_{\mathcal{K}}^{\mathcal{G}}$ is monotone. □

Corollary 5.67. *Let Γ be a composition-expanding cops and robber game type. Let $\mathcal{G}$ be a graph, and let $\mathcal{K}_n$ be the complete graph on n vertices. Then:*

1. *The robber-monotone $(\Gamma, \varphi_{\max})$-width of $\mathcal{G} \bullet \mathcal{K}_n$ is n times the robber-monotone $(\Gamma, \varphi_{\max})$-width of $\mathcal{G}$.*
2. *The cop-monotone $(\Gamma, \varphi_{\max})$-width of $\mathcal{G} \bullet \mathcal{K}_n$ is n times the cop-monotone $(\Gamma, \varphi_{\max})$-width of $\mathcal{G}$.*

5.5 Complexity results

To conclude this chapter we consider the complexity of the problem of determining the (Γ, φ)-width of a graph. More precisely, for a graph searching game type Γ and an order-preserving resource measure φ, we are interested in the complexity of the following problem:

(Γ, φ)-WIDTH	
Instance:	A graph $\mathcal{G}$ and $k \in \omega$
Problem:	Is the (Γ, φ)-width of $\mathcal{G}$ at most k?

Of course, the complexity of this problem is dependent on how difficult it is to compute the arena of $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ and the resource function φ. To have a sensible analysis, we assume that we can compute these in amortized constant time, that is, we can compute a path of length n in the arena, or the φ-value of a sequence of n sets in time $O(n)$. In practice computing edges of the arena and values of φ are more likely to require time polynomial in the size of the graph, but as the bounds we obtain are generally exponential in the size of the graph, this assumption is not going to significantly affect the overall complexity.

From Definition 5.1, we know that a graph searching game $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ defined by $(\mathcal{L}_s, \mathcal{L}_f, \mathcal{A})$ is a simple game, so it might appear at first that determining if the searchers have a winning strategy can be decided in time linear in the size of the arena, as per Theorem 2.60. However, for an arbitrary resource measure φ, whether a vertex of the arena is winning for the searchers in the resource-bounded game is dependent on the play to that vertex. So it could be the case that for any strategy, all possible consistent plays have to be checked to ensure the resource measure is bounded. Hence it may not be possible to do better than to iterate through all possible strategies and all consistent searches, or equivalently, all possible plays in the arena. However, as we observed after Definition 5.36, we need only consider plays that are simple paths in the arena, so this is at least decidable. Since every play can be characterized by a search, and a search is a sequence of positions, there are at most $O(|V_0(\mathcal{A})|!)$ plays that might have to be checked. Now $V_0(\mathcal{A})$ consists of pairs of subsets of $\mathrm{Elts}(\mathcal{G})$, thus $|V_0(\mathcal{A})| = O(4^{|\mathrm{Elts}(\mathcal{G})|}) = O(4^{\|\mathcal{G}\|})$, giving us the following bound:

Proposition 5.68. *Let Γ be a graph searching game type and φ an order-preserving resource measure.* (Γ, φ)-WIDTH *can be decided in time $O(4^n!)$.*

We can do considerably better by considering specific resource measures, in particular the measure $\varphi_{\max}$. In Lemma 5.40, we saw how the existence of a resource bounded winning strategy is equivalent to the existence of a winning strategy in a game with a smaller arena: the parameterized game defined in Definition 5.26. We can use Theorem 2.60 to decide if the cops have a winning strategy in this parameterized game in linear time, and therefore determine if the cops have a resource bounded winning strategy in the original game. More precisely,

Proposition 5.69. *Let Γ be the cops and visible robber game type defined in Definition 5.24. Then* $(\Gamma, \varphi_{\max})$-WIDTH *can be decided in time $O(n^{2k+4})$.*

Proof. Suppose $\mathcal{G}$, an undirected graph, and $k \in \omega$ are given. Let Γ' be the k-cops and visible robber game type defined in Definition 5.26, and suppose $\Gamma'(\mathcal{G}) = (\mathcal{L}_c, \mathcal{L}_r, \mathcal{A})$. From Lemma 5.40, we have that k cops have a winning strategy in $\mathbb{G}^{\Gamma}_{\mathcal{G}}$ if, and only if, the cops have a winning strategy in $\mathbb{G}^{\Gamma'}_{\mathcal{G}}$. From Theorem 2.60, we can determine if the cops have a winning

strategy in $\mathbb{G}_{\mathcal{G}}^{\Gamma'}$ in time $O(|E(\mathcal{A})|)$, so it suffices to find an upper bound on $|E(\mathcal{A})|$. From the definition of the game, we observe that for each $X, X' \in \mathcal{L}_c$ there are at most $|V(\mathcal{G})|$ sets R such that $(X, R) \in V_0(\mathcal{A})$ and $(X, X', R) \in V_1(\mathcal{A})$. Therefore, from the definition of $\mathcal{A}$ we see that each element (X, X', R) of $V_1(\mathcal{A})$ has a unique incoming edge (from (X, R)) and at most $|V(\mathcal{G})|$ outgoing edges (to (X', R')). Thus the number of edges is at most $(|V(\mathcal{G})| + 1)|V_1(\mathcal{A})|$. From the definition of $\mathcal{L}_c$, we have $|\mathcal{L}_c| \leq |V(\mathcal{G})|^{k+1}$, thus $|V_1(\mathcal{A})|$ is at most $|\mathcal{L}_c||\mathcal{L}_c||V(\mathcal{G})| \leq |V(\mathcal{G})|^{2k+3}$. Therefore, the number of edges of $\mathcal{A}$ is bounded by $O(|V(\mathcal{G})|^{2k+4})$, and the result follows. □

The parameterized class of games we defined in Definition 5.26 is easily extended to other graph searching game types, so we can use a similar argument as above to decide $(\Gamma, \varphi_{\max})$-WIDTH more efficiently than Proposition 5.68. In the more general case, we may not be able to bound the size of $V_1(\mathcal{A})$ as efficiently, nor the number of outgoing edges from elements of $V_1(\mathcal{A})$. However, we observe that $V_1(\mathcal{A}) \subseteq \mathcal{L}_s \times \mathcal{L}_s \times \mathcal{L}_f$, so $|V_1(\mathcal{A})| \leq \|\mathcal{G}\|^k \cdot \|\mathcal{G}\|^k \cdot 2^{\|\mathcal{G}\|}$, and there are at most $|\mathcal{L}_f| \leq 2^{\|\mathcal{G}\|}$ outgoing edges from any element of $V_1(\mathcal{A})$. This gives us the following improvement for deciding $(\Gamma, \varphi_{\max})$-WIDTH:

Proposition 5.70. *Let Γ be a graph searching game type.* $(\Gamma, \varphi_{\max})$-WIDTH *can be decided in time $O(n^{2k+2}4^n)$.*

We observe that all the algorithms we have so far considered are constructive: if the algorithm returns a positive answer, then it is possible to extract a winning strategy for the searchers.

We conclude the section by considering the complexity of determining the fugitive-monotone and searcher-monotone widths of a graph. As we observed following Lemma 5.11, the restriction to fugitive-monotone strategies can be enforced by removing edges from the arena. It therefore follows that the bounds we obtained for the general games are applicable to the fugitive-monotone case.

Proposition 5.71. *Let Γ be a graph searching game type.*

(i) FUGITIVE-MONOTONE $(\Gamma, \varphi_{\max})$-WIDTH *can be decided in time $O(n^{2k+2}4^n)$, and*

(ii) If Γ is the cops and visible robber game type defined in Definition 5.24. Then FUGITIVE-MONOTONE $(\Gamma, \varphi$ *can be decided in time $O(n^{2k+4})$.*

Unfortunately, for searcher-monotone strategies the situation is not as straightforward. Indeed, just as with arbitrary resource measures, the algorithm of Theorem 2.60 cannot, in general, be used as the set of successors available from (X, R) is dependent on the play to (X, R). Thus in the searcher-monotone case, we can in general do no better than the bounds obtained for an arbitrary resource measure.

Proposition 5.72. *Let Γ be a graph searching game type.* SEARCHER-MONOTONE $(\Gamma, \varphi_{\max})$-WIDTH *can be decided in time $O(4^n!)$.*

Chapter 6

Digraph measures: DAG-width

In Chapter 4 we discussed the problem of finding a measure of complexity for digraphs. We reviewed the definition of tree-width, arguably one of the most suitable measures of complexity for undirected graphs, and we considered the problem of finding a suitable generalization of tree-width for directed graphs. In Chapter 5 we introduced graph searching games, a useful tool for developing robust measures of graph complexity, and saw that several such games can be used to characterize tree-width. In this chapter we introduce a complexity measure for directed graphs which we argue is a more natural generalization of tree-width than directed tree-width. We introduce a decomposition which, unlike arboreal decompositions, is defined in a similar manner to tree decompositions. Just as tree decompositions are decompositions based on trees, our decompositions are based on directed, acyclic graphs (DAGs), so we use the name *DAG-decompositions*. And just as tree decompositions give rise to tree-width, DAG-decompositions give rise to a graph parameter which we call *DAG-width*.

We show that DAG-decompositions and DAG-width enjoy many properties similar to tree decompositions and tree-width. For example, in Theorem 6.28, we show that we may assume a DAG-decomposition satisfies certain conditions similar to those of nice tree decompositions, introduced in [Bod97]. This normalized form is particularly useful for designing dynamic programming algorithms which run efficiently on classes of directed graphs of bounded DAG-width. We see this in Section 6.3.3 when we present such an algorithm for parity games. But perhaps the strongest point in favour of DAG-width being a more natural generalization of tree-width is that it can be characterized by a natural generalization of the cops and visible robber game, a graph searching game which we saw in Chapter 5 characterizes tree-width. As the generalized game is particularly dependent on directed paths in the graph, this suggests that DAG-width is a good indicator of the directed connectivity of a digraph, a notion we discussed in Chapter 4.

The game characterization of DAG-width also provides support for the argument that DAG-width is a good measure of digraph complexity. For example, it is straightforward to show that DAG-width does not increase under the taking of subgraphs, and that the DAG-width of a graph can be computed from the DAG-width of its strongly connected components.

After we introduce DAG-width and its associated graph searching game, we consider the algorithmic benefits of DAG-width. As a digraph measure, DAG-width lies between tree-width and directed tree-width. That is, classes of graphs of bounded tree-width have bounded DAG-width and graphs of bounded DAG-width have bounded directed tree-width. In particular this implies that algorithms which are efficient on graphs of bounded directed tree-width are effi-

cient on graphs of bounded DAG-width, so in particular Theorem 4.13 applies also to graphs of bounded DAG-width. In this chapter we extend this algorithmic result and show that parity games can be decided in polynomial time on arenas of bounded DAG-width, something which is not currently known for graphs of bounded directed tree-width. We also show that DAG-width, tree-width and directed tree-width are different measures by exhibiting a class of digraphs with bounded DAG-width and unbounded tree-width and a class of digraphs with bounded directed tree-width and unbounded DAG-width. This suggests that weak connectivity, directed connectivity and strong connectivity are three very different properties of directed graphs.

The chapter is arranged as follows. In Section 6.1 we introduce the cops and visible robber game for directed graphs and we establish some results to help gain an understanding of the game. We then define DAG-decompositions in Section 6.2, and show the equivalence between DAG-width and the number of cops required to capture the fugitive with a monotone strategy. In Section 6.3 we discuss some algorithmic aspects of DAG-width. We also prove the existence of a polynomial time algorithm for solving parity games on arenas of bounded DAG-width, and in Section 6.4 we relate DAG-width to other measures of graph connectivity, in particular tree-width, directed tree-width and directed path-width.

6.1 Cops and visible robber game

We recall from Chapter 5 the cops and visible robber game from Example 5.2.1. In this game a number of cops and a robber occupy vertices of an undirected graph and the objective of the cops is to capture the robber. The cops move by removing some of their number from the graph and announcing a set of vertices to be occupied. Following this, the robber can move at great speed along paths in the graph to avoid capture, however he is not permitted to pass through any cop which remains on the graph. The cops then occupy the vertices that were announced, and if the robber is located on one of these vertices then he is captured. The location of the robber in the graph is always known to the cops. In Theorem 5.37 we saw that the minimum number of cops required to capture a robber on an undirected graph is equal to one more than the tree-width of the graph.

We now consider the natural extension of this game to directed graphs, where the robber is constrained to move along directed cop-free paths. More precisely,

Definition 6.1 (Cops and visible robber game)**.** Let $\mathcal{G}$ be a directed graph. The *cops and visible robber game on* $\mathcal{G}$ is the cops and robber game defined by $(\mathcal{L}_c, \mathcal{L}_r, \mathcal{A})$, where

- $\mathcal{L}_c = \mathcal{P}(V(\mathcal{G}))$ and $\mathcal{L}_r = \mathcal{P}(V(\mathcal{G})) \setminus \{\emptyset\}$,
- $V_0(\mathcal{A})$ consists of $(\emptyset, V(\mathcal{G}))$ together with pairs $(X, R) \in \mathcal{L}_c \times \mathcal{L}_r$ such that $R = \mathit{Reach}_{\mathcal{G}\setminus X}(r)$ for some $r \in V(\mathcal{G})$,
- $V_1(\mathcal{A})$ consists of triples $(X, X', R) \in V_1(\mathcal{A})$ for all $(X, R) \in V_0(\mathcal{A})$ and all $X' \in \mathcal{L}_c$,
- For all $(X, R) \in V_0(\mathcal{A})$ and all $X' \in \mathcal{L}_c$ there is an edge in $E(\mathcal{A})$ from (X, R) to (X, X', R), and
- If $R' = \mathit{Reach}_{\mathcal{G}\setminus X'}(r')$ then there is an edge in $E(\mathcal{A})$ from (X, X', R) to (X', R') if, and only if, $r' \in \mathit{Reach}_{\mathcal{G}\setminus(X\cap X')}(R)$.

Remark. In the sequel, it may be more convenient to view (non-initial) positions of the game as pairs (X, r) with $X \subseteq V(\mathcal{G})$ and $r \in V(\mathcal{G})$ to represent the position (X, R) where $R = \textit{Reach}_{\mathcal{G}\setminus X}(r)$.

We recall from Chapter 5 the definitions of a *search* and a *strategy*. As with the game characterizing tree-width, we are interested in the minimum number of cops required to capture the robber. Because of this, and from the definition of the game, it follows that we may assume the first move of the cops is to not place any cops on the graph and "wait and see" where the robber moves: if the robber can win from $(\emptyset, r_1)$ for some $r_1 \in V(\mathcal{G})$ then he can win from $(\emptyset, V(\mathcal{G}))$, and conversely, if the cops have a winning strategy σ which uses at most k cops from $(\emptyset, r)$ for all $r \in V(\mathcal{G})$, then the strategy defined by playing $\emptyset$ at $(\emptyset, V(\mathcal{G}))$ and σ otherwise is also a winning strategy which uses at most k cops. In view of this, and the above remark, we introduce a more practical definition of a strategy where the strategy is only defined for positions (X, r) where $X \subseteq V(\mathcal{G})$, $|X| \leq k$, and $r \in V(\mathcal{G})$.

Definition 6.2 (k-cop strategy)**.** Let $\mathcal{G}$ be a directed graph, and consider the cops and visible robber game on $\mathcal{G}$. A *(k-cop) strategy* for the cops is a function $\sigma : [V(\mathcal{G})]^{\leq k} \times V(\mathcal{G}) \to [V(\mathcal{G})]^{\leq k}$. A search $(X_1, r_1)(X_2, r_2)\cdots$ is *consistent* with a strategy σ if $X_{i+1} = \sigma(X_i, r_i)$ for all i. A strategy σ is a *winning strategy*, if every search consistent with σ is finite.

In a similar way, we can define a strategy for the robber against k cops.

Definition 6.3 (Strategy against k cops)**.** Let $\mathcal{G}$ be a directed graph, and consider the cops and visible robber game on $\mathcal{G}$. A *strategy against k cops* is a function $\rho : [V(\mathcal{G})]^{\leq k} \times [V(\mathcal{G})]^{\leq k} \times V(\mathcal{G}) \to V(\mathcal{G})$ such that for all $X, X' \subseteq V(\mathcal{G})$ and $r \in V(\mathcal{G})\setminus X$, $\rho(X, X', r) \in \textit{Reach}_{\mathcal{G}\setminus(X\cap X')}(r)$. A search $(X_1, r_1)(X_2, r_2)\cdots$ is *consistent* with a strategy ρ if $r_{i+1} = \rho(X_i, X_{i+1}, r_i)$ for all i.

We observe that, similar to the game on undirected graphs, variants of the cops and visible robber game where only one cop can be moved at a time, or the cops are lifted and placed in separate moves are all equivalent in that the number of cops required to capture the robber on a graph does not depend on the variant.

We call the graph searching width (recall Definition 5.36) associated with this game and the resource we are interested in bounding, the *cop number* of the graph. That is,

Definition 6.4 (Cop number)**.** The *cop number* of a directed graph $\mathcal{G}$ is the least k such that k cops have a strategy to win the cops and visible robber game on $\mathcal{G}$.

Before we introduce the technical aspects of this game needed in later sections, we present a couple of results that illustrate some of its properties.

Lemma 6.5. *Let $\mathcal{G}$ be a (finite) non-empty directed graph. At least one cop is required to capture a visible robber on $\mathcal{G}$ and exactly one cop is required if, and only if, $\mathcal{G}$ is acyclic.*

Proof. As we have no requirement that the robber moves, as long as there is one vertex, the robber can defeat zero cops by remaining at that vertex. That is, if $v \in V(\mathcal{G})$, then function ρ defined by $\rho(\emptyset, \emptyset, v) = v$ is clearly a winning strategy against 0 cops.

If $\mathcal{G}$ is acyclic, then one cop can catch the robber by always playing to the current position of the robber. Eventually, the robber will not be able to move and the cops will capture him. More precisely, define $\sigma(X, r) = \{r\}$. Then for any search $(X_0, r_0)(X_1, r_1)\cdots$ consistent with σ, we observe that for all i, $r_i \neq r_{i+1}$ and there is a directed path from r_i to r_{i+1}. Since $\mathcal{G}$ is finite and

acyclic, it follows that every search consistent with σ must be finite and therefore winning for the cops.

Conversely, if $\mathcal{G}$ has a cycle $(v_1, v_2, \ldots, v_m)$, then the robber can defeat one cop by forever staying in the cycle. That is, for all $r \in V(\mathcal{G})$ and $X \in [V(\mathcal{G})]^{\leq 1}$ let $\rho(X, X', r) = v_1$ for all X' such that $v_1 \notin X'$ and $\rho(X, \{v_1\}, r) = v_2$. This is clearly a strategy for the robber against one cop, and as any search consistent with ρ can be extended to an infinite search, it is winning for the robber. □

The cops and visible robber games we have already seen Chapter 5 characterizing tree-width and directed tree-width have the property that they are invariant under edge reversal. That is, the number of cops required to catch the robber does not change if the directions of all the edges of the graph are reversed. As we see below, this is not the case for the game we consider here. One exception is graphs of cop number 1, that is, acyclic graphs. We recall from Section 1.1.2, the definition of $\mathcal{G}^{op}$.

Proposition 6.6. *The cop number of a directed graph $\mathcal{G}$ is 1 if, and only if, the cop number of $\mathcal{G}^{op}$ is 1.*

Proof. This follows from Lemma 6.5 by observing that $\mathcal{G}$ is acyclic if, and only if, $\mathcal{G}^{op}$ is acyclic. □

Proposition 6.7. *For any j, k with $2 \leq j \leq k$, there exists a graph $\mathcal{T}_k^j$ with cop number j such that the cop number of $(\mathcal{T}_k^j)^{op}$ is k.*

Proof. Informally, $\mathcal{T}_k^j$ is a binary branching tree of height k such that every vertex has edges to all its descendants, and edges back to its $j-1$ nearest ancestors.[1] More precisely, $\mathcal{T}_k^j$ is the directed graph defined as follows:

- $V(\mathcal{T}_k^j) = \{w \in \{0,1\}^* : |w| < k\}$, and
- $(w_1, w_2) \in E(\mathcal{T}_k^j)$ if, and only if, either $w_1 \prec w_2$ or $w_2 \prec w_1$ and $|w_1| - |w_2| < j$, where $\prec$ is the prefix ordering on $\{0,1\}^*$.

We now show that the cop number of $\mathcal{T}_k^j$ is j and the cop number of $(\mathcal{T}_k^j)^{op}$ is k. First we see that j cops have a winning strategy on $\mathcal{T}_k^j$ by initially playing on the root then following the robber down, in a leap-frogging manner, whichever subtree he plays in. More precisely, we inductively define the strategy σ as follows. Initially, $\sigma(\emptyset, V(\mathcal{G})) = \{\epsilon\}$. We observe that from the definition of the edge relation, if the robber chooses to respond by moving to a vertex w with first symbol 0, then he is unable to reach any vertex w' with first symbol 1. Similarly if the robber chooses to move to a vertex with first symbol 1, he cannot reach any vertex in the 0-subtree. Now suppose the cops are on X and the robber is on w_r and X and w_r satisfy the following:

$$\text{There exists } w_{\min} \text{ and } w_{\max} \text{ such that } X = \{w : w_{\min} \preceq w \prec w_{\max}\} \text{ and } \textit{Reach}_{\mathcal{T}_k^j \setminus X}(w_r) = \{w : w_{\max} \preceq w\}. \qquad (*)$$

[1] To aid informal descriptions we view this graph as a directed tree with additional structure. Thus we use descendants, ancestors, root and leaves to refer to various vertices in the graph as they would be in the underlying directed tree.

Then $w_{\max}$ is the next vertex to be occupied by a cop. If $|X| < j$, then $\sigma(X, w_r) = X \cup \{w_{\max}\}$, otherwise if $|X| = j$, $\sigma(X, w_r) = X \setminus \{w_{\min}\} \cup \{w_{\max}\}$. Let w'_r be the next location of the robber after the cops move from X to $X' = \sigma(X, w_r)$. We show that the resulting position (X', w'_r) satisfies $(*)$. Clearly from the definition of σ, we have either $X' = \{w : w_{\min} \preceq w \preceq w_{\max}\}$ or $X' = \{w : w_{\min} \prec w \preceq w_{\max}\}$, so the first part of $(*)$ is true. Next we show that $w'_r \in \mathit{Reach}_{\mathcal{T}_k^j \setminus X}(X) w_r \setminus \{w_{\max}\}$. Clearly, if $X' \supseteq X$ this is true, so we need only consider the case when $|X| = j$. But this implies $|w_{\max}| - |w_{\min}| = j$, thus there are no edges from $w_{\max}$ to $w_{\min}$. As $w_{\min}$ is the only vertex vacated and every vertex reachable from w_r is reachable from $w_{\max}$, the set of vertices reachable by the robber must decrease. Now let w' be the shortest word which is a prefix of w'_r and for which $w_{\max}$ is a proper prefix. It follows from the definition of the edge relation that every vertex which the robber can reach must have w' as a prefix. Thus $\mathit{Reach}_{\mathcal{T}_k^j \setminus X}(X') w'_r = \{w : w' \preceq w\}$. Clearly the strategy σ is a strategy for j cops, we now show that it is winning. We observe that for every search consistent with σ, the sequence of $w_{\max}$ is a sequence of words of increasing length. So after k moves there will be no vertex available for the robber to move to. Thus σ is a winning strategy for j cops. A winning strategy for k cops on $(\mathcal{T}_k^j)^{op}$ can be similarly defined, replacing j with k in the above definition. Note that when $|X| = k$ there is no vertex available for the robber, so the cops never have to make a "leap-frog" move.

We now show that the robber can defeat $j - 1$ cops on $\mathcal{T}_k^j$ and $k - 1$ cops on $(\mathcal{T}_k^j)^{op}$. The strategy for the robber involves choosing some leaf. Whenever a cop moves to that leaf, a simple counting argument shows that there must be at least one unoccupied ancestor which the robber can reach with at least one clear path to a leaf below. The robber then plays to that ancestor and along that path to the leaf. More precisely, let $L = \{w \in V(\mathcal{T}_k^j) : |w| = k - 1\}$. For each $X, X' \in [V(\mathcal{G})]^{<j}$ and $w_r \in V(\mathcal{G})$, let $\rho(X, X', w_r) = w'$ for some $w' \in (L \cap \mathit{Reach}_{\mathcal{T}_k^j \setminus X}((X \cap X'))r) \setminus X'$. Clearly if ρ is well defined, it describes a winning strategy for the robber against $j - 1$ cops. We now show that there always exists some such w'. Since $|L| = 2^{k-1} > j - 1$, the robber can always choose an element of L initially, so we may assume that $w_r \in L$. If $w_r \notin X'$ then choosing $w' = w_r$ suffices, so suppose $w_r \in X'$. Since $w_r \notin X$ and $|X|, |X'| < j$, it follows that $|X \cap X'| < j - 1$. Thus there exists $w'' \prec w_r$ such that $|w_r| - |w''| < j$ and $\{w : w'' \preceq w \text{ and } w'_r \not\preceq w\} \cap X' = \emptyset$ where w'_r is the shortest word which is a prefix of w_r and for which w'' is a proper prefix. Thus for every $w \in L$ such that w'' is a prefix of w, there is a path from w_r to w in $\mathcal{T}_k^j \setminus (X \cap X')$. Thus choosing $w' \in L$ such that w'' is a prefix of w gives a well-defined strategy. A winning strategy for the robber against $k - 1$ cops on $(\mathcal{T}_k^j)^{op}$ is defined similarly, replacing j with k in the above definition. □

6.1.1 Monotonicity

For the remainder of this chapter, we are primarily concerned with monotone strategies. We recall from Definition 5.8 the definitions of fugitive-monotone (robber-monotone) and searcher-monotone (cop-monotone) searches and strategies. We observe that, as with the cops and visible robber game on undirected graphs, the cops and visible robber game for directed graphs permits idling and is vacating sensitive. Thus from Lemma 5.11, we have:

Lemma 6.8. *A cop-monotone winning strategy for k cops is robber-monotone.*

We saw in Theorem 5.37 that for the cops and visible robber game on undirected graphs, the converse to this holds: if k cops have a robber-monotone winning strategy then k cops

have a cop-monotone winning strategy. In [JRST01] it was shown that this is not the case for the strongly connected visible robber game. The next result shows that as with the game on undirected graphs, for the game we are considering, the two notions of monotonicity coincide.

Lemma 6.9. *If k cops have a cop-monotone or robber-monotone winning strategy, then they have a winning strategy that is both cop-monotone and robber-monotone.*

Proof. From Lemma 6.8, it suffices to show that if k cops have a robber-monotone winning strategy then k cops have a cop-monotone winning strategy. Suppose the cops have a robber-monotone winning strategy, and let $(X_0, r_0)(X_1, r_1)\cdots$ be a search consistent with that strategy. From this we construct a sequence which can be used to define a cop-monotone strategy in the obvious way. Suppose $X_i \not\subseteq X_{i+1}$ and let $v \in X_i \setminus X_{i+1}$. As $v \in X_i$, the robber is unable to reach v when the cops are on X_i. As the strategy is robber-monotone, the robber is unable to reach v at any further stage, in particular, he cannot reach v when the cops are on X_{i+1}. Thus, no cop needs to revisit v in order to prevent the robber from reaching v. Thus, we can remove v from all X_j, $j > i$. Proceeding in this way results in a sequence $(X_0, r_0)(X'_1, r_1)\cdots$. The strategy which takes (X'_i, r_i) to X'_{i+1} is cop-monotone for this search. Repeating this for all plays (that is, every choice for robber) results in a cop-monotone strategy. Hence, whenever the cops have a robber-monotone winning strategy they also has a cop-monotone strategy. □

With this lemma in mind we define a *monotone winning strategy* in the obvious way. Note that we have actually proved a slightly stronger assertion:

Corollary 6.10. *If k cops have a monotone winning strategy in the cops and visible robber game on a digraph $\mathcal{G}$, then k cops have a winning strategy σ such that $\sigma(X, r) \subseteq X \cup \mathit{Reach}_{\mathcal{G}\setminus X}(r)$ for all $X \subseteq V(\mathcal{G})$ and $r \in V(\mathcal{G}) \setminus X$.*

In Theorem 5.37, we also saw that in the visible robber game on undirected graphs, if k cops have a winning strategy then k cops have a monotone winning strategy. An interesting question is whether this extends to the game on directed graphs. Kreutzer and Ordyniak [KO07] have recently shown that this is not the case.

Theorem 6.11 ([KO07])**.** *For any $m \in \mathbb{N}$, there exists a digraph for which $5m$ cops can capture a visible robber but $6m$ cops are required to do so with a monotone strategy.*

Of course, this result does not preclude the possibility that, as with the strong visible robber game, the number of cops required for a monotone capture is bounded by some function of the number of cops required for a winning strategy which is not necessarily monotone. This gives us the following interesting open problem:

Open problem 6.12. *Does there exist a function $f : \omega \to \omega$ such that for all digraphs $\mathcal{G}$, if k cops can capture a visible robber on $\mathcal{G}$ then $f(k)$ cops can capture the robber with a monotone strategy?*

6.2 DAG-decompositions and DAG-width

In this section, we present a decomposition of directed graphs that is somewhat similar in style to tree decompositions of undirected graphs. This leads to the definition of DAG-width, which can be seen as a measure of how close a given graph is to being acyclic. We show then that a

graph has DAG-width k if, and only if, k cops have a monotone winning strategy in the cops and robber game played on that graph. We conclude with some algorithmic properties enjoyed by DAG-width.

Definition 6.13 (Guarding)**.** Let $\mathcal{G}$ be a directed graph. A set $W \subseteq V(\mathcal{G})$ *guards* a set $V \subseteq V(\mathcal{G})$ if $W \cap V = \emptyset$ and whenever there is an edge $(u, v) \in E(\mathcal{G})$ such that $u \in V$ and $v \notin V$, then $v \in W$.

Definition 6.14 (DAG-decomposition)**.** Let $\mathcal{G}$ be a digraph. A *DAG-decomposition* of $\mathcal{G}$ is a pair $(\mathcal{D}, \mathcal{X})$ where $\mathcal{D}$ is a directed, acyclic graph and $\mathcal{X} = (X_d)_{d \in V(\mathcal{D})}$ is a family of subsets of $V(\mathcal{G})$ such that

(D1) $\bigcup_{d \in V(\mathcal{D})} X_d = V(\mathcal{G})$.

(D2) For all vertices $d \preceq_{\mathcal{D}} d' \preceq_{\mathcal{D}} d''$, $X_d \cap X_{d''} \subseteq X_{d'}$.

(D3) For all edges $(d, d') \in E(\mathcal{D})$, $X_d \cap X_{d'}$ guards $X_{\geq d'} \setminus X_d$, where $X_{\geq d'} := \bigcup_{d' \preceq_{\mathcal{D}} d''} X_{d''}$. For any root d, $X_{\geq d}$ is guarded by $\emptyset$.

The width of a DAG-decomposition $(\mathcal{D}, \mathcal{X})$ is defined as $\max\{|X_d| : d \in V(\mathcal{D})\}$. The *DAG-width* of a graph is defined as the minimal width of any of its DAG-decompositions.

The main result of this section is an equivalence between monotone strategies for the cop player and DAG-decompositions.

Theorem 6.15. *For any directed graph $\mathcal{G}$, there is a DAG-decomposition of $\mathcal{G}$ of width k if, and only if, k cops have a monotone winning strategy in the cops and visible robber game on $\mathcal{G}$.*

To prove this, we first need some simple observations about guarding.

Lemma 6.16. *Let $\mathcal{G}$ be a directed graph, and $W, X, Y, Z \subseteq V(\mathcal{G})$.*

(i) *X guards $\mathit{Reach}_{\mathcal{G} \setminus X}(Y)$.*

(ii) *If W guards Y, X guards Z, then $(W \cup X) \setminus (Y \cup Z)$ guards $Y \cup Z$.*

(iii) *If X guards Y, $Z \supseteq X$ and $Z \cap Y = \emptyset$, then Z guards Y.*

(iv) *If X guards Y then $X \cup Z$ guards $Y \setminus Z$*

Proof. (i): Clearly $X \cap \mathit{Reach}_{\mathcal{G} \setminus X}(Y) = \emptyset$. Now suppose $(v, w) \in E(\mathcal{G})$, $v \in \mathit{Reach}_{\mathcal{G} \setminus X}(Y)$ and $w \notin \mathit{Reach}_{\mathcal{G} \setminus X}(Y)$. It follows from the definition of $\mathit{Reach}_{\mathcal{G} \setminus X}(Y)$ that $w \in X$. Therefore X guards $\mathit{Reach}_{\mathcal{G} \setminus X}(Y)$.

(ii): Suppose $(v, w) \in E(\mathcal{G})$, $v \in Y \cup Z$ and $w \notin Y \cup Z$. If $v \in Y$, then $w \in W$, as W guards Y. Similarly, if $v \in Z$ then $w \in X$ as X guards Z. Hence $w \in (W \cup X) \setminus (Y \cup Z)$, and $(W \cup X) \setminus (Y \cup Z)$ guards $Y \cup Z$.

(iii): Suppose $(v, w) \in E(\mathcal{G})$, $v \in Y$ and $w \notin Y$. As X guards Y, $w \in X$. As $Z \supseteq X$, $w \in Z$. Therefore, Z guards Y.

(iv): Since $X \cap Y = \emptyset$ and $Z \cap (Y \setminus Z) = \emptyset$, it follows that $(X \cup Z) \cap (Y \setminus Z) = \emptyset$. Now suppose $(v, w) \in E(\mathcal{G})$, $v \in Y \setminus Z$ and $w \notin Y \setminus Z$. Thus, $w \notin Y$ or $w \in Z$. For the first case, $w \in X$ as X guards Y. Hence $w \in X \cup Z$. □

We now turn to the proof of Theorem 6.15.

Proof of Theorem 6.15. Suppose k cops have a monotone winning strategy σ in the cops and visible robber game on a directed graph $\mathcal{G}$. As σ is monotone, from Corollary 6.10 it follows that we may assume that cops are only ever placed on vertices that are reachable by the robber. That is,

$$\sigma(X, r) \subseteq X \cup \mathit{Reach}_{\mathcal{G}\setminus X}(r). \tag{6.1}$$

We recall the definition of a strategy DAG, $\mathcal{D}_\sigma$, from Definition 5.7. Since the nodes of $\mathcal{D}_\sigma$ are positions in the cops and robber game, the function σ is well defined for all $d \in V(\mathcal{D}_\sigma)$. We claim that $(\mathcal{D}_\sigma, \mathcal{X})$, with $\mathcal{X}$ defined by $X_d = \sigma(d)$ for all $d \in V(\mathcal{D}_\sigma)$, is a DAG-decomposition of $\mathcal{G}$ of width $\leq k$. To support our claim, we first observe the following simple facts. For $d = (X, r) \in V(\mathcal{D}_\sigma)$,

$$\mathit{Reach}_{\mathcal{G}\setminus X}(r) \subseteq \bigcup_{d \preceq_{\mathcal{D}_\sigma} d'} \sigma(d') \subseteq X \cup \mathit{Reach}_{\mathcal{G}\setminus X}(r). \tag{6.2}$$

The first inclusion follows from the fact that σ is a winning strategy for the cop player: at position (X, r) every vertex reachable by the robber ($\mathit{Reach}_{\mathcal{G}\setminus X}(r)$) will be occupied by a cop at some point in the future. The second inclusion follows from repeated application of (6.1). Further, for $d = (X, r) \in V(\mathcal{D}_\sigma)$,

$$\mathit{Reach}_{\mathcal{G}\setminus X}(r) = \mathit{Reach}_{\mathcal{G}\setminus(X\cap\sigma(X,r))}(r). \tag{6.3}$$

As $X \cap \sigma(X, r) \subseteq X$, $\mathit{Reach}_{\mathcal{G}\setminus X}(r) \subseteq \mathit{Reach}_{\mathcal{G}\setminus(X\cap\sigma(X,r))}(r)$. The reverse inclusion follows from the fact that σ is a robber-monotone strategy.

Equations (6.2) and (6.3) together imply for $d = (X, r)$:

$$\Big(\bigcup_{d \preceq_{\mathcal{D}_\sigma} d'} \sigma(d')\Big) \setminus X = \mathit{Reach}_{\mathcal{G}\setminus(X\cap\sigma(X,r))}(r). \tag{6.4}$$

We now show that $(\mathcal{D}_\sigma, \mathcal{X})$ is indeed a DAG-decomposition of width $\leq k$. For (D1), if there was a $v \in V(\mathcal{G}) \setminus \bigcup_{d\in V(\mathcal{D}_\sigma)} X_d$, then the robber could defeat σ by playing to v at the beginning and staying there indefinitely. Hence $\bigcup_{d\in V(\mathcal{D})} X_d = V(\mathcal{G})$. (D2) follows immediately from the (cop-)monotonicity of the winning strategy σ. Towards establishing (D3), let us first consider a root $d = (X, r)$ of $\mathcal{D}_\sigma$. From the definition of $\mathcal{D}_\sigma$, this root is unique, thus $X_{\geq d} = V(\mathcal{G})$ and is therefore guarded by $\emptyset$. Now suppose $(d, d') \in E(\mathcal{D}_\sigma)$. If $d' = (X', r')$ then $X_d = \sigma(d) = X'$. So by (6.4),

$$X_{\geq d'} \setminus X_d = \Big(\bigcup_{d' \preceq_{\mathcal{D}_\sigma} d''} \sigma(d'')\Big) \setminus X' = \mathit{Reach}_{\mathcal{G}\setminus(X'\cap\sigma(X',r'))}(r').$$

Therefore, from Lemma 6.16(i), $X_d \cap X_{d'} = X' \cap \sigma(X', r')$ guards $X_{\geq d'} \setminus X_d$. It follows that $(\mathcal{D}_\sigma, \mathcal{X})$ is a DAG-decomposition. To see that it has width $\leq k$, note that $\max\{|X_d| : d \in V(\mathcal{D}_\sigma)\} = \max\{|\sigma(d)| : d \in V(\mathcal{D}_\sigma)\} \leq k$.

Conversely, let $(\mathcal{D}, \mathcal{X})$ be a DAG-decomposition of width k. A strategy for k cops can then be defined as:

(1) Let the robber choose a vertex $v \in V(\mathcal{G})$. From (D1), there exists $d_v \in V(\mathcal{D})$ such that $v \in X_{d_v}$. Let d be a root of D which lies above d_v.

(2) Place cops on X_d.

(3) From (D3) and Lemma 6.16(iii), X_d guards $X_{\geq d} \setminus X_d$. Therefore, the robber can only move to vertices in $X_{\geq d} \setminus X_d$. Suppose the robber moves to $v' \in X_{d''}$. Let d' be a successor of d which lies above d''.

(4) Remove cops on $X_d \setminus X_{d'}$ (leaving cops on $X_d \cap X_{d'}$)

(5) As $X_d \cap X_{d'}$ guards $X_{\geq d'} \setminus X_d$, the robber can only move to vertices in $X_{\geq d'}$ – that is, the robber must remain in the sub-DAG rooted at d'.

(6) Return to step 2 with d' as d.

As $\mathcal{D}$ is a DAG, at some point the robber will not be able to move because $X_{\geq d} \setminus X_d$ is empty when d is a leaf. Hence, this is a winning strategy for k cops. To show that it is monotone, observe that (D2) ensures that at no point does a cop return to a vacated vertex. This concludes the proof of Theorem 6.15. □

We observe that as a strategy DAG is the underlying DAG in the decomposition $(\mathcal{D}, \mathcal{X})$ constructed in this proof, and a strategy DAG has a unique root, we have the following:

Corollary 6.17. *If a digraph $\mathcal{G}$ has a DAG-decomposition of width k, then $\mathcal{G}$ has a DAG-decomposition $(\mathcal{D}, \mathcal{X})$ of width $\leq k$ such that $\mathcal{D}$ has a unique root.*

In the sequel we show that we can make further simplifying assumptions about the structure of DAG-decompositions.

The remainder of this section looks at some properties of DAG-decompositions motivated by similar results for tree-width and tree decompositions. We first observe that the winning strategies for the cop player in Lemma 6.5 and Proposition 6.7 are monotone. These results therefore imply that a graph has DAG-width 1 if, and only if, it is acyclic (indeed, the graph itself will suffice as a decomposition) and that the DAG-width of a graph may change by an arbitrary amount if its edges are reversed. This last observation is particularly useful when searching for alternative characterizations of DAG-width, such as those we introduce in Chapter 8.

We further observe that, as with the game on undirected graphs, the cops and visible robber game enjoys the properties of graph searching games introduced in Section 5.4. In particular this means that DAG-width decreases when taking subgraphs, and suitably increases when taking lexicographic products.

Lemma 6.18. *Let $(\mathcal{D}, \mathcal{X})$ be a DAG-decomposition of a digraph $\mathcal{G}$, and let $\mathcal{G}'$ be a subgraph of $\mathcal{G}$. $(\mathcal{D}, \mathcal{X}|_{\mathcal{G}'})$ where $\mathcal{X}|_{\mathcal{G}'} := \big(X_d \cap V(\mathcal{G}')\big)_{d \in V(\mathcal{D})}$ is a DAG-decomposition of $\mathcal{G}'$.*

Proof. Clearly, (D1) and (D2) still hold for $(\mathcal{D}, \mathcal{X}|_{\mathcal{G}'})$. For (D3), we observe that, if X guards Y in $\mathcal{G}$, then $X \cap V(\mathcal{G}')$ guards $Y \cap V(\mathcal{G}')$ in $\mathcal{G}'$. This is because, if $v \in Y \cap V(\mathcal{G}')$, $w \in V(\mathcal{G}') \setminus Y$ and $(v, w) \in E(\mathcal{G}') \subseteq E(\mathcal{G})$, then $w \in X$ (as X guards Y), hence $w \in X \cap V(\mathcal{G}')$. Then, (D3) follows immediately from (D3) for the original decomposition $(\mathcal{D}, \mathcal{X})$. □

Corollary 6.19. *Let $\mathcal{G}$ and $\mathcal{G}'$ be directed graphs such that $\mathcal{G}'$ is a subgraph of $\mathcal{G}$. Then DAG-width$(\mathcal{G}') \leq$ DAG-width$(\mathcal{G})$.*

Lemma 6.20. *Let $\mathcal{G}$ be a directed graph and $\mathcal{K}_n$ the complete graph on n vertices. DAG-width$(\mathcal{G} \bullet \mathcal{K}_n) = n \cdot$ DAG-width$(\mathcal{G})$.*

Proof. From Theorem 5.64, it suffices to show that the cops and visible robber game is composition-expanding. We show that it satisfies conditions (I)–(IV) of Lemma 5.65. Clearly as the cops are free to make any move, conditions (I) and (II) are satisfied. For condition (III), suppose on $\mathcal{G}$ as the cops move from X to X', the robber can move from r to r'. It follows by the definitions of *Reach* and lexicographic product that if the cops move from $X \times V(\mathcal{K}_n)$ to $X' \times V(\mathcal{K}_n)$ in $\mathcal{G} \bullet \mathcal{K}_n$, the robber can move from (r, v) to (r', w') for all $v, w \in V(\mathcal{K}_n)$. Thus there is an edge in the arena (of the game on $\mathcal{G} \bullet \mathcal{K}_n$) from $(X \times V(\mathcal{K}_n), X' \times V(\mathcal{K}_n), R \times V(\mathcal{K}_n))$ to $(X' \times V(\mathcal{K}_n), R' \times V(\mathcal{K}_n))$ where $R =$ *Reach*$_{\mathcal{G}\setminus X}(r)$ and $R' =$ *Reach*$_{\mathcal{G}\setminus X'}(r')$. Finally, to show condition (IV), we observe that for $X \subseteq V(\mathcal{G} \bullet \mathcal{K}_n)$ and $(r, v) \in V(\mathcal{G} \bullet \mathcal{K}_n)$, *Reach*$_{(\mathcal{G}\bullet\mathcal{K}_n)\setminus X}(r, v)$ consists of those vertices $(r', v') \notin X$ such that r' *in*$\textit{Reach}_{\mathcal{G}\setminus Y}(r)$ where $Y = \{x \in V(\mathcal{G}) : (x, w) \in X \text{ for all } v \in V(\mathcal{K}_n)\}$. Thus, if there is an edge in the arena (for the game on $\mathcal{G}$) from (Y, Y', S) to (Y', S'), then there is an edge in the arena (for the game on $\mathcal{G} \bullet \mathcal{K}_n$) from (X, X', R) to (X', R') where X, X', Y, Y', R, R', S and S' are as defined in condition (IV) of Lemma 5.65. □

We also show that the DAG-width of graphs is closed under directed unions, which, as we discussed in Chapter 4, is an important property of a reasonable decomposition of directed graphs.

Lemma 6.21. *Let $\mathcal{G}$ be a directed union of the digraphs $\mathcal{G}_1$ and $\mathcal{G}_2$. Then*

$$\textit{DAG-width}(\mathcal{G}) = \max\{\textit{DAG-width}(\mathcal{G}_1), \textit{DAG-width}(\mathcal{G}_2)\}.$$

Proof. For DAG-decompositions $(\mathcal{D}^1, \mathcal{X}^1)$ and $(\mathcal{D}^2, \mathcal{X}^2)$ of $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively, the DAG $\mathcal{D}$ obtained by adding an edge from every leaf of $\mathcal{D}^1$ to every root of $\mathcal{D}^2$. together with $\mathcal{X} := (X^1_d)_{d\in V(\mathcal{D}^1)} \dot\cup (X^2_d)_{d\in V(\mathcal{D}^2)}$ forms a DAG-decomposition of $\mathcal{G}$. Conversely, any DAG-decomposition $(\mathcal{D}, \mathcal{X})$ of $\mathcal{G}$ can be restricted to $\mathcal{G}_1$ and $\mathcal{G}_2$ yielding DAG-decompositions for these graphs, according to Lemma 6.18. □

We observe that it follows that the DAG-width of a directed graph is the maximum DAG-width of all its strongly connected components.

For algorithmic purposes, it is often useful to have a normal form for decompositions. The following is similar to one for tree decompositions as presented in [Bod97].

Definition 6.22. [Nice DAG-decompositions] A DAG-decomposition $(\mathcal{D}, \mathcal{X})$ is *nice* if

(N1) $\mathcal{D}$ has a unique root.

(N2) Every $d \in V(\mathcal{D})$ has at most two successors.

(N3) If d_1, d_2 are two successors of d_0, then $X_{d_0} = X_{d_1} = X_{d_2}$.

(N4) If d_1 is the unique successor of d_0, then $|X_{d_0} \triangle X_{d_1}| \leq 1$.

The final result we establish in this section is that every graph with DAG-width k has a nice decomposition with width k. For this, we transform a DAG-decomposition into one which is nice that has the same width. To do this we formalize the transformations we use, and show that executing them (possibly subject to some constraints) does not violate any of the properties of a DAG-decomposition. First we require the following useful observation.

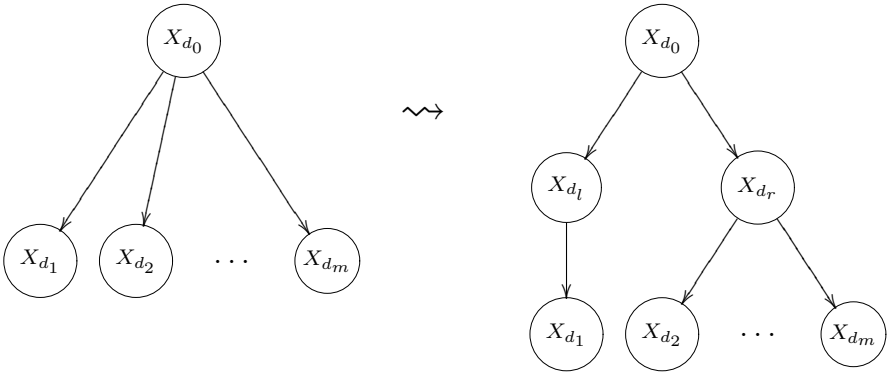


Figure 6.1: Splitting at d_0

Lemma 6.23. *Let $(\mathcal{D}, \mathcal{X})$ be a DAG-decomposition. For all $(d, d') \in E(\mathcal{D})$,*

$$X_{\geq d'} \setminus X_d = X_{\geq d'} \setminus (X_d \cap X_{d'}).$$

Proof. As $X_d \cap X_{d'} \subseteq X_d$, $X_{\geq d'} \setminus X_d \subseteq X_{\geq d'} \setminus (X_d \cap X_{d'})$. Conversely, suppose $v \in X_{\geq d'}$, that is, $v \in X_{d''}$ for some $d'' \succeq_{\mathcal{D}} d'$. We will show that $v \in X_d \cap X_{d'}$, or $v \notin X_d$. Suppose $v \in X_d$. Then as $d \preceq_{\mathcal{D}} d' \preceq_{\mathcal{D}} d''$, $v \in X_d \cap X_{d''} \subseteq X_{d'}$. Hence $v \in X_d \cap X_{d'}$. Thus, $X_{\geq d'} \setminus X_d \supseteq X_{\geq d'} \setminus (X_d \cap X_{d'})$. □

Definition 6.24 (Splitting)**.** Let $(\mathcal{D}, \mathcal{X})$ be a DAG-decomposition, and suppose $d_0 \in V(\mathcal{D})$ has $m > 1$ successors $d_1, d_2, \ldots, d_m$. The decomposition $(\mathcal{D}', \mathcal{X}')$ obtained from $(\mathcal{D}, \mathcal{X})$ by *splitting* d_0 is defined as follows:

(i) $V(\mathcal{D}') = V(\mathcal{D}) \dot{\cup} \{d_l, d_r\}$,

(ii) $$\begin{aligned} E(\mathcal{D}') \;=\; & \big(E(\mathcal{D}) \setminus \{(d_0, d_i) : 1 \leq i \leq m\}\big) \\ & \cup \{(d_0, d_l), (d_0, d_r), (d_l, d_1)\} \\ & \cup \{(d_r, d_i) : 2 \leq i \leq m\}, \text{ and} \end{aligned}$$

(iii) $X'_d = X_d$, for all $d \in V(\mathcal{D})$, and $X'_{d_l} = X'_{d_r} = X_{d_0}$.

Figure 6.1 gives a visual representation of this transformation.

Lemma 6.25. *Let $(\mathcal{D}, \mathcal{X})$ be a DAG-decomposition of a digraph $\mathcal{G}$ of width k, and suppose $d_0 \in V(\mathcal{D})$ has $m > 1$ successors $d_1, d_2, \ldots, d_m$. Then $(\mathcal{D}', \mathcal{X}')$ obtained from $(\mathcal{D}, \mathcal{X})$ by splitting d_0 is a DAG-decomposition of $\mathcal{G}$ of width k.*

Proof. First we observe that, as d_0 is the unique predecessor of d_l and d_r, for any $d \in V(\mathcal{D})$ such that $d \prec_{\mathcal{D}'} d_l$ or $d \prec_{\mathcal{D}'} d_r$, it must be the case that $d \preceq_{\mathcal{D}} d_0$. Thus, for all $d \in V(\mathcal{D})$,

$$X'_{\geq d} = \bigcup_{d \preceq_{\mathcal{D}'} d'} X'_{d'} = \bigcup_{d \preceq_{\mathcal{D}} d'} X_{d'} = X_{\geq d},$$

since if X_{d_l} or X_{d_r} is included in the union on the left, then so is X_{d_0}, and so neither X_{d_l} nor X_{d_r} contribute to the overall union.

Also, for all i such that $1 \leq i \leq m$, it is the case that $X_{d_0} \cap X_{d_i}$ guards $X_{\geq d_i} \setminus X_{d_0}$. Therefore, by Lemma 6.16(iii),

$$X_{d_0} \text{ guards } X_{\geq d_i} \setminus X_{d_0}. \tag{6.5}$$

It is easily seen that the edges added do not create any cycles, so $\mathcal{D}'$ is a DAG. Further, $\bigcup_{d\in V(\mathcal{D}')} X'_d = \bigcup_{d\in V(\mathcal{D})} X_d = V(\mathcal{G})$. To prove the connectivity condition (D2), let $d, d', d'' \in V(\mathcal{D}')$, be such that $d \preceq_{\mathcal{D}'} d' \preceq_{\mathcal{D}'} d''$. If $d' = d$ or d'' then trivially $X'_d \cap X'_{d''} \subseteq X'_{d'}$, so suppose $d \prec_{\mathcal{D}'} d' \prec_{\mathcal{D}'} d''$. We consider four cases:

- If none of d, d', d'' is d_l or d_r, then $d, d', d'' \in D$, and (D2) follows from the fact that $(\mathcal{D}, \mathcal{X})$ is a DAG-decomposition.
- If d is d_l or d_r then since all descendants of d are in $V(\mathcal{D})$, and $d_0 \in V(\mathcal{D})$ is the unique predecessor of d, we obtain the following chain of nodes in $\mathcal{D}$: $d_0 \prec_{\mathcal{D}} d' \prec_{\mathcal{D}} d''$. So $X'_d \cap X'_{d''} = X_{d_0} \cap X_{d''} \subseteq X_{d'} = X'_{d'}$.
- If d'' is d_l or d_r then from the comments at the beginning of the proof, $d \prec_{\mathcal{D}} d' \preceq_{\mathcal{D}} d_0$. Thus, $X'_d \cap X'_{d''} = X_d \cap X_{d_0} \subseteq X_{d'} = X'_{d'}$.
- Finally, if d' is d_l or d_r then by the same reasoning as the previous two cases, $d \preceq_{\mathcal{D}} d_0 \prec_{\mathcal{D}} d''$. So $X'_d \cap X'_{d''} = X_d \cap X_{d''} \subseteq X_{d_0} = X'_{d'}$.

Thus, in all cases, $X'_d \cap X'_{d''} \subseteq X'_{d'}$, showing that (D2) holds. To see that condition (D3) also holds, observe first that every root of $\mathcal{D}'$ is a root of $\mathcal{D}$ too. So $\emptyset$ guards $X_{\geq d} = X'_{\geq d}$. Now let $(d, d') \in E(\mathcal{D}')$. We consider three cases:

- $d' \in V(\mathcal{D})$ (i.e., $d' \neq d_l, d_r$). If $d = d_l$ or d_r, then $X'_d = X_{d_0}$. Otherwise $(d, d') \in E(\mathcal{D})$. In both cases, $X'_d \cap X'_{d'}$ guards $X'_{\geq d'} \setminus X'_d$.
- $d' = d_l$ (so $d = d_0$). Here $X'_{\geq d'} = X_{d_0} \cup X_{\geq d_1}$, so $X'_{\geq d'} \setminus X'_d = X_{\geq d_1} \setminus X_{d_0}$. Hence, by (6.5), $X_{d_0} = X'_d \cap X'_{d'}$ guards $X_{\geq d_1} \setminus X_{d_0} = X'_{\geq d'} \setminus X'_d$.
- $d' = d_r$ (so $d = d_0$). Here $X'_{\geq d'} = X_{d_0} \cup \bigcup_{2\leq i\leq m} X_{\geq d_i}$, and so $X'_{\geq d'} \setminus X'_d = (\bigcup X_{\geq d_i}) \setminus X_{d_0} = \bigcup (X_{\geq d_i} \setminus X_{d_0})$, where the unions are taken over i for $2 \leq i \leq m$. From Lemma 6.16(ii) and (6.5), $X'_d \cap X'_{d'} = X_{d_0}$ guards $\bigcup_{2\leq i\leq m}(X_{\geq d_i} \setminus X_{d_0}) = X'_{\geq d'} \setminus X'_d$.

As $X'_{d_l} = X'_{d_r} = X_{d_0}$, we have

$$\max\{|X'_d| : d \in V(\mathcal{D}')\} = \max\{|X_d| : d \in V(\mathcal{D})\} = k.$$

Consequently, the decomposition $(\mathcal{D}', \mathcal{X}')$ has width k. □

By the *decomposition resulting from splitting* d $m-1$ *times* we mean the decomposition resulting from splitting d, and then recursively splitting d_r until d_r has only one successor. A *complete split* of $(\mathcal{D}, \mathcal{X})$ is the decomposition $(\mathcal{D}', \mathcal{X}')$ obtained by recursively splitting every node with more than two successors.

Definition 6.26 (Adding)**.** Let $(\mathcal{D}, \mathcal{X})$ be a DAG-decomposition of a digraph $\mathcal{G}$. If $(d_0, d_1) \in E(\mathcal{D})$ and $X \subseteq V(\mathcal{G})$ the *decomposition resulting from adding* X *to* (d_0, d_1) is the pair $(\mathcal{D}', \mathcal{X}')$ with

(i) $V(\mathcal{D}') = V(\mathcal{D}) \dot\cup \{d_X\}$

(ii) $E(\mathcal{D}') = (E(\mathcal{D}) \setminus \{(d_0, d_1)\}) \cup \{(d_0, d_X), (d_X, d_1)\}$

(iii) $X'_{d_X} = X$, and for all $d \in V(\mathcal{D})$, $X'_d = X_d$.

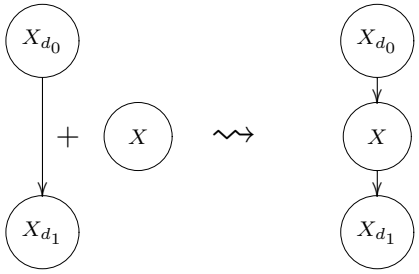


Figure 6.2: Adding X to (d_0, d_1)

See Figure 6.2 for a visual interpretation.

Lemma 6.27. *Let $(\mathcal{D}, \mathcal{X})$ be a DAG-decomposition of a digraph $\mathcal{G}$ of width k and let $(\mathcal{D}', \mathcal{X}')$ be the decomposition resulting from adding $X \subseteq V(\mathcal{G})$ to (d_0, d_1). If either*

(i) $X_{d_0} \cap X_{d_1} \subseteq X \subseteq X_{d_0}$, *or*

(ii) $X_{d_0} \cap X_{d_1} \subseteq X \subseteq X_{d_1}$,

then $(\mathcal{D}', \mathcal{X}')$ is a DAG-decomposition of $\mathcal{G}$ of width k.

Proof. We observe that for all $d \in V(\mathcal{D})$, if $d \prec_{\mathcal{D}'} d_X$, then, as $d_0 \in V(\mathcal{D})$ is the unique predecessor of d_X, we have $d \preceq_{\mathcal{D}} d_0$, and if $d_X \prec_{\mathcal{D}'} d$, then as $d_1 \in V(\mathcal{D})$ is the unique successor of d_X, we have $d_1 \preceq_{\mathcal{D}} d$. This implies, for all $d \in V(\mathcal{D})$

$$X'_{\geq d} = \bigcup_{d \preceq_{\mathcal{D}'} d'} X'_{d'} = \bigcup_{d \preceq_{\mathcal{D}} d'} X_{d'} = X_{\geq d},$$

since if X'_{d_X} is included in the union on the left, then both X'_{d_0} and X'_{d_1} are, and so in either case of the lemma $X'_{d_X} = X$ does not contribute to the overall union.

Further, $X_{d_0} \cap X_{d_1}$ guards $X_{\geq d_1} \setminus X_{d_0} = X_{\geq d_1} \setminus (X_{d_0} \cap X_{d_1})$ from Lemma 6.23.

Clearly, $\mathcal{D}'$ is a DAG. We now show that $(\mathcal{D}', \mathcal{X}')$ satisfies the properties (D1) to (D3). It is easily seen that $\bigcup_{d \in V(\mathcal{D}')} X'_d = X \cup \bigcup_{d \in V(\mathcal{D})} X_d = V(\mathcal{G})$. This shows (D1). Towards establishing condition (D2), suppose $d \preceq_{\mathcal{D}'} d' \preceq_{\mathcal{D}'} d''$. If $d' = d$ or $d' = d''$ then trivially $X'_d \cap X'_{d''} \subseteq X'_{d'}$, so suppose $d \prec_{\mathcal{D}'} d' \prec_{\mathcal{D}'} d''$. We consider four cases:

- If none of d, d', d'' is d_X then d, d', and d'' are all in $V(\mathcal{D})$, so (D2) follows from the fact that $(\mathcal{D}, \mathcal{X})$ is a DAG-decomposition.

- Suppose $d = d_X$. From the observations made at the beginning of the proof, we get the following chain of nodes in D: $d_0 \prec_{\mathcal{D}} d_1 \preceq_{\mathcal{D}} d' \prec_{\mathcal{D}} d''$. So in case (i) of the lemma, we have $X \subseteq X_{d_0}$. So $X'_d \cap X'_{d''} = X \cap X_{d''} \subseteq X_{d_0} \cap X_{d''} \subseteq X_{d'} = X'_{d'}$, by condition (D2) of $(\mathcal{D}, \mathcal{X})$. Otherwise, if $X \subseteq X_{d_1}$, then $X'_d \cap X'_{d''} = X \cap X_{d''} \subseteq X_{d_1} \cap X_{d''} \subseteq X_{d'} = X'_{d'}$.

- The other cases are similar. If $d'' = d_X$ then we obtain $d \prec_{\mathcal{D}} d' \preceq_{\mathcal{D}} d_0 \prec_{\mathcal{D}} d_1$. So if $X \subseteq X_{d_0}$, then $X'_d \cap X'_{d''} = X_d \cap X \subseteq X_d \cap X_{d_0} \subseteq X_{d'} = X'_{d'}$. If $X \subseteq X_{d_1}$, then $X'_d \cap X'_{d''} = X_d \cap X \subseteq X_d \cap X_{d_1} \subseteq X_{d'} = X'_{d'}$.

- Finally, assume $d' = d_X$. Then $d \preceq_{\mathcal{D}} d_0 \prec_{\mathcal{D}} d_1 \preceq_{\mathcal{D}} d''$. Hence $X_d \cap X_{d''} \subseteq X_{d_0}$ and $X_d \cap X_{d''} \subseteq X_{d_1}$. Thus, $X'_d \cap X'_{d''} = X_d \cap X_{d''} \subseteq X_{d_0} \cap X_{d_1} \subseteq X = X'_{d'}$.

Finally, towards (D3), if d is a root of $\mathcal{D}'$, then d is a root of $\mathcal{D}$. Hence $\emptyset$ guards $X_{\geq d} = X_{\geq d'}$. Now let $(d, d') \in E(\mathcal{D}')$. We consider three cases:

- $d_X \notin \{d, d'\}$, i.e., $(d, d') \in E(\mathcal{D})$. In this case, (D3) follows from the fact that $(\mathcal{D}, \mathcal{X})$ is a DAG-decomposition.

- Now suppose $d = d_X$ (so $d' = d_1$). If $X_{d_0} \cap X_{d_1} \subseteq X \subseteq X_{d_0}$, so we are in case (i) of the lemma, then

$$X_{\geq d_1} \setminus (X_{d_0} \cap X_{d_1}) \quad \supseteq \quad X_{\geq d_1} \setminus X \quad \supseteq \quad X_{\geq d_1} \setminus X_{d_0}.$$

  Further, by Lemma 6.23, $X_{\geq d_1} \setminus (X_{d_0} \cap X_{d_1}) = X_{\geq d_1} \setminus X_{d_0}$. Therefore $X_{\geq d_1} \setminus X = X_{\geq d_1} \setminus X_{d_0}$. As $(\mathcal{D}, \mathcal{X})$ is a DAG-decomposition, $X_{d_0} \cap X_{d_1}$ guards $X_{\geq d_1} \setminus X_{d_0}$, and as $X_{d_0} \cap X_{d_1} \subseteq X \cap X_{d_1}$, Lemma 6.16(iii) implies that $X'_d \cap X'_{d_1} = X \cap X_{d_1}$ guards $X_{\geq d_1} \setminus X_{d_0} = X'_{\geq d_1} \setminus X'_d$.

  Otherwise we are in case (ii) and we have $X_{d_0} \cap X_{d_1} \subseteq X \subseteq X_{d_1}$. Let $Z = X \setminus (X_{d_0} \cap X_{d_1})$. We know $(X_{d_0} \cap X_{d_1})$ guards $X_{\geq d_1} \setminus (X_{d_0} \cap X_{d_1})$, due to Lemma 6.23. Hence, by Lemma 6.16(iv), $X'_d \cap X'_{d_1} = X = (X_{d_0} \cap X_{d_1}) \cup Z$ guards

$$\begin{aligned}(X_{\geq d_1} \setminus (X_{d_0} \cap X_{d_1})) \setminus Z \;&=\; X_{\geq d_1} \setminus ((X_{d_0} \cap X_{d_1}) \cup Z) \\ &=\; X_{\geq d_1} \setminus X \;=\; X'_{\geq d_1} \setminus X'_{d'}.\end{aligned}$$

- Finally, suppose $d' = d_X$ (so $d = d_0$). Here we claim $X'_{\geq d_X} \setminus X'_{d_0} = X_{\geq d_1} \setminus X_{d_0}$. If $X \subseteq X_{d_0}$, then $X'_{\geq d_X} \setminus X'_{d_0} = (X \cup X_{\geq d_1}) \setminus X_{d_0} = (X \setminus X_{d_0}) \cup (X_{\geq d_1} \setminus X_{d_0}) = X_{\geq d_1} \setminus X_{d_0}$. If $X \subseteq X_{d_1}$, then since $d_X \preceq_{\mathcal{D}'} d_1$, $X'_{\geq d_X} = X'_{\geq d_1} = X_{\geq d_1}$. Now $X \supseteq X_{d_0} \cap X_{d_1}$, so by Lemma 6.16(iii), $X'_{d'} = X$ guards $X_{\geq d_1} \setminus X_{d_0} = X'_{\geq d_X} \setminus X'_{d_0}$.

Note that since $X \subseteq X_{d_0}$ or X_{d_1}, $\max\{|X'_d| : d \in V(\mathcal{D}')\} = \max\{|X_d| : d \in V(\mathcal{D})\} = k$. So $(\mathcal{D}', (X'_d)_{d \in V(\mathcal{D}')})$ has width k. □

If $X_1, X_2, \ldots, X_n$ is a sequence of subsets of $V(\mathcal{G})$, the *decomposition resulting from adding* $X_1, X_2, \ldots, X_n$ *to* (d_0, d_1) is the decomposition resulting from adding X_1 to (d_0, d_1) and then recursively adding X_{i+1} to (d_{X_i}, d_1).

We can now describe how to transform a DAG-decomposition into one which is nice and has the same width.

Theorem 6.28. *If $\mathcal{G}$ has a DAG-decomposition of width k, then $\mathcal{G}$ has a nice DAG-decomposition of width k.*

Proof. Let $(\mathcal{D}, \mathcal{X})$ be a DAG-decomposition of width k. From Corollary 6.17, we may assume that $\mathcal{D}$ has a unique root. We carry out each of the following steps.

1. We apply a complete split on $(\mathcal{D}, \mathcal{X})$ to obtain a DAG-decomposition such that every node has at most two successors, and if d has two successors d_1 and d_2, then $X_d = X_{d_1} = X_{d_2}$. This establishes (N2) and (N3).

2. To satisfy (N4), we require two stages. First, for each $(d_0, d_1) \in E(\mathcal{D})$ with $X_{d_0} \neq X_{d_1}$, we add $X_{d_0} \cap X_{d_1}$ to (d_0, d_1) to obtain a DAG-decomposition such that for every $(d, d') \in E(\mathcal{D}')$, X_d is either a subset or a superset of $X_{d'}$.

3. Secondly, for each $(d, d') \in E(\mathcal{D})$ with $|X_d| - |X_{d'}| = m > 1$ (or $|X_{d'}| - |X_d| = m > 1$), let $X_0 = X_d, X_1, \ldots, X_m = X_{d'}$ be a strictly decreasing (increasing) sequence of subsets. Such a sequence exists because at the previous step we finished with a DAG-decomposition such that $X_d \subseteq X_{d'}$ or $X_d \supseteq X_{d'}$. Add $X_1, X_2, \ldots, X_{m-1}$ to (d, d'). At this point we have a decomposition which satisfies (N1) to (N4), and is therefore nice.

Finally, from Lemmas 6.25 and 6.27, at each step we have a DAG-decomposition of width k. □

6.3 Algorithmic aspects of DAG-width

We now consider algorithmic applications of DAG-width as well as the complexity of deciding the DAG-width of a graph and computing a DAG-decomposition.

6.3.1 Computing DAG-width and decompositions

Because deciding if the tree-width of a graph is at most a given integer is NP-complete, it is no surprise that deciding if the DAG-width of a graph is at most a given integer is intractable. Indeed, the following is a direct consequence of the NP-completeness of the TREE-WIDTH decision problem and Proposition 6.36.

Theorem 6.29. *Given a digraph $\mathcal{G}$ and a natural number k, deciding if the DAG-width of $\mathcal{G}$ is at most k is* NP*-hard.*

Despite the similarity to tree-width, it is currently unknown whether deciding if the DAG-width of a graph is bounded by a given value is in NP. However, we strongly believe that this is the case, giving us the following:

Conjecture 6.30. *Given a digraph $\mathcal{G}$ and a natural number k, deciding if the DAG-width of $\mathcal{G}$ is at most k is* NP*-complete.*

However, for any fixed k, it is possible, in polynomial time, to decide if a graph has DAG-width at most k and to compute a DAG-decomposition of this width if it has. This follows in a similar manner to Proposition 5.71, so for the proof of the next result we refer the reader to Section 5.5.

Theorem 6.31. *Let $\mathcal{G}$ be a directed graph and let $k < \omega$. Deciding if k cops have a monotone winning strategy in the cops and visible robber game on $\mathcal{G}$, and computing such a strategy if it exists can be executed in time $O(|V(\mathcal{G})|^{2k+4})$.*

Note also that the translation of strategies into decompositions is computationally easy, that is, it can be done in polynomial time. Since winning strategies can be computed in polynomial time in the size of the graph, we get the following.

Proposition 6.32. *Given a graph $\mathcal{G}$ of DAG-width k, a DAG-decomposition of $\mathcal{G}$ of width k can be computed in time $O(|\mathcal{G}|^{O(k)})$.*

6.3.2 Algorithms on graphs of bounded DAG-width

We can use DAG-decompositions, particularly nice DAG-decompositions, to define dynamic programming algorithms similar to those used with tree decompositions. Working bottom-up from the leaves of the underlying DAG $\mathcal{D}$, for each node $d \in V(\mathcal{D})$ we compute a data set containing information for the subgraph induced by $X_{\geq d} := \bigcup_{d' \geq_{\mathcal{D}} d} X_d$. The general pattern is described in Algorithm 6.1. We observe that if the starting decomposition is nice, then the *combine* and *expand* steps become significantly simplified. Indeed, the *combine* step can be seen as applying to inner nodes with two successors and the *update* steps apply to inner nodes with only one successor.

Algorithm 6.1 Dynamic programming using a DAG-decomposition

Given a DAG-decomposition $(\mathcal{D}, \mathcal{X})$:

Leaves: Compute the data set for X_d for all leaves d.

Combine: If $d \in V(\mathcal{D})$ is an inner node with successors $d_1, \ldots, d_m$, combine the data sets computed for $X_{\geq d_1}, \ldots, X_{\geq d_m}$ to a data set for the union $\bigcup_{i=1}^{m} X_{\geq d_i}$.

Expand: Finally, expand the data set to include X_d.

As the directed tree-width of a graph is bounded above by a constant factor of its DAG-width (see Proposition 6.37), any graph property that can be decided in polynomial time on classes of graphs of bounded directed tree-width can be decided on classes of graphs of bounded DAG-width also. This implies that properties such as Hamiltonicity that are known to be polynomial time on graphs of bounded directed tree-width can be solved efficiently on graphs of bounded DAG-width too. We give a nontrivial application of DAG-width in Section 6.3.3 where we show that parity games can be solved efficiently on arena of bounded DAG-width, something which is not known for directed tree-width.

We observe that the arena used in the proof of Theorem 2.64 has DAG-width 2: place one cop on vertex q_φ and the remaining graph is acyclic and can be searched monotonely with one cop. This implies that, unlike parity games, win-set games (and, consequently, Muller games, Zielonka DAG games, Emerson-Lei games and circuit games) remain hard on arenas of bounded DAG-width.

Proposition 6.33. *Deciding win-set games on arenas of DAG-width* 2 *is* PSPACE*-hard.*

As for the relation to undirected tree-width, it is clear that not all graph properties that can be decided in polynomial time on graphs of bounded tree-width can also be decided efficiently on graphs of bounded DAG-width. For instance, the 3-colourability problem is known to be decidable in polynomial time on graphs of bounded tree-width. However, the problem does not depend on the direction of edges. For any given (undirected) graph, we can simply direct the edges in such a way that it becomes acyclic. Thus, arbitrary instances are polynomial-time reducible to instances of DAG-width 1. As 3-colourability over arbitrary graphs is NP-hard, it follows that the problem cannot be solved in polynomial time on graphs of bounded DAG-width, unless PTIME = NP.

6.3.3 Parity Games on Graphs of Bounded DAG-Width

Using the algorithm scheme of Algorithm 6.1, we now outline a dynamic programming algorithm for solving parity games. The advantage of such an algorithm is that on any class of arenas of bounded DAG-width it runs in polynomial time, giving us a large class of graphs for which there exists a tractable algorithm for solving parity games. Full details of the algorithm can be found in [BDHK06].

Given an arena $\mathcal{A}$, a DAG-decomposition of $\mathcal{A}$ is a DAG-decomposition of the underlying directed graph $(V(\mathcal{A}), E(\mathcal{A}))$.

Theorem 6.34. *For any k, given a parity game $(\mathcal{A}, \chi)$ where the DAG-width of $\mathcal{A}$ is at most k, determining if Player 0 has a winning strategy can be decided in polynomial time.*

Let us fix a parity game $(\mathcal{A}, \chi)$ where $\chi : V(\mathcal{A}) \to \mathsf{P}$, and let $n = |V(\mathcal{A})|$. We assume that every vertex in $\mathcal{A}$ has out-degree at most 2. It is easy to see that the arena resulting from the transformation described in Theorem 2.59, replacing vertices that have out-degree more than 2 with binary branching trees, requires at most one more cop to capture a visible robber. Thus such a transformation results in an arena with DAG-width at most $k+1$. Let $(\mathcal{D}, \mathcal{X})$ be a DAG-decomposition of $\mathcal{A}$ of width k which we assume is nice. For technical reasons, we also assume that for the root d of $\mathcal{D}$, $X_d = \emptyset$. From Proposition 6.32 we can compute such a decomposition in polynomial time. The idea is that we utilise the restrictions imposed by a DAG-decomposition to bound the number of strategies we need to consider. Although memoryless strategies are sufficient for parity games, we do not assume the strategies we consider are memoryless.

Consider $U \subseteq V(\mathcal{A})$ and a set W that guards U. Fix a pair of strategies σ and τ. For any $v \in U$, there is exactly one play $\pi = v_0 v_1 \cdots$ that is consistent with Player 0 playing σ and Player 1 playing τ. Let π' be the maximal prefix of π that is contained in U. The *outcome* of the pair of strategies (σ, τ) (given U and v) is defined as follows.

$$\text{out}_{\sigma,\tau}(U, v) := \begin{cases} \mathsf{win}_0 & \text{if } \pi' = \pi \text{ and } \pi \text{ is winning for Even;} \\ \mathsf{win}_1 & \text{if } \pi' = \pi \text{ and } \pi \text{ is winning for Odd;} \\ (v_{i+1}, p) & \text{if } \pi' = v_0 \cdots v_i \text{ and } p = \max\{\chi(v_j) : j \leq i+1\}. \end{cases}$$

That is to say that, if the play consistent with Player 0 playing σ and Player 1 playing τ leads to a cycle contained entirely within U, then the outcome simply records which player wins the game. However, if the winner is not determined entirely within U, the outcome records the vertex w in W in which the play emerges from U and the largest priority that is seen in the play π starting in v and ending in w, including the end points.

By construction, if $\text{out}_{\sigma,\tau}(U, v) = (w, p)$ then $w \in W$. More generally, for any set $W \subseteq V$, define the set of potential outcomes in W, written pot-out(W), to be the set $\{\mathsf{win}_0, \mathsf{win}_1\} \cup \{(w, p) : w \in W \text{ and } p \in \mathsf{P}\}$.

We recall from Chapter 3, the definition of the *reward order* $\sqsubseteq$. We now define a partial order $\trianglelefteq$ on pot-out(W) which orders potential outcomes according to how good they are for Player 1. It is the least partial order satisfying the following conditions:

(i) $\mathsf{win}_1 \trianglelefteq o$ for all outcomes o;
(ii) $o \trianglelefteq \mathsf{win}_0$ for all outcomes o;
(iii) $(w, p) \trianglelefteq (w, p')$ if $p \sqsubseteq p'$ for all $w \in W$.

In particular, (w, p) and (w', p') are incomparable if $w \neq w'$. The idea is that if τ and τ' are strategies such that $\text{out}_{\sigma,\tau}(U, v) \trianglelefteq \text{out}_{\sigma,\tau'}(U, v)$ then Player 1 is better off playing strategy τ rather than τ' in response to Player 0 playing according to σ.

A single outcome is the result of fixing the strategies played by both players in the subgame induced by a set of vertices U. If we fix the strategy of Player 0 to be σ but consider all possible strategies that Player 1 may play, we can order these strategies according to their outcome. If one strategy achieves outcome o and another o' with $o \trianglelefteq o'$, there is no reason for Player 1 to consider the latter strategy. Thus, we define $\text{result}_\sigma(U, v)$ to be the set of outcomes that are achieved by the best strategies that Player 1 may follow, in response to Player 0 playing according to σ. More formally, $\text{result}_\sigma(U, v)$ is the set of $\trianglelefteq$-minimal elements in the set $\{o : o = \text{out}_{\sigma,\tau}(U, v) \text{ for some } \tau\}$. Thus, $\text{result}_\sigma(U, v)$ is an anti-chain in the partial order $(\text{pot-out}(W), \trianglelefteq)$, where W is a set of guards for U. Finally, we write $\textsc{Result}(U, v)$ for the set $\{\text{result}_\sigma(U, v) : \sigma \text{ is a strategy for Player } 0\}$.

The data structure which we wish to compute is defined as follows. For any $d \in V(\mathcal{D})$, let $V_d = X_{\geq d} \setminus X_d$. Let

$$\textsc{Frontier}(d) = \{(v, r) : v \in V_d \text{ and } r \in \textsc{Result}(V_d, v)\}.$$

We show how to compute in polynomial time $\textsc{Frontier}(d)$ for all $d \in V(\mathcal{D})$. It follows from the definitions that if $\mathsf{win}_0 \in \textsc{Result}(V(\mathcal{A}), v)$, then Player 0 has a winning strategy from v. Thus, as $X_{\geq r} = V(\mathcal{A})$ when r is the root of $\mathcal{D}$, it follows that $\mathsf{win}_0 \in \textsc{Result}(X_{\geq r}, v_I(\mathcal{A}))$ if, and only if, Player 0 wins the game.

We observe that Since $X_{\geq d} \setminus X_d$ is guarded by X_d, $|X_d| \leq k$ and $|V_d| \leq n$, the number of distinct values of $\text{result}_\sigma(V_d, v)$ as σ ranges over all possible strategies is at most $(n+1)^k + 2$. This bound on the number of possible values of $\text{result}_\sigma(V_d, v)$ guarantees that $|\textsc{Frontier}(d)| \leq n\big((n+1)^k + 2\big)$.

We now outline how we compute $\textsc{Frontier}(d)$ for each stage of the dynamic programming scheme presented earlier.

Leaves: If $d \in V(\mathcal{D})$ is a leaf, then as $|V_d| \leq k$, it is clear that for all $v \in V_d$, $\textsc{Result}(V_d, v)$, and hence $\textsc{Frontier}(d)$, can be computed in constant time.

Combine: If $d \in V(\mathcal{D})$ is a node with two successors d_1 and d_2, then as $X_d = X_{d_1} = X_{d_2}$, it follows that $V_d = V_{d_1} \cup V_{d_2}$. In this case, as X_d guards V_{d_1} and V_{d_2} there is no path from a vertex in V_{d_1} to a vertex in V_{d_2} except through X_d. It is straightforward to show that $\textsc{Frontier}(d) = \textsc{Frontier}(d_1) \cup \textsc{Frontier}(d_2)$.

Expand: If $d \in V(\mathcal{D})$ is a node with one successor d', we consider three cases.

Case 1: $X_d = X_{d'}$. In this case, $\textsc{Frontier}(d) = \textsc{Frontier}(d')$.

Case 2: $X_d \setminus X_{d'} = \{u\}$. Then, by (D2), $u \notin V_{d'}$. Also, by the definition of V_d, $u \notin V_d$. We conclude that $V_d = V_{d'}$. Moreover, since $X_{d'}$ guards $V_{d'}$ (by Lemma 6.16(iii)), there is no path from any element of $V_{d'}$ to u except through $X_{d'}$. Thus, if $(w, p) \in \text{result}_\sigma(V_d, v)$ for some v and σ, it must be the case that $w \in X_{d'}$. Hence, $\textsc{Frontier}(d) = \textsc{Frontier}(d')$.

Case 3: $X_{d'} \setminus X_d = \{u\}$. This is the critical case. Here $V_d = V_{d'} \cup \{u\}$ and in order to construct $\textsc{Frontier}(d)$ we must determine the results of all plays beginning at u. If u has one successor, then this is trivial, so let us assume u has 2 successors u_1 and u_2. We observe that for $i \in \{1, 2\}$ either $u_i \in X_d$ or $u_i \in V_{d'}$. If $u_i \in X_d$, let $R_i =$

$\{(u_i, \max\{p, q\})\}$, where $p = \chi(u)$ and $q = \chi(u_i)$. Otherwise let $R_i = \text{RESULT}(V_{d'}, u_i)$. Thus R_i is the set of outcomes obtained if the play proceeds from u to u_i.

Consider a play from $v \in V_d$. If it does not reach u, then we can read, from $\text{RESULT}(V_{d'}, v) \in \text{FRONTIER}(d')$, the outcome of the play. Otherwise, if the play reaches u, it continues to either u_1 or u_2. If both u_1 and u_2 are in $V_{d'}$ then either the play returns to u, in which case we know the winner of the play, or the play reaches a vertex in X_d. This latter case also occurs if either of u_1 or u_2 is in X_d. Thus to compute $\text{RESULT}(V_d, v)$, and hence $\text{FRONTIER}(d)$, we proceed as follows.

For each $r \in \text{RESULT}(V_{d'}, v)$, we do the following. If there is no $p \in \mathsf{P}$ such that $(u, p) \in r$ add r to $\text{RESULT}(V_d, v)$. Otherwise, let $(u, p) \in r$ for some p. We now consider two cases. If $u \in V_1(\mathcal{A})$ then for each $r_1 \in R_1$ and $r_2 \in R_2$, let $R = r_1 \cup r_2$. Replace each $(w, q) \in R$ with $(w, \max\{p, q\})$. Let $R' = R \cup (r \setminus \{(u, p)\})$. If $(u, q) \in R'$ for some odd q then Player 1 wins the play for the chosen strategies, so replace (u, q) with win_1. Similarly, replace $(u, q) \in R'$ for q even with win_0. Finally, we remove the elements of R' which are not $\trianglelefteq$-minimal and add R' to $\text{RESULT}(V_d, v)$.

Now suppose $u \in V_0(\mathcal{A})$ for each $r' \in R_1 \cup R_2$, if $(u, q) \in r'$ and $\max p, q$ is odd, replace r' with win_1 and add it to $\text{RESULT}(V_d, v)$. Otherwise, let $R = \big(r \setminus \{(u, p)\}\big) \cup \big\{(w, q) : (w, q') \in r' \text{ and } q = \max\{p, q'\}\big\}$. If R contains a pair (u, q) then q must be even and we replace this pair in R by win_0. Finally, we add the $\trianglelefteq$-minimal elements of R to $\text{RESULT}(V_d, v)$.

In a similar way, we can also compute the set $\text{RESULT}(V_d, u)$.

It is clear from the bounds on the size of $\text{FRONTIER}(d)$ that at each stage, $\text{FRONTIER}(d)$ can be computed in polynomial time. Since the DAG-decomposition has size at most $O(n^{2k+4})$, it follows that this algorithm runs in polynomial time. This completes the outline of the proof of Theorem 6.34.

6.4 Relation to other graph connectivity measures

As a structural measure for undirected graphs, the concept of tree-width is of unrivalled robustness. On the realm of directed graphs, however, its heritage seems to be split among several different concepts. In the sequel we compare DAG-width with several other connectivity measures for directed graphs, namely tree-width, directed tree-width, and directed path-width. We show that, despite their similar nature, the measures are all significantly different.

6.4.1 Undirected tree-width

First we formalize the relationship between DAG-width and undirected tree-width alluded to in previous sections. We recall from Chapter 4, the definition of tree-width. We also recall that the tree-width of a directed graph $\mathcal{G}$ is defined as the tree-width of the undirected graph obtained from $\mathcal{G}$ by forgetting the orientation of the edges.

Proposition 6.35.

(i) If a directed graph $\mathcal{G}$ has tree-width k, it has DAG-width at most $k + 1$.

(ii) There exists a family of directed graphs with arbitrarily large tree-width and DAG-width 1.

Proof. (i): Suppose $(\mathcal{T}, \mathcal{W})$ is a tree decomposition of $\mathcal{G}$ of width k, with $\mathcal{W} = (W_t)_{t \in V(\mathcal{T})}$. Choose some $r \in V(\mathcal{T})$ and orient the edges of $\mathcal{T}$ away from r. That is, if $\{s, t\} \in E(\mathcal{T})$ and s is on the unique path from r to t, then change $\{s, t\}$ to (s, t). Since $\mathcal{T}$ is a tree, every edge has a unique orientation in this manner. Let $\mathcal{D}$ be the resulting DAG. For all $d \in V(\mathcal{D})$, set $X_d := W_t$ where t is the node of $\mathcal{T}$ corresponding to d. We claim that $(\mathcal{D}, \mathcal{X})$ with $\mathcal{X} = (X_d)_{d \in V(\mathcal{D})}$ is a DAG-decomposition of $\mathcal{G}$ of width $k + 1$. The condition (D1) is trivial from (T1); (D2) follows from (T2). The orientation ensures $\mathcal{D}$ has one root r, so $X_{\geq r} = V(\mathcal{G})$. Condition (D3) is hence satisfied at the root. For the other nodes, (D3) follows from Lemma 4.2. Let $(d, d') \in E(\mathcal{D})$ and suppose $v \in X_{\geq d'} \setminus X_d$. Suppose also that $(v, w) \in E(\mathcal{G})$ and $w \notin X_{\geq d'} \setminus X_d$. As there is a path (of length 1) from v to w, it follows from Lemma 4.2 that either $v \in X_d \cap X_{d'}$ or $w \in X_d \cap X_{d'}$. Since $v \notin X_d$, $w \in X_d \cap X_{d'}$ and (D3) holds.

(ii): For any integer $n \in \mathbb{N}$, let $\mathcal{K}_n$ be the (undirected) complete graph with n vertices $v_1, v_2, \ldots, v_n$. Orient the edges of $\mathcal{K}_n$ such that (v_i, v_j) is an edge if and only if $i < j$. The resulting directed graph is acyclic and therefore has DAG-width 1, but the underlying undirected graph is a complete graph of n vertices and therefore has tree-width $n - 1$. □

We now observe that DAG-width is equivalent to tree-width on undirected graphs if we view an undirected graph as a directed graph in the natural way. We recall from Section 1.1.2, the directed graph $\overleftrightarrow{\mathcal{G}}$ obtained from an undirected graph $\mathcal{G}$ by replacing each edge $\{u, v\}$ with two anti-parallel edges (u, v) and (v, u).

Proposition 6.36. *Let $\mathcal{G}$ be an undirected graph. $\mathcal{G}$ has tree-width $k - 1$ if, and only if, $\overleftrightarrow{\mathcal{G}}$ has DAG-width k.*

Proof. It is easily seen that the k-cops and robber game for undirected graphs on $\mathcal{G}$ is equivalent to the k-cops and robber game for directed graphs on $\overleftrightarrow{\mathcal{G}}$. The result follows from the correspondence between the measures and existence of monotone winning strategies. □

6.4.2 Directed tree-width

In Chapter 4 we saw directed tree-width from [JRST01] and in Chapter 5 we discussed how it was characterized by the strong visible robber game. We can use this game characterization to relate directed tree-width and DAG-width: as the strong visible robber game is defined similarly to the cops and visible robber game with added restrictions on movement of the robber, we see that a (robber-monotone) winning strategy for k cops in the cops and visible robber game is a (robber-monotone) winning strategy for k cops in the strong visible robber game. Thus, we can use Lemma 5.41 to obtain a bound on the directed tree-width. Towards a converse to this, we show that directed tree-width and DAG-width are very different measures by exhibiting a class of graphs with small directed tree-width and arbitrarily large DAG-width.

Proposition 6.37.

(i) If a directed graph $\mathcal{G}$ has DAG-width k, it has directed tree-width at most $3k + 1$.

(ii) There exists a family of graphs with arbitrarily large DAG-width and directed tree-width 1.

Proof. (i): If $\mathcal{G}$ has DAG-width k then k cops can win the cops and visible robber game on $\mathcal{G}$. Thus, k cops can win the strongly visible robber game on $\mathcal{G}$, as the robber is more restricted in this game. From Lemma 5.41, it follows that $\mathcal{G}$ has directed tree-width at most $3k+1$.

(ii): Consider the family $\{(\mathcal{T}_k^2)^{op} : k \geq 2\}$ of graphs defined in Proposition 6.7. Note that $(\mathcal{T}_k^2)^{op}$ is a binary branching tree of height k with back-edges from every vertex to each of its ancestors. We have shown that $(\mathcal{T}_k^2)^{op}$ has cop number k, and it is clear that the strategy described for k cops is monotone, so $(\mathcal{T}_k^2)^{op}$ has DAG-width k. On the other hand, consider the directed tree $\mathcal{T}$ obtained from $(\mathcal{T}_k^2)^{op}$ by removing back-edges. For each $t' \in V(\mathcal{T})$, let $B_{t'} := \{t, s\}$ where t is the vertex in $V((\mathcal{T}_k^2)^{op})$ corresponding to t' and s is the predecessor of t (if t' is not the root of T), and let $W_{(s',t')} := \{s\}$ for all $(s', t') \in E(\mathcal{T})$. Then, it is easy to see that $(\mathcal{T}, (B'_t)_{t' \in V(\mathcal{T})}, (W_e)_{e \in E(\mathcal{T})})$ is a directed tree decomposition of $(\mathcal{T}_k^2)^{op}$ of width 1. For $k \geq 2$, $(T_k^2)^{op}$ is not acyclic and therefore has directed tree-width exactly 1. □

6.4.3 Directed path-width

We saw in Chapter 4 the definition of *path-width*. According to Barát [Bar05], Reed, Seymour and Thomas defined a natural extension of path-width to directed around 1995, however [Tho02] seems to be the first occurrence of the definition in the literature. The definition mirrors the definition of path-width, however the direction of the edges is accounted for by fully utilising the linear ordering present in a sequence.

Definition 6.38 (Directed path decompositions and directed path-width [Bar05])**.** Let $\mathcal{G}$ be a directed graph. A *directed path decomposition* of $\mathcal{G}$ is a sequence $X_1, \ldots, X_n$ of subsets of $V(\mathcal{G})$ such that:

(DP1) $\bigcup_{i=1}^{n} X_i = V(\mathcal{G})$,

(DP2) If $i \leq j \leq k$ then $X_i \cap X_k \subseteq X_j$, and

(DP3) For each $e = (u, v) \in E(\mathcal{G})$, there exists $i \leq j$ such that $u \in X_i$ and $v \in X_j$.

The *width* of a directed path decomposition, $X_1, \ldots, X_n$, is $\max\{|X_i| : 1 \leq i \leq k\} - 1$. The *directed path-width* of $\mathcal{G}$ is the smallest width of any directed path decomposition of $\mathcal{G}$.

Just as tree-width can be characterized by the cops and visible robber game, we saw in Chapter 5 that path-width can also be characterized by a cops and robber game: the cops and invisible robber game of Example 5.2.2. In [Bar05] Barát considered the natural extension of this cops and robber game to directed graphs and showed that it the number of cops required to capture an invisible robber lies within one of the directed path-width of the graph. He also observed that the number of cops required to capture an invisible robber with a cop-monotone strategy is equal to one more than the directed path-width of the graph.

It is therefore not surprising that directed path-width is intimately related to DAG-width. From the game characterizations, it appears that directed path-width is to DAG-width as path-width is to tree-width. Indeed, as we see from the definitions the two are closely connected. In fact, a DAG-decomposition can be seen as a generalization of a directed path decomposition where we replace the linear order of the subsets of $V(\mathcal{G})$ with a partial order. This means that a directed path decomposition is a DAG-decomposition where the underlying DAG is a directed path. It is therefore not surprising that DAG-width bounds directed path-width below and there are families of graphs of bounded DAG-width and unbounded directed path-width. Just as the

class of binary trees is a class of graphs with bounded tree-width and unbounded directed path-width, we now show that the class of bidirected binary trees is a class of graphs with bounded DAG-width and unbounded directed path-width.

Proposition 6.39.

(i) If a directed graph $\mathcal{G}$ has directed path-width k, it has DAG-width at most $k+1$.

(ii) There exists a family of graphs with arbitrarily large directed path-width and DAG-width 2.

Proof. (i): Let $W_1, \ldots, W_n$ be a directed path decomposition of $\mathcal{G}$ of width k. Let $\mathcal{D}_n$ be the directed path with n vertices. That is $V(\mathcal{D}_n) = \{d_1, \ldots, d_n\}$ and $(d_i, d_j) \in E(\mathcal{D}_n)$ if, and only if, $j = i+1$. Set $X_{d_i} := W_i$ for all $d_i \in V(\mathcal{D}_n)$. We claim $(\mathcal{D}_n, (X_d)_{d \in V(\mathcal{D}_n)})$ is a DAG-decomposition of $\mathcal{G}$ of width $k+1$. Condition (D1) follows from (DP1) and (D2) follows from (DP2). To show (D3) for $1 \leq i < n$, suppose $v \in X_{\geq d_{i+1}} \setminus X_{d_i}$ and $(v, w) \in E(\mathcal{G})$. From (DP3) there exist $i' \leq j'$ such that $v \in W_{i'}$ and $w \in W_{j'}$. If $i' \leq i$, then by (DP2) $v \in X_{d_i}$, contradicting the choice of v. Thus, $i < i' \leq j'$ and $w \in X_{\geq d_{i+1}}$. If $w \notin X_{\geq d_{i+1}} \setminus X_{d_i}$ then $w \in X_{d_i}$ and therefore $w \in X_{d_{i+1}}$ by (DP2). Thus, $X_{d_i} \cap X_{d_{i+1}}$ guards $X_{\geq d_{i+1}} \setminus X_{d_i}$.

(ii): Let $\mathcal{T}_k$ be the (undirected) binary tree of height $k \geq 2$. From Proposition 6.36, $\overleftrightarrow{\mathcal{T}_k}$ has DAG-width 2. It is easy to see that on $\overleftrightarrow{\mathcal{T}_k}$, an invisible robber can defeat $k-1$ cops, but k cops have a winning strategy. Therefore, from [Bar05], $\overleftrightarrow{\mathcal{T}_k}$ must have directed path-width at least $k-2$. Thus, the family $\{\overleftrightarrow{\mathcal{T}_k} : k \geq 2\}$ satisfies the proposition. □

Chapter 7

Digraph measures: Kelly-width

In Chapter 4 we introduced the concept of tree-width as a measure of graph complexity. We remarked on its usefulness for algorithmic purposes, and discussed the importance of the problem of extending tree-width to directed graphs. In this chapter, we continue investigating this extension by considering other characterizations of tree-width and their natural generalizations to digraphs.

Part of the reason why tree-width is such a good measure of graph complexity is that many other measures arising from different areas of graph theory can be shown to be equivalent to tree-width. For instance, we saw in Chapter 5 that the number of cops required to capture a visible robber in a graph-searching game is equivalent to the tree-width of that graph. In this chapter we consider three other characterizations of tree-width: partial k-trees, elimination orders and a graph searching game in which an invisible robber attempts to avoid capture by a number of cops, subject to the restriction that he may only move if a cop is about to occupy his position. Partial k-trees are the historical forerunner of tree-width and are therefore associated with graph structure theory [Ros70]. In fact, many of the original algorithmic results for tree-width were formulated in terms of partial k-trees (see, for example [AP89]). Elimination orderings are particularly useful in the analysis of (symmetric) matrix factorizations such as Cholesky decompositions [Liu90]. For example, elimination orders can be used to determine the parallel time required to factorize a symmetric matrix with Gaussian elimination [BGHK95]. Finally, as we saw in Chapter 5 (and also [DKT97, FHT04]), graph searching games have recently been used to explore and generate robust measures of graph complexity. We generalize all these to directed graphs, resulting in partial k-DAGs, directed elimination orderings, and an inert robber game on digraphs. We show that all these generalizations are equivalent on digraphs and are also equivalent to the width-measure associated to a new kind of decomposition we introduce. As the game is reminiscent of capturing hideout-based outlaws, we propose the name Kelly-decompositions, after the infamous Australian bushranger Ned Kelly. The fact that all these notions are equivalent on digraphs as they are on undirected graphs suggests that this might be a robust measure of complexity and connectivity of digraphs.

As with tree-decompositions and DAG-decompositions, Kelly-decompositions have a structure that is well suited for designing dynamic programming algorithms that will run in polynomial time when the width of these decompositions is bounded. However, unlike DAG-decompositions (as far as is currently known), the size of Kelly-decompositions can be made linear in the size of the graph it decomposes, significantly reducing the space complexity of such algorithms. As with the previous chapter, we will introduce a general scheme for produc-

ing dynamic programming algorithms that use the additional structural information provided by Kelly-decompositions. We illustrate its use by producing algorithms for solving NP-complete problems such as Hamiltonian cycle, and computing the winner of a parity game. Both these algorithms run in polynomial time on graphs of bounded Kelly-width.

The chapter is organised as follows. In the first section we formally define inert robber games, elimination orders, and partial k-trees and k-DAGs. We show that on digraphs the associated width measures are all equivalent. In Section 7.2, we introduce Kelly-decompositions and Kelly-width and show that it also coincides with the measures defined in Section 7.1. In Section 7.3, we present applications: Algorithms for Hamiltonian cycle, weighted disjoint paths and parity games that all run in polynomial time on graphs of bounded Kelly-width, and detail a connection between Kelly-decompositions and asymmetric matrix factorization. Finally, we compare Kelly-width to some of the other directed graph measures we have already seen such directed tree-width and DAG-width, showing that it is a unique measure of complexity. However, we also provide evidence to suggest that Kelly-width and DAG-width are measuring the same fundamental property of digraphs.

7.1 Inert robber games, elimination orderings, and partial k-DAGs

7.1.1 Inert robber game

The cops and robber game we consider for this chapter is the cops and inert robber game from Example 5.2.5. This game consists of an invisible robber who is able to run from his position along any path which does not pass through a cop, however he may only move if a cop is about to land on his position. For convenience, we say that he is *inert*. The natural generalization of this game to directed graphs is defined as followed.

Definition 7.1 (Cops and inert robber game)**.** Let $\mathcal{G}$ be a directed graph. The *cops and inert robber game on* $\mathcal{G}$ is the cops and robber game defined by $(\mathcal{L}_c, \mathcal{L}_r, \mathcal{A})$, where

- $\mathcal{L}_c = \mathcal{P}(V(\mathcal{G}))$ and $\mathcal{L}_r = \mathcal{P}(V(\mathcal{G})) \setminus \{\emptyset\}$,
- $V_0(\mathcal{A})$ consists of pairs $(X, R) \in \mathcal{L}_c \times \mathcal{L}_r$ such that $X \cap R = \emptyset$,
- $V_1(\mathcal{A})$ consists of triples $(X, X', R) \in V_1(\mathcal{A})$ for all $(X, R) \in V_0(\mathcal{A})$ and all $X' \in \mathcal{L}_c$,
- For all $(X, R) \in V_0(\mathcal{A})$ and all $X' \in \mathcal{L}_c$ there is an edge in $E(\mathcal{A})$ from (X, R) to (X, X', R), and
- There is an edge in $E(\mathcal{A})$ from (X, X', R) to (X', R') if, and only if,

$$R' = \big(R \cup \mathit{Reach}_{\mathcal{G}\setminus(X\cap X')}(X' \cap R)\big) \setminus X'.$$

We recall from Chapter 5 the definitions of a *search*, *monotonicity* and a *strategy*. As with the game characterizing tree-width, we are interested in the minimum number of cops required to capture the robber, so we also recall the definition of a *strategy for k cops* from Definition 5.32. Since R' is uniquely defined from X, R and X', the inert robber game is in actuality a

single player game. As we mentioned earlier, this is typical for games with an invisible robber. One consequence is that given a strategy for the cops, there is a unique play consistent with that strategy. We call this the play *associated* with the strategy. In the remainder of this chapter we are primarily concerned with robber-monotone strategies. However, we first show that the added constraint on the movement of the invisible robber does not affect the existence of a cop-monotone winning strategy for k cops.

Proposition 7.2. *Let $\mathcal{G}$ be a digraph. Then k cops have a cop-monotone winning strategy in the cops and invisible robber game on $\mathcal{G}$ if, and only if, k cops have a cop-monotone winning strategy in the cops and inert robber game on $\mathcal{G}$.*

Proof. Since the cops and inert robber game is more restrictive on the robber than the cops and invisible robber game, a winning strategy in the latter is a winning strategy in the former. We now show how a cop-monotone winning strategy, σ, for k cops in the cops and inert robber game is also a cop-monotone winning strategy for k cops in the cops and invisible robber game. Let $(X_0, R_0)(X_1, R_1)\cdots(X_n, R_n)$ be the unique search associated with σ in the cops and inert robber game. We define a k-cop cop-monotone strategy, σ', for the cops and invisible robber game as follows. Define R'_i inductively as: $R'_0 = V(\mathcal{G})$, and for $1 \leq i \leq n$, $R'_i = \textit{Reach}_{\mathcal{G}\setminus(X_i\cap X_{i-1})}(R'_{i-1}) \setminus X_i$. Then define $\sigma'(X_i, R'_i) = X_{i+1}$, so σ' is essentially the strategy resulting from playing σ in the cops and invisible robber game. By definition, $(X_0, R'_0)(X_1, R'_1)\cdots$ is the search associated with σ', and it is clearly a cop-monotone strategy for k cops. We now show that it is winning. In particular, we prove by induction on i that $R'_i = R_i$ for $0 \leq i \leq n$.

Since $R_0 = V(\mathcal{G}) = R'_0$ our claim is clearly true for $i = 0$. Now suppose $R_i = R'_i$ for some $i \geq 0$. Since $R_i \cup \textit{Reach}_{\mathcal{G}\setminus(X_i\cap X_{i+1})}(R_i \cap X_{i+1}) \subseteq \textit{Reach}_{\mathcal{G}\setminus(X_i\cap X_{i+1})}(R'_i)$, we have $R_{i+1} \subseteq R'_{i+1}$. So suppose $R_{i+1} \not\supseteq R'_{i+1}$. Then there exists $w \in \textit{Reach}_{\mathcal{G}\setminus(X_i\cap X_{i+1})}(R_i) \setminus X_{i+1}$ such that $w \notin R_i \cup \textit{Reach}_{\mathcal{G}\setminus(X_i\cap X_{i+1})}(R_i \cap X_{i+1}) \setminus X_{i+1}$. Thus $w \notin X_i \cup R_i$. Note that since $w \notin R_i$, we have $i \geq 1$. Furthermore, there exists $v \in R_i \setminus X_{i+1}$ such that there is a path from v to w in $\mathcal{G} \setminus (X_i \cap X_{i+1})$. Let v' be the last element of R_i on this path, and let $w' \notin R_i$ be the successor of v' on this path. Since the path is in $\mathcal{G} \setminus (X_i \cap X_{i+1})$, $w' \notin X_i \cap X_{i+1}$. Suppose $w' \notin X_i$. Then since $X_i \cap R_i = \emptyset$, there is a path from v to w' in $\mathcal{G} \setminus X_i$. Therefore, as $v \in R_i = R'_i = \textit{Reach}_{\mathcal{G}\setminus(X_{i-1}\cap X_i)}(R_{i-1}) \setminus X_i$ we have $w' \in R_i$, contradicting the definition of w'. Thus

$$w' \in X_i \setminus X_{i+1}.$$

Now let $j \geq i+1$ be such that $v' \in R_j \setminus R_{j+1}$. Since σ is winning, and $v' \in R_{i+1}$, there is such a j. By the definition of the cops and inert robber game, it must be that $v' \in X_{j+1} \setminus X_j$. We claim that $w' \in R_{j+1}$. Since $i \leq i+1 \leq j+1$, and $w \in X_i \setminus X_{i+1}$, by the cop-monotonicity of σ, $w' \notin X_{j+1}$. Therefore, as $(v', w') \in E(\mathcal{G})$,

$$w' \in \textit{Reach}_{\mathcal{G}\setminus(X_j\cap X_{j+1})}(R_j \cap X_{j+1}) = R_{j+1}.$$

Now let $l \geq j+1$ be such that $w' \in R_l \setminus R_{l+1}$. Since σ is winning and $w' \in R_{j+1}$, such an l exists. By the definition of the cops and inert robber game, it must be that $w' \in X_{l+1}$. But since $i \leq i+1 \leq l+1$, and $w' \in X_i$, by the cop-monotonicity of σ, $w' \in X_{i+1}$ – contradiction. Thus $R_i \supseteq R'_i$, and therefore $R_i = R'_i$. □

7.1.2 Elimination orderings

Our next definition extends the idea of vertex elimination to digraphs. Vertex elimination, for undirected and directed graphs, has been researched for many years in the study of linear programming [RT75]. One technique for solving a system of equations is to combine equations so that the value of some variables can easily be determined, thereby eliminating those variables and reducing the system to a simpler one. This elimination process may introduce new relations between the remaining variables, and capturing this process in a more general setting is the motivation behind vertex elimination of graphs. We can represent a system of equations as a graph with a vertex for each variable occurring in the system, and an edge between variables that are related by some equation in the system. Vertex elimination is then a symbolic representation of variable elimination.

More precisely, let $\mathcal{G}$ be an undirected graph and $v \in V(\mathcal{G})$. To *eliminate* v from $\mathcal{G}$, we remove v, but add edges (if necessary) between any two vertices adjacent to v. In this way, we see that vertex elimination is the process of removing vertices from a graph but adding edges to preserve reachability. It is this concept that we extend to directed graphs.

Definition 7.3 (Directed elimination)**.** Let $\mathcal{G}$ be a digraph and $v \in V(\mathcal{G})$. The graph resulting from *directed elimination of v from $\mathcal{G}$* is the graph $\mathcal{G}'$ obtained from $\mathcal{G}$ by deleting v and adding new edges (u, w) (if necessary) if (u, v) and $(v, w) \in E(\mathcal{G})$.

We can use vertex elimination to define a complexity measure on undirected graphs. Let $\mathcal{G}$ be an undirected graph. A linear order $\lhd = (v_1, v_2, \ldots, v_n)$ on $V(\mathcal{G})$ defines a sequence of eliminations whereby the vertices of $\mathcal{G}$ are successively eliminated in the order specified by $\lhd$. For convenience we call $\lhd$ an *elimination ordering* and this sequence of eliminations, the *elimination defined by* $\lhd$. We define the *width* of $\lhd$ to be the maximum of the degrees of the vertices when they are eliminated. These definitions easily translate to directed graphs, but the complexity measure we are interested in is the maximum out-degree of eliminated vertices.

Definition 7.4 ((Partial) Directed elimination ordering)**.** Let $\mathcal{G}$ be a digraph and let $V \subseteq V(\mathcal{G})$ be a subset of vertices. A *partial directed elimination ordering on V* is a linear ordering $\lhd = (v_1, v_2, \ldots, v_n)$ of V. A *directed elimination ordering* is a partial directed elimination ordering on $V(\mathcal{G})$. The *(partial) directed elimination defined by* $\lhd$ is the following sequence of directed graphs. We define $\mathcal{G}_0^{\lhd} := \mathcal{G}$, and let $\mathcal{G}_{i+1}^{\lhd}$ be the graph resulting from directed elimination of v_{i+1} from $\mathcal{G}_i^{\lhd}$. The *width* of $\lhd$ is the maximum over all i of the out-degree of v_i in $\mathcal{G}_i^{\lhd}$. For convenience we also define the *support of v_i with respect to* $\lhd$ as $\text{supp}_{\lhd}(v_i) := \{v_j : (v_i, v_j) \in E(\mathcal{G}_i^{\lhd})\}$.

We observe that the width of a directed elimination ordering is the maximum cardinality of all its supports.

Immediately from the definitions, we have this simple lemma relating the support of an element in an elimination ordering to the set of vertices reachable from that vertex.

Lemma 7.5. *Let $\lhd$ be a directed elimination ordering of a graph $\mathcal{G}$ and let $v \in V(\mathcal{G})$. Let $R := \{u : v \lhd u\}$. Then $\text{supp}_{\lhd}(v) = \{u : v \lhd u$ and there is $v' \in \text{Reach}_{\mathcal{G}\setminus R}(v)$ such that $(v', u) \in E(\mathcal{G})\}$.*

7.1.3 Partial k-trees and partial k-DAGs

The class of k-trees and, more generally, chordal graphs are important and widely studied classes of undirected graphs. A graph is *chordal* if any cycle of four or more vertices contains a chord – an edge between a pair of vertices not adjacent in the cycle, and a chordal graph is a *k-tree* if it contains no $(k+2)$-clique as a subgraph. These structural restrictions are algorithmically beneficial: for example, chordal graphs have a linear number of maximal cliques, so problems such as finding a clique of a given size, which are in general NP-complete, can be efficiently solved on chordal graphs and k-trees.

An equivalent way to characterize the class of k-trees is as a class of graphs generated by a generalization of how one might construct a tree.

Definition 7.6 ((Partial) k-trees)**.** The class of k-trees is defined recursively as follows:

- The complete graph on k vertices is a k-tree.
- A k-tree with $n+1$ vertices is obtained from a k-tree $\mathcal{H}$ with n vertices by adding a vertex and making it adjacent to a k-clique in $\mathcal{H}$.

A *partial k-tree* is a subgraph[1] of a k-tree.

The last concept we define in this section is a generalization of partial k-trees, called partial k-DAGs. Just as k-trees are a generalization of trees, k-DAGs are a class of digraphs generated by a generalization of how one might construct a directed, acyclic graph in a top-down manner.

Definition 7.7 ((Partial) k-DAG)**.** The class of *k-DAGs* is defined recursively as follows:

- A complete digraph with k vertices is a k-DAG.
- A k-DAG with $n+1$ vertices is obtained from a k-DAG $\mathcal{H}$ with n vertices by adding a vertex v and edges satisfying the following:
  - Edges from v to $X \subseteq V(\mathcal{H})$ where $|X| \leq k$
  - An edge from $u \in V(\mathcal{H})$ to v if $(u,w) \in E(\mathcal{H})$ for all $w \in X \setminus \{u\}$.

A *partial k-DAG* is a subgraph of a k-DAG.

The second condition on the edges provides a method to add as many edges as possible going to the new vertex without introducing cycles. Note that if $X = \emptyset$, the antecedent of this condition is true for all $u \in V(\mathcal{H})$, so a digraph is a partial 0-DAG if, and only if, it is a directed acyclic graph.

We also observe that this definition generalizes k-trees, for if the vertices (X) adjacent to the new vertex (v) induce a clique, we will add edges back from X to v, effectively creating bidirected edges between v and X (and possibly some additional edges from $\mathcal{H} \setminus X$ to v). The following result shows that k-DAGs generalize the alternative characterization of k-trees we presented initially.

Lemma 7.8. *Let $\mathcal{G}$ be a k-DAG. Then:*

[1] Technically a partial graph is a spanning subgraph, that is, subgraph with the same vertex set. However, for the results we establish the distinction is not significant.

(i) *$\mathcal{G}$ contains no $(k+2)$-clique as a subgraph,*

(ii) *Any cycle in $\mathcal{G}$ with at least three vertices contains a chord, and*

(iii) *Any bidirected cycle with at least four vertices contains a bidirected chord.*

Proof. (i): Let $W \subseteq V(\mathcal{G})$ be a set of $k+2$ vertices. Suppose $v \in W$ was the last vertex of W to be added in the construction of $\mathcal{G}$. Since all other vertices of W were added before v, all edges from v to W were added as part of the first condition on the added edges. Therefore, there must be at most k outgoing edges from v to vertices in W, and so W cannot be the vertex set of a $(k+2)$-clique.

(ii): Let $C = (v_1, v_2, \ldots, v_n)$ be a cycle of length $n \geq 3$ in $\mathcal{G}$. Without loss of generality, assume v_0 was the last vertex of C to be added in the construction of $\mathcal{G}$. Since there is an edge from v_n to v_1, it follows that there must be an edge from v_n to all successors of v_1 added before v_1, in particular to v_2. Thus (v_n, v_2) is a chord of C.

(iii): Let $C = (v_1, v_2, \ldots, v_n)$ be a bidirected cycle of length $n \geq 4$. Again we assume v_1 was the last vertex of C to be added in the construction of $\mathcal{G}$. From the proof of *(ii)*, there is an edge $(v_n, v_2) \in E(\mathcal{G})$. Since $(v_1, v_n, \ldots, v_2)$ is also a cycle, the same argument implies there is also an edge $(v_2, v_n) \in E(\mathcal{G})$. These two edges make up a bidirected chord of C. □

Lemma 7.8 does not provide an equivalent characterization for k-DAGs because the given properties are invariant under edge-reversal. We see in Proposition 7.40 that the class of k-DAGs is not closed under this operation.

7.1.4 Equivalence results

We have introduced three notions that can be used to define the complexity of digraphs, all of which naturally extend measures for undirected graphs. On undirected graphs, the three measures are equivalent to each other, and also to tree-width [DKT97]. Our main result of this section is that the three measures introduced are equivalent on digraphs.

Theorem 7.9. *Let $\mathcal{G}$ be a digraph. The following are equivalent:*

1. *$k+1$ cops have a robber-monotone winning strategy to capture an inert robber on $\mathcal{G}$.*
2. *$\mathcal{G}$ has a directed elimination ordering of width $\leq k$.*
3. *$\mathcal{G}$ is a partial k-DAG.*

Proof. $1 \Rightarrow 2$: Suppose $k+1$ cops have a robber-monotone winning strategy σ. Without loss of generality, we assume that only one cop is placed at a time. Let $(X_0, R_0)(X_1, R_1)\cdots$ be the (unique) search consistent with σ. For each $v \in V(\mathcal{G})$, let $x_v = \min\{i : v \in X_i\}$. Since σ involves placing one cop at a time, for distinct $v, w \in V(\mathcal{G})$, $x_v \neq x_w$. Let $\lhd = (v_1, v_2, \ldots, v_n)$ be the order defined as: $v_i \lhd v_j$ if, and only if, $x_{v_j} < x_{v_i}$. For convenience, let $V_i = \{v_1, \ldots, v_i\}$, and $x_i = x_{v_i}$ for all i. We observe that from the definition of x_i, $V_i \cap X_{x_i} = \{v_i\}$.

We claim $\lhd$ has width $\leq k$. If this were not the case, there must exist v_i such that $|\text{supp}_{\lhd}(v_i)| \geq k+1$. As $|\text{supp}_{\lhd}(v_i)| \geq k+1$ and $|X_{x_i}| \leq k$ it follows that there exists $v_j \in \text{supp}_{\lhd}(v_i) \setminus X_{x_i}$. From the definition of $\text{supp}_{\lhd}(v_i)$, we have $v_i \lhd v_j$, so $x_j < x_i$. Furthermore, from Lemma 7.5, $v_j \in \mathit{Reach}_{\mathcal{G}[V_i \cup \{v_j\}]}(v_i)$. Therefore, since $V_i \cap X_{x_i - 1} \cap X_{x_i} = \emptyset$ and

$v_i \in X_{x_i}$ it follows that $v_j \in R_{x_i}$. But since $v_j \notin R_{x_j}$, the robber-monotonicity of σ implies $v_j \notin R_l$ for all $l \geq x_j$, contradicting the fact that $v_j \in R_{x_i}$. Thus there exists no such v_i with $|\text{supp}_\lhd(v_i)| \geq k+1$, and $\lhd$ has width $\leq k$.

$2 \Rightarrow 3$: Let $\lhd = (v_1, v_2, \ldots, v_n)$ be a directed elimination ordering of $\mathcal{G}$ of width k. For ease of notation, define $X_i := \text{supp}_\lhd(v_i)$, and let $m = n - k$. Let $\mathcal{K}_0$ be the complete graph on the vertices $\{v_{m+1}, v_{m+2}, \ldots, v_n\}$, and let $\mathcal{K}_j$ ($j \geq 1$) be the k-DAG formed by adding v_{m-j+1} to $\mathcal{K}_{j-1}$, and edges from v_{m-j+1} to X_{m-j+1} (together with the other edges added from $\mathcal{K}_{j-1}$ to $v_{n-k-j+1}$ in the definition of k-DAGs). We claim that for all $0 \leq j \leq m$, $\mathcal{G}^\lhd_{m-j}$ is a subgraph of $\mathcal{K}_j$. The result then follows by taking $j = m$. We prove our claim by induction on j. For the base case ($j = 0$) the result is trivial as $\mathcal{K}_j$ is a complete graph. Now assume the result is true for $j \geq 0$, and consider the graph $\mathcal{G}^\lhd_{m-j-1}$. For simplicity let $i = m - j$. By the definition of directed elimination, for every edge $(u, v) \in E(\mathcal{G}^\lhd_{i-1})$ either:

(a) $v_i \notin \{u, v\}$,
(b) $u = v_i$, or
(c) $v = v_i$.

In the first case, $(u, v) \in E(\mathcal{G}^\lhd_i)$ and therefore in $E(\mathcal{K}_j) \subseteq E(\mathcal{K}_{j+1})$ by the induction hypothesis. For the second case, (u, v) is added during the construction of $\mathcal{K}_{j+1}$. For the final case, for any $w \in X_i$, (v_i, w) is an edge of $\mathcal{G}^\lhd_{i-1}$, so (u, w) is an edge of $\mathcal{G}^\lhd_i$ (for $u \neq w$), and therefore of $\mathcal{K}_j$ by the induction hypothesis. Thus (u, v_i) is added during the construction of $\mathcal{K}_{j+1}$, and $E(\mathcal{G}^\lhd_{i-1}) \subseteq E(\mathcal{K}_{j+1})$ as required.

$3 \Rightarrow 1$: Let $\mathcal{G}$ be a partial k-DAG. Suppose $\mathcal{G}$ is a subgraph of the k-DAG, $\mathcal{K}$, formed from a complete graph on the vertices $X_k := \{v_1, v_2, \ldots, v_k\}$, and then by adding the vertices $v_{k+1}, v_{k+2}, \ldots, v_n$. For $1 \leq i \leq n-k$ let $X_{k+i} \subseteq \{v_1, \ldots, v_{k+i-1}\}$ denote the set of successors of v_{k+i}. That is, when v_{k+i} is added during the construction of $\mathcal{K}$, edges are added from v_{k+i} to each vertex in X_{k+i}. Note that for all i, $|X_i| \leq k$. We define a (history dependent) strategy σ for the cops inductively as follows. For all R, $\sigma(\emptyset, R) = X_k$. If $X = X_i$ for some i,

$k \leq i \leq n$ then for all R, $\sigma(X, R) = X_i \cup \{v_i\}$. If $X = X_i \cup \{v_i\}$ for some i, $k \leq i < n$, then for all R, $\sigma(X, R) = X_{i+1}$. We claim that this defines a monotone winning strategy for $k+1$ cops. Let $R_i = \{v_j : j > i\}$, then from the definition of k-DAGs and the X_i, it is easy to see that the search associated with the strategy is:

$$(\emptyset, V(\mathcal{G}))(X_k, R_k)(X_{k+1}, R_k)(X_{k+1} \cup \{v_{k+1}\}, R_{k+1}) \cdots (X_n \cup \{v_n\}, \emptyset).$$

As $R_i \supseteq R_{i+1}$ for all i, the strategy is monotone and winning as required. □

7.2 Kelly-decompositions and Kelly-width

Theorem 7.9 shows that the concepts introduced in the previous section define a sound measure of digraph complexity which naturally generalizes tree-width. We now turn to the problem of finding a closely related digraph decomposition. The decomposition we introduce is a partition of the vertices, arranged as a directed acyclic graph, together with sets of vertices which guard against paths in the graph that do not respect this arrangement. We have an additional restriction to avoid trivial decompositions: vertices in the guard sets must appear either to the left or earlier in the decomposition. Before we present the formal definition, we recall from Definition 6.13, the definition of *guarding*.

Definition 7.10 (Kelly-decomposition and Kelly-width)**.** A *Kelly-decomposition* of a digraph $\mathcal{G}$ is a triple $\mathbb{D} := (\mathcal{D}, \mathcal{B}, \mathcal{W})$ where $\mathcal{D}$ is a DAG and $\mathcal{B} = (B_d)_{d \in V(\mathcal{D})}$ and $\mathcal{W} = (W_d)_{d \in V(\mathcal{D})}$ are families of subsets of $V(\mathcal{G})$ such that

(K1) $\mathcal{B}$ is a partition of $V(\mathcal{G})$,

(K2) for all $d \in V(\mathcal{D})$, W_d guards $B_{\geq d} := \bigcup_{d' \geq_{\mathcal{D}} d} B_{d'}$, and

(K3) for all $d \in V(\mathcal{D})$ there is a linear order on its successors $d_1, \ldots, d_p$ so that for all $1 \leq i \leq p$, $W_{d_i} \subseteq B_d \cup W_d \cup \bigcup_{j<i} B_{\geq d_j}$. Similarly, there is a linear order on the roots such that $W_{r_i} \subseteq \bigcup_{j<i} B_{\geq r_j}$.

The *width* of $\mathbb{D}$ is $\max\{|B_d \cup W_d| : d \in V(\mathcal{D})\}$. The *Kelly-width* of $\mathcal{G}$ is the minimal width of any of its Kelly-decompositions.

Our main result of this section is that Kelly-decompositions do in fact correspond with the complexity measure defined at the end of the previous section.

Theorem 7.11. *Let $\mathcal{G}$ be a digraph. The following are equivalent:*

1. *k cops have a robber-monotone winning strategy to capture an inert robber on $\mathcal{G}$.*
2. *$\mathcal{G}$ has Kelly-width $\leq k$.*

Proof. $2 \Rightarrow 1$: Let $(\mathcal{D}, \mathcal{B}, \mathcal{W})$ a Kelly-decomposition of $\mathcal{G}$ of width k. Let $\mathcal{T}$ be the spanning tree of $\mathcal{D}$ obtained from the depth-first traversal of $\mathcal{D}$ which always chooses the greatest successor according to the ordering on successors guaranteed by (K3). Let $(t_1, t_2, \ldots, t_n)$ be the order of $V(\mathcal{T})$ (and hence, $V(\mathcal{D})$) visited in the depth-first traversal of $\mathcal{T}$ which always chooses the least successor according to the ordering. So t_1 will always be the root of $\mathcal{D}$ which is first in the linear order on the roots, t_2 will be the least successor of t_1 which is not a descendant of any greater root, or the next root of $\mathcal{D}$ in the ordering if no such successor exists, and so on. We observe that by the construction of this ordering, every descendant t_j of t_i in $\mathcal{D}$ is either a descendant of t_i in $\mathcal{T}$, or t_i and t_j have a common ancestor from which t_i is a descendant of a lesser successor than t_j. In both cases $j \geq i$ from the depth-first traversal of $\mathcal{T}$. It follows that

$$\bigcup_{j<i} B_{t_j} \cap B_{\geq t_i} = \emptyset. \tag{7.1}$$

We now define the strategy. For $1 \leq i \leq n$, let $X_{2i-1} = W_{t_i}$ and $X_{2i} = W_{t_i} \cup B_{t_i}$. We define a (history dependent) strategy σ inductively as $\sigma(\emptyset, R) = X_1$ and $\sigma(X_i, R) = X_{i+1}$ for all $R \subseteq V(\mathcal{G})$. We claim that σ is a robber-monotone winning strategy for k cops. Let $(X_0, R_0) \cdots (X_{2n}, R_{2n})$ be the search associated with the strategy. We show by induction on i that for $0 < i \leq n$, $R_{2i-2} = R_{2i-1} = \bigcup_{j \geq i} B_{\geq t_j}$. It follows immediately that the strategy must be monotone and winning. Since $X_1 = W_{t_1} = \emptyset$, we have $R_1 = R_0 = V(\mathcal{G}) = \bigcup_{j \geq 1} B_{\geq t_j}$. Now let us assume $R_{2i-2} = R_{2i-1} = \bigcup_{j \geq i} B_{\geq t_j}$ for some $i \geq 1$. From (K2), we observe that $\mathit{Reach}_{\mathcal{G} \setminus W_{t_i}}(B_{t_i}) \subseteq B_{\geq t_i} \subseteq R_{2i-1}$. Thus

$$\begin{aligned} R_{2i} &= \big(R_{2i-1} \cup \mathit{Reach}_{\mathcal{G} \setminus (X_{2i-1} \cap X_{2i})}(R_{2i-1} \cap X_{2i})\big) \setminus X_{2i} \\ &= \big(R_{2i-1} \cup \mathit{Reach}_{\mathcal{G} \setminus W_{t_i}}(B_{t_i})\big) \setminus B_{t_i} \\ &= \textstyle\bigcup_{j \geq i} B_{\geq t_j} \setminus B_{t_i} \\ &= \textstyle\bigcup_{j \geq i+1} B_{\geq t_j} \quad \text{(from Equation 7.1).} \end{aligned}$$

Since $W_d \cap B_{\geq d} = \emptyset$ for all $d \in V(\mathcal{D})$, it follows from (K3) and the construction of the ordering that $W_{t_{i+1}} \subseteq \bigcup_{j \leq i} B_{t_j}$. Therefore, from Equation 7.1, we have $R_{2i} \cap W_{t_{i+1}} \subseteq \bigcup_{j>i} B_{\geq t_j} \cap \bigcup_{j \leq i} B_{t_j} = \emptyset$. Hence,

$$\begin{aligned} R_{2i+1} &= \left(R_{2i} \cup \textit{Reach}_{\mathcal{G} \setminus (X_{2i} \cap X_{2i+1})}(R_{2i} \cap X_{2i+1})\right) \setminus X_{2i+1} \\ &= (R_{2i} \cup \emptyset) \setminus W_{t_{i+1}} \\ &= R_{2i}, \end{aligned}$$

completing the inductive step.

$1 \Rightarrow 2$: It follows from Theorem 7.9 that it suffices to show that if $\mathcal{G}$ has a directed elimination ordering of width $k-1$ then $\mathcal{G}$ has Kelly-width $\leq k$. Let $\lhd = (v_1, v_2, \ldots, v_n)$ be a directed elimination ordering of $\mathcal{G}$ of width $k-1$. We define $(\mathcal{D}, \mathcal{B}, \mathcal{W})$ as follows. $V(\mathcal{D}) := V(\mathcal{G})$. For all $d \in V(\mathcal{D})$ let $B_d := \{d\}$ and $W_d := \text{supp}_{\lhd}(d)$ and define $\mathcal{B} := (B_d)_{d \in V(\mathcal{D})}$ and $\mathcal{W} := (W_d)_{d \in V(\mathcal{D})}$. Towards defining the edge relation of $\mathcal{D}$, let $d \in V(\mathcal{D})$ be a node. For convenience we write $\mathcal{G}_d$ for the induced subgraph $\mathcal{G}[\{w : w \lhd d\} \cup \{d\}]$. Let $C_1, \ldots, C_p$ be the strongly connected components of $\mathcal{G}_d \setminus d$. Let $d_1, \ldots, d_p$ be the $\lhd$-maximal elements of $C_1, \ldots, C_p$, respectively. We put an edge (d, d_i) between d and d_i if d_i is reachable from d in $\mathcal{G}_d$ and there is no d_j with $d_i \lhd d_j \lhd d$ such that d_j is reachable from d in $\mathcal{G}_d$ and d_i is reachable from d_j in $\mathcal{G}_d \setminus d$.

We claim that $(\mathcal{D}, \mathcal{B}, \mathcal{W})$ is a Kelly-decomposition of width $\leq k$. Clearly, $\mathcal{D}$ is a DAG, as all the edges in $E(\mathcal{D})$ are oriented following the ordering $\lhd$. Further, the width of the decomposition is clearly at most one more than the width of $\lhd$.

To establish (K2), we first show the following claim.

Claim. For all $d \in V(\mathcal{D})$, $\textit{Reach}_{\mathcal{G}_d}(d) = B_{\geq d}$.

Proof of claim. We first show by induction on the index i of d in $\lhd$ that $\textit{Reach}_{\mathcal{G}_d}(d) \subseteq B_{\geq d}$. For $i = 1$ there is nothing to show. Suppose the claim has been proven for all $j < i$. Let $v \in \textit{Reach}_{\mathcal{G}_d}(d)$. Let $C_1, \ldots, C_m$ be the strongly connected components of $G_d \setminus d$. Without loss of generality we assume that $v \in C_1$. Let s be the $\lhd$-maximal element of C_1 and let d' be the $\lhd$-maximal element such that

- d' is the $\lhd$-maximal element of some C_i
- there is a directed path from d to d' in $\mathcal{G}_d$
- there is a directed path from d' to s in $\mathcal{G}_d \setminus d$.

By construction, there is an edge $(d, d') \in E(\mathcal{D})$. If $d' = v$, or in fact if d' is the $\lhd$-maximal element of C_1, then there is nothing more to show. Otherwise, if d' and v are not in the same strongly connected component of $\mathcal{G}_d \setminus d$, then s, and hence v, must be reachable from d' in $\mathcal{G}_{d'}$. For, by construction, s is reachable from d' in $\mathcal{G}_d \setminus d$ and d' is the $\lhd$-maximal element reachable from d in $\mathcal{G}_d$ and from which s can be reached in $\mathcal{G}_d \setminus d$. Thus, if s was not reachable from d' in $\mathcal{G}_{d'}$ then the only path from d' to s in $\mathcal{G}_d \setminus d$ must include an element $w \lhd d$ such that $d' \lhd w$, contradicting the maximality of d'. Hence, v is reachable from d' in $\mathcal{G}_{d'}$ and therefore, by induction hypothesis, $v \in B_{\geq t'} \subseteq B_{\geq t}$.

A simple induction on the height of the nodes in $\mathcal{D}$ establishes the converse. $\dashv$

It remains to show that for all $d \in V(\mathcal{D})$ there is a linear ordering $\sqsubset$ of the successors d satisfying the ordering condition required by the definition of Kelly-decompositions. For successors $v \neq v'$ of d define $v \sqsubset v'$ if $v' \lhd v$, that is, $\sqsubset$ is the inverse ordering of $\lhd$.

Let $d_1, \ldots, d_m$ be the successors of d ordered by $\sqsubset$. We claim that for all $i \in \{1, \ldots, m\}$,

$$W_{d_i} \subseteq B_d \cup W_d \cup \bigcup_{j<i} B_{\geq d_j}.$$

If $v \in B_d$ there is nothing to show. If $d \lhd v$ then $v \in W_d$ as $d_i \lhd d$ is reachable from d and therefore $W_{d_i} \cap \{x : d \trianglelefteq x\} = \text{supp}_{\lhd}(d_i) \cap \{x : d \trianglelefteq x\} \subseteq \text{supp}_{\lhd}(d) \cap \{x : d \trianglelefteq x\} = W_d \cap \{x : d \trianglelefteq x\}$. Finally, suppose $v \lhd d$. But then, $v \in B_{\geq d}$ and hence $v \in B_{\geq d_j}$ for some $1 \leq j \leq m$. By definition of support sets, $v \notin B_{\geq d_i}$ and $d_i \lhd v$. But then, $v \notin B_{\geq d_j}$ for all $j \sqsupset i$, that is, $j \lhd i$, as then $d_j \lhd v$ and by construction, $w \lhd d_j$ for all $w \in B_{\geq d_j}$. Hence, $v \in B_{\geq d_l}$ for some $d_l \rhd d_i$. This completes the proof of the theorem. □

The proof of Theorem 7.11 is constructive in that given an elimination ordering of width $k-1$ we construct a Kelly-decomposition of width k, and conversely. In fact, the proofs establish a slightly stronger statement.

Corollary 7.12. *Every digraph $\mathcal{G}$ of Kelly-width k has a Kelly-decomposition $\mathbb{D} = (\mathcal{D}, \mathcal{B}, \mathcal{W})$ of width k such that for all $d \in V(\mathcal{D})$:*

- $|B_d| = 1$,
- *W_d is the minimal set which guards $B_{\geq d}$, and*
- *Every vertex $v \in B_{\geq d}$ is reachable in $\mathcal{G} \setminus W_d$ from the unique $w \in B_d$.*

Further, if $\mathcal{G}$ is strongly connected, then $\mathcal{D}$ has only one root.

We call such a decomposition *special*.

Just as with the cops and visible robber game, it is easy to see that the cops and inert robber game satisfies the properties introduced in Section 5.4. The characterization of Kelly-width by such graph searching games implies that Kelly-width is well behaved under important structural relations. The proofs of the following results are similar to those presented in Section 6.3.

Lemma 7.13. *Let $(\mathcal{D}, \mathcal{B}, \mathcal{W})$ be a Kelly-decomposition of a digraph $\mathcal{G}$, and let $\mathcal{G}'$ be a subgraph of $\mathcal{G}$. $(\mathcal{D}, \mathcal{B}|_{\mathcal{G}'}, \mathcal{W}|_{\mathcal{G}'})$ where $\mathcal{B}|_{\mathcal{G}'} := (B_d \cap V(\mathcal{G}'))_{d \in V(\mathcal{D})}$ and $\mathcal{W}|_{\mathcal{G}'} := (W_d \cap V(\mathcal{G}'))_{d \in V(\mathcal{D})}$ is a Kelly-decomposition of $\mathcal{G}'$.*

Corollary 7.14. *Let $\mathcal{G}$ and $\mathcal{G}'$ be directed graphs such that $\mathcal{G}'$ is a subgraph of $\mathcal{G}$. Then Kelly-width$(\mathcal{G}') \leq$ Kelly-width$(\mathcal{G})$.*

Lemma 7.15. *Let $\mathcal{G}$ be a directed graph and $\mathcal{K}_n$ the complete graph on n vertices. Kelly-width$(\mathcal{G} \bullet \mathcal{K}_n) = n \cdot$ Kelly-width$(\mathcal{G})$.*

Lemma 7.16. *Let $\mathcal{G}$ be a directed union of the digraphs $\mathcal{G}_1$ and $\mathcal{G}_2$. Then*

$$\text{Kelly-width}(\mathcal{G}) = \max\{\text{Kelly-width}(\mathcal{G}_1), \text{Kelly-width}(\mathcal{G}_2)\}.$$

We observe that from this last result it follows that the Kelly-width of a directed graph is the maximum Kelly-width of all its strongly connected components.

7.3 Algorithmic aspects of Kelly-width

7.3.1 Computing Kelly-decompositions

In this section we mention several algorithms for computing Kelly-width and Kelly-decompositions. The proofs of Theorems 7.9 and 7.11 show that Kelly-decompositions can efficiently be constructed from directed elimination orderings or monotone winning strategies, so we concern ourselves with the problem of finding any of the equivalent characterizations.

In a recent paper Bodlaender et al. [BFK$^+$06] study exact algorithms for computing the (undirected) tree-width of a graph. Their algorithms are based on dynamic programming to compute an elimination ordering of the graph. The algorithms translate easily to directed elimination orderings and can therefore be used to compute Kelly-width, giving us the following theorem:

Theorem 7.17. *The Kelly-width of a graph with n vertices and m edges can be determined in*

- $O\big((n+m)\cdot 2^n\big)$ *time and* $O\big(n\cdot 2^n\big)$ *space, or*
- $O\big((n+m)\cdot 4^n\big)$ *time and* $O(n^2)$ *space.*

Proof. The algorithms we require for these bounds are presented as Algorithm 7.1 and Algorithm 7.2 respectively. Lemmas 7.18 and 7.20 prove that these algorithms are correct, and Lemmas 7.19 and 7.21 establish the running times and space requirements. □

Algorithm 7.1 KELLY-WIDTH-DP($\mathcal{G}$)

```
let KW(∅) = 0
for k = 1 to |V(G)| do
  for each S ∈ [V(G)]^k do
    for each v ∈ S do
      Compute supp_S(v) := N_out(Reach_S(v)) ∪ {v}
    let KW(S) = min_{v∈S} max{KW(S \ {v}), |supp_S(v)|}
return KW(V(G))
```

Lemma 7.18. *For any digraph $\mathcal{G}$,* KELLY-WIDTH-DP($\mathcal{G}$) *outputs the Kelly-width of $\mathcal{G}$.*

Proof. We observe that for a directed elimination ordering $\lhd = (v_1, \ldots, v_n)$, $\text{supp}_{\lhd}(v_i)$ is not dependent on the order of the vertices $\{v_1, \ldots, v_{i-1}\}$. The algorithm uses this observation to reduce the number of possible orderings which need to be considered from $n!$ to 2^n. It is easily seen that $|\text{supp}_S(v)|$ is v together with the support set of v in any directed elimination ordering where v is preceded by some ordering of the remaining elements of S. Thus $\max\{KW(S \setminus \{v\}), |\text{supp}_S(v)|\}$ is one more than the minimal width of a partial directed elimination ordering on S where v is the last vertex eliminated. It follows that $KW(S)$ returns one more than the minimal width of a partial directed elimination ordering on S, and thus $KW(V(\mathcal{G}))$ returns the Kelly-width of $\mathcal{G}$. □

Lemma 7.19. *Let $\mathcal{G}$ be a digraph with n vertices and m edges.* KELLY-WIDTH-DP($\mathcal{G}$) *requires at most $O\big((n+m)\cdot 2^n\big)$ time and $O\big(n\cdot 2^n\big)$ space.*

Proof. For a set $S \subseteq V(\mathcal{G})$ and a vertex $v \in V(\mathcal{G})$, it is readily seen that $\mathit{Reach}_S(v)$ can be computed with a depth-first search from v. Since such a search can be executed in time $O(n+m)$ [CLR96], it follows that $\mathrm{supp}_S(v)$ can be computed in time $O(n+m)$. The innermost **for** loop is executed once for each $S \subseteq V(\mathcal{G})$, and loops $|S|$ times. So if each value for $KW(S)$ is stored as it is computed so that its value can be found in constant time, the total running time for the algorithm is $O(n+m)\sum_{S\subseteq V(\mathcal{G})} O(|S|) = O\big((n+m)\cdot 2^n\big)$. □

Algorithm 7.2 KELLY-WIDTH-REC$(\mathcal{G}, L, S)$

if $S = \{v\}$ for some v **then**
  return $\mathrm{supp}_L(v)$
let $\mathsf{Opt} = \infty$
for each $S' \subseteq S$ with $|S'| = \lfloor |S|/2 \rfloor$ **do**
  Compute $w_1 = $ KELLY-WIDTH-REC$(\mathcal{G}, L, S')$
  Compute $w_2 = $ KELLY-WIDTH-REC$(\mathcal{G}, L \cup S', S \setminus S')$
  let $\mathsf{Opt} = \min\big\{\mathsf{Opt}, \max\{w_1, w_2\}\big\}$
return Opt

Lemma 7.20. *For any digraph $\mathcal{G}$,* KELLY-WIDTH-REC$(\mathcal{G}, \emptyset, V(\mathcal{G}))$ *outputs the Kelly-width of $\mathcal{G}$.*

Proof. We prove by induction on $|S|$ that KELLY-WIDTH-REC$(\mathcal{G}, L, S)$ returns one more than the minimal width of a partial directed elimination ordering on $L \cup S$ where the first $|L|$ vertices are elements of L. From our observations regarding $\mathrm{supp}_L(v)$ in the proof of Lemma 7.18, we see this is true for $|S| = 1$. Now suppose it is true for $|S| \leq s$, we show that it is true for all S with $|S| \leq 2s$. Consider a single execution of the **for** loop. Since $|S'| = \lfloor |S|/2 \rfloor$, it follows by the induction hypothesis that w_1 is one more than the minimal width of a partial directed elimination ordering on $L \cup S'$ where the first $|L|$ elements are from L and w_2 is one more than the minimal width of a partial directed elimination ordering on $L \cup S$ where the first $|L| + |S'|$ elements are from $L \cup S'$. Thus, the maximum of w_1 and w_2 is one more than the minimal width of a partial directed elimination ordering on $L \cup S$ where the first $|L|$ elements are from L, and the next $|S'|$ elements are from S'. Opt stores the minimum of all these maxima, over all subsets S' with $|S'| = \lfloor |S|/2 \rfloor$. As the minimal width of a partial directed elimination ordering of $L \cup S$ where the first $|L|$ elements are from L must be the minimal width of a partial directed elimination ordering of $L \cup S$ where the first $|L|$ elements are from L and the next $\lfloor |S|/2 \rfloor$ elements are from S' for some $S' \subseteq S$, it follows that Opt stores the required value. Thus KELLY-WIDTH-REC$(\mathcal{G}, \emptyset, V(\mathcal{G}))$ returns the Kelly-width of $\mathcal{G}$. □

Lemma 7.21. *Let $\mathcal{G}$ be a directed graph with n vertices and m edges. Then* KELLY-WIDTH-REC$(\mathcal{G}, \emptyset, V(\mathcal{G}))$ *runs in $O\big((n+m)\cdot 4^n\big)$ time and $O(n^2)$ space.*

Proof. Let $T(s)$ be the time required to compute KELLY-WIDTH-REC$(\mathcal{G}, L, S)$ when $|S| = s$. We prove by induction on s that $T(s) = O\big((n+m)\cdot 4^s\big)$ time. If $s = 1$, as we saw in Lemma 7.21, $\mathrm{supp}_L(v)$ can be computed in $O(n+m)$ time, so the assumption holds for this case. For $s > 1$, the algorithm runs in time $2\binom{s}{s/2}T(s/2)$. Using asymptotic approximations of Catalan numbers [GKP98], $\binom{2n}{n} \in O(4^n)$, so $T(s) = O(4^{s/2})T(s/2) = O\big((n+m)\cdot 4^s\big)$. The space requirement follows from the observation that at each stage of the recursion we need

$O(n)$ space to store the current subset S' of S and the values we have computed. Since the recursion tree has maximum height $\lceil \log |S| \rceil \leq n$, we obtain the space bound of $O(n^2)$. □

For a given k, the problem whether a digraph $\mathcal{G}$ has Kelly-width $\leq k$ is decided in exponential time with the above algorithms. As the minimization problem is NP-complete (it generalizes the NP-complete problem of deciding the tree-width of an undirected graph), we cannot expect polynomial time algorithms to exist. However, the exact complexity of determining if a digraph has Kelly-width $\leq k$ for fixed k is currently unknown. Clearly a digraph has Kelly-width equal to 1 if, and only if, it is acyclic, and recently Meister, Telle and Vatshelle [MTV07] exhibited a polynomial time algorithm for determining if a digraph has Kelly-width 2. So for $k \leq 2$ the problem can be solved in polynomial time. For $k > 2$ it is an open problem.

Open problem 7.22. *For a fixed $k > 2$, what is the complexity of the following problem: Given a digraph $\mathcal{G}$ does $\mathcal{G}$ have Kelly-width $\leq k$?*

It seems plausible that, as in the case of DAG-width, studying strategies in the inert robber game will lead to a polynomial time algorithm when k is fixed.

7.3.2 Algorithms on graphs of small Kelly-width

In this section we present algorithmic applications of Kelly-decompositions, including a general scheme that can be used to construct algorithms based on a decomposition. We assume that a Kelly-decomposition (or even an elimination ordering) has been provided or pre-computed. We give an example algorithm based on this to compute the winner of a parity game, which runs in polynomial time on graphs of bounded Kelly-width. As the algorithm is similar to the algorithm of the previous chapter, we outline the major difference between the two.

Dynamic programming algorithms using Kelly-decompositions follow a pattern similar to algorithms that use tree-decompositions and DAG-decompositions. Starting with a special Kelly-decomposition $(\mathcal{D}, \mathcal{B}, \mathcal{W})$ and then working bottom up to compute for each node $d \in V(\mathcal{D})$ a data set containing information on the set $B_{\geq d} := \bigcup_{d' \geq d} B_d$. The general pattern is described in Algorithm 7.3.

Algorithm 7.3 Dynamic programming using a Kelly-decomposition

Given a special Kelly-decomposition $(\mathcal{D}, \mathcal{B}, \mathcal{W})$:

Leaves: Compute the data set for B_d for all leaves d.

Combine: If $d \in V(\mathcal{D})$ is an inner node with successors $d_1, \ldots, d_m$ ordered by the ordering guaranteed by the Kelly-decomposition (we observe that such an ordering can be computed easily with a greedy algorithm), combine the data sets computed for $B_{\geq d_1}, \ldots, B_{\geq d_m}$ to a data set for the union $\bigcup_{i=1}^{m} B_{\geq d_i}$.

Update: Update the data set computed in the previous step so that the new vertex u with $B_d = \{u\}$ is taken into account. Usually, the vertex u will have been part of at least some guard sets W_{d_i}.

Expand: Finally, expand the data set to include guards in $W_d \setminus \bigcup_i W_{d_i}$ and also paths starting at u.

We illustrate this pattern by briefly presenting an algorithm for computing the winner of a parity game. The full algorithm can be found in [HK07]. The algorithm is similar to the algorithm based on DAG-decompositions, however the separation of guard sets in Kelly-decompositions makes the presentation more straightforward. As with DAG-decompositions, we define a Kelly-decomposition of an arena $\mathcal{A}$ as a Kelly-decomposition of the underlying directed graph $(V(\mathcal{A}), E(\mathcal{A}))$.

Theorem 7.23. *For any k, given a parity game $(\mathcal{A}, \chi)$ and a Kelly-decomposition of $\mathcal{A}$ of width $\leq k$, determining if Player 0 has a winning strategy from $v_I(\mathcal{A})$ can be computed in polynomial time.*

To prove the theorem, we first need some preparation. For the rest of this section fix a parity game $(\mathcal{A}, \chi)$ where $\chi : V(\mathcal{A}) \to \mathsf{P}$. We assume that the maximal out-degree of any vertex in $V(\mathcal{A})$ is 2. Using the inert robber game, it is straightforward to show that the graph resulting from the modification described in Theorem 2.59 has Kelly-width at most one more than the original graph.

We recall from the proof of Theorem 6.34 the definitions of $\text{result}_\sigma(U, v)$ and $\textsc{Result}(U, v)$ for a (not necessarily memoryless) strategy σ for Player 0, a subset of vertices $U \subseteq V(\mathcal{A})$ and a vertex $v \in V(\mathcal{A})$. We show how, for a fixed k and given a special Kelly-decomposition $(\mathcal{D}, \mathcal{B}, \mathcal{W})$ of $\mathcal{A}$ of width k, to compute $\textsc{Result}(B_{\geq d}, v)$ for each $d \in V(\mathcal{D})$ and $v \in B_{\geq d}$ in polynomial time. As with Theorem 6.34 we observe that as $B_{\geq d}$ has at most k guards (W_d), $|\textsc{Result}(B_{\geq d}, v)| \leq (n+1)^k + 2$.

The dynamic programming algorithm can then be presented as follows.

Leaves: It follows with the same argument as the *Leaves* step in the proof of Theorem 6.34, that for any leaf $d \in V(\mathcal{D})$, and vertex $v \in B_d$ the set $\textsc{Result}(B_d, v)$ can be computed in constant time.

Combine: Let d be an inner node of $\mathcal{D}$ with successors $d_1, \ldots, d_m$ ordered according to the ordering guaranteed by (K3). For $1 \leq i \leq m$, let $B_i := \bigcup_{j \leq i} B_{\geq d_i}$ and let $B := B_m = \bigcup_{1 \leq i \leq m} B_{\geq d_i}$. We aim to compute the set $\textsc{Result}(B, u)$ for each $u \in B$. We observe that if $i < j$ and $u \in B_{\geq d_i}$ then every path from u to a vertex $v \in B_{\geq d_j} \setminus B_{\geq d_i}$ must go through W_d. Hence, if $u \in B_{\geq d_i}$ then $\textsc{Result}(B, u) = \textsc{Result}(B_i, u)$. We compute for each $i \leq m$ and $u \in B_i$ the set $\textsc{Result}(B_i, u)$ by induction on i. For $i = 1$, $\textsc{Result}(B_1, u) = \textsc{Result}(d_1, u)$. Let $i > 1$ and let $u \in B_i \setminus B_{i-1}$.

To compute $\textsc{Result}(B_i, u)$, we do the following. Let $r = \text{result}_\sigma(B_{\geq d_i}, u) \in \textsc{Result}(B_{\geq d_i}, u)$ be a set of results against a strategy σ for Player 0. The result set r gives us the set of vertices $v \in W_{d_i}$ to which Player 1 can force the play against σ and also the best priority he can achieve in doing so. Now, if $v \in W_{d_i} \cap B_{i-1}$ is a guard contained in B_{i-1} then once the play has reached v it can never return to $B_i \setminus B_{i-1}$ and continues in B_{i-1} until it reaches a vertex in W_d. Hence, once the play has reached v, we can determine the results of possible strategies in B_{i-1} from $\textsc{Result}(B_{i-1}, v)$.

This suggests the following algorithm for computing $\textsc{Result}(B_i, u)$. For each $r \in \textsc{Result}(B_{\geq d_i}, u)$ we compute a set R_r of sets as follows. Let $R := \{(w, p) \in r : w \in W_{d_i} \setminus W_d\}$ be the set of outcomes in r for plays which end in vertices in B_{i-1}. Let $(w_1, p_1), \ldots, (w_s, p_s)$ list the elements of R. For each tuple $\rho = (r_1, \ldots, r_s)$ with $r_j \in \textsc{Result}(B_{i-1}, w_j)$ Let R_ρ be defined as follows. For each $(v, p) \in r \setminus R$ add (v, p)

to R_ρ. If $(v, q) \in r_j$ add $(v, \max\{p_j, q\})$ to R_ρ. Then, add the set R_ρ to R_r. Then, RESULT(B_i, u) contains for each $R_\rho \in R_r$ the set of $\trianglelefteq$-minimal pairs in R_ρ.

Update and Expand: We now consider how to update the data structure to take account of paths that include vertices entering $B_{\geq d}$. The argument is similar to the *Expand* step of the proof of Theorem 6.34, so we refer the reader there for the details.

We observe that each step of the above algorithm, and hence the entire algorithm, runs in polynomial time. This completes the outline of the proof of Theorem 7.23.

7.3.3 Asymmetric matrix factorization

We saw in Section 7.1.2 that the idea of vertex elimination was motivated by the practical application of solving systems of linear equations. Such systems are more commonly represented as matrix equations: $M\mathbf{x} = \mathbf{b}$, with the goal being to find a solution for the $n \times 1$ vector of variables, $\mathbf{x}$, given an $m \times n$ matrix M, and an $m \times 1$ vector $\mathbf{b}$. A straightforward solution to such an equation is to find M^{-1}, the inverse of M, to obtain $\mathbf{x} = M^{-1}\mathbf{b}$, however a more common approach is to factorize M in such a way that solutions may be easily computed. Cholesky decompositions and LU-factorizations are two such examples of this. If M is an $m \times n$ matrix, an *LU-factorization* (or *LU-decomposition*) of M is an $m \times m$ lower triangular matrix L and an $m \times n$ upper triangular matrix U such that $M = LU$. If, in addition M is symmetric and positive definite, then there is an LU-factorization of M where $U = L^T$. Such a decomposition is called a *Cholesky decomposition*. When a matrix has an LU-factorization we can solve the equation $M\mathbf{x} = \mathbf{b}$ as follows: first we use forward substitution to solve $L\mathbf{y} = \mathbf{b}$, and then backward substitution to solve $U\mathbf{x} = \mathbf{y}$.

The elimination process we described in Section 7.1.2, also known as Gaussian elimination, is one of the most common methods for computing an LU-factorization or a Cholesky decomposition. More precisely, Gaussian elimination is the process of transforming a matrix into an upper triangular matrix via row operations: adding a multiple of one row to another (including itself), or interchanging two rows (also known as *pivoting*). The resulting upper triangular matrix is the U factor of a LU-factorization, and the row operations can be represented by a sequence of transformation matrices, the product of which form the L factor of the LU-factorization. If the original matrix was symmetric and positive definite, this process will generate a Cholesky decomposition.

Since Gaussian elimination can be used to compute LU-factorizations and Cholesky decompositions, it is not surprising that elimination orderings and two associated structures we introduce here, elimination trees and elimination DAGs, are useful for investigating the complexity of computing these matrix decompositions. We first define the particular relationship between graphs and matrices that we are interested in.

Definition 7.24. Let $M = (a_{ij})$ be a square $n \times n$ matrix. We define $\mathcal{G}_M$ as the directed graph with $V(\mathcal{G}_M) = \{v_1, \ldots, v_n\}$, and for $i \neq j$, $(v_i, v_j) \in E(\mathcal{G}_M)$ if, and only if, $a_{ij} \neq 0$. We also define the elimination ordering $\lhd_M$ as $\lhd_M := (v_1, \ldots, v_n)$.

When M is a symmetric matrix, we view $\mathcal{G}_M$ as an undirected graph rather than a bidirected graph.

One structure that is particularly useful for analysing symmetric matrix factorization is the *elimination tree*.

Definition 7.25 (Elimination tree)**.** Let $\mathcal{G}$ be an undirected graph, and $\lhd$ an elimination ordering for $\mathcal{G}$. The *elimination tree* defined by $\lhd$ is a pair $(\mathcal{T}, \lambda)$ where $\mathcal{T}$ is a rooted tree and $\lambda : V(\mathcal{T}) \to V(\mathcal{G})$ is a bijection such that if $s \in V(\mathcal{T})$ is the parent of $t \in V(\mathcal{T})$, then $\lambda(s) = \min_{\lhd}\big(\text{supp}_{\lhd}(\lambda(t))\big)$.

Liu [Liu90] observed that elimination trees can be used to investigate many aspects of Cholesky decompositions, for example the row and column structure of the Cholesky factors can be extracted directly from an elimination tree. Another observation, from Bodlaender et al. [BGHK95], is that the height of an elimination tree gives the parallel time required to compute a Cholesky decomposition of a symmetric matrix using Gaussian elimination.

In [GL93], Gilbert and Liu introduced a generalization of elimination trees, called elimination DAGs, which can be similarly used to analyse factorizations in the asymmetric case. We recall that a *transitive reduction* of a directed graph is a minimal graph with the same transitive closure and we observe that an acyclic graph has a unique transitive closure.

Definition 7.26 (Upper and Lower elimination DAGs [GL93])**.** Let M be a square matrix that can be decomposed as $M = LU$ without pivoting. The *upper (lower) elimination DAG* is the transitive reduction of the directed graph $\mathcal{G}_U$ ($\mathcal{G}_L$ respectively).

Gilbert and Liu [GL93] observed that elimination DAGs enjoy many properties similar to elimination trees. For instance, they are an efficient storage scheme for sparse matrices, and an upper and lower pair of elimination DAGs are sufficient to capture the path structure of a graph: if there is a directed path from u to v in the graph, then there is a vertex w such that there is a path from u to w in the upper elimination DAG, and a path from w to v in the lower elimination DAG. They also showed that when the matrix is symmetric, the upper elimination DAG is isomorphic to the elimination tree, as is the lower elimination DAG when its edges are reversed.

The Kelly-decomposition constructed in the proof of Theorem 7.11 captures the upper and lower elimination DAGs in a very direct manner.

Theorem 7.27. *Let M be a square matrix that can be decomposed as $M = LU$ without pivoting. Let $(\mathcal{D}, \mathcal{B}, \mathcal{W})$ be the Kelly-decomposition of $\mathcal{G}_M$ obtained by applying the proof of Theorem 7.11 with elimination order $\lhd_M$. Then*

(a) $(\mathcal{D}, \mathcal{B})$ is isomorphic to the lower elimination DAG, and

(b) $\mathcal{G}_U = (V(\mathcal{G}_M), \{(v, w) : w \in W_v\})$, thus the upper elimination DAG is isomorphic to the transitive reduction of the relation $\{(v, w) : w \in W_v\}$.

Proof. For $v \in V(\mathcal{G}_M)$, let $X_v = \{v\} \cup \{w \in V(\mathcal{G}_M) : w \lhd_M v\}$. First, from Theorem 1 of [RT78]:

$$(E(\mathcal{G}_L))^{TC} = \{(v, w) : w \lhd_M v, \text{ and there is a path from } v \text{ to } w \text{ in } \mathcal{G}_M[X_v]\},$$

where R^{TC} denotes the transitive closure of R. We observe that in the construction of the Kelly-decomposition, $E(\mathcal{D})$ is the transitive reduction of the right-hand side. Since, by construction, elements of $\mathcal{B}$ are singletons, we can view $\mathcal{B}$ as a bijection between $V(\mathcal{D})$ and $V(\mathcal{G})$, and the first result follows. Secondly, from Theorem 4.6 of [GL93], we have

$$E(\mathcal{G}_U) = \{(v, w) : v \lhd_M w, \text{ and there is a } v' \in \textit{Reach}_{X_v}(v) \text{ with } (v', w) \in E(\mathcal{G}_M)\}.$$

The second result then follows from Lemma 7.5, which shows that $\{(v,w) : w \in W_v\} = \{(v,w) : w \in \text{supp}_{\lhd_M}(v)\}$ is equivalent to the right-hand side. □

We can use the results of [GL93] to make the following observation when we construct Kelly-decompositions on undirected graphs.

Corollary 7.28. *Let $\mathcal{G}$ be an undirected graph, $\lhd$ an elimination order on $\mathcal{G}$ and $(\mathcal{D}, \mathcal{B}, \mathcal{W})$ the Kelly-decomposition of $\mathcal{G}$ (considered as a bidirected graph) obtained by applying the proof of Theorem 7.11 with elimination order $\lhd$. Then $\mathcal{D}$ is a tree, and more precisely, $(\mathcal{D}^{op}, \mathcal{B})$ is isomorphic to the elimination tree associated with the (undirected) elimination order $\lhd$.*

7.4 Comparing Kelly-width and DAG-width

In this section we use graph searching games to compare Kelly-width to DAG-width and directed tree-width. In the undirected case, all the games we consider require the same number of searchers, however we show that in the directed case there are graphs on which all three measures differ by an arbitrary amount. We show that Kelly-width bounds directed tree-width within a constant factor, but the converse fails as there are classes of graphs of bounded directed tree-width and unbounded Kelly-width. We also provide evidence to suggest that Kelly-width and DAG-width are within a constant factor of each other.

We recall from Definition 6.1 the cops and robber game used to characterize DAG-width. For convenience, we will refer to this as the visible robber game. In Example 5.2.1 we discussed another cops and robber game that partially characterizes directed tree-width: the strongly connected visible robber game. The following theorem summarizes Theorems 6.15 and Lemma 5.41:

Theorem 7.29. *Let $\mathcal{G}$ be a digraph.*

1. *$\mathcal{G}$ has DAG-width k if, and only if, k cops have a monotone winning strategy in the visible robber game on $\mathcal{G}$.*
2. *$\mathcal{G}$ has directed tree-width $\leq 3k+1$ or k cops do not have a winning strategy in the strongly connected visible robber game on $\mathcal{G}$.*

For the undirected case, the following proposition sums up results from [DKT97] and [ST93].

Proposition 7.30. *On any undirected graph $\mathcal{G}$, the following are equivalent*

1. *k cops have a winning strategy in the visible robber game.*
2. *k cops have a robber-monotone and cop-monotone winning strategy in the visible robber game.*
3. *k cops have a winning strategy in the inert robber game.*
4. *k cops have a robber monotone winning strategy in the inert robber game.*
5. *The tree-width of $\mathcal{G}$ is at most $k-1$.*

It follows from these results that Kelly-width is a generalization of tree-width in the following sense.

Corollary 7.31. *Let $\mathcal{G}$ be an undirected graph. $\mathcal{G}$ has tree-width k if, and only if, $\overleftrightarrow{\mathcal{G}}$ has Kelly-width k.*

On general directed graphs, the situation is more complicated. As we saw in Theorem 6.11, monotonicity is not sufficient for the visible robber game. Kreutzer and Ordyniak [KO07] have also recently shown that monotonicity is not sufficient for the inert robber game.

Theorem 7.32 ([KO07])**.** *For any $m \in \mathbb{N}$, there exists a graph for which $6m$ cops can capture an invisible, inert robber but $7m$ cops are required to do so with a robber-monotone strategy.*

Of course, as with Theorem 6.11, this does not preclude the possibility that the number of cops required for monotonicity is bounded by some factor of the number of cops required with any strategy.

Open problem 7.33. *Does there exist a function $f : \omega \to \omega$ such that for all digraphs $\mathcal{G}$, if k cops can capture an inert robber on $\mathcal{G}$ then $f(k)$ cops can capture the robber with a robber-monotone strategy?*

Before we compare Kelly-width with directed tree-width and DAG-width, we first observe that Proposition 7.2 allows us to compare Kelly-width and directed path-width. As we mentioned previously, Barát [Bar05] observed that the directed path-width of a digraph was one less than the minimum number of cops required to capture an invisible robber with a cop-monotone strategy. Thus, using the observation that a cop-monotone strategy in the cops and inert robber game is also robber-monotone, and the example from Proposition 6.39, we obtain the following relationship between Kelly-width and directed path-width.

Proposition 7.34.

(i) If a directed graph $\mathcal{G}$ has directed path-width k, it has Kelly-width at most $k + 1$.

(ii) There exists a family of graphs with arbitrarily large directed path-width and Kelly-width 2.

Our next comparison result shows that a robber-monotone winning strategy in the inert robber game can be translated to a (not necessarily monotone) winning strategy in the visible robber game.

Theorem 7.35. *Let $\mathcal{G}$ be a directed graph. If k cops can catch an inert robber with a robber-monotone strategy on $\mathcal{G}$, then $2k - 1$ cops can catch a visible robber on $\mathcal{G}$.*

Proof. Suppose k cops have a robber-monotone winning strategy in the inert robber game on a digraph $\mathcal{G}$. By Theorem 7.9 this implies that there is a directed elimination ordering $\lhd$ on $\mathcal{G}$ of width $\leq k - 1$. We use the elimination ordering to describe the winning strategy of $2k - 1$ cops against a visible robber, thereby establishing the result.

The cops are split into two groups, k cops called the *blockers* and $k - 1$ cops called the *chasers*. Similarly, the cop moves are split in two phases, a blocking move and a chasing phase.

In the first move, k cops are placed on the k highest elements with respect to $\lhd$. These cops form the set of blockers. Let the robber choose some element v. This concludes the first (blocking) move. We observe:

> If u is the $\lhd$-smallest vertex occupied by a blocker, then every directed path from v to a vertex greater than u has at least one vertex occupied by a cop. $(*)$

This invariant is maintained by the blocking cops during the play. Now suppose after r rounds have been played, the robber occupies vertex v and the blockers occupy vertices in X so that the invariant $(*)$ is preserved. Let u be the $\lhd$-smallest element in X and let $C_1, \ldots, C_s$ be the set of strongly connected components of $\mathcal{G}[\{u' : u' \lhd u\}]$. Further, let $\sqsubset$ be a linear ordering on $\mathcal{C} := \{C_1, \ldots, C_s\}$ so that $C_i \sqsubset C_j$ if, and only if, the $\lhd$-maximal element in C_i is $\lhd$-smaller than the $\lhd$-maximal element of C_j. Now the cops move as follows. Let $C \in \mathcal{C}$ be the component such that $v \in C$ and let $w \in C$ be the $\lhd$-maximal element in C. The cops place the $k-1$ cops not currently on the graph on $\mathrm{supp}_{\lhd}(w)$. These cops are the chasers. As the chasers approach, the robber has two options. Either he stays within C or he escapes to a vertex in a different strongly connected component C'. If the robber runs to a vertex $x \in C$ or $x \in C'$ for some $C' \sqsubset C$ then after the chasers land on $S := \mathrm{supp}_{\lhd}(w)$ there is no path from x to a node u such that $u \rhd u'$ for the $\lhd$-minimal vertex u' in S. Hence, the chasers become blockers and the chasing phase is completed. Otherwise, if the robber escapes to a C' with $C \sqsubset C'$, then the chasers repeat the procedure and move to $\mathrm{supp}_{\lhd}(w')$ for the $\lhd$-maximal element in C'. However, as the robber always escapes to a $\sqsubset$-larger strongly connected component and also can not bypass the blockers, this chasing phase must end after finitely many steps with the robber being on a vertex $v \in C$ for some component C and the chasers being on $\mathrm{supp}_{\lhd}(w)$ for the $\lhd$-maximal element in C. At this point the chasers become blockers. One of the old blockers is now placed on w and all others are removed from the board. The cop on w makes sure that in each such step the robber space shrinks by at least one vertex. By construction, the invariant in $(*)$ is maintained. Further, as the robber space shrinks by at least one after every chasing-phase, the robber is eventually caught by the cops. □

An immediate consequence of this is that the Kelly-width of a graph bounds the directed tree-width of the graph.

Corollary 7.36. *Let $\mathcal{G}$ be a directed graph with Kelly-width k. Then $\mathcal{G}$ has directed tree-width $\leq 6k-2$.*

Since it is not known whether the number of cops required for a winning strategy in the visible robber game bounds the number of cops required for a monotone winning strategy, we cannot obtain a similar bound for DAG-width. We can, however, ask whether we can improve the bound. That is, assuming that k cops have a robber-monotone winning strategy against an invisible, inert robber can we define a winning strategy for less than $2k-1$ cops in the visible robber game? Although it might be possible to improve the result, the next theorem shows that we cannot do better than with $\frac{4}{3}k$ cops.

Theorem 7.37. *For every $m \in \mathbb{N}$, there is a graph such that $3m$ cops have a robber-monotone winning strategy in the inert robber game but no fewer than $4m$ cops can catch a mobile visible robber.*

Proof. Consider the graph $\mathcal{G}$ in Figure 7.1. We show that on $\mathcal{G}$, 3 cops do not have a (non-monotone winning) strategy to catch a visible robber, however 4 cops do. Consider the partition of $V(\mathcal{G})$, $\mathcal{H} = \big\{\{v_1, v_2, v_4\}, \{v_3\}, \{v_5\}, \{v_6\}\big\}$. The strategy for the robber against 3 cops is to move to any element of $\mathcal{H}$ which is not occupied by a cop. As long as the robber moves to one of $\{v_1, v_4\}$ when the cops occupy $\{v_3, v_5, v_6\}$, it will always be possible for him to move to such an element when the cops move. However 4 cops can capture a visible robber with a monotone strategy by occupying the following sequence of sets of vertices: $\{v_3, v_4, v_5, v_6\}, \{v_2, v_3, v_5, v_6\}, \{v_1, v_2, v_3\}$.

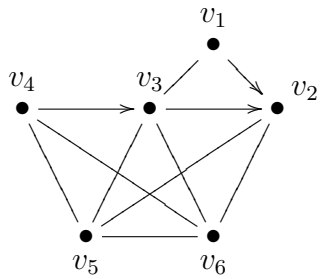


Figure 7.1: Graph $\mathcal{G}$ showing the difference between DAG-width and inert robber game

On the other hand, 3 cops suffice to capture an invisible, inert robber with a robber-monotone strategy by occupying the following sequence of sets of vertices: $\{v_4, v_5, v_6\}$, $\{v_3, v_5, v_6\}$, $\{v_2, v_5, v_6\}$, $\{v_2, v_3\}$, $\{v_1, v_2, v_3\}$. The result follows by taking the lexicographic product of this graph with the complete graph on m vertices. □

Since 4 cops can capture a visible robber with a monotone strategy on the graph in the previous proof, we have the following:

Corollary 7.38. *For all $m \in \mathbb{N}$ there are graphs of DAG-width $4m$ and Kelly-width $3m$.*

Despite this $\frac{4}{3}$ bound, for graphs of small Kelly-width we can do better.

Theorem 7.39. *For $k = 1$ or 2, if $\mathcal{G}$ has Kelly-width k, $\mathcal{G}$ has DAG-width k.*

Proof. If $\mathcal{G}$ has Kelly-width 1, then it must be acyclic, as all guard sets are empty. Thus it has DAG-width 1. If $\mathcal{G}$ has Kelly-width 2, then it has an elimination ordering $\lhd = (v_1, v_2, \ldots, v_n)$ of width 1. A cop-monotone strategy for two cops against a visible robber is as follows. Initially, let $i = n$ and place one cop on v_i. At this point, the robber is restricted to $\{v_1, \ldots, v_{i-1}\}$. Let $j < i$ be the maximal index such that the robber can reach v_j. Place a cop on v_j. After the cop has landed, we claim that the robber is unable to reach both v_i and v_j. For otherwise, let r be the maximal index such that the robber can reach v_r (with cops on v_i and v_j) and from v_r can reach v_i (with a cop on v_j) and v_j (with a cop on v_i). By the maximality of j, $r < j$. Let $s > r$ be the first index greater than r which occurs on a path from v_r to v_i that does not go through v_j, and $t > r$ be the first index greater than r which occurs on a path from v_r to v_j that does not go through v_i. Then from the maximality of r, $s \neq t$. Furthermore, $\{v_s, v_t\} \subseteq \text{supp}_{\lhd}(r)$, so $|\text{supp}_{\lhd}(v_r)| > 1$, contradicting the width of the ordering. So we can remove the cop from whichever vertex the robber can no longer reach without changing the robber space, and either the robber is now restricted to $\{v_1, \ldots, v_j\}$ or the maximal index which the robber can reach is smaller. Clearly, this is a monotone winning strategy for two cops. □

We now turn to the converse problem, what can be said about the Kelly-width of graphs given their directed tree-width or DAG-width?

Firstly we observe the following analogue of Proposition 6.7 for Kelly-width.

Proposition 7.40. *For any j, k with $2 \leq j < k$, there exists a graph $\mathcal{T}_k^j$ such that Kelly-width$(\mathcal{T}_k^j) = j$ and Kelly-width$((\mathcal{T}_k^j)^{op}) = k$.*

Proof. Consider the graph $\mathcal{T}_k^j$ from Proposition 6.7. In the proof of Proposition 6.7, the strategies described for the cops and the robber are also winning strategies in the inert robber game.[2] □

It follows, using the same argument of Proposition 6.37 that there are families of graphs of bounded directed tree-width and unbounded Kelly-width.

Corollary 7.41. *There exist families of digraphs with directed tree-width* 2 *and unbounded Kelly-width.*

Our final result is a step towards relating Kelly-width to DAG-width by showing how to translate a monotone strategy in the visible robber game to a (not necessarily monotone) strategy in the inert robber game.

Theorem 7.42. *If* $\mathcal{G}$ *has DAG-width* $\leq k$*, then* k *cops have a winning strategy in the inert robber game.*

Proof. Given a DAG-decomposition $(\mathcal{D}, \mathcal{X})$ of $\mathcal{G}$ of width k, the strategy for k cops against an invisible, inert robber is to follow a depth-first search on the decomposition. More precisely, we assume the decomposition has a single root r, and we have an empty stack of nodes of $\mathcal{D}$.

1. Initially, place the cops on X_r and push r onto the stack.
2. At this point we assume d is on the top of the stack and the cops are on X_d. We next "process" the successors of d in turn. To process a successor d' of d, we remove all cops not on $X_d \cap X_{d'}$, place cops on $X_{d'}$, push d' onto the stack, and return to step 2. Note that a node may be processed more than once.
3. Once all the successors of a node have been processed, we pop the node off the stack and if the stack is non-empty, return to step 2.

Because the depth-first search covers all nodes of the DAG and hence all vertices of the graph are eventually occupied by a cop, the robber will be forced to move at some point. Due to the guarding condition for DAG-decompositions, when the robber is forced to move this strategy will always force the robber into a smaller region and eventually capture him. □

Again we observe that it is unknown if, in the inert robber game, the number of cops required to capture the robber with a robber-monotone strategy is bounded by the number of cops required to capture him with any strategy. So this result does not allow us to directly compare Kelly-width and DAG-width. However, we strongly believe that the number of cops required for monotone strategies is bounded in both the inert robber game and the visible robber game, giving us the following conjecture:

Conjecture 7.43. *The Kelly-width and DAG-width of a digraph lie within constant factors of one another.*

[2] Indeed, the winning strategy for the robber is winning even if the robber is visible and inert.

Chapter 8

Havens, Brambles and Minors for Directed Connectivity

In this chapter we present some preliminary work towards a structure theory for directed graphs based on directed connectivity. The aim of such a structure theory would be to obtain generalizations of some of the significant results for undirected graphs, for example finding a directed analogue of the Graph Minor Theorem. However, as we show, even determining some of the basic building blocks of such a structure theory leads to some interesting open problems. We work on the assumption that DAG-width, Kelly-width and the non-monotone versions of their cops and robber games are all approximately the same and can therefore be used to measure the directed connectivity of a digraph. Then, using the premise that DAG-width or Kelly-width measures the complexity of a graph, we consider the following two questions: What structural features are present in directed graphs which are "complex"?; and what relation on directed graphs indicates "structural simplification"?

As we observed with Theorem 4.7 the existence of a bramble or a haven in an undirected graph indicates that the tree-width is not going to be small. Similarly, Theorems 4.7 and 4.11 show that the existence of the natural generalizations of havens and brambles imply that the directed tree-width is not going to be small. So in order to address the first question, we consider generalizations of havens and brambles which correspond to DAG-width and Kelly-width. Although we are unable to show full equivalence as with Theorems 4.7 and 4.11, we can show, via cops and robber games, that they do provide obstructions for DAG-width and Kelly-width. That is, their existence in a graph places restrictions on the DAG-width or Kelly-width of that graph.

Towards finding a relation which indicates structural simplification, we consider the problem of extending the minor relation to directed graphs. As we mentioned in Chapter 4, the minor relation is an important relation in the structural theory of undirected graphs as it indicates whether one graph is structurally more simple than another. So having a minor relation for directed graphs is the cornerstone of any digraph structure theory. We argue that the existing definitions in the literature of minors for directed graphs are not sufficient, in the sense that a structure theory based on them would not be able to produce similar results to those of undirected graph structure theory. While it may be the case that there is no appropriate relation for directed graphs, we provide some examples which may take the investigation further.

8.1 Havens and brambles

The aim of this section is to define various structural properties which may lead to a minimax theorem for DAG-width and Kelly-width, similar to Theorem 4.7. To achieve this, we introduce some generalizations of havens and brambles and show how they relate to DAG-width and Kelly-width. We recall from Chapter 4, the definitions and theorem that we wish to generalize:

Definition 4.5 (Haven)**.** Let $\mathcal{G}$ be an undirected graph and $k \in \mathbb{N}$. A *haven of order* k in $\mathcal{G}$ is a function $\beta : [V(\mathcal{G})]^{<k} \to \mathcal{P}(V(\mathcal{G}))$ such that for all $X \subseteq V(\mathcal{G})$ with $|X| < k$:

(H1) $\beta(X)$ is a non-empty connected component of $\mathcal{G} \setminus X$, and

(H2) If $Y \subseteq X$, then $\beta(Y) \supseteq \beta(X)$.

Definition 4.6 (Bramble)**.** Let $\mathcal{G}$ be an undirected graph. A *bramble* in $\mathcal{G}$ is a set $\mathcal{B}$ of connected subsets of $V(\mathcal{G})$ such that for all pairs $B, B' \in \mathcal{B}$ either $B \cap B' \neq \emptyset$, or there exists $\{u, v\} \in E(\mathcal{G})$ with $u \in B$ and $v \in B'$. The *width* of a bramble $\mathcal{B}$ is the minimum size of a set which has a non-empty intersection with every element of $\mathcal{B}$.

Theorem 4.7 ([ST93])**.** *Let $\mathcal{G}$ be an undirected graph. The following are equivalent:*

1. *$\mathcal{G}$ has tree-width $\geq k - 1$*
2. *$\mathcal{G}$ has a haven of order k.*
3. *$\mathcal{G}$ has a bramble of width k.*

We saw with Theorems 4.11 and 4.12, that the natural extension of these definitions to directed graphs – replacing "connected components" with "strongly connected components" – results in structural properties closely related to directed tree-width. In this section we introduce some less obvious extensions that are closer to DAG-width and Kelly-width. One of the major obstacles to finding such definitions, and the reason why the extensions we consider are less obvious is that the definitions have to be dependent on edge direction. That is, a bramble or haven of a graph should not necessarily be a bramble or haven of the graph obtained by reversing the direction of the edges. The above definitions of haven and bramble do not have obvious extensions which satisfy this property, however the definitions we introduce next are dependent on edge direction.

Definition 8.1 (D-Haven)**.** Let $\mathcal{G}$ be a directed graph and $k \in \mathbb{N}$. A *D-haven of order* k in $\mathcal{G}$ is a function $\beta : [V(\mathcal{G})]^{<k} \to \mathcal{P}(V(\mathcal{G}))$ such that for all $X \subseteq V(\mathcal{G})$ with $|X| < k$:

(DH1) $\beta(X)$ is a non-empty subset of $V(\mathcal{G} \setminus X)$, and

(DH2) If $Y \subseteq X$ then $\beta(Y) \supseteq \beta(X)$ and $\forall y \in \beta(Y)$, $\beta(X) \cap \mathit{Reach}_{\beta(Y)}(y) \neq \emptyset$.

As suggested by the nomenclature, and as we observed in Chapter 5, on undirected graphs havens describe winning strategies for the robber in the cops and visible robber game. That is, when the cops are on X, $\beta(X)$ suggests the locations the robber should occupy to defeat the cops. The analogous result for the game on directed graphs suggests that D-havens are the "correct" extension of havens for DAG-width. More precisely,

Proposition 8.2. *Let $\mathcal{G}$ be a directed graph. The robber can defeat k cops in the visible cops and robber game on $\mathcal{G}$ if, and only if, $\mathcal{G}$ has a D-haven of order $k + 1$.*

Proof. If $\mathcal{G}$ has a D-haven β of order $k+1$, then the strategy for the robber is to remain in $\beta(X)$ whenever the cops are on X. The D-haven axioms guarantee that this is always possible. More precisely, we define the following strategy for the robber: $\rho(X, X', R) = \mathit{Reach}_{\mathcal{G}\setminus X'}(r')$ for some $r' \in \beta(X') \cap \mathit{Reach}_{\mathcal{G}\setminus(X\cap X')}(r)$. We observe that at every position (X, r), $r \in \beta(X)$ and show that this implies that such a choice is always possible. Since $X \supseteq X' \cap X$, it follows from (DH2) that $r \in \beta(X \cap X')$. Then, since $X' \supseteq X \cap X'$, $\beta(X') \cap \mathit{Reach}_{\beta(X\cap X')}(r) \neq \emptyset$, so ρ is well defined. Finally, since $\rho(X, X', r) \in \mathit{Reach}_{\mathcal{G}\setminus(X\cap X')}(r)$ by definition, ρ is a valid strategy for the robber in the cops and visible robber game.

For the converse, suppose the robber has a winning strategy, ρ, against k cops. Define, for $X \in [V(\mathcal{G})]^{\leq k}$,

$$\beta(X) := \bigcup \{R : \text{ the robber wins from } (X, R) \text{ playing } \rho\}.$$

We show that β is a D-haven of order $k+1$. We observe that $\rho(\emptyset, X, V(\mathcal{G})) \subseteq \beta(X)$, so as ρ is a winning strategy, $\beta(X)$ is non-empty for all $X \in [V(\mathcal{G})]^{\leq k}$. Thus (DH1) holds. For (DH2) we observe from the definition of the cops and visible robber game that if the robber can win from (X, R) then he can win from (Y, R) for all $Y \subseteq X$. Thus, if $Y \subseteq X$, then $\beta(Y) \supseteq \beta(X)$. □

Immediately from this result and Lemmas 6.18, 6.20 and 6.21,we observe that D-havens behave as we expect under subgraphs, lexicographic product, and directed union. Also as a consequence of Proposition 8.2, the existence of a D-haven in a digraph imposes a restriction on the DAG-width of the graph.

Corollary 8.3. *Let $\mathcal{G}$ be a digraph. If $\mathcal{G}$ has a D-haven of width k then the DAG-width of $\mathcal{G}$ is at least k.*

Since a D-haven corresponds to a winning strategy for the robber against any cop strategy and DAG-width corresponds to monotone winning strategies, the converse of Corollary 8.3 is equivalent to the monotonicity question for the cops and visible robber game: if k cops have a winning strategy, do k cops have a monotone winning strategy? As there are graphs where more cops are required to capture the robber with a monotone strategy [KO07], we know that this does not hold in general. However, a result similar to Theorem 4.11 would provide a solution to the more general monotonicity problem posed in Open Problem 6.12.

Obdržálek [Obd06] observed that the relaxation of connected components in (H1) to subsets in (DH1) is necessary if we require havens to correspond to strategies for the robber. More precisely, let us say that a D-haven, β, is *connected* if it also satisfies:

(DH1′) $\beta(X)$ is a non-empty weakly connected component of $\mathcal{G} \setminus X$.

Proposition 8.4 ([Obd06])**.** *There exists a directed graph $\mathcal{G}$ such that the robber can defeat 2 cops in the cops and visible robber game on $\mathcal{G}$, but there is no connected D-haven of order 2 in $\mathcal{G}$.*

The graph that illustrates this result is shown in Figure 8.1. It is difficult to define a notion of haven that corresponds to the inert robber game for two reasons. First, because the motility of the robber is dependent on the move of the cops, there may be a number of "responses" to a given cop position in this game. So having a function defined only for sets of cop locations is not going to be sufficient. Secondly, as we observed in Chapters 5 and 7, the cops and robber game with an invisible robber is essentially a single player game. Thus there is only one strategy for

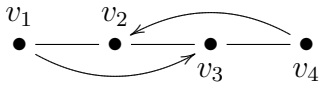


Figure 8.1: Graph to show that D-havens may be disconnected

the robber and it is either winning or it is not. So having a function which dynamically suggests a strategy for the robber is not going to be particularly interesting. A more practical approach would be to identify the structural features which are present when the strategy for the robber is winning. This leads us to the problem of extending the definition of brambles.

Before we introduce the extension of brambles we are interested in, we need to introduce the concept of initial and terminal components.

Definition 8.5 (Initial and Terminal Component)**.** Let $\mathcal{G}$ be a directed graph, and $\mathcal{H}$ a strongly connected component of $\mathcal{G}$. $\mathcal{H}$ is an *initial component* if it is closed under predecessors. That is, if $v \in V(\mathcal{G})$ with $(v, w) \in E(\mathcal{G})$ for some $w \in V(\mathcal{H})$, then $v \in V(\mathcal{H})$. $\mathcal{H}$ is a *terminal component* if it is closed under successors. That is, if $v \in V(\mathcal{G})$ with $(w, v) \in E(\mathcal{G})$ for some $w \in V(\mathcal{H})$, then $v \in V(\mathcal{H})$.

We denote by $\text{Init}(\mathcal{G})$ the set of all vertices in initial components, and $\text{Term}(\mathcal{G})$ the set of all vertices in terminal components. For a subset of vertices $B \subseteq V(\mathcal{G})$ we write $\text{Init}(B)$ and $\text{Term}(B)$ for $\text{Init}(\mathcal{G}[B])$ and $\text{Term}(\mathcal{G}[B])$ respectively when $\mathcal{G}$ is clear from the context.

Another way to view initial and terminal components are as the roots and leaves (respectively) of the block graph of $\mathcal{G}$: the directed acyclic graph with the strongly connected components of $\mathcal{G}$ as vertices and an edge (C, C') if there is an edge in $\mathcal{G}$ from some vertex in C to some vertex in C'. With this interpretation it is straightforward to show that initial and terminal components are well-behaved with respect to the structural relations for directed graphs we consider important.

Lemma 8.6. *Let $\mathcal{G}$, $\mathcal{G}'$ and $\mathcal{G}''$ be non-empty directed graphs and $C \subseteq \mathcal{G}$ an initial (terminal) component of $\mathcal{G}$.*

1. *If $\mathcal{G}'$ is a subgraph of $\mathcal{G}$ with $C \cap V(\mathcal{G}') \neq \emptyset$ then there is an initial (terminal) component $C' \subseteq \mathcal{G}'$ such that $C' \subseteq C$.*

2. *If $\mathcal{G}$ is a directed union of $\mathcal{G}'$ and $\mathcal{G}''$ then C is either an initial (terminal) component of $\mathcal{G}_1$ or an initial (terminal) component of $\mathcal{G}_2$.*

3. *If $\mathcal{G}'$ is a directed union of $\mathcal{G}$ and $\mathcal{G}''$ (directed union of $\mathcal{G}''$ and $\mathcal{G}$) then C is an initial (terminal) component of $\mathcal{G}'$.*

4. *If either $|C| \geq 2$ or $\mathcal{G}'$ is strongly connected, then $C \bullet \mathcal{G}'$ is an initial (terminal) component of $\mathcal{G} \bullet \mathcal{G}'$.*

5. *If $\mathcal{G} = \mathcal{G}' \bullet \mathcal{G}''$ then $\pi_1(C) = \{v : (v, w) \in C\}$ is an initial (terminal) component of $\mathcal{G}'$.*

Definition 8.7 (Initial bramble)**.** Let $\mathcal{G}$ be a directed graph. An *initial bramble* in $\mathcal{G}$ is a set $\mathcal{B}$ of subsets of $V(\mathcal{G})$ such that for all pairs $B, B' \in \mathcal{B}$ and for all $x \in \text{Init}(B)$, there exists $y \in \text{Init}(B')$ such that $y \in \textit{Reach}_{B \cup \text{Init}(B')}(x)$.

Definition 8.8 (Terminal bramble)**.** Let $\mathcal{G}$ be a directed graph. A *terminal bramble* in $\mathcal{G}$ is a set $\mathcal{B}$ of subsets of $V(\mathcal{G})$ such that for all pairs $B, B' \in \mathcal{B}$ and for all $x \in \text{Term}(B)$, there exists $y \in \text{Term}(B')$ such that $y \in \textit{Reach}_{\text{Term}(B)\cup B'}(x)$.

Definition 8.9 (Bramble width)**.** Let $\mathcal{G}$ be a directed graph and $\mathcal{B}$ an initial or terminal bramble in $\mathcal{G}$. The *width* of $\mathcal{B}$ is the size of the smallest hitting set of $\mathcal{B}$. That is, the size of the minimal $X \subseteq V(\mathcal{G})$ such that $X \cap B \neq \emptyset$ for all $B \in \mathcal{B}$.

Although it would appear that initial and terminal brambles are similar entities, we show that there are graphs where the smallest width of an initial bramble differs from the smallest width of a terminal bramble. It might also seem that, since an initial component of a graph $\mathcal{G}$ is a terminal component of the graph $\mathcal{G}^{op}$ obtained by reversing the direction of the edges of $\mathcal{G}$, that an initial bramble in $\mathcal{G}$ is a terminal bramble in $\mathcal{G}^{op}$. However, the ordering of the quantifiers in each of the definitions means that this is not necessarily the case: an initial bramble in $\mathcal{G}$ is, in $\mathcal{G}^{op}$, a set of subsets such that for all pairs B, B' and all $x \in \text{Term}(\mathcal{G}^{op}[B])$, there exists $y \in \text{Term}(\mathcal{G}^{op}[B'])$ such that $x \in \textit{Reach}_{\mathcal{G}^{op}[\text{Term}(B')\cup B]}(y)$. Before we show how initial and terminal brambles differ, we show how they correspond to DAG-width and Kelly-width, and establish some robustness results.

Lemma 8.10. *Let $\mathcal{G}$ be a directed graph.*

1. *If $\mathcal{G}$ has an initial bramble of width k then the robber can defeat $k-1$ cops in the cops and visible robber game.*

2. *If $\mathcal{G}$ has a terminal bramble of width k then the robber can defeat $k-1$ cops in the cops and inert robber game.*

Proof. 1: Suppose $\mathcal{G}$ has an initial bramble $\mathcal{B}$ of width k. Then, for any set X with $|X| \leq k-1$ there exists $B_X \in \mathcal{B}$ such that $B_X \cap X = \emptyset$. The strategy for the robber is to be on some vertex in $\text{Init}(B_X)$ whenever the cops are located on X. It is clear from the definition of an initial bramble that such a move is always possible. As the robber is able to do this forever, it follows that this is a winning strategy for the visible robber against $k-1$ cops.

2: Now suppose $\mathcal{G}$ has a terminal bramble $\mathcal{B}$ of width k. Again, for any set X with $|X| \leq k-1$ there exists $B_X \in \mathcal{B}$ such that $B_X \cap X = \emptyset$. The "strategy" for the robber is, when he can move and when the cops are on X, to move to the first element of a strongly connected component of $\text{Term}(B_X)$ that will be occupied by the cops. More precisely, we show that after every cop move, there exists $B \in \mathcal{B}$ such that $\text{Term}(B)$ is contaminated. Clearly this is true at the beginning, as every vertex is contaminated. Now suppose the cops are moving from X to X' and for some $B \in \mathcal{B}$ and some terminal component C of $\mathcal{G}[B]$, $X \cap C = \emptyset$ and there exists a contaminated vertex $v \in X' \cap C$. As $B_{X'} \cap X' = \emptyset$, and C is a terminal component, the path in $\text{Term}(B) \cup B_{X'}$ from v to some $w \in \text{Term}(B_{X'})$ is cop-free. Thus $B_{X'}$ is now an element of $\mathcal{B}$ such that $\text{Term}(B_{X'})$ is contaminated. □

An immediate corollary from the game characterizations of DAG-width and Kelly-width is that initial and terminal brambles provide obstructions for DAG-width and Kelly-width.

Corollary 8.11. *Let $\mathcal{G}$ be a directed graph.*

1. *If $\mathcal{G}$ has an initial bramble of width k then $\mathcal{G}$ has DAG-width $\geq k$.*

2. *If $\mathcal{G}$ has a terminal bramble of width k then $\mathcal{G}$ has Kelly-width $\geq k$.*

Unfortunately, it is not known whether the converse to Lemma 8.10 holds.

It is relatively straightforward to show that brambles behave manner to the cops and robber games under various graph operations. For example a bramble of a graph is a bramble of any supergraph, and the width of a bramble increases by an appropriate factor under lexicographic products. This strongly suggests that the converse of Lemma 8.10 does hold.

Conjecture 8.12. *Let $\mathcal{G}$ be a directed graph.*

1. *If the robber can defeat $k-1$ cops in the cops and visible robber game on $\mathcal{G}$ then $\mathcal{G}$ has an initial bramble of width k.*

2. *If the robber can defeat $k-1$ cops in the cops and inert robber game on $\mathcal{G}$ then $\mathcal{G}$ has a terminal bramble of width k.*

We observe that since monotonicity is not sufficient in either cops and robber game [KO07], we know that the converse of Corollary 8.11 does not hold. However, as with D-havens, a result along the lines of Theorem 4.12 would resolve Open Problems 6.12 and 7.33.

We conclude this section by combining these results with some results from Chapter 7 to show that initial brambles and terminal brambles are different.

Proposition 8.13. *For all $m \in \mathbb{N}$, there exists a directed graph with an initial bramble of width $4m$ but no terminal bramble of width $\geq 3m+1$.*

Proof. Consider the graph $\mathcal{G}$ in Figure 7.1. As we observed in the proof of Theorem 7.37, 3 cops suffice to capture an inert robber on $\mathcal{G}$. We also showed that $\mathcal{G}$ has an initial bramble of width 4: $\big\{\{v_1, v_2, v_4\}, \{v_3\}, \{v_5\}, \{v_6\}\big\}$. The result follows by taking the lexicographic product of this graph with $\mathcal{K}_m$, the complete digraph on m vertices. □

8.2 Directed minors

In this section we investigate the problem of finding a relation on directed graphs which represents structural simplification. Such relations are ubiquitous throughout mathematics, for example in algebra or model theory homomorphisms describe structural simplifications, and in geometry or topology homeomorphisms are the key structural relations. Graphs can be viewed both as relational structures and as topological complexes, so there are well-defined notions of graph homomorphisms and graph homeomorphisms. However, for undirected graphs at least, the minor relation is arguably the most suitable relation for comparing fundamental graph structural properties such as connectivity and cyclicity. Intuitively, a graph $\mathcal{G}$ is a minor of a graph $\mathcal{H}$ if $\mathcal{G}$ can be embedded in $\mathcal{H}$ modulo connected sets. That is, if we consider connected sets in $\mathcal{H}$ as "vertices", then $\mathcal{G}$ is a subgraph of this "graph". More precisely,

Definition 8.14 (Minor)**.** Let $\mathcal{G}$ and $\mathcal{H}$ be undirected graphs. $\mathcal{G}$ is a *minor* of $\mathcal{H}$, written $\mathcal{G} \leq \mathcal{H}$, if there exists a function $\xi : V(\mathcal{G}) \to \mathcal{P}(V(\mathcal{H}))$ which maps distinct vertices to disjoint sets such that:

- For all $v \in V(\mathcal{G})$, $\mathcal{H}[\xi(v)]$ is a connected graph, and
- For all $\{v, w\} \in E(\mathcal{G})$ there exists $\{v', w'\} \in E(\mathcal{H})$ such that $v' \in \xi(v)$ and $w' \in \xi(w)$.

So why is the minor relation a good indicator of structural simplification? As we observed above, there are well-defined notions of graph homomorphisms and graph homeomorphisms. A homomorphism preserves relational structure and a homeomorphism preserves topological shape, so injective homomorphisms or subgraph homeomorphisms would seem to be reasonable indicators of structural simplification. However, the minor relation subsumes these. We see from the definition that the minor relation can be considered a generalization of injective graph homomorphisms: $\mathcal{G}$ is a minor of $\mathcal{H}$ if there is a homomorphic-like injective map from $V(\mathcal{G})$ to connected sets of $\mathcal{H}$. Presently we will also show that if $\mathcal{G}$ is homeomorphic to a subgraph of $\mathcal{H}$ then $\mathcal{G}$ is a minor of $\mathcal{H}$. So the minor relation can be seen as a generalization of both relational and topological structure simplification. We now turn to the problem of finding an extension of the minor relation to directed graphs which enjoys similar properties.

The definition of a minor has two obvious extensions to directed graphs: either map vertices to weakly connected sets or map vertices to strongly connected sets. However, as we argue below, neither of these truly reflect the notion of structural simplification that complexity measures like directed tree-width, DAG-width and Kelly-width suggest. In the remainder of this section we identify the characteristics of the minor relation that make it useful and we introduce several definitions of digraph minor relations and compare them against these criteria. First we show how we can view the minor relation operationally, and how this implies that the minor relation is a generalization of subgraph homeomorphism.

Definition 8.15 (Edge contraction)**.** Let $\mathcal{G}$ be a graph, and $e = (v, w) \in E(\mathcal{G})$. The graph $\mathcal{G}'$ obtained from $\mathcal{G}$ by *contracting* e is defined as:

- $V(\mathcal{G}') = V(\mathcal{G}) \setminus \{v\}$,
- $E(\mathcal{G}') = \big(E(\mathcal{G}) \cup \{(u, w) : (u, v) \in E(\mathcal{G})\} \cup \{(w, u) : (v, u) \in E(\mathcal{G})\}\big) \setminus \{(u, v), (v, u) : u \in V(\mathcal{G})\}$.

The following result follows easily from the definitions and is often used as an alternative definition of the minor relation.

Lemma 8.16. *Let $\mathcal{G}$ and $\mathcal{H}$ be undirected graphs. The following are equivalent:*

1. *$\mathcal{G}$ is a minor of $\mathcal{H}$,*
2. *$\mathcal{G}$ is isomorphic to a subgraph of a graph obtained by contracting edges of $\mathcal{H}$, and*
3. *$\mathcal{G}$ is isomorphic to a graph obtained by contracting edges of a subgraph of $\mathcal{H}$.*

Proof. $1 \Rightarrow 2$: Suppose $\mathcal{G}$ is a minor of $\mathcal{H}$ and let $\xi : V(\mathcal{G}) \to \mathcal{P}(V(\mathcal{H}))$ be the function witnessing this. Let $\mathcal{H}'$ be the graph obtained from $\mathcal{H}$ by contracting, for each $v \in V(\mathcal{G})$ the edges in $\mathcal{H}[\xi(v)]$. Now ξ can be viewed as an injective mapping from $V(\mathcal{G})$ to $V(\mathcal{H}')$ such that for each $\{v, w\} \in E(\mathcal{G})$, $\{\xi(v), \xi(w)\} \in E(\mathcal{H}')$. That is, ξ is an embedding of $\mathcal{G}$ in $\mathcal{H}'$, so $\mathcal{G}$ is isomorphic to a subgraph of $\mathcal{H}'$, a graph resulting from contracting edges of $\mathcal{H}$.

$2 \Leftrightarrow 3$: Let us view the subgraph relation as the operation of deleting edges and isolated vertices. That is, $\mathcal{G}$ is a subgraph of $\mathcal{H}$ if $\mathcal{G}$ can be obtained by deleting edges and isolated vertices of $\mathcal{H}$. We observe that edge and isolated vertex deletion and edge contraction commute, that is we obtain the same graph independent of the order of the operations. Thus if we perform all edge contractions first and then all deletions we obtain the same graph by performing all deletions first followed by all edge contractions and vice versa. Thus any subgraph of a

graph obtained by contracting edges is a graph obtained by contracting edges of a subgraph and conversely.

$3 \Rightarrow 1$: Suppose $\mathcal{G}$ is isomorphic to a graph obtained by contracting edges of $\mathcal{H}'$ where $\mathcal{H}'$ is a subgraph of $\mathcal{H}$. For convenience, we will assume that $\mathcal{G}$ is a graph obtained by contracting edges of $\mathcal{H}'$. For each $v \in V(\mathcal{G})$ define $\xi(v)$ as the set of vertices $w \in V(\mathcal{H}')$ such that there is a path from w to v consisting of edges which are contracted to obtain $\mathcal{G}$. From the definition of ξ, $\mathcal{H}[\xi(v)] = \mathcal{H}'[\xi(v)]$ is connected. Now suppose $\{v, w\} \in E(\mathcal{G})$. It follows from the definition of edge contractions that there exists $\{v', w'\} \in E(\mathcal{H}')$ such that there are paths from v' to v and from w' to w consisting of edges which are contracted to obtain $\mathcal{G}$. That is $v' \in \xi(v)$ and $w' \in \xi(w)$. As $V(\mathcal{H}') \subseteq V(\mathcal{H})$, h is a function from $V(\mathcal{G})$ to $\mathcal{P}(V(\mathcal{H}))$, so $\mathcal{G}$ is a minor of $\mathcal{H}$. □

Indeed, as subgraphs and edge contractions are well-defined for directed graphs, this lemma suggests the following natural definition of a minor relation on directed graphs.

Definition 8.17 (Minor for digraphs)**.** Let $\mathcal{G}$ and $\mathcal{H}$ be directed graphs. $\mathcal{G}$ is a *minor* of $\mathcal{H}$, $\mathcal{G} \leq \mathcal{H}$, if $\mathcal{G}$ is isomorphic to a graph obtained from $\mathcal{H}$ by a sequence of edge and isolated vertex deletions and edge contractions.

It is clear from Lemma 8.16 that this definition is equivalent to the minor relation on the underlying undirected graphs, hence the notation. That is,

Proposition 8.18. *Let $\mathcal{G}$ and $\mathcal{H}$ be digraphs. $\mathcal{G} \leq \mathcal{H}$ if, and only if, $\overline{\mathcal{G}} \leq \overline{\mathcal{H}}$.*

We also observe that the $\leq$-minor relation corresponds to the weakly connected "natural" generalization of the minor relation.

Proposition 8.19. *Let $\mathcal{G}$ and $\mathcal{H}$ be digraphs. $\mathcal{G} \leq \mathcal{H}$ if, and only if, there exists a function $\xi : V(\mathcal{G}) \to \mathcal{P}(V(\mathcal{H}))$ such that:*

- *if $v \neq w$ then $\xi(v)$ is disjoint from $\xi(w)$,*
- *for all $v \in V(\mathcal{G})$, $\mathcal{H}[\xi(v)]$ is a weakly connected graph, and*
- *for all $(v, w) \in E(\mathcal{G})$ there exists $(v', w') \in E(\mathcal{H})$ such that $v' \in \xi(v)$ and $w' \in \xi(w)$.*

These observations show that the minor relation has a straightforward extension to directed graphs. However, just the simple extension of tree-width to directed graphs is not an ideal measure of complexity, we argue below that this definition is not restrictive enough to be a suitable relation for structural simplification for digraphs. In particular a minor of an acyclic digraph need not be acyclic, which goes against our tenet that acyclic graphs are structurally the least complex graphs. However, all the minor relations we introduce in the Section 8.2.2 are restrictions of this relation.

Lemma 8.16 also demonstrates how minors can be seen as a generalization of subgraph homeomorphisms. First we recall the definition of a subgraph homeomorphism.

Definition 8.20 (Subgraph homeomorphism)**.** Let $\mathcal{G}$ and $\mathcal{H}$ be (directed) graphs. We say $\mathcal{G}$ is *homeomorphic to a subgraph* of $\mathcal{H}$ if there is an injective function $\eta : V(\mathcal{G}) \to V(\mathcal{H})$ and a mapping p from edges of $\mathcal{G}$ to pairwise internal-vertex-disjoint paths in $\mathcal{H}$ such that for $e = (v, w) \in E(\mathcal{G})$, $p(e)$ is a (directed) path from $\eta(v)$ to $\eta(w)$.

Lemma 8.21. *Let $\mathcal{G}$ and $\mathcal{H}$ be undirected graphs. If $\mathcal{G}$ is homeomorphic to a subgraph of $\mathcal{H}$ then $\mathcal{G}$ is a minor of $\mathcal{H}$.*

Proof. We observe that if $\mathcal{G}$ is homeomorphic to a subgraph of $\mathcal{H}$, then $\mathcal{G}$ is isomorphic to a graph obtained from a subgraph of $\mathcal{H}$ by repeatedly replacing vertices of degree 2 with an edge joining its neighbours. But this operation can also be viewed as contracting edges that have at least one endpoint with degree 2. Therefore $\mathcal{G}$ is isomorphic to a graph obtained by contracting edges of a subgraph of $\mathcal{H}$, so by Lemma 8.16, $\mathcal{G}$ is a minor of $\mathcal{H}$. □

8.2.1 What makes a good minor relation?

We now consider the properties we expect a reasonable definition of a minor relation for directed graphs to satisfy. First and foremost, the relation should respect digraph complexity. That is, if $\mathcal{G}$ is a minor of $\mathcal{H}$ then $\mathcal{G}$ should not be more structurally complex than $\mathcal{H}$. But which notion of digraph complexity should we use? As we mentioned at the start of the chapter we are primarily interested in a relation corresponding to directed connectivity, so DAG-width or Kelly-width or their associated cops and robber games would be suitable. However, there is also no known appropriate relation for strong connectivity, so we also consider directed tree-width. In Section 8.2.3 we consider various graph properties that are preserved under the operation "taking a minor" and use these to identify unsuitable candidates.

The second property we are interested in is being able to obtain generalizations of theorems concerning the minor relation. In particular, we are concerned with trying to extend two results: the Graph Minor Theorem, which asserts that the minor relation is a well-quasi order, and the algorithmic result that for a fixed graph $\mathcal{H}$, determining if $\mathcal{H}$ is a minor of $\mathcal{G}$ can be decided in cubic time. The latter result implies that any class characterized by a finite set of excluded minors can be decided in polynomial time, and the former implies that any minor-closed property can be characterized by a finite set of excluded minors. Although we show that many of our defined relations fail to satisfy this property, the investigation raises some interesting questions.

Our final requirement for a reasonable notion of a minor relation for directed graphs is that it should be an extension of the minor relation for undirected graphs. In particular, if $\mathcal{G}$ and $\mathcal{H}$ are undirected graphs such that $\mathcal{G}$ is a minor of $\mathcal{H}$ then $\overleftrightarrow{\mathcal{G}}$ should be a minor of $\overleftrightarrow{\mathcal{H}}$. Furthermore, it should also generalize subgraph homeomorphisms (for directed graphs). That is, if we replace internal-vertex-disjoint paths with single edges we should obtain a minor of the original graph. Although many of our defined relations do satisfy both these requirements, some interesting relations do not, including the strongly connected "natural" generalization of the minor relation and two relations which occur in the literature: the butterfly minor relation and the topological minor relation.

8.2.2 Directed minor relations

In this section we define several minor relations for digraphs. We adopt the operational definition of minor implied by Lemma 8.16 and generate variations by considering different restrictions on the edge contraction operation. For the results we establish, it is convenient to consider two types of edge contraction operation: one which contracts a single edge, and one which contracts multiple edges simultaneously. We call the first kind *edge contractions* and the second *set contractions*. We observe that when a sequence of edge contractions are performed, it does not matter in which order they are performed, the resulting graphs are all isomorphic. Thus to

"simultaneously" contract a set of edges, we can contract them individually in some arbitrary order. We now define the edge and set contractions we use to define our minor relations.

Definition 8.22. Let $\mathcal{G}$ be a directed graph and $e = (u, v) \in E(\mathcal{G})$.

- We say e can be *topologically contracted* if either
  - u has in-degree 1 and out-degree 1, or
  - v has in-degree 1 and out-degree 1.
- We say e can be *butterfly contracted* if either
  - u has out-degree 1, or
  - v has in-degree 1.
- We say e can be *D-contracted* unless either
  - there is a directed path from u to v edge disjoint from (u, v), or
  - there exists two vertex disjoint cycles C_1, C_2, each with at least two vertices, such that $u \in C_1$ and $v \in C_2$.

Before we introduce the set contractions, we observe that the above definitions of edge contractions are ordered from most restrictive to least restrictive. That is,

Lemma 8.23. *Let $\mathcal{G}$ be a directed graph and $e = (u, v) \in E(\mathcal{G})$. If e can be topologically contracted then e can be butterfly contracted, and if e can be butterfly contracted then e can be D-contracted.*

Proof. If e can be topologically contracted then clearly e can be butterfly contracted. Now suppose e can be butterfly contracted. If u has out-degree 1 then e is the only outgoing edge from u so there is no path from u to v which is edge disjoint from e and there is no cycle which contains u and does not contain v. Thus e can be D-contracted. Otherwise v has in-degree 1 and e is the only incoming edge to v. Again, there can be no path from u to v which is edge disjoint from e and there is no cycle which contains v and does not contain u. So e can be D-contracted. □

Definition 8.24. Let $\mathcal{G}$ be a directed graph and $E \subseteq E(\mathcal{G})$.

- If $E = \{(u, v), (v, u)\}$ then the simultaneous contraction of E is an *anti-parallel contraction*.
- If $\mathcal{G}[E]$ is a strongly connected graph, then the simultaneous contraction of E is a *strong contraction*.

Clearly these definitions are also ordered from most restrictive to least restrictive. We include the result for completeness.

Lemma 8.25. *Let $\mathcal{G}$ be a directed graph and $E \subseteq E(\mathcal{G})$. An anti-parallel contraction of E is a strong contraction of E.*

We now combine these edge and set contractions with the subgraph relation to obtain a number of minor relations.

Definition 8.26 (Subgraph minor)**.** Let $\mathcal{G}$ and $\mathcal{H}$ be directed graphs. $\mathcal{G}$ is a *subgraph minor* of $\mathcal{H}$, $\mathcal{G} \Subset \mathcal{H}$, if $\mathcal{G}$ is isomorphic to a graph obtained from $\mathcal{H}$ by a sequence of edge and isolated vertex deletions. $\mathcal{G}$ is an *anti-parallel subgraph minor* of $\mathcal{H}$, $\mathcal{G} \Subset^{AP} \mathcal{H}$, if $\mathcal{G}$ is isomorphic to a graph obtained from $\mathcal{H}$ by a sequence of edge and isolated vertex deletions and anti-parallel contractions. $\mathcal{G}$ is a *strong subgraph minor* of $\mathcal{H}$, $\mathcal{G} \Subset^{S} \mathcal{H}$, if $\mathcal{G}$ is isomorphic to a graph obtained from $\mathcal{H}$ by a sequence of edge and isolated vertex deletions and strong contractions.

Definition 8.27 (Topological minor)**.** Let $\mathcal{G}$ and $\mathcal{H}$ be directed graphs. $\mathcal{G}$ is a *topological minor* of $\mathcal{H}$, $\mathcal{G} \dashv \mathcal{H}$, if $\mathcal{G}$ is isomorphic to a graph obtained from $\mathcal{H}$ by a sequence of edge and isolated vertex deletions and topological contractions. $\mathcal{G}$ is an *anti-parallel topological minor* of $\mathcal{H}$, $\mathcal{G} \dashv^{AP} \mathcal{H}$, if $\mathcal{G}$ is isomorphic to a graph obtained from $\mathcal{H}$ by a sequence of edge and isolated vertex deletions, and anti-parallel and topological contractions. $\mathcal{G}$ is a *strong topological minor* of $\mathcal{H}$, $\mathcal{G} \dashv^{S} \mathcal{H}$, if $\mathcal{G}$ is isomorphic to a graph obtained from $\mathcal{H}$ by a sequence of edge and isolated vertex deletions, and strong and topological contractions.

Definition 8.28 (Butterfly minor)**.** Let $\mathcal{G}$ and $\mathcal{H}$ be directed graphs. $\mathcal{G}$ is a *butterfly minor* of $\mathcal{H}$, $\mathcal{G} \ll \mathcal{H}$, if $\mathcal{G}$ is isomorphic to a graph obtained from $\mathcal{H}$ by a sequence of edge and isolated vertex deletions and butterfly contractions. $\mathcal{G}$ is an *anti-parallel butterfly minor* of $\mathcal{H}$, $\mathcal{G} \ll^{AP} \mathcal{H}$, if $\mathcal{G}$ is isomorphic to a graph obtained from $\mathcal{H}$ by a sequence of edge and isolated vertex deletions, and anti-parallel and butterfly contractions. $\mathcal{G}$ is a *strong butterfly minor* of $\mathcal{H}$, $\mathcal{G} \ll^{S} \mathcal{H}$, if $\mathcal{G}$ is isomorphic to a graph obtained from $\mathcal{H}$ by a sequence of edge and isolated vertex deletions, and strong and butterfly contractions.

Definition 8.29 (D-minor)**.** Let $\mathcal{G}$ and $\mathcal{H}$ be directed graphs. $\mathcal{G}$ is a *D-minor* of $\mathcal{H}$, $\mathcal{G} \trianglelefteq \mathcal{H}$, if $\mathcal{G}$ is isomorphic to a graph obtained from $\mathcal{H}$ by a sequence of edge and isolated vertex deletions and D-contractions. $\mathcal{G}$ is an *anti-parallel D-minor* of $\mathcal{H}$, $\mathcal{G} \trianglelefteq^{AP} \mathcal{H}$, if $\mathcal{G}$ is isomorphic to a graph obtained from $\mathcal{H}$ by a sequence of edge and isolated vertex deletions, and anti-parallel and D-contractions. $\mathcal{G}$ is a *strong D-minor* of $\mathcal{H}$, $\mathcal{G} \trianglelefteq^{S} \mathcal{H}$, if $\mathcal{G}$ is isomorphic to a graph obtained from $\mathcal{H}$ by a sequence of edge and isolated vertex deletions, and strong and D-contractions.

Remark. Unlike the case for the undirected minor relation, the edge and set contractions we have defined here do not commute with edge and vertex deletion: an edge may not be edge contractible until some other edges have been deleted, and a set of edges may no longer be set contractible after some edges have been deleted. However, for our definitions it is the case that the reverse holds: if an edge is edge contractible before some other edges or vertices have been deleted, then it is still edge contractible after those deletions, and if a set of edges is set contractible after some deletions then it is set contractible before those deletions. So we may assume that to obtain a minor we perform a sequence of set contractions, followed by a sequence of edge and isolated vertex deletions, followed by a sequence of edge contractions.

Before we establish some results, we define a useful function which captures the inverse of edge contraction.

Definition 8.30 (Vertex expansion)**.** Let $\preceq$ be a minor relation, and let $\mathcal{G}$ and $\mathcal{H}$ be directed graphs such that $\mathcal{G} \preceq \mathcal{H}$. A $\preceq$*-vertex expansion* of $\mathcal{G}$ to $\mathcal{H}$ is a function $\xi : V(\mathcal{G}) \to \mathcal{P}(V(\mathcal{H}))$ defined to be ξ^n in the following construction. Let $\mathcal{G}_0 \succeq \mathcal{G}_1 \succeq \ldots \succeq \mathcal{G}_n$ be a sequence of

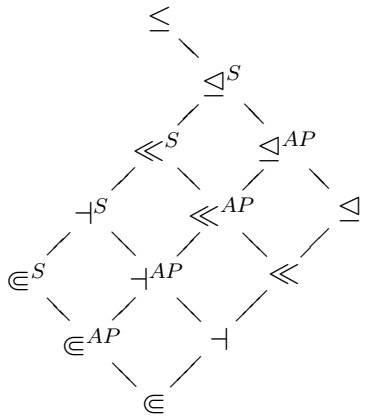


Figure 8.2: Inclusion diagram for the introduced minor relations

graphs such that $\mathcal{G}_0 = \mathcal{H}$, $\mathcal{G}_n = \mathcal{G}$ and $\mathcal{G}_{i+1}$ is obtained from $\mathcal{G}_i$ by a single edge deletion, vertex deletion, or edge contraction[1]. For each $i \leq n$ define $\xi^i : V(\mathcal{G}_i) \to \mathcal{P}(V(\mathcal{H}))$ as follows. $\xi^0(v) = \{v\}$. If $\mathcal{G}_{i+1}$ is obtained from $\mathcal{G}_i$ by contracting (u, v) then $\xi^{i+1}(v) = \xi^i(v) \cup \xi^i(u)$ and $\xi^{i+1}(w) = \xi^i(w)$ for all $w \neq u, v$ (recall that $u \notin V(\mathcal{G}_{i+1})$). Otherwise we let $\xi^{i+1}(w) = \xi^i(w)$ for all $w \in V(\mathcal{G}_{i+1})$.

Lemmas 8.23 and 8.25 imply that all the minor relations we have so far defined can be arranged as in the inclusion diagram of Figure 8.2. Presently we will show that each inclusion in Figure 8.2 is strict, however first we need to show that these minor relations are well-behaved with respect to directed connectivity.

Theorem 8.31. *Let $\mathcal{G}$ and $\mathcal{H}$ be directed graphs, with $\mathcal{G} \trianglelefteq^S \mathcal{H}$. If k cops can capture a visible robber on $\mathcal{H}$ then k cops can capture a visible robber on $\mathcal{G}$.*

Proof. As a consequence of Lemma 6.18, it suffices to show that the number of cops required decreases after either a D-contraction or a strong contraction. Let ξ be a $\trianglelefteq^S$-vertex expansion from $\mathcal{G}$ to $\mathcal{H}$. The idea is that if any of the vertices of $\xi(w)$ is occupied by a cop, then we occupy w with a cop. It is clear that if $\mathcal{G}$ is obtained from $\mathcal{H}$ by strong contractions only, then this describes a winning strategy for the cops as the robber is more restricted in his movement. So it suffices to consider the case when $\mathcal{G}$ is obtained from $\mathcal{H}$ by a single D-contraction of the edge (u, v). In this case, the robber may be able to reach some vertices in $\mathcal{G}$ that he could not reach in $\mathcal{H}$ by a directed path through the contraction of u and v. Let $U \subseteq V(\mathcal{H})$ be the set of vertices w, not including u, for which there is a path from u to w edge disjoint from (u, v), and let $V \subseteq V(\mathcal{H})$ be the set of vertices w, not including v, for which there is a path from w to v edge disjoint from (u, v). We observe that after (u, v) is contracted, the robber is able to move from vertices in V to vertices in U. We argue that he can only do this once.

Since (u, v) can be D-contracted, U and V are disjoint, as otherwise there would be a path from u to v edge disjoint from (u, v). Thus, any path from U to V must include the edge (u, v). For any $x \in V$, suppose there is a directed path in $\mathcal{G}$ to some $y \in U$ such that there is a directed path from y to some $z \in V$. Since such a path in $\mathcal{H}$ must go through (u, v), it follows that there

[1]We treat set contractions as sequences of single edge contractions, so $\mathcal{G}_i$ might not necessarily be a minor of $\mathcal{G}_{i+1}$

is a path from y to u and a path from v to z. Thus u and y are two distinct vertices in a cycle, as are v and z, contradicting the assumption that (u, v) could be D-contracted.

The strategy for the cops is now as follows. Play as before, occupying $w \in V(\mathcal{G})$ if some vertex in $\xi(w)$ is occupied. If the robber never moves from V to U, then each move of the robber can be simulated on $\mathcal{H}$. Otherwise, if the robber does move from V to U, he can never return to V, so we can discard this part of the graph and continue playing the winning strategy on the subgraph of $\mathcal{H}$. □

We now have sufficient tools to demonstrate that each minor relation we have defined is distinct from the others, and that there are no other inclusions other than those we have already identified.

Theorem 8.32. *The inclusion diagram of Figure 8.2 is strict and complete.*

Proof. To prove the result, it suffices to show the following six inequations:

(I) $\dashv \not\subseteq \Subset^S$

(II) $\ll \not\subseteq \dashv^S$

(III) $\trianglelefteq \not\subseteq \ll^S$

(IV) $\leq \not\subseteq \trianglelefteq^S$

(V) $\Subset^{AP} \not\subseteq \trianglelefteq$

(VI) $\Subset^S \not\subseteq \trianglelefteq^{AP}$

Now consider Table 8.1. We show that for each pair of minor relations $(\preceq, \preceq')$, $\mathcal{G}_2 \preceq \mathcal{G}_1$ but $\mathcal{G}_2 \not\preceq' \mathcal{G}_1$. It is easy to see that in each example, $\mathcal{G}_2 \preceq \mathcal{G}_1$. We therefore show that $\mathcal{G}_2 \not\preceq' \mathcal{G}_1$.

(I) – (III): We observe that in each example, the graph $\mathcal{G}_2$ has only one less edge than $\mathcal{G}_1$. It is easily checked that deleting any edge from $\mathcal{G}_1$ will not result in the graph $\mathcal{G}_2$, thus the only possible way for $\mathcal{G}_2 \preceq' \mathcal{G}_1$ is from edge contractions. In (I), by symmetry any edge will suffice. But no single edge is contractible under strong contractions, thus $\mathcal{G}_2 \not\Subset^S \mathcal{G}_1$. In (II) and (III), to obtain a vertex of degree 4, the only edge which can be contracted is the vertical edge. However, in (II) both endpoints of this edge have out-degree 2 and therefore it cannot be topologically contracted, and in (III) this edge is neither the only outgoing edge of its tail nor the only incoming edge of its head, thus it cannot be butterfly contracted. In both cases it cannot be contracted using a strong contraction, thus in (II) $\mathcal{G}_2 \not\dashv^S \mathcal{G}_1$, and in (III) $\mathcal{G}_2 \not\ll^S \mathcal{G}_1$.

(IV): This follows directly from Theorem 8.31, as $\mathcal{G}_1$ is acyclic and $\mathcal{G}_2$ is not.

(V): We observe that it is not possible to D-contract any edge in $\mathcal{G}_1$. Thus if $\mathcal{G}_2$ is a D-minor of $\mathcal{G}_1$, $\mathcal{G}_2$ must be a D-minor of some subgraph of $\mathcal{G}_1$ with at least one edge deleted. However, only two cops are required to capture a robber on $\mathcal{G}_1$ with any edge deleted, whereas three cops are required to capture a robber on $\mathcal{G}_2$. Thus, from Theorem 8.31, $\mathcal{G}_2$ cannot be a D-minor of $\mathcal{G}_1$.

(VI): We observe that it is not possible to D-contract any edge in $\mathcal{G}_1$ without first deleting some edges. As anti-parallel contractions reduce the number of anti-parallel pairs of edges, we cannot obtain $\mathcal{G}_2$ from $\mathcal{G}_1$ through anti-parallel contractions alone. Thus if $\mathcal{G}_2 \trianglelefteq^{AP} \mathcal{G}_1$, to obtain $\mathcal{G}_2$ from $\mathcal{G}_1$ we must first delete some edges. However, it is easy to check that after any edge is deleted from $\mathcal{G}_1$, three cops have a winning strategy to capture a visible robber:

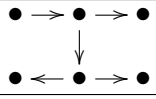
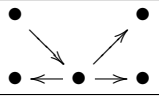
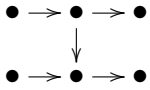
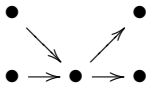
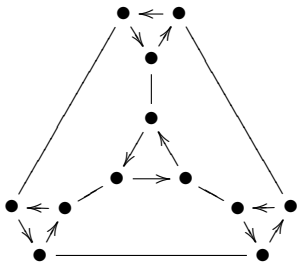
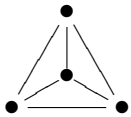
$\preceq_A \not\subseteq \preceq_B$	$\mathcal{G}_1$	$\mathcal{G}_2$
(I) $\dashv \not\subseteq \Subset^S$		
(II) $\ll \not\subseteq \dashv^S$		
(III) $\trianglelefteq \not\subseteq \ll^S$		
(IV) $\leq \not\subseteq \trianglelefteq^S$		
(V) $\Subset^{AP} \not\subseteq \trianglelefteq$		
(VI) $\Subset^S \not\subseteq \trianglelefteq^{AP}$		

Table 8.1: Separating examples of the introduced minor relations

intuitively, removing an edge makes one of the small cycles "weaker" than the others, either by removing one of the edges which leaves the cycle, or removing one of the edges in the cycle. The strategy for three cops is then to chase the robber into this weaker cycle, and then use the weakness to capture him. As four cops are required to capture the robber on $\mathcal{G}_2$, it follows from Theorem 8.31 that $\mathcal{G}_2 \not\trianglelefteq^{AP} \mathcal{G}_1$. □

Remark. Example (IV), which shows that $\leq \not\subseteq \trianglelefteq^S$, illustrates that a $\leq$-minor of an acyclic graph may not necessarily be acyclic. This supports our earlier claim that $\leq$ was not restrictive enough to be a reasonable indicator of structural simplification for directed graphs.

Before we consider some other structural properties which are preserved by the minor relations we have defined, we show that the relations we have introduced include other digraph relations that we have already considered. First we show that topological minors correspond to directed subgraph homeomorphisms.

Proposition 8.33. *Let $\mathcal{G}$ and $\mathcal{H}$ be directed graphs. $\mathcal{G} \dashv \mathcal{H}$ if, and only if, $\mathcal{G}$ is homeomorphic to a subgraph of $\mathcal{H}$.*

Proof. Using the proof of Lemma 8.21 we see that if $\mathcal{G}$ is homeomorphic to a subgraph of $\mathcal{H}$ then $\mathcal{G} \dashv \mathcal{H}$, as the edge contractions used in the proof are all topological contractions. For the converse, suppose $\mathcal{G} \dashv \mathcal{H}$. Without loss of generality, we may assume $\mathcal{G}$ is obtained by a sequence of edge and vertex deletions followed by a sequence of topological contractions. Thus $\mathcal{G}$ is obtained from a subgraph $\mathcal{H}'$ of $\mathcal{H}$ by a sequence of topological contractions. Let $\xi : V(\mathcal{G}) \to \mathcal{P}(V(\mathcal{H}'))$ be a $\dashv$-vertex expansion. We show how ξ can be used to define a (directed) subgraph homeomorphism. From the definition of topological contraction, we observe that for

each $u \in V(\mathcal{G})$, there is at most one $u' \in \xi(u)$ with out-degree ≥ 1, as otherwise it would not be possible to contract $\xi(u)$ to a single vertex. This means that intuitively, $\mathcal{H}'[\xi(u)]$ looks like a star with one central vertex, paths radiating outwards, and a path from u to the central vertex. We define $\eta : V(\mathcal{G}) \to V(\mathcal{H})$ by setting $\eta(u)$ to be either the vertex in $\xi(u)$ with more than one successor, or u if there is no such vertex. We observe the following: if the in-degree of u is greater than 1, then $\eta(u) = u$; there is a directed path in $\xi(u)$ from u to $\eta(u)$; and there is a directed path in $\xi(u)$ from $\eta(u)$ to all vertices in $\xi(u)$ with a successor outside of $\xi(u)$. Now let $(u, v) \in E(\mathcal{G})$ be an edge in $\mathcal{G}$. From the definition of edge contraction, there exists $w \in \xi(u)$ such that $(w, v) \in E(\mathcal{H}')$. From our observations regarding $\eta(u)$ and $\xi(u)$, it follows that there exists a path from $\eta(u)$ to v. Since there is a path from v to $\eta(v)$, it follows that there is a path from $\eta(u)$ to $\eta(v)$. To show this path is vertex distinct (excluding end-points) from any other, we observe that for any $v' \neq v$ such that $(u, v') \in E(\mathcal{G})$, the path from $\eta(u)$ to v' is disjoint (except for $\eta(u)$) to the path from $\eta(u)$ to v, and if $u' \neq u$ is a predecessor of v in $\mathcal{G}$, then $\eta(v) = v$, so the paths from $\eta(u)$ to $\eta(v)$ and from $\eta(u')$ to $\eta(v)$ are disjoint. □

Now we observe that the strong subset minor relation corresponds to the strongly connected "natural" generalization of the minor relation.

Proposition 8.34. *Let $\mathcal{G}$ and $\mathcal{H}$ be digraphs. $\mathcal{G} \Subset^S \mathcal{H}$ if, and only if, there exists an function $\xi : V(\mathcal{G}) \to \mathcal{P}(V(\mathcal{H}))$ which maps distinct vertices to disjoint sets such that:*

- *for all $v \in V(\mathcal{G})$, $\mathcal{H}[\xi(v)]$ is a strongly connected graph, and*
- *for all $(v, w) \in E(\mathcal{G})$ there exists $(v', w') \in E(\mathcal{H})$ such that $v' \in \xi(v)$ and $w' \in \xi(w)$.*

Proof. Let ξ be a $\Subset^S$-vertex expansion of $\mathcal{G}$ in $\mathcal{H}$. From the definition of strong contraction and edge contraction, it follows that ξ satisfies the requirements. □

Finally we observe from Lemma 8.16 that if a minor relation allows anti-parallel contractions, then on bidirected graphs the relation is equivalent to the minor relation for undirected graphs.

Proposition 8.35. *Let $\mathcal{G}$ and $\mathcal{H}$ be undirected graphs, and $\preceq$ a minor relation such that $\preceq \supseteq \Subset^{AP}$. Then $\mathcal{G} \leq \mathcal{H}$ if, and only if $\overleftrightarrow{\mathcal{G}} \preceq \overleftrightarrow{\mathcal{H}}$.*

8.2.3 Preservation results

Theorem 8.31 showed that all the minor relations we introduced respect complexity as defined by directed connectivity. We now consider some other structural properties that are preserved under the operation of taking a minor. Our first result shows that the taking of butterfly minors preserves non-reachability, or equivalently, a butterfly minor vertex expansion preserves reachability.

Proposition 8.36. *Let $\mathcal{G}$ and $\mathcal{H}$ be digraphs such that $\mathcal{G} \ll^S \mathcal{H}$. Let ξ be a $\ll^S$-vertex expansion of $\mathcal{G}$ in $\mathcal{H}$. Let $u, v \in V(\mathcal{G})$. If there is a directed path from u to v then there exists $u' \in \xi(u)$ and $v' \in \xi(v)$ such that there is a directed path from u' to v'.*

Proof. Clearly if $\mathcal{G}$ is a subgraph of $\mathcal{H}$ then the result holds, and similarly if $\mathcal{G}$ can be obtained from $\mathcal{H}$ by strong contractions. Thus it suffices to assume that $\mathcal{G}$ can be obtained from $\mathcal{H}$ by butterfly contractions. Let $w \in V(\mathcal{G})$ be a vertex of $\mathcal{G}$ Since $\xi(w)$ butterfly contracts to a single vertex, it follows that there exists a vertex $w' \in \xi(w)$ such that there is a path to w' from all vertices in $\xi(w)$ with in-degree greater than 1, and there is a path from w' to all vertices in $\xi(w)$ with out-degree greater than 1. Furthermore, there is a path from w to w' and a path from w' to all vertices in $\xi(w)$ with a successor not in $\xi(w)$. If $w_0 w_1 \cdots w_n$ is a path in $\mathcal{G}$ from $u = w_0$ to $v = w_n$, let w'_i be the vertex in $\xi(w_i)$ which satisfies the above observation. It follows from the definition of edge contraction, that for all $i \geq 0$, there is a path in $\mathcal{H}$ from w'_i to w'_{i+1} (in $\xi(w_i) \cup \xi(w_{i+1})$). Thus there exists a path from $u' = w'_0$ to $v' = w'_n$, as required. □

Example (III) in Table 8.1 shows that Proposition 8.36 does not hold for D-minors. However, D-minors do preserve a more restrictive structural property: strong connectivity.

Proposition 8.37. *Let $\mathcal{G}$ and $\mathcal{H}$ be digraphs such that $\mathcal{G} \trianglelefteq^S \mathcal{H}$. Let ξ be a $\trianglelefteq^S$-vertex expansion of $\mathcal{G}$ in $\mathcal{H}$. Let $u, v \in V(\mathcal{G})$. If there are directed paths from u to v and from v to u then there exists $u' \in \xi(u)$ and $v' \in \xi(v)$ such that there are directed paths from u' to v' and from u' to v'.*

Proof. As with Proposition 8.36, we observe that we can assume that $\mathcal{G}$ can be obtained from $\mathcal{H}$ by D-contractions. For $w \in V(\mathcal{G})$, we observe from the definition of D-contractions that $\mathcal{H}[\xi(w)]$ takes the following form: a directed tree, rooted at w, such that if $w_1 w_2 \cdots w_n$ is a path in $\mathcal{H}$ with $w_1, w_n \in \xi(w)$, then w_n is an ancestor of w_1 in $\mathcal{H}[\xi(w)]$. For if this were not the case, then it would not be possible to D-contract $\xi(w)$ to a single vertex. The result now follows by expanding the vertices in the cycle containing u and v in a similar way to Proposition 8.36. □

8.2.4 Algorithmic results

We now consider the algorithmic aspects of the minor relations we have defined. In particular, we are concerned with the following decision problem:

$(\mathcal{G}, \preceq)$-MINOR	
Instance:	A directed graph $\mathcal{H}$
Problem:	Is $\mathcal{G} \preceq \mathcal{H}$?

In [RS95], it was shown that for undirected graphs and the standard minor relation, $(\mathcal{G}, \leq)$-MINOR is solvable in cubic time, so it is worth investigating if any of the minor relations we have defined enjoy a similar property. Unfortunately, we show that this is not the case unless NP = PTIME, as the problem is in general NP-complete for most of the relations we have defined.

Fortune, Hopcroft and Wyllie [FHW80] showed a dichotomy result for the directed subgraph homeomorphism problem for a fixed pattern graph $\mathcal{G}$. If $\mathcal{G}$ is a star, that is there is a unique source or sink which is the tail or head (respectively) of every edge, then deciding if a given graph with a given node mapping has a subgraph homeomorphic to $\mathcal{G}$ is solvable in polynomial time. Otherwise it is NP-complete. Not surprisingly, from Proposition 8.33, this result is partly applicable to $(\mathcal{G}, \dashv)$-MINOR. The difference is that in [FHW80] it is assumed that the node mapping was given. That is, they were asking if given a node mapping could be extended to a subgraph homeomorphism. The $(\mathcal{G}, \dashv)$-MINOR problem corresponds to the case when the node mapping is not given. This case was discussed in [FHW80] where it was observed that

firstly the polynomial time result carries over, as there are at most a polynomial number of node mappings, and secondly with some additional structure in the pattern graph, the node mapping required for NP-completeness can be forced to be the only possible node mapping, so the NP-completeness result holds for a large class of directed graphs (but not quite the complement of the star graphs). Summarizing their results in the terminology of this chapter gives us:

Theorem 8.38 ([FHW80])**.** *If $\mathcal{G}$ is a directed graph which is a star then $(\mathcal{G}, \dashv)$-*MINOR *is solvable in polynomial time.*

Theorem 8.39 ([FHW80])**.** *If $\mathcal{G}$ is a directed graph with at least four distinct vertices $\{v_1, v_2, v_3, v_4\}$ and edges (v_1, v_2) and (v_3, v_4) such that for $i \leq 4$ the degree of v_i is greater than 3 and different from the degree of v_j for $j \neq i$, then $(\mathcal{G}, \dashv)$-*MINOR *is* NP*-complete.*

Corollary 8.40. *Let $\preceq$ be a minor relation which includes $\dashv$ and let $\mathcal{G}$ be a directed graph which satisfies the requirements of Theorem 8.39. Then $(\mathcal{G}, \preceq)$-*MINOR *is* NP*-complete.*

Because of the additional structure required in the pattern graph to show NP-completeness when the node mapping is not specified, we no longer have the dichotomy result. Indeed it is an interesting problem to investigate the complexity of the problem when $\mathcal{G}$ is neither a star nor a directed graph satisfying the requirements of Theorem 8.39, for example if the maximum degree of any vertex in $\mathcal{G}$ is 3. This gives us the following problem for further investigation.

Open problem 8.41. *Characterize $\mathcal{G}$, $\preceq$ and the class of graphs $\mathcal{H}$ such that $(\mathcal{G}, \preceq)$-*MINOR *is NP-complete*

8.2.5 Well-quasi order results

We conclude this chapter by showing that only a few of the relations we have introduced can be used to generalize one of the most significant theorems associated with the minor relation: the *Graph Minor Theorem* of Robertson and Seymour [RS04]. Recalling the definition of a well-quasi order from Section 1.1.1, the theorem can be stated as:

Theorem 8.42 (Graph Minor Theorem [RS04])**.** *The minor relation is a well-quasi order.*

In particular this implies that for any infinite set of graphs there is a pair of graphs such that one is the minor of the other. From this, it follows that any family of graphs which is closed under the minor relation can be characterized by a finite list of *forbidden minors*. That is, if $\mathbb{F}$ is a family of graphs such that $\mathcal{H} \in \mathbb{F}$ and $\mathcal{G} \leq \mathcal{H}$ implies $\mathcal{G} \in \mathbb{F}$, then there exists a finite set of graphs $\{\mathcal{G}_1, \ldots, \mathcal{G}_m\}$ such that $\mathcal{G} \in \mathbb{F}$ if, and only if, $\mathcal{G}_i \not\leq \mathcal{G}$ for all $i \leq m$. Together with the observation that for a fixed graph $\mathcal{G}$, determining if $\mathcal{G}$ is a minor of a given graph can be decided in cubic time, this we obtain the following important algorithmic consequence.

Corollary 8.43 ([RS04])**.** *Let $\mathbb{F}$ be a minor-closed family of graphs. The problem of deciding if $\mathcal{G} \in \mathbb{F}$ can be computed in cubic time.*

Thus it is an interesting problem to see if we can generalize the Graph Minor Theorem to directed graphs. Unfortunately, for most of the minor relations we have defined, this is not the case.

Theorem 8.44. *$\trianglelefteq$ and $\ll^S$ are not well-quasi orders.*

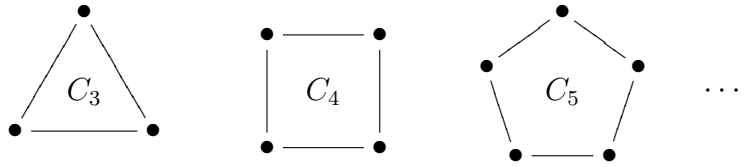


Figure 8.3: An infinite anti-chain for the $\trianglelefteq$ relation

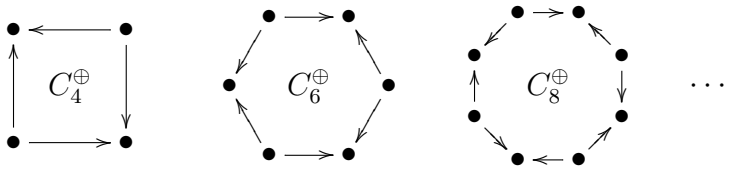


Figure 8.4: An infinite anti-chain for the $\ll^S$ relation

Proof. Consider the sequence of bidirected cycles $C_3, C_4, C_5, \ldots$ pictured in Figure 8.3. Using the same argument as in the proof of Theorem 8.32, Example (V), it is easy to see that $C_i \not\trianglelefteq C_j$ for $i < j$. Thus $\trianglelefteq$ is not a well-quasi order.

Now consider the sequence of graphs $C_4^{\oplus}, C_6^{\oplus}, \ldots$ pictured in Figure 8.4. It is easy to see that for all even $i \geq 4$, an edge in $C_i^{\oplus}$ can neither be butterfly contracted nor strong contracted, and the deletion of any edge results in a graph with an acyclic underlying graph. Thus for all $i < j$, $C_i^{\oplus} \not\ll^S C_j^{\oplus}$, and so $\ll^S$ is not a well-quasi order. □

Chapter 9

Conclusion and Future work

In this dissertation we examined the role of infinite games on finite graphs in two aspects of complexity: computational complexity and structural complexity. The research resolved some unanswered questions in the literature and opened up some interesting avenues for further research. We conclude this dissertation by recalling the major results established, and discussing possible areas for future study.

9.1 Summary of results

In Chapter 1 we stated the two main goals of this dissertation: to investigate the computational complexity of infinite games on finite graphs, and to use infinite games to define an algorithmically useful notion of structural complexity for directed graphs. The first of these goals was predominantly addressed in Chapters 2, 3, 6 and 7, while the second was catered for in Chapters 4 to 8. We now summarize the contribution each chapter made to each goal.

Complexity of Infinite Games

In Chapter 2 we considered the general class of infinite games on finite graphs. We introduced a generalization of bisimulation called *game simulation* which enables us to translate strategies from one game to another. We then introduced the notion of a *condition type*, which gives us a general framework for comparing many types of games which occur in the literature, for example *Muller games* [Mul63], *Rabin games* [Rab72], *Streett games* [Str82] and *parity games* [Mos91, EJ91]. The notion of *translatability* between condition types lets us compare the computational complexity of two games via the expressibility and succinctness of their winning conditions. We considered the computational complexity of deciding the winner in Muller games. We provided polynomial time algorithms for explicitly presented Muller games under various restrictions on the family of sets which specified the winning condition, namely simple games, and games where the condition is an anti-chain. We showed that deciding the winner of win-set games was PSPACE-complete. Following our work on translatability, it follows that the problems of deciding the winner of Muller games where the winning condition is specified as a Muller, Zielonka DAG, Emerson-Lei, or a circuit condition are all also PSPACE-complete, thus closing one of the open problems relating to the complexity of Muller games that we discussed in Chapter 1. We showed that the completeness results carries over to arenas of bounded tree-width for games specified by a Muller condition. We also gave examples of union-closed and

upward-closed games for which deciding the winner is co-NP-complete. We ended the chapter by showing how the lower bounds for deciding win-set games can be used to establish that the non-emptiness and model-checking problems for Muller automata are also PSPACE-complete, thereby resolving an open question in the field of automata theory.

Our foray into the sticky world of parity games began in Chapter 3, where we analysed one of the best performing algorithms for deciding parity games in an effort to establish tighter bounds on the running time. We interpreted the algorithm from a combinatorial perspective, in particular as a method for finding a global sink on an acyclic unique sink oriented hypercube. Using techniques from combinatorics, we improved the upper bound for the running time. We also provided an example which shows that the hypercube orientations resulting from parity games are not pseudomodular.

In Chapters 6 and 7, we demonstrated how the structure of the arena affects the complexity of deciding the winner of parity games. We used DAG-decompositions in Chapter 6 and Kelly-decompositions in Chapter 7 to produce two dynamic programming style algorithms for solving parity games. The upshot of such algorithms is that on a class of arenas of bounded DAG-width or bounded Kelly-width, there is a polynomial time algorithm for deciding the winner of a parity game. As DAG-width and Kelly-width encompass other graph parameters such as tree-width, this gives us the largest class of graphs so far known on which parity games can be solved in polynomial time.

Complexity by Infinite Games

In Chapter 4 we discussed the properties that a good measure of digraph structural complexity should have. We cited tree-width as an example to aspire towards, and discussed why tree-width is not suitable as a measure for directed graphs. We also discussed why the established notion of directed tree-width from [JRST01] is also not entirely suitable.

In Chapter 5 we introduced a framework for defining reasonable structural complexity measures via *graph searching games*, a form of the infinite games we have been considering. We showed how these games encompass many similar games in the literature, including those that can be used to characterize tree-width.

In Chapter 6 we used the work from Chapter 5 to define an extension of tree-width to directed graphs, *DAG-width*. Unlike directed tree-width and Kelly-width, the definition of a DAG-decomposition closely resembles tree decompositions. After showing that cop-monotonicity and robber-monotonicity coincide in this game, we showed that DAG-width is equivalent to the number of cops required to capture a visible robber with a monotone strategy, thereby demonstrating that it is a reasonable measure of structural complexity for directed graphs. We also showed that DAG-width defines an algorithmically useful complexity measure by showing that a number of problems, including deciding the winner of a parity game, can be solved in polynomial time on graphs of bounded DAG-width. We concluded the chapter by demonstrating that DAG-width is markedly different from three other measures defined in the literature: tree-width, directed tree-width and directed path-width.

In Chapter 7 we considered the generalization to directed graphs of three characterizations of tree-width: partial k-trees, elimination orderings and the cops and inert robber graph searching game. This results in *partial k-DAGs*, *directed elimination orderings*, and the cops and inert robber game for directed graphs. We showed that the graph parameters defined by these three generalizations were all equivalent, and these, in turn, were equivalent to the width of a

decomposition we introduced called a *Kelly-decomposition*. As with DAG-width, we demonstrated the algorithmic potential of Kelly-width by exhibiting polynomial time algorithms for a number of problems, including deciding the winner of a parity game, on graphs of bounded Kelly-width. We concluded the chapter by showing that, as with DAG-width, Kelly-width is quite different from tree-width, directed tree-width and directed path-width. However, its relation to DAG-width is somewhat more complex. We showed that, in the graph searching games which characterize DAG-width and Kelly-width, a monotone winning strategy for the cops in one game implies a winning strategy in the other (with possibly twice as many cops). Without a result in either game relating the number of cops required for a monotone strategy to the number of cops with a winning strategy, we are unable to compare DAG-width and Kelly-width directly. However, we do show that there are graphs on which DAG-width and Kelly-width differ (by an arbitrary amount).

Finally, in Chapter 8 we presented preliminary results towards a directed graph structure theory, based on the notions of structural complexity we have developed. We introduced generalizations of havens and brambles which appear to correspond with DAG-width and Kelly-width. The brambles for DAG-width are dual to the brambles for Kelly-width, suggesting that DAG-width and Kelly-width are very closely connected. We also considered the problem of extending the minor relation to directed graphs. We introduced a number of distinct relations ranging from the subgraph relation to the minor relation on the underlying undirected graphs. We showed that these relations do not enjoy the algorithmic properties of the minor relation, as deciding if a fixed subgraph is a minor of a given graph is, in general, NP-complete for most of the minor relations we considered. We concluded the chapter by showing that all except two of the minor relations we introduced contain infinite anti-chains. This implies that to consider a generalization of the Graph Minor Theorem using the minor relations we defined, we need to use either the anti-parallel D-minor or the strong D-minor relation.

9.2 Future work

The work we have presented in this dissertation raises a number of interesting questions and directions for further research. We now discuss some of these, roughly in the order they arose during the dissertation.

The exact complexity for deciding Muller games when the winning condition is explicitly presented remains open, as does the question for union-closed games with an explicitly presented winning condition. We saw in Theorem 2.62 that if the winning condition is an anti-chain then the game can be solved efficiently. Thus it is possible that the complexity of the former problem can be derived from the complexity of the latter. This would also be an interesting question to investigate.

The exact complexity for deciding parity games also remains an interesting open problem. Characterizing the acyclic unique sink orientations that arise from valuations in parity games could either establish a polynomial time algorithm for parity games, or give a super-polynomial lower bound for the strategy improvement algorithm.

Monotonicity questions frequently arise in the study of graph searching games. An interesting line of research would be to characterize the properties of graph searching games necessary for monotonicity to be sufficient. For example, extending the work of Fomin and Thilikos [FT03]. On a more specific level, for the cops and visible robber game on directed graphs an important open problem is finding a relation between the number of cops required

for a monotone winning strategy and the number of cops required for a winning strategy which is not necessarily monotone. Such a correspondence allows us to compare DAG-width with other parameters we have considered such as D-havens and Kelly-width. Similarly, finding a relation between the number of cops required for a robber-monotone winning strategy and the number of cops required for a not necessarily monotone winning strategy in the inert robber game allows us to compare Kelly-width to other measures.

Two important questions regarding the complexity of DAG-width and Kelly-width still remain open. First is the question of whether deciding if a digraph has DAG-width at most a given integer is in NP. Second is the question of whether, for a fixed k if deciding whether a digraph has Kelly-width at most k is decidable in polynomial time. An improved bound from $O(n^k)$ on the size of a DAG-decomposition of a graph would benefit the first question.

Finally, the preliminary work on a structure theory based on directed connectivity raises a number of interesting questions. For example, determining the precise relationship between DAG-width, Kelly-width, and initial and terminal brambles; characterizing the pattern graphs $\mathcal{G}$ for which $(\mathcal{G}, \preceq)$-MINOR is solvable in polynomial time; determining if any of the introduced minor relations is a well-quasi order; and characterizing classes of graphs via forbidden minors.

9.3 Conclusion

In conclusion, this dissertation has made a significant contribution towards the analysis of the complexity of infinite games and to the development of a notion of structural complexity for directed graphs, and opened up exciting possibilities for future research. We resolved the open questions regarding the exact complexity of deciding Muller games and Muller automata non-emptiness and model-checking, and we made substantial progress towards answering the question for parity games. We introduced two similar measures of structural complexity for directed graphs which appear to measure the *directed connectivity* of a digraph, a metric which lies between weak connectivity and strong connectivity and is distinct from both. We demonstrated their algorithmic benefits by providing efficient algorithms for problems not known to be decidable in polynomial time.

Bibliography

[ACP87] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embedding in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8:277–284, 1987.

[Adl05] Isolde Adler. Directed tree-width examples. To appear in *Journal of Combinatorial Theory (Series B)*, 2005.

[AP89] Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 23:11–24, 1989.

[Arn85] Stefan Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability — A survey. *BIT*, 25:2–33, 1985.

[Bar05] János Barát. Directed path-width and monotonicity in digraph searching. To appear in *Graphs and Combinatorics*, 2005.

[BDHK06] Dietmar Berwanger, Anuj Dawar, Paul Hunter, and Stephan Kreutzer. DAG-width and parity games. In *Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science*, pages 524–536, 2006.

[BFK+06] Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. On exact algorithms for treewidth. In *Proceedings of the European Symposium on Algorithms*, 2006.

[BG04] Dietmar Berwanger and Erich Grädel. Entanglement – A measure for the complexity of directed graphs with applications to logic and games. In *Proceedings of the 11th conference on Logic Programming and Automated Reasoning*, pages 209–223, 2004.

[BGHK95] Hans L. Bodlaender, John R. Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995.

[BL69] J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[Bod88] Hans L. Bodlaender. Dynamic programming on graphs of bounded treewidth. In *Proceedings of the 15th International Colloquium on Automata Languages and*

Programming, volume 317 of *Lecture Notes in Computer Science*, pages 105–118, 1988.

[Bod96] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[Bod97] Hans L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36, 1997.

[Bre67] R. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers*, VI:72–78, 1967.

[BRST91] Daniel Bienstock, Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a forest. *Journal of Combinatorial Theory (Series B)*, 52(2):274–283, 1991.

[BS91] Daniel Bienstock and Paul D. Seymour. Monotonicity in graph searching. *Journal of Algorithms*, 12:239–245, 1991.

[BSV03] Henrik Björklund, Sven Sandberg, and Sergei Vorobyov. On combinatorial structure and algorithms for parity games. Technical Report 002, Uppsala University, Department of Information Technology, January 2003.

[CLR96] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1996.

[Cou90] Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwan, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pages 193–242. Elsevier, 1990.

[Dam94] Mads Dam. CTL* and ECTL* as fragments of the modal mu-calculus. *Theoretical Computer Science*, 126(1):77–96, 1994.

[Die05] Reinhard Diestel. *Graph Theory*. Springer, 3rd edition, 2005.

[DJW97] Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, pages 99–110, 1997.

[DK05] Reinhard Diestel and Daniela Kühn. Graph minor hierarchies. *Discrete Applied Mathematics*, 145(2):167–182, 2005.

[DKT97] Nick D. Dendris, Lefteris M. Kirousis, and Dimitrios M. Thilikos. Fugitive-search games on graphs and related parameters. *Theoretical Computer Science*, 172(1-2):233–254, 1997.

[EF99] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer, 2nd edition, 1999.

[EJ88] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs (extended abstract). In *Proceedings for the 29th IEEE Symposium on Foundations of Computer Science*, pages 328–337, 1988.

[EJ91] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *Proceedings for the 32nd Annual Symposium on Foundations of Computer Science*, pages 368–377, 1991.

[EJS01] E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model checking for the μ-calculus and its fragments. *Theoretical Computer Science*, 258(1–2):491–522, 2001.

[EL85] E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time strikes back. In *Proceedings of the 12th Annual ACM Symposium on Principles of Programming Languages*, pages 84–96, 1985.

[FFN05] Fedor V. Fomin, Pierre Fraigniaud, and Nicholas Nisse. Nondeterministic graph searching: From pathwidth to treewidth. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 364–375, 2005.

[FG00] Fedor V. Fomin and Petr A. Golovach. Graph searching and interval completion. *SIAM Journal of Discrete Mathematics*, 13:454–464, 2000. (electronic).

[FHT04] Fedor V. Fomin, Pinar Heggernes, and Jan Arne Telle. Graph searching, elimination trees, and a generalization of bandwidth. *Algorithmica*, 41(2):73–87, 2004.

[FHW80] Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 1980.

[FT03] Fedor V. Fomin and Dimitrios M. Thilikos. On the monotonicity of games generated by symmetric submodular functions. *Discrete Applied Mathematics*, 131(2):323–335, 2003.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[GKP98] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, 1998.

[GL93] John R. Gilbert and Joseph W. H. Liu. Elimination structures for unsymmetric sparse LU factors. *SIAM Journal of Matrix Analysis and Applications*, 14:334–352, 1993.

[GLS01] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: Game theoretic and logical characterizations of hypertree width. In *Proceedings of the 20th Symposium on Principles of Database Systems*, pages 195–201, 2001.

[GM06] Martin Grohe and D. Marx. Constraint solving via fractional edge covers. In *Proceedings of the 17th Symposium on Discrete Algorithms*, pages 289–298, 2006.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[Hal76] Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 8(1–2):171–186, 1976.

[HD05] Paul Hunter and Anuj Dawar. Complexity bounds for regular games. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 495–506. Springer, 2005.

[HK07] Paul Hunter and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms*, New York, NY, USA, 2007. ACM Press.

[HS96] Jaakko Hintikka and Gabriel Sandu. A revolution in logic? *Nordic Journal of Philosophical Logic*, 1(2):169–183, 1996.

[HSLdW88] Peter L. Hammer, Bruno Simeone, Thomas M. Liebling, and Dominique de Werra. From linear separability to unimodality: A hierarchy of pseudo-boolean functions. *SIAM Journal of Discrete Mathematics*, 1(2):174–184, 1988.

[IK02] Hajime Ishihara and Bakhadyr Khoussainov. Complexity of some infinite games played on finite graphs. In *Proceedings of the 28th International Workshop on Graph Theoretical Concepts in Computer Science*, volume 2573 of *Lecture Notes in Computer Science*. Springer, 2002.

[JPZ06] Marcin Jurdziński, Mike Paterson, and Uri Zwick. A deterministic subexponential algorithm for solving parity games. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 117–123, 2006.

[JRST01] Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *Journal of Combinatorial Theory (Series B)*, 82(1):138–154, 2001.

[JRST02] Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Addendum to "Directed tree-width", 2002. www.math.gatech.edu/∼thomas/PAP/diradd.pdf.

[Jur00] Marcin Jurdziński. Small progress measures for solving parity games. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301, 2000.

[Kla94] Nils Klarlund. Progress measures, immediate determinacy, and a subset construction for tree automata. *Annals of Pure and Applied Logic*, 69(2-3):243–268, 1994.

[KO07] Stephan Kreutzer and Sebastian Ordyniak. Personal communication, April 2007.

[KP86] Lefteris M. Kirousis and Christos H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(3):205–218, 1986.

[LaP93] Andrea S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2):224–245, 1993.

[Liu90] Joseph W. H. Liu. The role of elimination trees in sparse factorization. *SIAM Journal of Matrix Analysis and Applications*, 11(1):134–172, 1990.

[LTMN02] Salvatore La Torre, Aniello Murano, and Margherita Napoli. Weak Muller acceptance conditions for tree automata. In *Proceedings of the 3rd International Workshop on Verification, Model Checking and Abstract Interpretation*, volume 2294 of *Lecture Notes in Computer Science*, pages 240–254. Springer, 2002.

[Mar75] Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102:363–375, 1975.

[McN66] Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966.

[McN93] Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

[Mos91] Andrzej Wlodzimierz Mostowski. Games with forbidden positions. Technical Report 78, Instytut Matematyki, Uniwersytet Gdański, Poland, 1991.

[MS99] Yishay Mansour and Satinder P. Singh. On the complexity of policy iteration. In *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30-August 1, 1999*, pages 401–408, 1999.

[MTV07] Daniel Meister, Jan Arne Telle, and Martin Vatshelle. Characterization and recognition of digraphs of bounded kelly-width. To appear in proceedings of WG, 2007.

[Mul63] David E. Muller. Infinite sequences and finite machines. In *Proceedings of the 4th IEEE Symposium on Switching Circuit Theory and Logical Design*, pages 3–16, 1963.

[NRY96] Anil Nerode, Jeffery B. Remmel, and Alexander Yakhnis. McNaughton games and extracting strategies for concurrent programs. *Annals of Pure and Applied Logic*, 78(1-3):203–242, 1996.

[Obd03] Jan Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In *Proceedings of 15th International Conference on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 80–92. Springer, 2003.

[Obd06] Jan Obdržálek. DAG-width: Connectivity measure for directed graphs. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 814–821, 2006.

[Pap95] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1995.

[Par78] T. D. Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, volume 642 of *Lecture Notes in Mathematics*, pages 426–441. Springer, 1978.

[Rab72] Michael O. Rabin. Automata on infinite objects and church's problem. *American Mathematical Society*, 1972.

[Ros70] Donald J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32:597–609, 1970.

[RS82] Arnold L. Rosenberg and Ivan Hal Sudborough. Bandwidth and pebbling. *Computing*, 31:115–139, 1982.

[RS83] Neil Robertson and Paul D. Seymour. Graph minors I: Excluding a forest. *Journal of Combinatorial Theory (Series B)*, 35:39–61, 1983.

[RS84] Neil Robertson and Paul D. Seymour. Graph minors III: Planar tree-width. *Journal of Combinatorial Theory (Series B)*, 36:49–63, 1984.

[RS95] Neil Robertson and Paul D. Seymour. Graph minors XIII: The disjoint path problem. *Journal of Combinatorial Theory (Series B)*, 63:65–110, 1995.

[RS04] Neil Robertson and Paul D. Seymour. Graph minors XX: Wagner's conjecture. *Journal of Combinatorial Theory (Series B)*, 92:325–357, 2004.

[RT75] Donald J. Rose and R. Endre Tarjan. Algorithmic aspects of vertex elimination. In *Proceedings of the seventh annual ACM Symposium on Theory of Computing*, pages 245–254, 1975.

[RT78] Donald J. Rose and R. Endre Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *SIAM Journal of Applied Mathematics*, 34(1):176–197, 1978.

[Saf05] Mohammad Ali Safari. D-width: A more natural measure for directed tree width. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 745–756. Springer, 2005.

[SS05] Ingo Schurr and Tibor Szabó. Jumping doesn't help in abstract cubes. In *Proceedings of the 11th International Integer Programming and Combinatorial Optimization Conference*, pages 225–235, 2005.

[ST93] Paul D. Seymour and Robin Thomas. Graph searching, and a min-max theorem for tree-width. *Journal of Combinatorial Theory (Series B)*, 58:22–33, 1993.

[Str82] Robert S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1-2):121–141, 1982.

[SW01] Tibor Szabó and Emo Welzl. Unique sink orientations of cubes. In *Proceedings for the 42nd Annual Symposium on Foundations of Computer Science*, pages 547–555, 2001.

[Tho02] Robin Thomas. Directed tree-width. Slides from a lecture at the Regional NSF-CBMS Conference on Graph Structure and Decomposition, 2002. www.math.gatech.edu/∼thomas/SLIDE/CBMS/dirtrsl.pdf.

[VJ00a] Jens Vöge and Marcin Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *Proceedings of 12th International Conference on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215. Springer, 2000.

[VJ00b] Jens Vöge and Marcin Jurdziński. A discrete strategy improvement algorithm for solving parity games. Technical Report 00-48, BRICS, 2000.

[WH88] Kathy Williamson Hoke. Completely unimodal numberings of a simple polytope. *Discrete Applied Mathematics*, 20:69–81, 1988.

[Yan97] M Yannakakis. Computational complexity. In Emile Aarts and Jan K. Lenstra, editors, *Local search in combinatorial optimization*, pages 19–55. Princeton University Press, 1997.

[Zie98] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200:135–183, 1998.