

# The DAG-Width of Directed Graphs

Dietmar Berwanger<sup>a</sup>, Anuj Dawar<sup>b</sup>, Paul Hunter<sup>c</sup>,  
Stephan Kreutzer<sup>c</sup>, Jan Obdržálek<sup>d</sup>

<sup>a</sup>*LSV, CNRS & ENS de Cachan, France*

<sup>b</sup>*University of Cambridge Computer Laboratory, UK*

<sup>c</sup>*Oxford University Computing Laboratory, UK*

<sup>d</sup>*Faculty of Informatics, Masaryk University, Brno, Czech Republic*

---

## Abstract

Tree-width is a well-known metric on undirected graphs that measures how tree-like a graph is and gives a notion of graph decomposition that proves useful in algorithm development. Tree-width can be characterised by a graph searching game where a number of cops attempt to capture a robber. We consider the natural adaptation of this game to directed graphs and show that monotone strategies in the game yield a measure that can be seen to describe how close a directed graph is to a directed acyclic graph (DAG). We also provide an associated decomposition and show how it is useful for developing algorithms on directed graphs. In particular, we show that the problem of determining the winner of a parity game is solvable in polynomial time on graphs of bounded DAG-width. We also consider the relationship between DAG-width and other measures of connectivity such as entanglement and directed tree-width. One consequence we obtain is that certain NP-complete problems such as Hamiltonicity and disjoint paths are polynomial-time computable on graphs of bounded DAG-width.

---

## 1 Introduction

The groundbreaking work of Robertson and Seymour in their graph minor project has focused much attention on tree decompositions of graphs and associated measures of graph connectivity such as tree-width [1]. Aside from their interest in graph-structure theory, these notions have also proved very useful in the development of algorithms. The tree-width of a graph is a measure of how tree-like the

---

*Email addresses:* [dwb@lsv.ens-cachan.fr](mailto:dwb@lsv.ens-cachan.fr) (Dietmar Berwanger),  
[anuj.dawar@cl.cam.ac.uk](mailto:anuj.dawar@cl.cam.ac.uk) (Anuj Dawar), [paul.hunter@comlab.ox.ac.uk](mailto:paul.hunter@comlab.ox.ac.uk) (Paul Hunter),  
[kreutzer@comlab.ox.ac.uk](mailto:kreutzer@comlab.ox.ac.uk) (Stephan Kreutzer), [obdrzalek@fi.muni.cz](mailto:obdrzalek@fi.muni.cz) (Jan Obdržálek).

graph is, and it is found that small tree-width allows for graph decompositions along which recursive algorithms can work. Many problems that are intractable in general can be solved efficiently on graphs of bounded tree-width. These include such classical NP-complete problems as finding a Hamiltonian cycle in a graph or detecting if a graph is three-colourable. Indeed, a general result of Courcelle [2] shows that any property definable in monadic second-order logic is solvable in linear time on graphs of fixed tree-width.

The idea of designing algorithms that work on tree decompositions of the input has been generalised from graphs to other kinds of structures. Usually the tree-width of a structure is defined as that of the underlying connectivity (or Gaifman) graph. For instance, the tree-width of a directed graph is simply that of the undirected graph we get by forgetting the direction of edges, a process which leads to some loss of information. This loss may be significant if the algorithmic problems we are interested in are inherently directed. A good example is the problem of detecting Hamiltonian cycles. While we know that this can be solved easily on graphs with small tree-width, there are also directed graphs with very simple connectivity structure which have large tree-width. A directed acyclic graph (DAG) is a particularly simple structure, but we lose sight of this when we erase the direction on the edges and find the underlying undirected graph to be dense. Several proposals have been made (see [3–7]) which extend notions of tree decompositions and tree-width to directed graphs. In particular, Johnson et al. [4] introduce the notion of *directed tree-width* where directed acyclic graphs have width 0 and they show that Hamiltonicity can be solved for graphs of bounded directed tree-width in polynomial time. However, the definition and characterisations of this measure are somewhat unwieldy and they have not, so far, resulted in many further developments in algorithms.

We are especially interested in one particular problem on directed graphs, that of determining the winner of a *parity game*. This is an infinite two-player game played on a directed graph where the vertices are labelled by priorities. The players take turns pushing a token along edges of the graph. The winner is determined by the parity of the least priority occurring infinitely often in this infinite play. Parity games have proved useful in the development of model-checking algorithms used in the verification of concurrent systems. The modal  $\mu$ -calculus, introduced in [8], is a widely used logic for the specification of such systems, encompassing a variety of modal and temporal logics. The problem of determining, given a system  $\mathcal{A}$  and a formula  $\varphi$  of the  $\mu$ -calculus, whether or not  $\mathcal{A}$  satisfies  $\varphi$  can be turned into a parity game (see [9]). The exact complexity of solving parity games is an open problem that has received a large amount of attention. It is known [10] that the problem is in  $\text{NP} \cap \text{co-NP}$  but no polynomial time algorithm is known. From the general result of Courcelle [2], it follows that there is a linear time algorithm that solves parity games with a fixed number of priorities on graphs of bounded tree-width. Obdržálek [11] exhibited a polynomial time algorithm for games with an arbitrary number of priorities on graphs of bounded tree-width. He points out that the algorithm would not give good bounds, for instance, on directed acyclic graphs

even though solving the games on such graphs is easy. He asks whether there is a structural property of directed graphs that would allow a fast algorithm on both bounded tree-width structures and on DAGs.

In this article, we give just such a generalisation. We introduce a measure of the connectivity of graphs that we call DAG-width. It is intermediate between tree-width and directed tree-width, in that for any graph  $G$ , the directed tree-width of  $G$  is no greater than its DAG-width which, in turn, is no greater than its tree-width. Thus, the class of structures of DAG-width  $k + 1$  or less includes all structures of tree-width  $k$  and more (in particular, DAGs of arbitrarily high tree-width all have DAG-width 1). This measure was introduced independently in two conference papers [12,13] to which the present paper is a follow-up.

The notion of DAG-width can be understood as a simple adaptation of the *cops and robber game* (which characterises tree-width) to directed graphs. The game is played by two players, one of whom controls a set of  $k$  cops attempting to catch a robber controlled by the other player. The cop player can move any set of cops to any vertices on the graph, while the robber can move along any path in the graph as long as there is no cop currently on the path. Such games have been extensively studied (see [14–17,5]). It is known [14] that the cop player has a winning strategy on an undirected graph  $G$  using  $k + 1$  cops if, and only if,  $G$  has tree-width  $k$ . We consider the natural adaptation of this game to directed graphs, by constraining the robber to move along directed paths. We show that the class of directed graphs where there is a monotone (in a sense we make precise) strategy for  $k$  cops to win is characterised by its width in a decomposition that is a generalisation of tree decompositions. We are then able to show that the problem of determining the winner of a parity game is solvable in polynomial time on the class of graphs of DAG-width  $k$ , for any fixed  $k$ .

In Section 2, we introduce some notation. Section 3 introduces the cops and robber game, DAG-decompositions and DAG-width and shows the equivalence between the existence of monotone winning strategies and DAG-width. Also in Section 3 we discuss some algorithmic aspects of DAG-width. Section 4 proves the existence of a polynomial time algorithm for solving parity games on such graphs, and Section 5 relates DAG-width to other measures of graph connectivity.

## 2 Preliminaries

We first fix some notation used throughout the article. All graphs used are finite, directed and simple unless otherwise stated.

We write  $\omega$  for the set of finite ordinals, i.e., natural numbers. For every  $n \in \omega$ , we write  $[n]$  for the set  $\{1, \dots, n\}$ . For a set  $V$  and a number  $k \in \omega$ , we write  $[V]^{\leq k}$

for the set of all  $X \subseteq V$  with  $|X| \leq k$ . Given sets  $A, B$ , we denote their symmetric difference  $(A \setminus B) \cup (B \setminus A)$  by  $A \Delta B$ .

Let  $G$  be a directed graph. We write  $V(G)$  for the set of its vertices and  $E(G)$  for the set of its edges. Let  $V \subseteq V(G)$  be a set of vertices. We write  $G[V]$  for the subgraph induced by  $V$ . Further,  $G^{op}$  is the graph with the same set of vertices as  $G$  and with a set of edges that results from reversing the edges in  $E(G)$ , i.e.,  $E(G^{op}) = \{(w, v) : (v, w) \in E(G)\}$ .

**Definition 1.** A *tree decomposition* of a graph  $G$  is a pair  $(T, \mathcal{X})$ , where  $T$  is a tree and  $\mathcal{X} = (X_t)_{t \in V(T)}$  is a family of subsets of  $V(G)$  such that

- $\bigcup_{t \in V(T)} X_t = V(G)$ ,
- for each edge  $(u, v) \in E(G)$ , there is a  $t \in V(T)$  such that  $\{u, v\} \subseteq X_t$ , and
- for each vertex  $v \in V(G)$ , the set  $\{t \in V(T) : v \in X_t\}$  forms a connected subtree of  $T$ .

The *width* of a tree decomposition is one less than the cardinality of the largest  $X_t$ . The *tree-width* of  $G$  is the smallest  $k$  such that  $G$  has a tree decomposition of width  $k$ .

Let  $D$  be a directed, acyclic graph (DAG). The partial order  $\preceq_D$  on  $D$  is the reflexive, transitive closure of  $E(D)$ . A *root* of a set  $X \subseteq V(D)$  is a  $\preceq_D$ -minimal element of  $X$ , that is,  $r \in X$  is a root of  $X$ , if there is no  $y \in X$  such that  $y \preceq_D r$  and  $y \neq r$ . Analogously, a *leaf* of  $X \subseteq V(D)$  is a  $\preceq_D$ -maximal element.

### 3 Games, Strategies and Decompositions

This section contains the graph-theoretical part of this article. We define DAG-width and its relation to graph searching games. As mentioned in the introduction, the notion of tree-width has a natural characterisation in terms of a cops and robber game. Directed tree-width has also been characterised in terms of such games [4], but these games appear to be less intuitive. In this article, we consider the straightforward extension of the cops and robber game from undirected graphs to directed graphs. We show that these games give a characterisation of the graph connectivity measure that we call DAG-width and introduce in Section 3.2. We comment on algorithmic properties in Section 3.3.

#### 3.1 Cops and robber games

The cops and robber game on a digraph is a game where  $k$  cops try to catch a robber who may run along paths in the digraph. While the robber is confined to moving

along paths in the graph, the cops may move to any vertex at any time. A formal definition follows.

**Definition 2** (Cops and robber game). Given a graph  $G := (V, E)$ , the  $k$ -cops and robber game on  $G$  is played between two players, the *cop* and the *robber* player, as follows:

- At the beginning, the cop player chooses  $X_0 \in [V]^{\leq k}$ , and the robber player chooses a vertex  $r_0$  of  $V$ , giving position  $(X_0, r_0)$ .
- From position  $(X_i, r_i)$ , if  $r_i \notin X_i$  then the cop player chooses  $X_{i+1} \in [V]^{\leq k}$ , and the robber player chooses a vertex  $r_{i+1}$  such that there is a directed path from  $r_i$  to  $r_{i+1}$  which does not include any vertex in  $X_i \cap X_{i+1}$ . If  $r_i \in X_i$ , then the cops have captured the robber and the robber player loses.

A *play* in the game is a (finite or infinite) sequence  $\pi := (X_0, r_0), (X_1, r_1), \dots$  of positions such that the transition from  $(X_i, r_i)$  to  $(X_{i+1}, r_{i+1})$  is a valid move by the rules above and such that the play is finite if, and only if,  $r_n \in X_n$  for the final position  $(X_n, r_n)$ . A play is winning for the robber player, if it is infinite.

**Definition 3** (Cop number). The *cop number* of a directed graph  $G$  is the least  $k$  such that the cop player has a strategy to win the  $k$ -cops and robber game on  $G$ .

Variants of the game where the robber moves first, or 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 cop number of a graph does not depend on the variant.

Before we introduce the technical aspects of these games needed in later sections, we present a couple of results that illustrate some of their properties.

**Lemma 4.** *The cop number of any finite, non-empty, directed graph  $G$ , is at least 1 and it equals 1 if, and only if,  $G$  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 on that vertex. If  $G$  is acyclic, then one cop can catch the robber by always playing to the robber's current position. Eventually, the robber will not be able to move and the cop will capture him. Conversely, if  $G$  has a cycle, then the robber can defeat one cop by forever staying in the cycle.  $\square$

Games similar to the one defined above have been used to give game characterisations of measures such as tree-width [14] and directed tree-width [4]. In Section 5 we investigate this in more detail. One property of these measures is that they are invariant under edge reversal, that is, they do not change if the directions of the edges of the graph are reversed. As we see below, this is not the case for the game we have defined. One exception is graphs of cop number 1, that is, acyclic graphs.

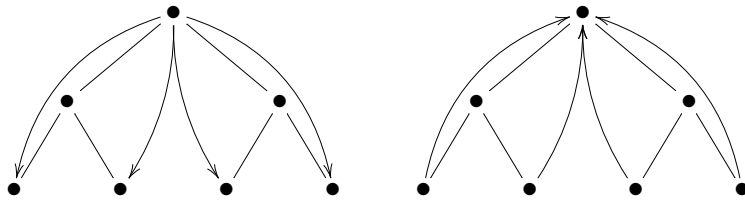


Fig. 1. The graphs  $T_3^2$  (cop number 2) and  $(T_3^2)^{op}$  (cop number 3)

**Proposition 5.** *The cop number of a directed graph  $G$  is 1 if, and only if, the cop number of  $G^{op}$  is 1.*

*Proof.* This follows from Lemma 4 by observing that  $G$  is acyclic if, and only if,  $G^{op}$  is acyclic.  $\square$

**Proposition 6.** *For any  $j, k$  with  $2 \leq j \leq k$ , there exists a graph  $T_k^j$  with cop number  $j$  such that the cop number of  $(T_k^j)^{op}$  is  $k$ .*

*Proof.* Let  $T_k^j$  be a binary branching tree of height  $k$  (i.e., with  $2^k - 1$  vertices) with edges oriented away from the root. In addition, every vertex  $v$  has a “forward-edge”<sup>1</sup> to each of its descendants, i.e., the forward edges form the transitive closure of the original tree-edges, and a “back-edge” to its  $j - 1$  nearest ancestors. Figure 1 illustrates the case for  $k = 3$  and  $j = 2$ . Here, as with Figure 2, undirected edges represent a pair of anti-parallel edges.

To show that the cop number of  $T_k^j$  is  $j$ , first we see that  $j$  cops have a winning strategy by initially playing on the root then following the robber down, in a leap-frogging manner, whichever subtree he plays in. To defeat  $j - 1$  cops, the robber chooses any leaf. Whenever a cop moves to that leaf, a simple counting argument shows that there must be at least one unoccupied ancestor with at least one clear path to a leaf below. The robber then plays to that ancestor and along that path to the leaf.

For  $(T_k^j)^{op}$ , the robber is captured by  $k$  cops occupying the root of whichever subtree he is in. To defeat  $k - 1$  cops, the robber plays the strategy that defeats  $j - 1$  cops in  $T_k^j$ .  $\square$

As always when dealing with games we are less interested in a single play of the game as in strategies that allow a player to win every play of the game. We first start by looking at the winning strategies for the robber player. One way of describing a winning strategy for the robber is given by the following proposition.

<sup>1</sup> To aid the description of strategies, we prefer to view  $T_k^j$  as a directed binary branching tree with some additional structure. But note that there is no actual difference between edges, forward-edges, and back-edges

**Proposition 7.** A robber can defeat  $k$  cops on a graph  $G$  if, and only if, there exists a function  $\sigma$  mapping each  $X \in [V(G)]^{\leq k}$  to a nonempty union of strongly connected components of  $G \setminus X$  such that if  $X \subseteq Y \in [V(G)]^{\leq k}$  then:

- (1)  $\sigma(X) \supseteq \sigma(Y)$ , and
- (2) For all  $x \in \sigma(X)$ , there is a  $y \in \sigma(Y)$  such that there is a directed path from  $x$  to  $y$  in  $G \setminus X$ .

*Proof.* If such a function  $\sigma$  exists, the robber can remain uncaptured by occupying any element of  $\sigma(X)$  when the cops occupy  $X$ . Conversely, suppose that the robber has a winning strategy against  $k$  cops. Then for each  $X \in [V(G)]^{\leq k}$  let  $\sigma(X)$  be the set of all vertices from which the robber can guarantee a win when the cops occupy  $X$ . Then it is easy to show that  $\sigma$  satisfies the proposition.  $\square$

The function  $\sigma$  above plays the same role as havens for tree-width or directed tree-width. The main difference is that  $\sigma(X)$  need not be connected. Indeed, as Figure 2 shows, there are examples where  $\sigma(X)$  is not connected. A robber can defeat 2 cops on this graph, yet for any appropriate function  $\sigma$ ,  $\sigma(\{b, c\})$  must be precisely  $\{a, d\}$ .

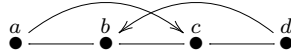


Fig. 2. Graph showing havens may not be connected

Winning strategies for the cop player play a central role throughout this article. We therefore give a precise definition of this notion.

**Definition 8.** Let  $G := (V, E)$  be a directed graph.

- (i) A ( $k$ -cop) strategy for the cop player is a function  $f$  from  $[V]^{\leq k} \times V$  to  $[V]^{\leq k}$ . A play  $(X_0, r_0), (X_1, r_1), \dots$  is consistent with a strategy  $f$  if  $X_{i+1} = f(X_i, r_i)$  for all  $i$ . The strategy  $f$  is called a winning strategy, if every play consistent with  $f$  is winning for the cop player.
- (ii) A strategy for the cop player is cop-monotone if in playing the strategy, no vertex is visited twice by cops. That is, if  $(X_0, r_0), (X_1, r_1), \dots$  is a play consistent with the strategy, then for every  $0 \leq i < n$  and  $v \in X_i \setminus X_{i+1}$ , we have  $v \notin X_j$  for all  $j > i$ .
- (iii) A strategy for the cop player is robber-monotone if in playing the strategy, the set of vertices reachable by the robber is non-increasing.

**Lemma 9.** If the cop player has a cop-monotone or robber-monotone winning strategy then he also has a winning strategy that is both cop- and robber-monotone.

*Proof.* Suppose the cop player has a robber-monotone winning strategy, and let  $(X_0, r_0), (X_1, r_1), \dots$  be a play 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  $X_i \supseteq X_{i+1}$  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  $X_{i+1}$  (i.e., not on  $v$ ). 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  $(X_0, r_0), (X'_1, r_1), \dots$ . The strategy which takes  $(X'_i, r_i)$  to  $X'_{i+1}$  is cop-monotone for this play. Repeating this for all plays (i.e., every choice for robber) results in a cop-monotone strategy. Hence, whenever the cop player has a robber-monotone winning strategy he also has a cop-monotone strategy.

We show next that any cop-monotone winning strategy for the cop player is actually robber-monotone also. This proves the lemma. Suppose the cop player has a cop-monotone winning strategy. Let  $(X_0, r_0), (X_1, r_1), \dots$  be a play consistent with the strategy, and  $R_0, R_1, \dots, R_n$  be the corresponding robber space, i.e., the set of vertices reachable for the robber. By the definition of the game, we can assume the strategy alternates placing and removing (possibly no) cops. Clearly we only need to consider the action of removing the cops, i.e., the case where  $X_i \supseteq X_{i+1}$ , as adding cops can only reduce the robber space. Let  $v \in X_i \setminus X_{i+1}$ . As  $v \notin X_j$  for all  $j > i$ , the robber is unable to reach  $v$  otherwise he could play to  $v$  and sit there indefinitely, contradicting the assumption that cop player is playing a winning strategy. Thus, the robber is unable to reach any of the vertices in  $X_i \setminus X_{i+1}$  and is therefore unable to reach any new vertices. Hence  $R_i \supseteq R_{i+1}$ , so the strategy is robber-monotone.  $\square$

With this lemma in mind we define a *monotone winning strategy* in the obvious way.

### 3.2 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, the cop player has a monotone winning strategy in the  $k$ -cops and robber game played on that graph. We conclude with some properties enjoyed by DAG-width.

**Definition 10.** Let  $G := (V, E)$  be a graph. A set  $W \subseteq V$  *guards* a set  $V' \subseteq V$  if whenever there is an edge  $(u, v) \in E$  such that  $u \in V'$  and  $v \notin V'$ , then  $v \in W$ .

**Definition 11** (DAG-decomposition). Let  $G := (V, E)$  be a directed graph. A DAG-decomposition of  $G$  is a pair  $(D, \mathcal{X})$  where  $D$  is a DAG and  $\mathcal{X} = (X_d)_{d \in V(D)}$  is a family of subsets of  $V(G)$  such that

- (D1)  $\bigcup_{d \in V(D)} X_d = V$ .
- (D2) For all vertices  $d \preceq_D d' \preceq_D d''$ ,  $X_d \cap X_{d''} \subseteq X_{d'}$ .
- (D3) For all edges  $(d, d') \in E(D)$ ,  $X_d \cap X_{d'}$  guards  $X_{\succeq d'} \setminus X_d$ , where  $X_{\succeq d'}$  stands for  $\bigcup_{d'' \succeq_D d'} X_{d''}$ . For any root  $d$ ,  $X_{\succeq d}$  is guarded by  $\emptyset$ .

The width of a DAG-decomposition  $(D, \mathcal{X})$  is defined as  $\max\{|X_d| : d \in V(D)\}$ . 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 12.** *For any graph  $G$ , there is a DAG-decomposition of  $G$  of width  $k$  if, and only if, the cop player has a monotone winning strategy in the  $k$ -cops and robber game on  $G$ .*

To prove this, we first need some observations about guarding.

**Lemma 13.**

- (i) If  $X_0$  guards  $Y_0$  and  $X_1$  guards  $Y_1$ , then  $X_0 \cup X_1$  guards  $Y_0 \cup Y_1$ .
- (ii) If  $X$  guards  $Y$  and  $Z \supseteq X$ , then  $Z$  guards  $Y$ .
- (iii) If  $X$  guards  $Y$  then  $X \cup Z$  guards  $Y \setminus Z$ .

*Proof.* (i). Suppose  $(v, w) \in E(G)$ ,  $v \in Y_0 \cup Y_1$  and  $w \notin Y_0 \cup Y_1$ . Let  $v \in Y_i$ , then  $w \in X_i$ , as  $X_i$  guards  $Y_i$ . Hence  $w \in X_0 \cup X_1$ , and  $X_0 \cup X_1$  guards  $Y_0 \cup Y_1$ .

(ii). Suppose  $(v, w) \in E(G)$ ,  $v \in Y$  and  $w \notin Y$ . As  $X$  guards  $Y$ ,  $w \in X$ . As  $Z \supseteq X$ ,  $w \in Z$ . Therefore,  $Z$  guards  $Y$ .

(iii). Suppose  $(v, w) \in E(G)$ ,  $v \in Y \setminus Z$  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  $w \in X \cup Z$ .  $\square$

We now turn to the proof of Theorem 12. Suppose the cop player has a monotone winning strategy  $f$  in the  $k$ -cops and robber game on a graph  $G$ . We can assume that the first move defined by  $f$  is to place no cops. We can also assume, as  $f$  is (cop-)monotone, that cops are only ever placed on vertices that are reachable by the robber. That is,

$$f(X, r) \subseteq X \cup \text{Reach}_{G \setminus X}(r). \quad (1)$$

First, we capture the  $k$ -cops and robber game on  $G$  by a simpler token-moving game. This game is played on a graph  $\Pi_k(G)$  where  $V(\Pi_k(G)) = [V(G)]^{\leq k} \times V(G)$  and  $E(\Pi_k(G))$  consists of all pairs  $((X, r), (X', r'))$  with the property that  $r' \in \text{Reach}_{G \setminus (X \cap X')}(r)$ . That is,  $((X, r), (X', r'))$  is an edge if going from  $(X, r)$  to

$(X', r')$  is a legal move in the  $k$ -cops and robber game. The  $k$ -cops and robber game can then be seen as a game in which the robber player moves a token around  $\Pi_k(G)$ , along edges nominated by the cop player. A slight variant of this game (in which the two players alternately move the token on a larger graph) is used in Theorem 26.

Let  $D$  be the subgraph of  $\Pi_k(G)$  obtained by restricting  $\Pi_k(G)$  to nodes from which the cop player wins playing  $f$  and to edges  $\left((X, r), (X', r')\right)$  such that  $X' = f(X, r)$ , i.e., edges which are consistent with  $f$ . As  $f$  is a winning strategy for the cop player,  $D$  is an acyclic subgraph of  $\Pi_k(G)$ . We call this the *strategy DAG* defined by  $f$ . Note that the nodes of  $D$  are positions in the cops and robber game. Hence, the function  $f$  is well defined for all  $d \in V(D)$ . We claim that  $(D, \mathcal{X})$ , with  $\mathcal{X}$  determined by  $X_d = f(d)$  for all  $d \in V(D)$ , is a DAG-decomposition of  $G$  of width  $\leq k$ . To support our claim, we first observe the following simple facts. For  $d = (X, r) \in V(D)$ ,

$$\text{Reach}_{G \setminus X}(r) \subseteq \bigcup_{d \preceq_D d'} f(d') \subseteq X \cup \text{Reach}_{G \setminus X}(r).$$

The first inclusion follows from the fact that  $f$  is a winning strategy for the cop player: at position  $(X, r)$  every vertex reachable by the robber ( $\text{Reach}_{G \setminus X}(r)$ ) will be occupied by a cop at some point in the future. The second inclusion follows from repeated application of (1). Further, for  $d = (X, r) \in V(D)$ ,

$$\text{Reach}_{G \setminus X}(r) = \text{Reach}_{G \setminus (X \cap f(X, r))}(r). \quad (2)$$

As  $X \cap f(X, r) \subseteq X$ ,  $\text{Reach}_{G \setminus X}(r) \subseteq \text{Reach}_{G \setminus (X \cap f(X, r))}(r)$ . The reverse inclusion follows from the fact that  $f$  is a robber-monotone strategy.

The two equations together imply for  $d = (X, r)$ :

$$\left( \bigcup_{d \preceq_D d'} f(d') \right) \setminus X = \text{Reach}_{G \setminus (X \cap f(X, r))}(r). \quad (3)$$

We now show that  $(D, \mathcal{X})$  is indeed a DAG-decomposition of width  $\leq k$ . As observed above,  $D$  is a DAG, otherwise  $f$  would not be a winning strategy for the cop player. For (D1), if there was a  $v \in V \setminus \bigcup_{d \in V(D)} X_d$ , then, as we assumed  $f$  initially placed no cops, the robber could defeat  $f$  by playing to  $v$  and staying there indefinitely. Hence  $\bigcup_{d \in V(D)} X_d = V$ . (D2) follows immediately from the (cop-)monotonicity of the winning strategy  $f$ . Towards establishing (D3), let us first consider a root  $d = (X, r)$  of  $D$ . As we assumed that the first move of  $f$  is to place no cops,  $X$  must be  $\emptyset$ . By (3),

$$X_{\succeq d} = \left( \bigcup_{d \preceq_D d'} f(d') \right) = \text{Reach}_G(r)$$

is therefore guarded by  $\emptyset$ . Finally suppose  $(d, d') \in E(D)$ . If  $d' = (X', r')$  then  $X_d = f(d) = X'$ . So by (3),

$$X_{\succeq d'} \setminus X_d = \left( \bigcup_{d' \preceq_D d''} f(d'') \right) \setminus X' = \text{Reach}_{G \setminus (X' \cap f(X', r'))}(r').$$

Therefore,  $X_d \cap X_{d'} = X' \cap f(X', r')$  guards  $X_{\succeq d'} \setminus X_d$ . It follows that  $(D, \mathcal{X})$  is a DAG-decomposition. To see that it has width  $\leq k$ , note that  $\max\{|X_d| : d \in V(D)\} = \max\{|f(d)| : d \in V(D)\} \leq k$ .

Conversely, let  $(D, \mathcal{X})$  be a DAG-decomposition of width  $k$ . A strategy for  $k$  cops can then be defined as:

- (1) Let the robber choose a vertex  $v \in V$ . From (D1), there exists  $d_v \in V(D)$  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 13(ii),  $X_d$  guards  $X_{\succeq d} \setminus X_d$ . Therefore, the robber can only move to vertices in  $X_{\succeq d} \setminus X_d$ . Suppose the robber moves to  $v' \in X_{d''}$ . Let  $d'$  be a child 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  $X_{\succeq d'} \setminus X_d$ , the robber can only move to vertices in  $X_{\succeq d'}$  – that is, the robber must remain in the sub-DAG rooted at  $d'$ .
- (6) Return to step 2 with  $d'$  as  $d$ .

As  $D$  is a DAG, at some point the robber player will not be able to move (since  $X_{\succeq d} \setminus X_d$  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 12.  $\square$

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 4 and Proposition 6 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.

For the remaining results, we require the following useful observations.

**Lemma 14.** *Let  $(D, \mathcal{X})$  be a DAG-decomposition. For all  $(d, d') \in E(D)$ ,*

$$X_{\succeq d'} \setminus X_d = X_{\succeq d'} \setminus (X_d \cap X_{d'}).$$

*Proof.* As  $X_d \cap X_{d'} \subseteq X_d$ ,  $X_{\succeq d'} \setminus X_d \subseteq X_{\succeq d'} \setminus (X_d \cap X_{d'})$ . Conversely, suppose  $x \in X_{\succeq d'}$ , that is,  $x \in X_{d''}$  for some  $d'' \succeq_D d'$ . We will show that  $x \in X_d \cap X_{d'}$ , or  $x \notin X_d$ . Suppose  $x \in X_d$ . Then as  $d \preceq_D d' \preceq_D d''$ ,  $x \in X_d \cap X_{d''} \subseteq X_{d'}$ . Hence  $x \in X_d \cap X_{d'}$ . Thus,  $X_{\succeq d'} \setminus X_d \supseteq X_{\succeq d'} \setminus (X_d \cap X_{d'})$ .  $\square$

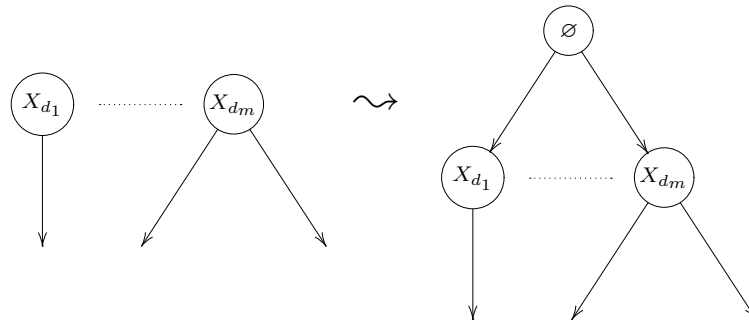


Fig. 3. Forming a unique root

**Lemma 15.** *Let  $(D, \mathcal{X})$  be a DAG-decomposition of a graph  $G$ . For  $W \subseteq V(G)$ ,  $(D, \mathcal{X}|_W)$  with  $\mathcal{X}|_W := (X_d \cap W)_{d \in V(D)}$  is a DAG-decomposition of  $G[W]$ .*

*Proof.* Clearly, (D1) and (D2) still hold for  $(D, \mathcal{X}|_W)$ . For (D3), we observe that, if  $X$  guards  $Y$  in  $G$ , then  $X \cap W$  guards  $Y \cap W$  in  $G[W]$ . This is because, if  $v \in Y \cap W$ ,  $w \in W \setminus Y$  and  $(v, w) \in E(G)$ , then  $w \in X$  (as  $X$  guards  $Y$ ), hence  $w \in X \cap W$ . Then, (D3) follows immediately from (D3) for the original decomposition  $(D, \mathcal{X})$ .  $\square$

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 [18].

**Definition 16.** A DAG-decomposition  $(D, (X_d)_{d \in V(D)})$  is *nice* if

- (N1)  $D$  has a unique root.
- (N2) Every  $d \in V(D)$  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  $|X_{d_0} \triangle X_{d_1}| = 1$ .

We show next 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. First we formalise 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.

**Lemma 17 (Unique root).** *Let  $(D, \mathcal{X})$  be a DAG-decomposition of width  $k$ , and let  $d_1, d_2, \dots, d_m$  be the roots of  $D$ . Then, the pair  $(D', \mathcal{X}')$  with*

- (i)  $V(D') := V(D) \dot{\cup} \{r\}$ ,
- (ii)  $E(D') := E(D) \cup \{(r, d_i) : 1 \leq i \leq m\}$ ,
- (iii)  $X'_r := \emptyset$ , and  $X'_d = X_d$ , for all other  $d \in V(D')$ ,

*is a DAG-decomposition of width  $k$ .*

*Proof.* As we have only added edges from  $r \notin V(D)$ , the graph  $D'$  is acyclic.

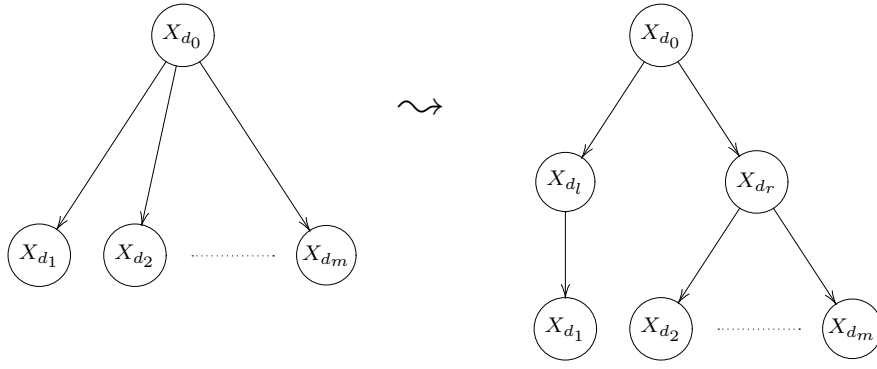


Fig. 4. Splitting at  $d_0$

Condition (D1) is trivially satisfied as we have only added a node. If  $d \preceq_{D'} d' \preceq_{D'} d''$ , then either  $d = r$  in which case  $X'_d \cap X'_{d''} = \emptyset \subseteq X'_{d'}$ , or  $d \in V(D)$ , in which case  $X'_d \cap X'_{d''} \subseteq X'_{d'}$  follows from the fact that  $(D, \mathcal{X})$  is a DAG-decomposition. This establishes (D2). For any root  $r$ , (D3) is again trivially satisfied, as  $X'_{\succeq r} = V(G)$ . Further, for  $(r, d_i) \in E(D')$ ,  $X'_r \cap X'_{d_i} = \emptyset$  guards  $X'_{\succeq d_i} = X'_{\succeq d_i} \setminus X'_r$ . Otherwise  $(d, d') \in E(D)$  and (D3) follows from the fact that  $(D, \mathcal{X})$  is a DAG-decomposition. Since  $|X'_r| = 0$ ,  $(D', \mathcal{X}')$  has width  $k$ .

Figure 3 gives a visual representation of the construction.  $\square$

**Definition 18 (Splitting).** Let  $(D, \mathcal{X})$  be a DAG-decomposition, and suppose  $d_0 \in V(D)$  has  $m > 1$  successors  $d_1, d_2, \dots, d_m$ . The decomposition  $(D', \mathcal{X}')$  obtained from  $(D, \mathcal{X})$  by *splitting*  $d_0$  is defined as follows:

- (i)  $V(D') = V(D) \dot{\cup} \{d_l, d_r\}$ ,
- (ii)  $E(D') = \left( E(D) \setminus \{(d_0, d_i) : 1 \leq i \leq m\} \right) \cup \{(d_0, d_l), (d_0, d_r), (d_l, d_1)\} \cup \{(d_r, d_i) : 2 \leq i \leq m\}$ , and
- (iii)  $X'_d = X_d$ , for all  $d \in V(D)$ , and  $X'_{d_l} = X'_{d_r} = X_{d_0}$ .

Figure 4 gives a visual representation of this transformation.

**Lemma 19.** Let  $(D, \mathcal{X})$  be a DAG-decomposition of a graph  $G$  of width  $k$ , and suppose  $d_0 \in V(D)$  has  $m > 1$  successors  $d_1, d_2, \dots, d_m$ . Then  $(D', \mathcal{X}')$  obtained from  $(D, \mathcal{X})$  by *splitting*  $d_0$  is a DAG-decomposition of  $G$  of width  $k$ .

*Proof.* First we observe that, as  $d_0$  is the unique predecessor of  $d_l$  and  $d_r$ , for any  $d \in V(D)$  such that  $d \prec_{D'} d_l$  or  $d \prec_{D'} d_r$ , it must be the case that  $d \preceq_D d_0$ . Thus, for all  $d \in V(D)$ ,

$$X'_{\succeq d} = \bigcup_{d \preceq_{D'} d'} X'_{d'} = \bigcup_{d \preceq_D d'} X_{d'} = X_{\succeq d},$$

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 true that  $X_{d_0} \cap X_{d_i}$  guards  $X_{\succeq d_i} \setminus X_{d_0}$ . Therefore, by Lemma 13(ii),

$$X_{d_0} \text{ guards } X_{\succeq d_i} \setminus X_{d_0}. \quad (4)$$

It is easily seen that the edges added do not create any cycles, so  $D'$  is a DAG. Further,  $\bigcup_{d \in V(D')} X'_d = \bigcup_{d \in V(D)} X_d = V(G)$ . To prove the connectivity condition (D2), let  $d, d', d'' \in V(D')$ , be such that  $d \preceq_{D'} d' \preceq_{D'} d''$ . If  $d' = d$  or  $d'' = d$  then trivially  $X'_d \cap X'_{d''} \subseteq X'_{d'}$ , so suppose  $d \prec_{D'} d' \prec_{D'} d''$ . 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  $(D, \mathcal{X})$  is a DAG-decomposition.
- If  $d$  is  $d_l$  or  $d_r$  then since all descendants of  $d$  are in  $V(D)$ , and  $d_0 \in V(D)$  is the unique predecessor of  $d$ , we obtain the following chain of nodes in  $D$ :  $d_0 \prec_D d' \prec_D d''$ . 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,  $d \prec_D d' \preceq_D d_0$ . 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,  $d \preceq_D d_0 \prec_D d''$ . 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  $D'$  is a root of  $D$  too. So  $\emptyset$  guards  $X_{\succeq d} = X'_{\succeq d}$ . Now let  $(d, d') \in E(D')$ . We consider three cases:

- $d' \in V(D)$  (i.e.,  $d' \neq d_l, d_r$ ). If  $d = d_l$  or  $d_r$ , then  $X'_d = X_{d_0}$ . Otherwise  $(d, d') \in E(D)$ . In both cases,  $X'_d \cap X'_{d'}$  guards  $X'_{\succeq d'} \setminus X'_d$ .
- $d' = d_l$  (so  $d = d_0$ ). Here  $X'_{\succeq d'} = X_{d_0} \cup X_{\succeq d_1}$ , so  $X'_{\succeq d'} \setminus X'_d = X_{\succeq d_1} \setminus X_{d_0}$ . Hence, by (4),  $X_{d_0} = X'_d \cap X'_{d'}$  guards  $X_{\succeq d_1} \setminus X_{d_0} = X'_{\succeq d'} \setminus X'_d$ .
- $d' = d_r$  (so  $d = d_0$ ). Here  $X'_{\succeq d'} = X_{d_0} \cup \bigcup_{2 \leq i \leq m} X_{\succeq d_i}$ , and so  $X'_{\succeq d'} \setminus X'_d = (\bigcup X_{\succeq d_i}) \setminus X_{d_0} = \bigcup (X_{\succeq d_i} \setminus X_{d_0})$ , where the unions are taken over  $i$  for  $2 \leq i \leq m$ . From Lemma 13(i) and (4),  $X'_d \cap X'_{d'} = X_{d_0}$  guards  $\bigcup_{2 \leq i \leq m} (X_{\succeq d_i} \setminus X_{d_0}) = X'_{\succeq d'} \setminus X'_d$ .

As  $X'_{d_l} = X'_{d_r} = X_{d_0}$ , we have

$$\max\{|X'_d| : d \in V(D')\} = \max\{|X_d| : d \in V(D)\} = k.$$

Consequently, the decomposition  $(D', \mathcal{X}')$  has width  $k$ .  $\square$

By the *decomposition resulting from splitting  $d$   $m - 1$  times* we mean the decomposition resulting from splitting  $d$ , and then recursively splitting the successor of  $d$  with more than one successor until no such successor exists. A *complete split* of

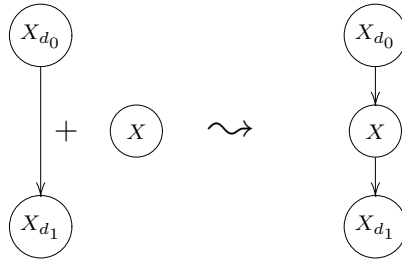


Fig. 5. Adding  $X$  to  $(d_0, d_1)$

$(D, \mathcal{X})$  is the decomposition  $(D', \mathcal{X}')$  obtained by recursively splitting every node with more than two children.

**Definition 20** (Adding). Let  $(D, \mathcal{X})$  be a DAG-decomposition of a graph  $G$ . If  $(d_0, d_1) \in E(D)$  and  $X \subseteq V(G)$  the *decomposition resulting from adding  $X$  to  $(d_0, d_1)$*  is the pair  $(D', \mathcal{X}')$  with

- (i)  $V(D') = V(D) \dot{\cup} \{d_X\}$
- (ii)  $E(D') = (E(D) \setminus \{(d_0, d_1)\}) \cup \{(d_0, d_X), (d_X, d_1)\}$
- (iii)  $X'_{d_X} = X$ , and for all  $d \in V(D)$ ,  $X'_d = X_d$ .

See Figure 5 for a visual interpretation.

**Lemma 21.** Let  $(D, \mathcal{X})$  be a DAG-decomposition of a graph  $G$  of width  $k$  and let  $(D', \mathcal{X}')$  be the decomposition resulting from adding  $X \subseteq V(G)$  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  $(D', \mathcal{X}')$  is a DAG-decomposition of  $G$  of width  $k$ .

*Proof.* We observe that for all  $d \in V(D)$ , if  $d \prec_{D'} d_X$ , then, as  $d_0 \in V(D)$  is the unique predecessor of  $d_X$ , we have  $d \preceq_D d_0$ , and if  $d_X \prec_{D'} d$ , then as  $d_1 \in V(D)$  is the unique successor of  $d_X$ , we have  $d_1 \preceq_D d$ . This implies, for all  $d \in V(D)$

$$X'_{\succeq d} = \bigcup_{d \preceq_{D'} d'} X'_{d'} = \bigcup_{d \preceq_D d'} X_{d'} = X_{\succeq d},$$

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  $X_{\succeq d_1} \setminus X_{d_0} = X_{\succeq d_1} \setminus (X_{d_0} \cap X_{d_1})$  from Lemma 14.

Clearly,  $D'$  is a DAG. We now show that  $(D', \mathcal{X}')$  satisfies the properties (D1) to (D3). It is easily seen that  $D'$  is a DAG and  $\bigcup_{d \in V(D')} X'_d = X \cup \bigcup_{d \in V(D)} X_d = V(G)$ . Also, if  $d$  is a root of  $D'$ , then  $d$  is a root of  $D$ . Hence  $\emptyset$  guards  $X_{\succeq d} = X_{\succeq d}$ . This shows (D1). Towards establishing condition (D2), suppose  $d \preceq_{D'} d' \preceq_{D'} d''$ .

If  $d' = d$  or  $d' = d''$  then trivially  $X'_d \cap X'_{d''} \subseteq X'_{d'}$ , so suppose  $d \prec_{D'} d' \prec_{D'} d''$ . We consider four cases:

- If none of  $d, d', d''$  is  $d_X$  then  $d, d',$  and  $d''$  are all in  $V(D)$ , so (D2) follows from the fact that  $(D, \mathcal{X})$  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  $D$ :  $d_0 \prec_D d_1 \preceq_D d' \prec_D d''$ . So if  $X \subseteq X_{d_0}$ , i.e., we are in case (i) of the lemma, then  $X'_d \cap X'_{d''} = X \cap X_{d''} \subseteq X_{d_0} \cap X_{d''} \subseteq X_{d'} = X'_{d'}$ , by condition (D2) of  $(D, \mathcal{X})$ . 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  $d \prec_D d' \preceq_D d_0 \prec_D d_1$ . 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  $d \preceq_D d_0 \prec_D d_1 \preceq_D d''$ . 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), let  $(d, d') \in E(D')$ . We consider three cases:

- $d_X \notin \{d, d'\}$ , i.e.,  $(d, d') \in E(D)$ . In this case, (D3) follows from the fact that  $(D, \mathcal{X})$  is a DAG-decomposition.
- Now suppose  $d = d_X$  (so  $d' = d_1$ ). If  $X_{d_0} \cap X_{d_1} \subseteq X \subseteq X_{d_0}$ , i.e., we are in case (i) of the lemma, then

$$X_{\succeq d_1} \setminus (X_{d_0} \cap X_{d_1}) \supseteq X_{\succeq d_1} \setminus X \supseteq X_{\succeq d_1} \setminus X_{d_0}.$$

Further, by Lemma 14,  $X_{\succeq d_1} \setminus (X_{d_0} \cap X_{d_1}) = X_{\succeq d_1} \setminus X_{d_0}$ . Therefore  $X_{\succeq d_1} \setminus X = X_{\succeq d_1} \setminus X_{d_0}$ . As  $(D, \mathcal{X})$  is a DAG-decomposition,  $X_{d_0} \cap X_{d_1}$  guards  $X_{\succeq d_1} \setminus X_{d_0}$ , and as  $X_{d_0} \cap X_{d_1} \subseteq X \cap X_{d_1}$ , Lemma 13(ii) implies that  $X'_d \cap X'_{d_1} = X \cap X_{d_1}$  guards  $X_{\succeq d_1} \setminus X_{d_0} = X'_{\succeq d_1} \setminus X'_{d_0}$ .

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  $X_{\succeq d_1} \setminus (X_{d_0} \cap X_{d_1})$ , due to Lemma 14. Hence, by Lemma 13(iii),  $X'_d \cap X'_{d_1} = X = (X_{d_0} \cap X_{d_1}) \cup Z$  guards

$$\begin{aligned} (X_{\succeq d_1} \setminus (X_{d_0} \cap X_{d_1})) \setminus Z &= X_{\succeq d_1} \setminus ((X_{d_0} \cap X_{d_1}) \cup Z) \\ &= X_{\succeq d_1} \setminus X = X'_{\succeq d_1} \setminus X'_{d_1}. \end{aligned}$$

- Finally, suppose  $d' = d_X$  (so  $d = d_0$ ). Here we claim  $X'_{\succeq d_X} \setminus X'_{d_0} = X_{\succeq d_1} \setminus X_{d_0}$ . If  $X \subseteq X_{d_0}$ , then  $X'_{\succeq d_X} \setminus X'_{d_0} = (X \cup X_{\succeq d_1}) \setminus X_{d_0} = (X \setminus X_{d_0}) \cup (X_{\succeq d_1} \setminus X_{d_0}) = X_{\succeq d_1} \setminus X_{d_0}$ . If  $X \subseteq X_{d_1}$ , then since  $d_X \preceq_{D'} d_1$ ,  $X'_{\succeq d_X} = X'_{\succeq d_1} = X_{\succeq d_1}$ . Now  $X \supseteq X_{d_0} \cap X_{d_1}$ , so by Lemma 13(ii),  $X'_{d'} = X$  guards  $X_{\succeq d_1} \setminus X_{d_0} = X'_{\succeq d_X} \setminus X'_{d_0}$ .

Note that since  $X \subseteq X_{d_0}$  or  $X_{d_1}$ ,  $\max\{|X'_d| : d \in V(D')\} = \max\{|X_d| : d \in V(D)\} = k$ . So  $(D', (X'_d)_{d \in V(D')})$  has width  $k$ .  $\square$

If  $X_1, X_2, \dots, X_n$  is a sequence of subsets of  $V(G)$ , the *decomposition resulting from adding  $X_1, X_2, \dots, X_n$  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 22.** *If  $G$  has a DAG-decomposition of width  $k$ , it also has a nice DAG-decomposition of width  $k$ .*

*Proof.* Let  $(D, \mathcal{X})$  be a DAG-decomposition of width  $k$ . We carry out each of the following steps and reset  $(D, \mathcal{X})$  to be the resulting decomposition.

- (1) We apply Lemma 17 to obtain a decomposition with a unique root, therefore satisfying (N1).
- (2) We apply a complete split on  $(D, \mathcal{X})$  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).
- (3) To satisfy (N4), we require two stages. First, for each  $(d_0, d_1) \in E(D)$  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  $(d, d') \in E(D')$ ,  $X_d$  is either a subset or a super-set of  $X_{d'}$ .
- (4) Secondly, for each  $(d, d') \in E(D)$  with  $|X_d| - |X_{d'}| = m > 1$  (or  $|X_{d'}| - |X_d| = m > 1$ ), let  $X_0 = X_d, X_1, \dots, X_m = X_{d'}$  be a strictly decreasing (increasing) sequence of sets. 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  $X_1, X_2, \dots, X_{m-1}$  to  $(d, d')$ . At this point we have a decomposition which satisfies (N1) to (N4), and is therefore nice.

Finally, from Lemmas 17, 19, and 21, at each step we have a DAG-decomposition of width  $k$ . □

Finally, we show that the DAG-width of graphs is closed under directed unions, which is considered (see [4]) as an important property of a reasonable decomposition of directed graphs.

**Definition 23.** Let  $G$  and  $H$  be (disjoint) directed graphs. The *directed union* of  $G$  and  $H$ , denoted  $G \vec{\cup} H$  is defined as:

$$G \vec{\cup} H := (V(G) \cup V(H), E(G) \cup E(H) \cup (V(G) \times V(H))).$$

A *partial directed union* of  $G$  and  $H$  is a graph  $(V(G) \cup V(H), E(G) \cup E(H) \cup E)$  where  $E \subseteq V(G) \times V(H)$ .

**Theorem 24.** *If  $G$  and  $H$  are digraphs, then*

$$\text{DAG-width}(G \vec{\cup} H) = \max\{\text{DAG-width}(G), \text{DAG-width}(H)\}.$$

*Proof.* For DAG-decompositions  $(D^G, \mathcal{X}^G)$  and  $(D^H, \mathcal{X}^H)$  of  $G$  respectively  $H$ , the DAG  $D$  obtained by putting an edge from every leaf of  $D^G$  to every root of  $D^H$  together with  $\mathcal{X} := (X_d^G)_{d \in V(D^G)} \dot{\cup} (X_d^H)_{d \in V(D^H)}$  forms a DAG-decomposition of  $G \vec{\cup} H$ . Conversely, any DAG-decomposition  $(D, \mathcal{X})$  of  $G \vec{\cup} H$  can be restricted to  $G$  and  $H$  yielding DAG-decompositions for these graphs, according to Lemma 15.  $\square$

We observe that this result also extends to partial directed unions.

### 3.3 Algorithmic aspects of bounded 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.

#### 3.3.1 Computing DAG-width and decompositions

The following is a direct consequence of the similar result for tree-width and Proposition 34.

**Theorem 25.** *Given a digraph  $G$  and a natural number  $k$ , deciding if the DAG-width of  $G$  is at most  $k$  is NP-hard.*

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. We give an algorithm for this that is based on computing monotone winning strategies in the  $k$ -cops and robber game.

**Theorem 26.** *Let  $G$  be a directed graph and let  $k < \omega$ . There is a polynomial time algorithm for deciding if the cop player has a monotone winning strategy in the  $k$ -cops and robber game on  $G$  and for computing such a strategy.*

*Proof.* Given a graph  $G$  and the number  $k$  of available cops we represent the  $k$ -cops and robbers game as a simple, alternating, token-moving game. The game is played on a finite, bi-partite graph, or arena,  $H(G) = (V_0 \dot{\cup} V_1, E)$  which is defined as follows. Let  $W_1 := [V]^{\leq k} \times V$  and  $W_2 := ([V]^{\leq k} \times [V]^{\leq k} \times V)$ .

- (i)  $V_0 := W_1$ ,
- (ii)  $V_1 := W_2 \dot{\cup} \{v_0\}$ , and
- (iii) From each node  $(X, r) \in W_1$  there is an edge to every node  $(X_1, X_2, r') \in W_2$  such that  $r = r'$ ,  $X = X_1$ , and the set of vertices reachable from  $r$  in  $G \setminus X_1$  contains the set of vertices reachable from  $r$  in  $G \setminus (X_1 \cap X_2)$ . Further, from a node  $(X_1, X_2, r) \in W_2$  there is an edge to a node  $(X, r) \in W_1$ , if  $X = X_2$ ,

$r \notin X$  and there is a path from  $r$  to  $r'$  in  $G \setminus (X_1 \cap X_2)$ . Finally, there is an edge from  $v_0$  to every node  $(\emptyset, r) \in W_1$ , where  $r \in V$ .

Note that  $H(G)$  can be constructed in polynomial time.

The game starts with a token at the node  $v_0$ . Player 0 moves the token whenever it is on a node in  $V_0$ , and Player 1 moves the token whenever it is on a node in  $V_1$ . The token may only be moved along an out-edge, on a path of length 1. If a player cannot move he loses. If the game lasts forever, Player 1 wins. Computing which player wins is thus an example of alternating reachability and is therefore decidable in polynomial time (with respect to the size of the arena).

It is easy to see that Player 0 wins this simple game if, and only if the cop player wins the  $k$ -cops and robber game following a (robber-)monotone strategy. As the arena  $H(G)$  is polynomial in the size of the input, and we can compute the winner of the simpler game in polynomial time, the theorem follows.  $\square$

Note also that the translation of strategies into decompositions is computationally easy, i.e., 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 27.** *Given a graph  $G$  of DAG-width  $k$ , a DAG-decomposition of  $G$  of width  $k$  can be computed in time  $\mathcal{O}(|G|^{\mathcal{O}(k)})$ .*

### 3.3.2 Algorithms on graphs of bounded DAG-width.

As the directed tree-width of a graph is bounded above by a constant factor of its DAG-width (see Proposition 35), 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 4 where we show that parity games can be solved efficiently on graphs of bounded DAG-width, something which is not known for directed tree-width.

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  $\text{PTIME} = \text{NP}$ . Furthermore, as 3-colourability is MSO-definable, this also implies that Courcelle’s theorem [2] does fail for DAG-width.

The obvious question that arises is whether one can define a suitable notion of “directed problem” and then show that every MSO-definable “directed” graph problem can be decided efficiently on graphs of bounded DAG-width. This is part of ongoing work.

#### 4 Parity Games on Graphs of Bounded DAG-Width

A *parity game* is a tuple  $(V, V_0, E, \Omega)$  where  $(V, E)$  is a directed graph,  $V_0 \subseteq V$  and  $\Omega : V \rightarrow \omega$  is a function assigning a priority to each vertex. There is no loss of generality in assuming that the range of  $\Omega$  is contained in  $[n]$  where  $n = |V|$  and we will make this assumption from now on.

Intuitively, two players called Odd and Even play a parity game by pushing a token along the edges of the graph with Even playing when the token is on a vertex in  $V_0$  and Odd playing otherwise. Formally, a *play* of the game is an infinite sequence  $\pi = (v_i \mid i \in \omega)$  such that  $(v_i, v_{i+1}) \in E$  for all  $i$ . We say  $\pi$  is *winning* for Even if  $\liminf_{i \rightarrow \infty} \Omega(v_i)$  is even and  $\pi$  is winning for Odd otherwise.

A *strategy* is a map  $f : V^{<\omega} \rightarrow V$  such that, for any sequence  $(v_0, \dots, v_i) \in V^{<\omega}$ , we have  $(v_i, f(v_0, \dots, v_i)) \in E$ . A play  $\pi = (v_i \mid i \in \omega)$  is *consistent* with Even playing  $f$  if whenever  $v_i \in V_0$ ,  $v_{i+1} = f(v_0, \dots, v_i)$ . Similarly,  $\pi$  is consistent with Odd playing  $f$  if whenever  $v_i \notin V_0$ ,  $v_{i+1} = f(v_0, \dots, v_i)$ . A strategy  $f$  is winning for Even from a vertex  $v$  if every play beginning at  $v$  that is consistent with Even playing  $f$  is winning for Even. A strategy is *memoryless* if whenever  $u_0, \dots, u_i$  and  $v_0, \dots, v_j$  are two sequences in  $V^{<\omega}$  with  $u_i = v_j$ , then  $f(u_0, \dots, u_i) = f(v_0, \dots, v_j)$ . It is known that parity games are determined, i.e. for any game and starting position, either Even or Odd has a winning strategy and indeed, a memoryless one. However, we do not assume in our construction that the strategies we consider are memoryless

The following ordering on  $[n]$  is useful in evaluating competing strategies. For priorities  $i, j \in [n]$  we say  $i \sqsubseteq j$  if either

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

Intuitively,  $i \sqsubseteq j$  if the priority  $i$  is “better” for player Odd than  $j$ .

We are interested in the problem of determining, given a parity game and a start-

ing vertex  $v$ , which player has a winning strategy from  $v$ . The complexity of this problem in general remains a major open question, as explained in Section 1. We demonstrate that parity games are tractable on arenas of bounded DAG-width by an algorithm similar in spirit to that of Obdržálek [11]. That algorithm relies on the fact that in a tree decomposition, a set of  $k$  nodes guards all entries and exits to the part of the graph below it, and thus all cycles must pass through this set. In the case of a DAG-decomposition, while the small set guards all exits from the subgraph below it, there may be an unlimited number of edges going into this subgraph. This is the main challenge that our algorithm addresses, and is specifically solved in Lemmas 28, 29 and 30.

For a parity game  $(V, V_0, E, \Omega)$ , consider  $U \subseteq V$  and a set  $W$  that guards  $U$ . Fix a pair of strategies  $f$  and  $g$ . For any  $v \in U$ , there is exactly one play  $\pi = (v_i : i \in \omega)$  starting at  $v$  that is consistent with Even playing  $f$  and Odd playing  $g$ . Let  $\pi'$  be the maximal initial segment of  $\pi$  that is contained in  $U$ . The *outcome* of the pair of strategies  $(f, g)$  (given  $U$  and  $v$ ) is defined as follows.

$$\text{out}_{f,g}(U, v) := \begin{cases} \text{winEven} & \text{if } \pi' = \pi \text{ and } \pi \text{ is winning for Even;} \\ \text{winOdd} & \text{if } \pi' = \pi \text{ and } \pi \text{ is winning for Odd;} \\ (v_{i+1}, p) & \text{if } \pi' = v_0, \dots, v_i \text{ and } p = \min\{\Omega(v_j) \mid j \leq i + 1\}. \end{cases}$$

That is to say that, if the play that results from Even playing  $f$  and Odd playing  $g$  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 lowest priority that is seen in the play  $\pi$  starting in  $v$  and ending in  $w$ , including the end points.

By construction, if  $\text{out}_{f,g}(U, v) = (w, p)$  then  $w \in W$ . More generally, for any set  $W \subseteq V$ , define the set of potential outcomes in  $W$ , written  $\text{pot-out}(W)$ , to be the set  $\{\text{winEven}, \text{winOdd}\} \cup \{(w, p) : w \in W \text{ and } p \in [n]\}$ . We define a partial order  $\trianglelefteq$  on  $\text{pot-out}(W)$  which orders potential outcomes according to how good they are for player Odd. It is the least partial order satisfying the following conditions:

- (i)  $\text{winOdd} \trianglelefteq o$  for all outcomes  $o$ ;
- (ii)  $o \trianglelefteq \text{winEven}$  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  $g$  and  $g'$  are strategies such that  $\text{out}_{f,g}(U, v) \trianglelefteq \text{out}_{f,g'}(U, v)$  then player Odd is better off playing strategy  $g$  rather than  $g'$  in response to Even playing according to  $f$ .

A single outcome is the result of fixing the strategies played by both players in the sub-game induced by a set of vertices  $U$ . If we fix the strategy of player Even to be  $f$  but consider all possible strategies that Odd may play, we can order these

strategies according to their outcome. If one strategy achieves outcome  $o$  and another  $o'$  with  $o \sqsubseteq o'$ , there is no reason for Odd to consider the latter strategy. Thus, we define  $\text{result}_f(U, v)$  to be the set of outcomes that are achieved by the best strategies that Odd may follow, in response to Even playing according to  $f$ . More formally,  $\text{result}_f(U, v)$  is the set of  $\sqsubseteq$ -minimal elements in the set  $\{o : o = \text{out}_{f,g}(U, v) \text{ for some } g\}$ . Thus,  $\text{result}_f(U, v)$  is an anti-chain in the partial order  $(\text{pot-out}(W), \sqsubseteq)$ , where  $W$  is a set of guards for  $U$ . We write  $\text{pot-res}(W)$  for the set of *potential results* in  $W$ . To be precise,  $\text{pot-res}(W)$  is the set of all anti-chains in the partial order  $(\text{pot-out}(W), \sqsubseteq)$ . By definition of the order  $\sqsubseteq$ , if either of  $\text{winEven}$  or  $\text{winOdd}$  is in the set  $\text{result}_f(U, v)$ , then it is the sole element of the set. Also, for each  $w \in W$ , there is at most one  $p$  such that  $(w, p) \in \text{result}_f(U, v)$  so the number of distinct values that  $\text{result}_f(U, v)$  can take is at most  $(n + 1)^{|W|} + 2$ . This is an upper bound on the cardinality of the set  $\text{pot-res}(W)$ .

We also abuse notation and extend the order  $\sqsubseteq$  to the set  $\text{pot-res}(W)$  pointwise. That is, for  $r, s \in \text{pot-res}(W)$  we write  $r \sqsubseteq s$  if, for each  $o \in s$ , there is an  $o' \in r$  with  $o' \sqsubseteq o$ . With this definition, the order  $\sqsubseteq$  on  $\text{pot-res}(W)$  admits greatest lower bounds. Indeed, the greatest lower bound  $r \sqcap s$  of  $r$  and  $s$  can be obtained by taking the set of  $\sqsubseteq$  minimal elements in the set of outcomes  $r \cup s$ . One further piece of notation we use is that we write  $\text{result}(U, v)$  for the set  $\{\text{result}_f(U, v) : f \text{ is a strategy}\}$ .

Suppose now that  $(V, V_0, E, \Omega)$  is a parity game  $(D, \mathcal{X})$  is a DAG-decomposition of  $(V, E)$  of width  $k$  that is *nice* in the sense of Definition 16. For each  $d \in V(D)$ , we write  $V_d$  for the set  $X_{\succeq d} \setminus X_d$ . The key to the algorithm is that we construct the set of results  $\text{result}(V_d, v)$  for each  $v \in V_d$ . Since  $V_d$  is guarded by  $X_d$  and  $|X_d| \leq k$ , the number of distinct values of  $\text{result}_f(V_d, v)$  as  $f$  ranges over all possible strategies is at most  $(n + 1)^k + 2$ .

We define the following, which is our key data structure:

$$\text{Frontier}(d) = \{(v, r) : v \in V_d \text{ and } r = \text{result}_f(V_d, v) \text{ for some strategy } f\}.$$

Note that in the definitions of  $\text{result}_f(U, v)$  and  $\text{Frontier}(d)$ ,  $f$  and  $g$  range over *all* strategies and not just memoryless ones. The bound on the number of possible values of  $\text{result}_f(V_d, v)$  guarantees that  $|\text{Frontier}(d)| \leq n((n + 1)^k + 2)$ . We aim to show how  $\text{Frontier}(d)$  can be constructed from the set of frontiers of the successors of  $d$  in polynomial time. There are four cases to consider.

*Case 1:*  $d$  is a leaf. In this case,  $V_d$  has at most  $k$  nodes and  $\text{Frontier}(d)$  can be constructed by explicitly considering all plays in the set.

*Case 2:*  $d$  has two successors  $e_1$  and  $e_2$ . In this case,  $X_d = X_{e_1} = X_{e_2}$  by the definition of a nice decomposition. Thus,  $V_d = V_{e_1} \cup V_{e_2}$ . Moreover, each of the three sets  $V_d, V_{e_1}$  and  $V_{e_2}$  is guarded by  $X_d$  so, in particular, there is no path from a vertex in  $V_{e_1} \setminus V_{e_2}$  into  $V_{e_2} \setminus V_{e_1}$  (or vice versa) except through  $X_d$ . We claim that  $\text{Frontier}(d) = \text{Frontier}(e_1) \cup \text{Frontier}(e_2)$ .

To see this, suppose first that  $(v, r) \in \text{Frontier}(e_1)$  (the case of  $\text{Frontier}(e_2)$  is symmetrical) and in particular  $r = \text{result}_f(V_{e_1}, v)$ . Now, if  $o \in r$  there is a  $g$  such that  $o = \text{out}_{f,g}(V_{e_1}, v)$ . If  $o$  is winEven or winOdd it is clear that  $o = \text{out}_{f,g}(U, v)$  for any  $U \supset V_{e_1}$  and in particular  $o = \text{out}_{f,g}(V_d, v)$ . If  $o = (w, p)$  then the play  $\pi$  determined by strategies  $f$  and  $g$  starting at  $v$  first leaves the set  $V_{e_1}$  at  $w$ . Since  $w \in X_{e_1} = X_d$  it also leaves the set  $V_d$  at this point and therefore again  $o = \text{out}_{f,g}(V_d, v)$ . We conclude that the set of available outcomes is the same and therefore the set of  $\preceq$ -minimal outcomes is the same. That is,  $r = \text{result}_f(V_d, v)$  and therefore  $(v, r) \in \text{Frontier}(d)$ .

In the other direction, suppose  $(v, r) \in \text{Frontier}(d)$  and that  $v \in V_{e_1}$  (again the case when  $v \in V_{e_2}$  is symmetrical). Let  $f$  be such that  $r = \text{result}_f(V_d, v)$ . Suppose  $o = \text{out}_{f,g}(V_d, v)$  for some strategy  $g$  and let  $\pi$  be the play starting at  $v$  determined by  $f$  and  $g$ . We claim that  $o = \text{out}_{f,g}(V_{e_1}, v)$ . If this is not the case, then the first occurrence in  $\pi$  of a node not in  $V_{e_1}$  must be contained in  $V_d$ . However, since any such node must be in  $X_d$ , which is disjoint from  $V_d$ , this is impossible. Thus, once again  $\text{out}_{f,g}(V_d, v) = \text{out}_{f,g}(V_{e_1}, v)$  and therefore  $r = \text{result}_f(V_{e_1}, v)$ .

Note that the above argument implies in particular that, for  $v \in V_{e_1} \cup V_{e_2}$ , we have  $\text{result}_f(V_{e_1}, v) = \text{result}_f(V_{e_2}, v)$ .

*Case 3:*  $d$  has one successor  $e$  and  $X_d \setminus X_e = \{u\}$ . Then, by (D2),  $u \notin V_e$ . Also, by definition of  $V_d$ ,  $u \notin V_d$ . We conclude that  $V_d = V_e$ . Moreover, since  $X_e$  guards  $V_e$  (by Lemma 13(ii)), there is no path from any element of  $V_e$  to  $u$  except through  $X_e$ . Thus, if  $(w, p) \in \text{result}_f(V_d, v)$  for some  $v$  and  $f$ , it must be the case that  $w \in X_e$ . Hence,  $\text{Frontier}(d) = \text{Frontier}(e)$ .

*Case 4:*  $d$  has one successor  $e$  and  $X_e \setminus X_d = \{u\}$ . This is the critical case. Here  $V_d = V_e \cup \{u\}$  and in order to construct  $\text{Frontier}(d)$  we must determine the results of all plays beginning at  $u$ .

Consider the set of vertices  $v$  in  $X_{\succeq d}$  such that  $(u, v) \in E(G)$ . These fall into two categories. Either  $v \in X_d$  or  $v \in V_e$ . Let  $x_1, \dots, x_s$  enumerate the first category and let  $v_1, \dots, v_m$  enumerate the second. Let  $O = \{(x_i, \min\{\Omega(x_i), \Omega(u)\}) : 1 \leq i \leq s\}$ . This is the set of outcomes obtained if play in the parity game proceeds directly from  $u$  to an element of  $X_d$ . Note that as no two outcomes in  $O$  are comparable with respect to  $\preceq$ ,  $O \in \text{pot-res}(X_d)$ . We write  $\mathcal{O}$  for  $\{\{o\} : o \in O\}$ . That is  $\mathcal{O}$  is the set of singleton results obtained from  $O$ . For each  $v_i$  we know, from  $\text{Frontier}(e)$ , the set  $\text{result}(V_e, v_i)$ . For each result  $r \in \text{result}(V_e, v_i)$ , we write  $\text{mod}(r)$  for the set of outcomes defined by modifying  $r$  as follows. First, if  $r$  contains an outcome  $(u, p)$ , we replace it by winEven if  $\min\{p, \Omega(u)\}$  is even and winOdd if it is odd. Secondly, for any pair  $(w, p) \in r$  where  $w \neq u$ , we replace it with  $(w, \min\{p, \Omega(u)\})$ . Finally, we take the set of  $\preceq$ -minimal elements from the resulting set. This is  $\text{mod}(r)$ . Note that  $\text{mod}(r) \in \text{pot-res}(X_d)$ . The intuition is that  $\text{mod}(\text{result}_f(V_e, v_i))$  defines the set of best possible outcomes for player Odd, if starting at  $u$ , the play goes to  $v_i$  and

from that point on, player Even plays according to strategy  $f$ . For each  $1 \leq i \leq m$ , let  $M_i = \{\text{mod}(r) : r \in \text{result}(V_e, v_i)\}$ .

We now wish to use the sets of results  $M_i$ ,  $O$  and  $\mathcal{O}$  to construct the set  $\text{result}(V_d, u)$ . We need to distinguish between the cases when  $u \in V_0$  (i.e. player Even plays from  $u$  in the parity game) and  $u \in V \setminus V_0$  (i.e. player Odd plays).

The simpler case is when  $u \in V_0$ .

**Lemma 28.** *If  $u \in V_0$ , then  $\text{result}(V_d, u) = \bigcup_i M_i \cup \mathcal{O}$ .*

*Proof.* Let  $f$  be a strategy. If  $f(u) = x_i$ , then  $\text{result}_f(V_d, u) \in \mathcal{O}$ . The other possibility is that  $f(u) = v_i$ , in which case, clearly,  $\text{result}_f(V_d, u) = \text{mod}(\text{result}_f(V_e, v_i))$  and this result is in  $M_i$ . For the converse, if  $r = \{(x_i, p)\} \in \mathcal{O}$ , it is clear that  $r = \text{result}_f(V_d, u)$  for any strategy  $f$  with  $f(u) = x_i$ . Now, let  $r \in M_i$  with  $r = \text{mod}(\text{result}_f(V_e, v_i))$ , then  $r = \text{result}_{f'}(V_d, u)$  where  $f'$  is the strategy that moves from  $u$  to  $v_i$  and then follows the strategy  $f$  from that point on.  $\square$

The case when  $u \notin V_0$  is somewhat trickier. To explain how we can obtain the set  $\text{result}(V_d, u)$  in this case, we formulate the following lemma.

**Lemma 29.** *If  $u \notin V_0$ , then  $r \in \text{result}(V_d, u)$  if, and only if, there is a function  $c$  on the set  $[m]$  with  $c(i) \in M_i$  such that  $r = O \sqcap \prod_{i \in [m]} c(i)$ .*

*Proof.*  $\Rightarrow$  Let  $r \in \text{result}(V_d, u)$ , i.e. there is a strategy  $f$  such that  $r = \text{result}_f(V_d, u)$ .

We define the function  $c$  by  $c(i) = \text{mod}(\text{result}_f(V_e, v_i))$ . Since player Odd can move to any of the vertices  $v_i$ , it is clear that  $r \sqsubseteq c(i)$ , for each  $i$ . Odd can also move to any of the  $x_i$  and therefore  $r \sqsubseteq O$ . Furthermore, for each outcome  $o \in r$ , there is a  $g$  such that  $o = \text{out}_{f,g}(V_d, u)$ . Either  $g(u) = v_i$ , in which case  $o \in c(i)$  by construction, or  $g(u) = x_j$  and  $o \in O$ . Together this establishes  $O \sqcap \prod_{i \in [m]} c(i) \sqsubseteq r$ .

$\Leftarrow$  Let  $c$  be a choice function with  $c(i) = \text{mod}(\text{result}_{f_i}(V_e, v_i))$  for each  $i$ . Let  $f$  be a strategy that agrees with  $f_i$  on all paths beginning with the two vertices  $u, v_i$ . Then, it is clear that  $\text{result}_f(V_d, u) = O \sqcap \prod_{i \in [m]} c(i)$ .  $\square$

Lemma 29 suggests constructing  $\text{result}(V_d, u)$  by considering all possible choice functions  $c$ . However, as each set  $M_i$  may have as many as  $(n+1)^k + 2$  elements, there are  $m^{(n+1)^k + 2}$  possibilities for  $c$  and our algorithm would be exponential. We consider an alternative way of constructing  $\text{result}(V_d, u)$ . Recall that  $\text{result}(V_d, u) \subseteq \text{pot-res}(X_d)$  and the latter set has at most  $(n+1)^k + 2$  elements. We check, for each  $r \in \text{pot-res}(X_d)$ , in polynomial time, whether there is a choice function  $c$  as in Lemma 29 that yields  $r$ . In particular, we take the following alternative characterisation of  $\text{result}(V_d, u)$ .

**Lemma 30.** *If  $u \notin V_0$ , then  $r \in \text{result}(V_d, u)$  if, and only if, there is a set  $D \subseteq [m]$  with  $|D| \leq |r|$  and a function  $d$  on  $D$  with  $d(i) \in M_i$  such that*

- (i)  $r = O \sqcap \prod_{i \in D} d(i)$ ; and
- (ii) for each  $i \notin D$  there is an  $r_i \in M_i$  with  $r \sqsubseteq r_i$ .

*Proof.*  $\Rightarrow$  Assume  $r \in \text{result}(V_d, u)$  and let  $c$  be the choice function according to Lemma 29. For each  $o \in r$ , if  $o \notin O$  select one  $i \in [m]$  such that  $o \in c(i)$ . Let  $D$  be the collection of indices  $i$  selected. By construction,  $|D| \leq |r|$ . Now, we define  $d(i) = c(i)$  for all  $i \in D$  and let  $r_i = c(i)$  for  $i \notin D$ .  
 $\Leftarrow$  Given  $D, d$  and the collection of  $r_i$  as specified, we define the choice function  $c$  by

$$c(i) := \begin{cases} d(i) & \text{if } i \in D; \\ r_i & \text{otherwise.} \end{cases}$$

Since by hypothesis  $r \sqsubseteq r_i$  and  $r = O \sqcap \prod_{i \in D} d(i)$ , it is then easily seen that  $r = O \sqcap \prod_{i \in [m]} c(i)$ .  $\square$

Now, any  $r \in \text{pot-res}(X_d)$  has at most  $k$  elements. Thus, to check whether such an  $r$  is in  $\text{result}(V_d, u)$  we cycle through all sets  $D \subseteq [m]$  with  $k$  or fewer elements (and there are  $\mathcal{O}(n^k)$  such sets) and for each one consider all candidate functions  $d$  (of which there are  $\mathcal{O}(n^{k^2})$ ). Having found a  $d$  which gives  $r = O \sqcap \prod_D d(i)$ , we then need to find a suitable  $r_i$  in each  $i \in [m] \setminus D$ . For this we must, at worst, go through all elements of all the sets  $M_i$  and compare them to  $r$ . This can be done in time  $\mathcal{O}(n^{k+1})$ .

We have now obtained the set  $\text{result}(V_d, u)$ . One barrier remains to completing the construction of  $\text{Frontier}(d)$ . Elements  $(v, r)$  of  $\text{Frontier}(e)$  may have outcomes in  $r$  of the form  $(u, p)$ . Since  $u$  is not in  $X_d$ , these must be resolved by combining them with results from  $\text{result}(V_d, u)$ . To be precise, let  $r \in \text{result}(V_e, v)$  for some  $v \in V_e$  and  $s \in \text{result}(V_d, u)$ . Define the combined result  $c(r, s)$  as follows:

- if  $r$  does not contain an outcome of the form  $(u, p)$ , then  $c(r, s) = r$ ;
- otherwise,  $r$  contains a pair  $(u, p)$ . Let  $s'$  be obtained from  $s$  by replacing every pair  $(w, q)$  by  $(w, \min\{p, q\})$ .  $c(r, s) = (r \setminus \{(u, p)\}) \sqcap s'$ .

Intuitively, if  $r = \text{result}_f(V_e, v)$  and  $s = \text{result}_{f'}(V_d, u)$  then  $c(r, s)$  is the set of  $\sqsubseteq$ -minimal outcomes that can be obtained if player Even plays according to  $f$  starting at  $v$  until the node  $u$  is encountered and then switches to strategy  $f'$ .

**Lemma 31.** *For any  $v \in V_e$ ,*

$$\text{result}(V_d, v) = \{c(r, s) : r \in \text{result}(V_e, v) \text{ and } s \in \text{result}(V_d, u)\}.$$

*Proof.* Clearly, for any strategy  $f$ ,  $\text{result}_f(V_d, v) = c(\text{result}_f(V_e, v), \text{result}_f(V_d, u))$ . Thus,  $\text{result}_f(V_d, v)$  is included in the set on the right hand side. For the converse, suppose first that  $r = \text{result}_f(V_e, v)$  is such that no outcome of the form  $(u, p)$  is in  $r$ . This means that when player Even plays according to  $f$ , there is no strategy  $g$  that Odd can play which will lead to the vertex  $u$ . Therefore,  $\text{result}_f(V_e, v) = \text{result}_f(V_d, v) = c(r, s)$  for all  $s$ . Now, let  $r = \text{result}_{f_1}(V_e, v)$  include an outcome  $(u, p)$  and set  $s = \text{result}_{f_2}(V_d, u)$ . Let  $f$  be the strategy which follows  $f_1$  for the path from  $v$  to  $u$  and follows  $f_2$  once  $u$  has been reached. It is easily checked that  $\text{result}_f(V_d, v) = c(r, s)$ .  $\square$

We now obtain  $\text{Frontier}(d) = \{(v, r) : r \in \text{result}(V_d, v)\}$ .

**Theorem 32.** *For each  $k$ , there is a polynomial  $p$  and an algorithm running in time  $\mathcal{O}(p(n))$  which determines the winner of parity games on all graphs of  $n$  vertices with DAG-width at most  $k$ .*

*Proof.* By Proposition 27, there is a polynomial-time algorithm that will produce a DAG-decomposition of the game graph of width  $k$ . This can be converted into a nice decomposition  $(D', \mathcal{X}')$  in time at most quadratic (in the size of the decomposition). Let  $a$  be the root of  $D'$  and let  $X_a = \{x_1, \dots, x_l\}$  where  $l \leq k$ . Consider the DAG  $D$  formed by adding  $l$  new elements  $a_0, \dots, a_{l-1}$  to  $D'$  in a simple directed path ending in  $a$ . Further, for each  $i$  define  $X_{a_i}$  to be the set  $\{x_1, \dots, x_i\}$ . In particular, the new root  $a_0$  is labelled by  $\emptyset$ . It is easily seen that the new labelled DAG  $(D, \mathcal{X})$  with  $\mathcal{X} = (X_d)_{d \in V(D)}$  still meets the definition of a nice decomposition. We then use the above construction to obtain  $\text{Frontier}(d)$  for each  $d$  in  $D$ , starting from the leaves and working our way to the root. Since the size of  $D$  is at most  $n^{2k} + k$ , the total time taken is bounded by a polynomial. Now, for the root  $a_0$  of  $D$  it is true that  $X_{\geq a_0} = V_{a_0} = V$ . Thus, if  $(v, r) \in \text{Frontier}(a_0)$  then  $r \subseteq \{\text{winEven}, \text{winOdd}\}$ . If  $\text{winEven} \in r$ , this means that player Even has a strategy to win the parity game beginning at vertex  $v$ , and if  $\text{winEven} \notin r$ , for any strategy played by player Even, Odd has a strategy to defeat it. We have thus determined the winner of the parity game starting at each vertex.  $\square$

## 5 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 directed tree-width introduced by Johnson et al. [4], directed path-width [17], and entanglement proposed by Berwanger and Grädel [5].

### 5.1 Undirected tree-width.

First we formalise the relationship between DAG-width and undirected tree-width alluded to in previous sections.

Recall that the tree-width of a directed graph  $G$  is defined as the tree-width of the undirected graph obtained from  $G$  by forgetting the orientation of the edges.

**Proposition 33.**

- (i) *If a directed graph  $G$  has tree-width  $k$ , its DAG-width is at most  $k + 1$ .*
- (ii) *There exists a family of directed graphs with arbitrarily large tree-width and DAG-width 1.*

*Proof.* (i). Suppose  $(T, \mathcal{W})$  is a tree decomposition of  $G$  of width  $k$ , with  $\mathcal{W} = (W_t)_{t \in V(T)}$ . Choose some  $r \in V(T)$  and orient the edges of  $T$  away from  $r$ . That is, if  $\{s, t\} \in E(T)$  and  $s$  is on the unique path from  $r$  to  $t$ , then change  $\{s, t\}$  to  $(s, t)$ . Since  $T$  is a tree, every edge has a unique orientation in this manner. Let  $D$  be the resulting DAG. For all  $d \in V(D)$ , set  $X_d := W_t$  where  $t$  is the node of  $T$  corresponding to  $d$ . We claim that  $(D, \mathcal{X})$  with  $\mathcal{X} = (X_d)_{d \in V(D)}$  is a DAG-decomposition of  $G$  of width  $k + 1$ . The condition (D1) is trivial; (D2) follows from the connectivity condition of tree decompositions. The orientation ensures  $D$  has one root  $r$ , so  $X_{\succeq r} = V(G)$ . Condition (D3) is hence satisfied at the root. For the other nodes, it follows from a similar condition for tree decompositions. Let  $(d, d') \in E(D)$  and suppose  $v \in X_{\succeq d'} \setminus X_d$ . Suppose also that  $(v, w) \in E(G)$  and  $w \notin X_{\succeq d'} \setminus X_d$ . We will show that  $w \in X_d \cap X_{d'}$ . Since  $v \notin X_d$  and  $v \in X_{\succeq d'}$ , any  $d''$  such that  $v \in X_{d''}$  must satisfy  $d' \preceq_D d''$  by the connectivity condition of tree decompositions. As  $(v, w) \in E(G)$ , there exists  $d'' \in V(D)$  such that  $\{v, w\} \subseteq X_{d''}$ . Thus,  $w \in X_{\succeq d'}$ . As  $w \notin X_{\succeq d'} \setminus X_d$ , it follows that  $w \in X_d$ . By (D2), we also have  $w \in X_{d'}$ , as  $w \in X_{\succeq d'}$ . Accordingly,  $w \in X_d \cap X_{d'}$  and (D3) holds.

- (ii). For any number  $n$ , let  $K_n$  be the (undirected) complete graph with  $n$  vertices  $v_1, v_2, \dots, v_n$ . Orient the edges of  $K_n$  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  $n - 1$ . □

If  $G$  is an undirected graph then let  $\overleftrightarrow{G}$  be the directed graph obtained by replacing each edge  $\{u, v\}$  in  $E(G)$  with two edges  $(u, v)$  and  $(v, u)$ .

**Proposition 34.**  *$G$  has tree-width  $k - 1$  if, and only if,  $\overleftrightarrow{G}$  has DAG-width  $k$ .*

*Proof.* It is easily seen that the  $k$ -cops and robber game for undirected graphs on  $G$

is equivalent to the  $k$ -cops and robber game for directed graphs on  $\overleftrightarrow{G}$ . The result follows from the correspondence between the measures and existence of monotone winning strategies.  $\square$

## 5.2 Directed tree-width

With the aim of recovering the effectiveness of tree decompositions in allowing divide-and-conquer algorithms, directed tree-width is associated with a tree-shaped representation of the input graph. It was proved that this representation leads to efficient algorithms for solving a particular class of NP-complete problems, including, e.g., Hamiltonicity, when directed tree-width is bounded. Unfortunately this generic method does not cover many interesting problems. In particular, the efficient solution of parity games on bounded tree-width has failed so far to generalise to directed tree-width.

In terms of games, directed tree-width is characterised by a restriction of the cops and robber games for DAG-width, in which the robber is only permitted to move to vertices where there exists a directed cop-free path from his intended destination back to the current position. In contrast to the case of undirected tree-width, for these games cop-monotonicity and robber-monotonicity differ and both cop- and robber-monotone strategies are known to not be sufficient [19]. However, the difference is within a constant factor.

On basis of the game characterisation, it is clear that undirected tree-width of a graph is a lower bound for its DAG-width. Conversely, the DAG-width of a graph cannot be bound in terms of its directed tree-width.

### Proposition 35.

- (i) *If a graph has DAG-width  $k$ , its directed tree-width is at most  $3k + 1$ .*
- (ii) *There exists a family of graphs with arbitrarily large DAG-width and directed tree-width 1.*

*Proof.* (i). If  $G$  has DAG-width  $k$  then  $k$  cops can win the  $k$ -cops and robber game on  $G$ . Thus,  $k$  cops can win the game defined in [4], and so  $G$  does not have a (directed) haven of size  $k$ . By a result of [4], this implies that  $G$  has a directed tree decomposition<sup>2</sup> of width  $3k + 1$ .

(ii). Consider the family  $\{(T_k^2)^{op} : k \geq 2\}$  of graphs defined in Proposition 6. Note that  $(T_k^2)^{op}$  is a binary branching tree of height  $k$  with back-edges from every vertex to each of its ancestors. We have shown that  $(T_k^2)^{op}$  has cop number  $k$ , and

---

<sup>2</sup> Directed tree decompositions are called *arboreal decompositions* in [4].

it is clear that the strategy described for  $k$  cops is monotone, so  $(T_k^2)^{op}$  has DAG-width  $k$ . On the other hand, consider the directed tree  $T$  obtained from  $(T_k^2)^{op}$  by removing back-edges. For each  $t' \in V(T)$ , let  $B_{t'} := \{t, s\}$  where  $t$  is the vertex corresponding to  $t'$  in  $(T_k^2)^{op}$  and  $s$  is the predecessor of  $t$  (if  $t'$  is not the root of  $T$ ), and let  $X_{(s', t')} := \{s\}$  for all  $(s', t') \in E(T)$ . Then, it is easy to show that  $(T, (B_{t'})_{t' \in V(T)}, (X_e)_{e \in E(T)})$  is a directed tree decomposition of  $(T_k^2)^{op}$  of width 1. For  $k \geq 2$ ,  $(T_k^2)^{op}$  is not acyclic and therefore has directed tree-width exactly 1.  $\square$

One requirement of directed tree decompositions is that the partition consists of non-empty sets. An interesting extension of directed tree decompositions is one without that condition, i.e., using pre-decompositions. The measure (which we call the *extended* directed tree-width here) nicely generalises DAG-width without using games.

**Proposition 36.** *If a graph  $G$  has DAG-width  $k$  its extended directed tree-width is at most  $k - 1$ .*

*Proof.* Let  $(D, \mathcal{X})$  be a DAG-decomposition of  $G$  of width  $k$ . By Lemma 17, we can assume that  $D$  has a unique root. Let  $T$  be a spanning tree of  $D$ . Let  $t_1, t_2, \dots$  be the sequence of nodes of  $T$  visited in a depth-first traversal of  $T$ , where a node is only added after all its children have been. Starting from  $i = 1$ , we define  $B_{t_i} := X_{d_i} \setminus \bigcup_{j < i} B_{t_j}$ , where  $d_i$  is the node of  $D$  corresponding to  $t_i$ . For  $(s, t) \in E(T)$  we set  $W_{(s, t)} := X_{s'} \cap X_{t'}$  where  $s', t'$  are the nodes of  $D$  corresponding to  $s$  and  $t$  respectively. It follows easily that  $(T, (B_{t_i})_{t_i \in V(T)}, (W_e)_{e \in E(T)})$  is a pre-decomposition of  $G$  of width at most  $k - 1$ .  $\square$

Tree-width on undirected graphs also has a useful characterisation in terms of balanced separators. From the proof of 3.3 in [4] we are able to extract a similar characterisation of directed tree-width in terms of small balanced *directed separators*.

**Definition 37.** Let  $G := (V, E)$  be a digraph and  $W \subseteq V$  be a set of vertices. A *directed separator of size  $k$  for  $W$*  is a set  $X$  with  $|X| \leq k$  such that  $W \setminus X$  can be partitioned into  $W_1, W_2 \subseteq W$  with  $W \setminus X = W_1 \dot{\cup} W_2$ ,  $W_1 \cap W_2 = \emptyset$ ,  $\frac{1}{3}|W| \leq |W_i| \leq \frac{2}{3}|W|$ ,  $i = 1, 2$ , and there is no directed path from a vertex in  $W_2$  to a vertex in  $W_1$  in  $G \setminus X$ .

A graph  $G$  has the *directed  $k$ -separator property* if every subset of  $V(G)$  has a directed separator of width  $k$ .

Note that directed separators are a direct generalisation of standard separators, as the two notions coincide on undirected graphs.

**Theorem 38.** *Every graph of directed tree-width at most  $k - 1$  has the directed  $k$ -separator property. Conversely, any graph with the directed  $k$ -separator property has directed tree-width at most  $6k$ .*

As the first part of the previous theorem can be proved via havens rather than by construction directed tree-decompositions, the same proof implies the following corollary which will be used below.

**Corollary 39.** *Every graph of DAG-width at most  $k$  has the directed  $k$ -separator property.*

### 5.3 Directed path-width

Directed path-width was introduced by Thomas [20] as a generalisation of path-width to directed graphs. Formally, a directed path decomposition of a directed graph  $G$  is a sequence  $W_1, W_2, \dots, W_n$  such that

- (P1)  $\bigcup_{i=1}^n W_i = V(G)$ ,
- (P2) If  $i < i' < i''$  then  $W_i \cap W_{i''} \subseteq W_{i'}$ ,
- (P3) For every edge  $(u, v) \in E(G)$ , there exist  $i \leq j$  such that  $u \in W_i$  and  $v \in W_j$ .

The width of  $W_1, W_2, \dots, W_n$  is  $\max\{|W_i| : 1 \leq i \leq n\} - 1$ , and the *directed path-width* of  $G$  is the minimal width of all directed path decompositions.

It is worth noting that for undirected graphs, path-width readily generalises to tree-width as a path decomposition is also a tree decomposition. For directed graphs however, this is not the case. We next show that DAG-width does generalise directed path-width in this way.

**Proposition 40.**

- (i) *If a graph  $G$  has directed path-width  $k$ , its DAG-width is at most  $k + 1$ .*
- (ii) *There exists a family of graphs with arbitrarily large directed path-width and DAG-width 2.*

*Proof.* (i). Let  $W_1, W_2, \dots, W_n$  be a directed path decomposition of  $G$  of width  $k$ . Let  $D_n$  be the directed path with  $n$  vertices. That is  $V(D_n) = \{d_1, \dots, d_n\}$  and  $(d_i, d_j) \in E(D_n)$  if and only if  $j = i + 1$ . Set  $X_{d_i} := W_i$  for all  $d_i \in V(D_n)$ . We claim  $(D_n, (X_d)_{d \in V(D_n)})$  is a DAG-decomposition of  $G$  of width  $k + 1$ . Condition (D1) follows from (P1) and (D2) follows from (P2). To show (D3) for  $1 \leq i < n$ , suppose  $v \in X_{\succeq d_{i+1}} \setminus X_{d_i}$  and  $(v, w) \in E(G)$ . From (P3) there exist  $i' \leq j'$  such that  $v \in W_{i'}$  and  $w \in W_{j'}$ . If  $i' \leq i$ , then by (P2)  $v \in X_{d_i}$ , contradicting the choice of  $v$ . Thus,  $i < i' \leq j'$  and  $w \in X_{\succeq d_{i+1}}$ . If  $w \notin X_{\succeq d_{i+1}} \setminus X_{d_i}$  then  $w \in X_{d_i}$  and therefore  $w \in X_{d_{i+1}}$  by (P2). Thus,  $X_{d_i} \cap X_{d_{i+1}}$  guards  $X_{\succeq d_{i+1}} \setminus X_{d_i}$ .

(ii). Let  $T_k$  be the (undirected) binary tree of height  $k \geq 2$ . From Proposition 34,  $\overleftrightarrow{T}_k$  has DAG-width 2. It is known that  $T_k$  has path-width exactly  $k - 1$ , and it is

straightforward to show that  $\overleftrightarrow{T}_k$  must therefore have directed path-width exactly  $k - 1$ . Thus, the family  $\{T_k : k \geq 2\}$  satisfies the proposition.  $\square$

In [17], Barát showed that directed path-width corresponds to the number of cops required to catch an invisible robber on a directed graph. It should therefore not be surprising that our measure generalises directed path-width.

#### 5.4 Entanglement

Entanglement measures the nesting depth of directed cycles in a graph. In terms of cops and robber games, it is obtained by restricting the mobility of both the robber and the cops so that in any round, the cop player may send one cop to the robber's current position (or do nothing) while the robber must move to a successor of his current residence.

Unlike the other graph measures considered here, entanglement is not associated with an efficient tree-shaped graph representation. Nevertheless, it was shown that parity games on arenas of bounded entanglement can be solved in polynomial time [5]. In fact, just a bound on the minimal entanglement of a subgraph induced by any winning strategy rather than of the input arena is required.

It is difficult to compare entanglement with DAG-width as the latter measure requires monotone strategies whereas the former does not. However, as the following proposition shows, entanglement does provide an upper bound for the number of cops required to catch the robber with a strategy that is not necessarily monotone. This suggests that having bounded DAG-width is more general than having bounded entanglement. On the other hand, the gap between DAG-width and entanglement, although arbitrarily large, can be at most logarithmic in the number of graph vertices.

#### **Proposition 41.**

- (i) *If a graph has entanglement  $k$ , its cop number is at most  $k + 1$ .*
- (ii) *There are graphs with arbitrarily large entanglement but with DAG-width 3.*
- (iii) *If a graph  $G$  has DAG-width  $k$ , its entanglement is at most  $(k + 1) \cdot \log |V(G)|$ .*

*Proof.* (i). Let  $G$  be a graph of entanglement  $k$ . We replicate a play of the cops and robber game on  $G$  with  $k + 1$  cops while keeping track of an auxiliary entanglement game with  $k$  cops on the same graph. Our aim is to transfer the moves of the robber from the first to the second game and those of the cops vice versa, so that the plays proceed, and end, simultaneously. The move of the cop player in the entanglement game will be simply copied to the cops and robber game. However, since in the

latter the robber is able to move along several edges at a time, we employ an additional cop, called *chaser*, who follows the robber's itinerary moving just one edge per round. Accordingly, each round of the entanglement game will be split into two phases of the cops and robber game, one to copy the cops' move and another one to post the chaser.

To describe the state at the beginning of round  $i$  we refer to the following three components: the set of positions  $X_i \subseteq [V]^{\leq k}$  occupied by cops and the position  $r_i$  of the robber in the entanglement game, and the position  $s_i$  of the robber in the cops and robber game. Note that  $r_i$  also represents the position of the chaser and that it is located on the itinerary of the robber in the cops and robber game, i.e., the path along which the robber moved throughout the play. We denote by  $\sigma_i$  the simple path from  $r_i$  to  $s_i$  that follows this itinerary.

We proceed the induction on  $i$ , maintaining the invariant that beyond its first position  $\sigma_i$  is cop-free. At the beginning of a play in the cops and robber game, the cop player chooses the empty set  $X_0 := \emptyset$  and the robber chooses the initial position  $r_0$ . In the second phase of this first round, the cop player then posts the chaser  $X_0 := \{r_0\}$  whereas the robber may move to some position  $s_0$ . In the auxiliary entanglement game, we set the cops idle, and post the robber at  $r_0$ . Thus, the plays are left in the configurations  $(\{r_0\}, s_0)$  and respectively  $(\emptyset, r_0)$ .

Let  $(X_i, r_i)$  and  $(X_i \cup \{r_i\}, s_i)$  be the starting configurations in round  $i$ , and let  $\sigma_i$  be the (cycle-free) path along the itinerary of the robber from  $r_i$  to  $s_i$ . We assume that the segment  $\sigma_i$  avoids  $X_i$  from the second position onwards. To decide the next move of the cop player in the cops and robber game, we first look at the entanglement game. There, the cop player may do either of the following:

- (1) do nothing:  $X_{i+1} = X_i$ , or
- (2) post a fresh cop to the robber's current position:  $X_{i+1} = X_i \cup \{r_i\}$ , or
- (3) move a cop from  $z \in X_i$  to the current position,  $X_{i+1} = (X_i \setminus z) \cup \{r_i\}$ .

Notice that in the cops and robber game the position  $r_i$  is already occupied by the chaser. To transfer the cops' move from the entanglement game, we can hence skip the first phase in case (1) and (2). Else, when the cop is removed from position  $z$ , we also choose the set  $(X_i \setminus z) \cup \{r_i\}$  for the cop player in the cops and robber game. Now, the robber moves to some position  $s$  that he can reach from  $s_i$  avoiding  $X_{i+1}$ . Hence, by our assumption, the path from  $r_i$  via  $\sigma_i$  to  $s$  following the robber's itinerary also avoids  $X_{i+1}$ . In particular, the successor of  $r_i$  along this path is cop-free. We choose this successor  $r_{i+1}$  as a new position for the chaser. Hence, in the second phase of the round, the chaser is removed from  $r_i$  and sent to  $r_{i+1}$ . After this cop-move, the robber prolongs his itinerary to some position  $s_{i+1}$ . However, notice that this prolongation either does not pass through  $r_i$  or it closes a cycle that will be discarded so that the new segment  $\sigma_{i+1}$  which links  $r_{i+1}$  to  $s_{i+1}$  still satisfies our invariant. Finally, we interpret the chaser's choice as a move of the robber in the

entanglement game, thus assuming the configuration  $(X_{i+1}, r_{i+1})$ .

Due to our invariant, the robber can freely move in the entanglement game as long as the robber in the cops and robber game is able to move. But since the cop player has a winning strategy for  $k$  cops this cannot be forever so that, eventually, the robber cannot prolong his itinerary and loses. This shows that  $k + 1$  cops have a winning strategy in the cops and robber game on  $G$ , or equivalently that the cop number of  $G$  is at most  $k + 1$ .

(ii). Let  $T_k^\downarrow$  be the full binary tree of height  $k$  with edges oriented downwards, and let  $T_k^\uparrow$  be the same tree with edges oriented upwards. Every node  $v^\downarrow \in T_k^\downarrow$  has a *double*  $v^\uparrow \in T_k^\uparrow$ , and vice versa. The graph  $G(3, k)$  is constructed by taking the union  $T_k^\downarrow \cup T_k^\uparrow$ , adding edges from each leaf to its double (in both directions), and adding the edges  $(u^\uparrow, v^\downarrow)$  for each edge  $(u^\uparrow, v^\uparrow)$  of  $T_k^\uparrow$ . It is easy to see that  $G(3, k)$  has DAG-width 3.

We claim that  $\text{entanglement}(G(3, k)) > k$ . To prove this, we describe a strategy by which the robber escapes against  $k$  cops. We call a path in  $G(3, k)$  *free* if neither the nodes on the path nor their doubles are occupied by a cop. We say that a node is *blocked* if both the node and its double are occupied. The robber moves according to the following strategy: *at a leaf  $w^\uparrow$ , he selects an ancestor  $u^\downarrow$  of  $w^\downarrow$  from which there is a free path to a leaf  $v^\downarrow$ . He goes to  $v^\downarrow$  by moving upwards through  $T_k^\uparrow$ , stepping over to  $u^\downarrow$  and moving downwards through  $T_k^\downarrow$ . Finally he steps over to  $v^\uparrow$ .* With this strategy, the robber is never below a blocked node. A leaf has  $k + 1$  ancestors (including itself) in  $T_k^\downarrow$ , so there is always an ancestor with a free path to a leaf. Thus, the robber can maintain this strategy and escape forever.

(iii). This follows from the  $k$ -separator property for graphs of DAG-width  $k$  established in Corollary 39. In the entanglement game on such a graph, a winning strategy for the cops can be described as follows: choose a balanced directed separator  $X$  and place a cop whenever the robber passes through a vertex in  $X$ . Intuitively, we intend to cut the graph into two partitions  $W_1, W_2$  using at most  $k$  cops, so that the robber is trapped in either one of them. The cut is complete when all vertices in  $X$  are occupied by cops. This procedure is applied recursively on the partition where the robber currently resides. Whenever he moves between partitions, e.g., from  $W_1$  to  $W_2$ , he needs to pass through the separator  $X$  triggering the posting of a new cop. After the robber leaves  $W_1$ , any cop posted in this partition can be removed. In case the robber again returns to  $W_1$ , one more vertex in the separator will be blocked, so that he is finally trapped in one partition. During a play, these partitions decrease up to size one, at which point the robber loses.  $\square$

Note that the proof of (iii) implies a stronger result in that if  $G$  has directed tree-width  $k - 1$  then its entanglement is at most  $(k + 1) \cdot \log |V(G)|$ . We conclude that, despite their conceptual affinity, directed tree-width, directed path-width, entanglement, and DAG-width are rather different measures. The following inequalities

summarise, up to constant factors, the results of this section.

$$\begin{aligned} \text{entanglement}(G) / \log |V(G)| & \\ & \leq \text{directed tree-width}(G) \\ & \leq \begin{cases} \text{DAG-width}(G) \leq \text{tree-width}(G) \\ \text{DAG-width}(G) \leq \text{directed path-width}(G) \\ \text{cop-number}(G) \leq \text{entanglement}(G). \end{cases} \end{aligned}$$

Furthermore, for any inequality above there exist families of graphs for which the inequality (up to constant factors) is strict.

## 6 Further Remarks

We conclude with a comment on a few recent and relevant results. In [21], Kreutzer and Ordyniak show that monotonicity is not sufficient in the cops and robber game on directed graphs. In particular, for any  $n \in \mathbb{N}$  there exist graphs which require  $n$  more cops to capture the robber with a monotone strategy than with a non-monotone strategy. Their examples do not preclude the possibility of bounded monotonicity cost; that is, the existence of a function  $f$  such that if  $k$  cops have a winning strategy then  $f(k)$  cops have a monotone winning strategy. We believe that these games have linearly bounded monotonicity cost, however the problem remains an active area of research.

The class of directed graphs for which the winner of a parity game can be efficiently decided has been extended by Obdržálek [22] to include graphs of bounded clique-width. As there are DAGs of arbitrary clique-width and graphs of fixed clique-width but arbitrary DAG-width, this result is incomparable with our own. Whether there exists a measure which generalises both clique-width and DAG-width, particularly with regard to efficiently solving parity games, remains an open problem.

Toward investigating other characterisations of tree-width and their extension to directed graphs, Hunter and Kreutzer [7] show that the natural generalisations of partial  $k$ -trees and elimination orderings result in a measure different from DAG-width which they call *Kelly-width*. The measures are similar in that the cop number of a graph bounds its Kelly-width and likewise, a non-monotone version of Kelly-width bounds DAG-width, up to constant factors. As a consequence, the authors conjecture that Kelly-width lies within a constant factor of DAG-width. Resolving this and similar questions would provide insight into the structure theory of directed graphs associated with DAG-width and is part of ongoing work.

## References

- [1] N. Robertson, P. Seymour, Graph Minors. III. Planar tree-width, *Journal of Combinatorial Theory, Series B* 36 (1984) 49–63.
- [2] B. Courcelle, Graph rewriting: An algebraic and logic approach, in: J. van Leeuwen (Ed.), *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, Elsevier, 1990, pp. 193–242.
- [3] B. A. Reed, Introducing directed tree width, in: 6th Twente Workshop on Graphs and Combinatorial Optimization, Vol. 3 of *Electron. Notes Discrete Math*, Elsevier, 1999, pp. 222–229.
- [4] T. Johnson, N. Robertson, P. D. Seymour, R. Thomas, Directed tree-width, *Journal of Combinatorial Theory, Series B* 82 (1) (2001) 138–154.
- [5] D. Berwanger, E. Grädel, Entanglement – A measure for the complexity of directed graphs with applications to logic and games, in: *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, 2004, pp. 209–223.
- [6] M. Safari, D-width: A more natural measure for directed tree width, in: *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2005, pp. 745–756.
- [7] P. Hunter, K. Kreutzer, Digraph measures: Kelly decompositions, games, and orderings, *TCS* 399 (3) (2008) 206–219.
- [8] D. Kozen, Results on the propositional  $\mu$ -calculus, *TCS* 27 (1983) 333–354.
- [9] E. Emerson, C. Jutla, A. Sistla, On model checking for the  $\mu$ -calculus and its fragments, *TCS* 258 (1-2) (2001) 491–522.
- [10] E. Emerson, C. Jutla, The complexity of tree automata and logics of programs (extended abstract), in: *Proceedings for the 29th IEEE Symposium on Foundations of Computer Science (FOCS)*, 1988, pp. 328–337.
- [11] J. Obdržálek, Fast  $\mu$ -calculus model checking when tree-width is bounded, in: *Proceedings of the 15th International Conference on Computer Aided Verification (CAV)*, 2003, pp. 80–92.
- [12] J. Obdržálek, Dag-width: connectivity measure for directed graphs, in: *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006, pp. 814–821.
- [13] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, Dag-width and parity games, in: *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, 2006, pp. 524–536.
- [14] P. Seymour, R. Thomas, Graph searching, and a min-max theorem for tree-width, *Journal of Combinatorial Theory, Series B* 58 (1993) 22–33.

- [15] N. D. Dendris, L. M. Kirousis, D. M. Thilikos, Fugitive-search games on graphs and related parameters, *TCS* 172 (1-2) (1997) 233–254.
- [16] G. Gottlob, N. Leone, F. Scarcello, Robbers, marshals, and guards: Game theoretic and logical characterizations of hypertree width, in: *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, 2001, pp. 195–201.
- [17] J. Barát, Directed path-width and monotonicity in digraph searching, to appear in *Graphs and Combinatorics* (2005).
- [18] H. L. Bodlaender, Treewidth: Algorithmic techniques and results, in: *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 1997, pp. 19–36.
- [19] I. Adler, Directed tree-width examples, to appear in *Journal of Combinatorial Theory, Series B* (2005).
- [20] R. Thomas, Directed tree-width, slides from a lecture at the Regional NSF-CBMS Conference on Graph Structure and Decomposition, 2002. Available at <http://www.math.gatech.edu/~thomas/SLIDE/CBMS/dirtrsl.pdf>.
- [21] S. Kreutzer, S. Ordyniak, Digraph decompositions and monotonicity in digraph searching, in: *Proceedings of the 34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, 2008, pp. 336–347.
- [22] J. Obdržálek, Clique-width and parity games, in: *Proceedings of the 21st International Workshop on Computer Science Logic (CSL)*, 2007, pp. 54–68.