DAG-Width and Parity Games

Dietmar Berwanger[1], Anuj Dawar[2], Paul Hunter[3], and Stephan Kreutzer[3]

[1] LaBRI, Université de Bordeaux 1, dwb@labri.fr
[2] University of Cambridge Computer Laboratory, anuj.dawar@cl.cam.ac.uk
[3] Logic and Discrete Systems, Institute for Computer Science, Humboldt-University Berlin, {hunter, kreutzer}@informatik.hu-berlin.de

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 is characterised by a game known as the cops-and-robber game where a number of cops chase a robber on the graph. We consider the natural adaptation of this game to directed graphs and show that monotone strategies in the game yield a measure with an associated notion of graph decomposition that can be seen to describe how close a directed graph is to a directed acyclic graph (DAG). This promises to be useful in 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 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 [13]. 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 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 [4] 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 [12, 8, 2, 14]) which extend notions of tree-decompositions and tree-width to directed graphs. In particular, Johnson et al. [8] 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 nodes 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 μ-calculus, introduced in [10], 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 φ of the μ-calculus, whether or not $\mathcal{A}$ satisfies φ can be turned into a parity game (see [6]). The exact complexity of solving parity games is an open problem that has received a large amount of attention. It is known [9] that the problem is in NP $\cap$ co-NP and no polynomial time algorithm is known. It follows from the general result of Courcelle [4] that there is a polynomial time algorithm that solves parity games on graphs of bounded tree-width. Obdržàlek [11] exhibited a particular such algorithm. 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 paper, we give just such a generalisation. We introduce a new measure of the connectivity of graphs that we call DAG-width[4]. It is intermediate between tree-width and directed tree-width, in that for any graph $\mathcal{G}$, the directed tree-width of $\mathcal{G}$ 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).

The notion of DAG-width can be understood as a simple adaptation of the game of *cops and robber* (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 nodes 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 [15, 5, 7, 1, 2]). It is known [15] that the cop player has a winning strategy on an undirected graph $\mathcal{G}$ using $k + 1$ cops if, and only if, $\mathcal{G}$ has tree-width k. We consider the natural

[4] We understand that Obdržàlek has defined a similar measure in a paper to appear at SODA'06. We have not yet had an opportunity to see that paper.

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. All proofs are in the appendix.

2 Preliminaries

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

We write ω for the set of finite ordinals, i.e. natural numbers. For every $n \in \omega$, we write $[n]$ for the set $\{1, \ldots, n\}$. For every set V and every $k \in \omega$, we write $[V]^k$ for the set of all k-element subsets of V, that is, $[V]^k := \{\{x_1, \ldots, x_k\} \subseteq V : x_i \neq x_j$ whenever $i \neq j\}$. We write $[V]^{\leq k}$ for the set of all $X \subseteq V$ with $|X| \leq k$.

Let $\mathcal{G}$ be a directed graph. We write $V^{\mathcal{G}}$ for the set of its vertices and $E^{\mathcal{G}}$ for the set of its edges. Let $V \subseteq V^{\mathcal{G}}$ be a set of vertices. We write $\langle V \rangle^{\mathcal{G}}$ for the sub-graph induced by V. Further, E^{op} denotes the set of edges that results from reversing the edges in E, i.e. $E^{op} = \{(w, v) : (v, w) \in E\}$. The graph $\mathcal{G}^{op}$ is defined to be (V, E^{op}).

The *block graph* of a graph $\mathcal{G}$ is the graph (H, I) where H is the set of strongly connected components of $\mathcal{G}$ and there is an edge $(u, v) \in I$ if there is an edge in $\mathcal{G}$ from a node in u to a node in v.

A tree-decomposition of a graph $\mathcal{G}$ is a labelled tree $(\mathcal{T}, (X_t)_{t \in V^{\mathcal{T}}})$ where $X_t \subseteq V^{\mathcal{G}}$ for each vertex $t \in V^{\mathcal{T}}$, for each edge $(u, v) \in E^{\mathcal{G}}$ there is a $t \in V^{\mathcal{T}}$ such that $\{u, v\} \subseteq X_t$, and for each $v \in V^{\mathcal{G}}$, the set $\{t \in V^{\mathcal{T}} : v \in X_t\}$ forms a connected subtree of $\mathcal{T}$. The *width* of a tree-decomposition is the cardinality of the largest X_t minus one. The tree-width of $\mathcal{G}$ is the smallest k such that $\mathcal{G}$ has a tree-decomposition of width k.

Let $\mathcal{D} := (D, A)$ be a directed, acyclic graph (DAG). The partial order $\preceq_{\mathcal{D}}$ (or $\preceq_A$) on D is the reflexive, transitive closure of A. A *root* of a set $X \subseteq D$ is a $\preceq_{\mathcal{D}}$-minimal element of X, that is, $r \in X$ is a root of X if there is no $y \in X$ such that $y \preceq_{\mathcal{D}} r$. Analogously, a *leaf* of $X \subseteq D$ is a $\preceq_{\mathcal{D}}$-maximal element.

3 Games, Strategies and Decompositions

This section contains the graph theoretical part of this paper. 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 [8], but these games appear to be less intuitive. In this paper, 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 Game. 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 3.1 (Cops and Robber Game). Given a graph $\mathcal{G} := (V, E)$, the *k-cops and robber* game on $\mathcal{G}$ 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 $V \setminus X_0$, giving position (X_0, r_0).
- From position (X_i, r_i), the cop player chooses $X_{i+1} \in [V]^{\leq k}$, and the robber player chooses a vertex r_{i+1} of $V \setminus X_{i+1}$ such that there is a path from r_i to r_{i+1} which does not pass through a vertex in $X_i \cap X_{i+1}$. If no such vertex exists then 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_{r+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 3.2 (Game-width). The *game-width gw*$(\mathcal{G})$ of $\mathcal{G}$ is the least k such that the cop player has a strategy to win the k-cops and robber game on $\mathcal{G}$.

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 game-width of a graph does not depend on the variant.

Lemma 3.3. *For every finite, non-empty, directed graph $\mathcal{G}$ the game-width gw$(\mathcal{G})$ is at least one and gw$(\mathcal{G}) = 1$ if, and only if, $\mathcal{G}$ is acyclic.*

Proof. We have no requirement that the robber moves, so as long as there is one vertex, the robber can defeat zero cops by remaining on that vertex. If $\mathcal{G}$ is acyclic then one cop can catch the robber by playing to the robber's current position. Eventually the robber will not be able to move and the cop will capture him. Conversely, if $\mathcal{G}$ has a cycle then the robber can defeat one cop by forever staying in the cycle. □

Games similar to the one defined above have been used to give game characterisations of concepts like undirected tree-width [15] and also the directed tree-width of [8]. Directed tree-width is invariant under reversing the edges of a graph. As we see below, this is not true of the game-width we have defined. One exception are graphs of game-width 1, i.e. acyclic graphs.

Proposition 3.4. *$gw(\mathcal{G}) = 1$ if, and only if $gw(\mathcal{G}^{op}) = 1$.*

Proof. If $\mathcal{G}$ has a cycle $(v_0, v_1, \ldots, v_n)$ then $\mathcal{G}^{op}$ also has a cycle: $(v_n, v_{n-1}, \ldots, v_0)$, and vice-versa. Hence, by Lemma 3.3, $\text{gw}(\mathcal{G}) \neq 1$ iff $\mathcal{G}$ has a cycle, iff $\mathcal{G}^{op}$ has a cycle, iff $\text{gw}(\mathcal{G}^{op}) \neq 1$. $\square$

Proposition 3.5. *For any j, k with $2 \leq j \leq k$, there exists a graph $\mathcal{T}_k^j$ such that $gw(\mathcal{T}_k^j) = j$ and $gw((\mathcal{T}_k^j)^{op}) = k$.*

Proof. Let $\mathcal{T}_k^j$ be a binary branching tree of height k (i.e. $2^k - 1$ vertices) where every vertex v has a "forward-edge"[5] 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 nearest ancestors. To show $\text{gw}(\mathcal{T}_k^j) = j$, 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, 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. Robber then plays to that ancestor and along that path to the leaf.

For $(\mathcal{T}_k^j)^{op}$, k cops suffice to chase the robber down from the root. To defeat $k - 1$ cops, the robber plays the strategy that defeats $j - 1$ cops in $\mathcal{T}_k^j$. $\square$

As always when dealing with games we are less interested in a single play in the game as in strategies that allow a player to win every play in the game. Winning strategies for the cop player play a crucial role throughout this paper. We therefore give a precise definition of this notion.

Definition 3.6. Let $\mathcal{G} := (V, E)$ 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 $(X_0, r_0), (X_1, r_1), \ldots$ 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 $(X_0, r_0), (X_1, r_1) \ldots$ is a play consistent with the strategy, then for every $0 \leq i < n$ and $v \in X_i \setminus X_{i+1}$, $v \notin X_j$ 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 3.7. *If the cop player has a cop-monotone or robber-monotone winning strategy then it 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 $(X_0, r_0), (X_1, r_1) \ldots$ 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 $v \in X_i$, robber is unable to reach v when the cops are on X_i.

[5] To aid description of strategies, we prefer to view $\mathcal{T}_k^j$ as a directed binary branching tree with some additional structure, there is no actual difference between edges, forward-edges and back-edges

As the strategy is robber-monotone, 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 robber from reaching v – if robber can reach v, then he must have reached a vertex which he could not when the cops were on X_{i+1}, contradicting the robber-monotonicity of the strategy. Thus we can remove v from all X_j, $j > i$. Proceeding in this way results in a sequence $(X_0, r_0), (X'_1, r_1), \ldots$. 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 it 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 $(X_0, r_0), (X_1, r_1) \ldots$ be a play consistent with the strategy, and $R_0, R_1, \ldots, R_n$ 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 $j > i$, robber is unable to reach v otherwise robber could play to v and sit there indefinitely – contradicting the fact that cop player is playing a winning strategy. Thus the robber is unable to reach any of the vertices in $X_i \setminus X_{i+i}$ and is therefore unable to reach any new vertices. Hence $R_i \supseteq R_{i+1}$, so the strategy is robber-monotone. $\square$

From this lemma we can define a *monotone winning strategy* in the obvious way.

3.2 DAG-Decompositions and DAG-Width

In this section we define the notion of DAG-width which measures how close a given graph is to being acyclic. We present a decomposition of directed graphs that is somewhat similar in style to tree-decompositions of undirected graphs. 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 3.8. Let $\mathcal{G} := (V, E)$ 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 3.9 (DAG-decomposition)**.** Let $\mathcal{G} := (V, E)$ be a directed graph. A DAG-*decomposition* is a tuple $\mathfrak{D} = (\mathcal{D}, (X_d)_{d \in V^{\mathcal{D}}})$ such that

(D1) $\mathcal{D}$ is a DAG.
(D2) $\bigcup_{d \in V^{\mathcal{D}}} X_d = V$.
(D3) For all $d \preceq_{\mathcal{D}} d' \preceq_{\mathcal{D}} d''$, $X_d \cap X_{d''} \subseteq X_{d'}$.
(D4) For a root d, $\mathcal{X}_d$ is guarded by $\varnothing$.
(D5) For all $(d, d') \in E^{\mathcal{D}}$, $X_d \cap X_{d'}$ guards $\mathcal{X}_{d'} \setminus X_d$, where $\mathcal{X}_{d'} := \bigcup_{d' \preceq_{\mathcal{D}} d''} X_{d''}$.

The width of $\mathfrak{D}$ is defined as $\max\{|X_d| : d \in V^{\mathcal{D}}\}$. The DAG-*width* of a graph is defined as the minimal width of any of its DAG-decompositions.

The main result of this section is an equivalence between monotone strategies for the cop player and DAG-decompositions.

Theorem 3.10. *For any graph $\mathcal{G}$ there is a* DAG*-decomposition of $\mathcal{G}$ of width k if, and only if, the cop player has a monotone winning strategy in the k-cops and robber game on $\mathcal{G}$.*

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

Lemma 3.11.

(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 $(v, w) \in E^{\mathcal{G}}$, $v \in Y_0 \cup Y_1$ 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 $(v, w) \in E^{\mathcal{G}}$, $v \in Y$ and $w \notin Y$. As X guards Y, $w \in X$. As $Z \supseteq X$, $w \in Z$. Therefore, Z guards Y.
(iii) Suppose $(v, w) \in E^{\mathcal{G}}$, $v \in Y \setminus Z$ and $w \notin Y \setminus Z$. Thus $w \notin Y$ or $w \in Z$. For the first case, $w \in X$ as X guards Y. Hence $w \in X \cup Z$.

□

We now turn to the proof of Theorem 3.10. Suppose the cop player has a monotone winning strategy f in the k-cops and robber game on a graph $\mathcal{G}$. 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,

$$f(X, r) \subseteq X \cup \mathit{Reach}_{\mathcal{G} \setminus X}(r). \tag{1}$$

To prove the theorem we first capture the k-cops and robber game on $\mathcal{G}$ by a simpler, token-moving game. This game is played on a graph $\Pi_k(\mathcal{G})$, where $V^{\Pi_k(\mathcal{G})} := [V^{\mathcal{G}}]^{\leq k} \times V^{\mathcal{G}}$ and $E^{\Pi_k(\mathcal{G})}$ consists of all pairs $\big((X, r), (X', r')\big)$ such that $r' \in \mathit{Reach}_{\mathcal{G} \setminus (X \cap X')}(r)$. That is, $\big((X, r), (X', r')\big)$ 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 which the robber player moves a token around $\Pi_k(\mathcal{G})$, 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 3.26.

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

$$\textit{Reach}_{\mathcal{G}\setminus X}(r) \subseteq \bigcup_{d \preceq_{\mathcal{D}} d'} f(d') \subseteq X \cup \textit{Reach}_{\mathcal{G}\setminus X}(r). \tag{2}$$

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

$$\textit{Reach}_{\mathcal{G}\setminus X}(r) = \textit{Reach}_{\mathcal{G}\setminus(X\cap f(X,r))}(r). \tag{3}$$

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

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

$$\big(\bigcup_{d \preceq_{\mathcal{D}} d'} f(d')\big) \setminus X = \textit{Reach}_{\mathcal{G}\setminus(X\cap f(X,r))}(r). \tag{4}$$

We now show that $\mathfrak{D}$ is indeed a DAG-decomposition of width $\leq k$. As observed above, $\mathcal{D}$ is a DAG as otherwise f would not be a winning strategy for the cop player. For (D2), if there was a $v \in V \setminus \bigcup_{d\in V^{\mathcal{D}}} X_d$, then, as we assumed f initially placed no cops, the robber could defeat f by playing to v and staying there indefinitely. Hence $\bigcup_{d\in V^{\mathcal{D}}} X_d = V$. (D3) follows immediately from the monotonicity of the winning strategy f. Towards establishing (D4), let $d = (X, r)$ be a root of $\mathcal{D}$. As we assumed the first move of f is to place no cops, X must be $\varnothing$. By Equation 4,

$$\mathcal{X}_d = \big(\bigcup_{d \preceq_{\mathcal{D}} d'} f(d')\big) = \textit{Reach}_{\mathcal{G}}(r),$$

and is therefore guarded by $\varnothing$. Finally, to show (D5), suppose $(d, d') \in E^{\mathcal{D}}$. If $d' = (X', r')$ then $X_d = f(d) = X'$. So by Equation 4,

$$\mathcal{X}_{d'} \setminus X_d = \big(\bigcup_{d' \preceq_{\mathcal{D}} d''} f(d'')\big) \setminus X' = \textit{Reach}_{\mathcal{G}\setminus(X'\cap f(X',r'))}(r').$$

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

Conversely, let $(\mathcal{D}, (X_d)_{d\in V^{\mathcal{D}}})$ 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 (D2), there exists $d_v \in V^{\mathcal{D}}$ such that $v \in X_{d_v}$. Let d be a root of $\mathcal{D}$ which lies above d_v.
(2) Place cops on X_d.
(3) From (D5) and Lemma 3.11(ii), X_d guards $\mathcal{X}_d \setminus X_d$. Therefore, the robber can only move to vertices in $\mathcal{X}_d \setminus X_d$. 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 $\mathcal{X}_{d'} \setminus X_d$, the robber can only move to vertices in $\mathcal{X}_{d'}$ – that is, the robber must remain in the sub-DAG rooted at d'.
(6) Return to step 2 with d' as d.

As $\mathcal{D}$ is a DAG, at some point the robber player will not be able to move (since $\mathcal{X}_d \setminus X_d$ is empty when d is a leaf). Hence this is a winning strategy for k cops. To show that it is monotone, observe that (D3) ensures that at no point does a cop return to a vacated vertex. This concludes the proof of Theorem 3.10. □

The remainder of this section looks at some properties of DAG-decompositions motivated by similar results for tree-width and tree decompositions. First we make two useful observations.

Lemma 3.12. *Let $(\mathcal{D}, (X_d)_{d \in V^\mathcal{D}})$ be a* DAG-*decomposition. For all $(d, d') \in E^\mathcal{D}$,*

$$\mathcal{X}_{d'} \setminus X_d = \mathcal{X}_{d'} \setminus (X_d \cap X_{d'}).$$

Proof. As $X_d \cap X_{d'} \subseteq X_d$, $\mathcal{X}_{d'} \setminus X_d \subseteq \mathcal{X}_{d'} \setminus (X_d \cap X_{d'})$. Conversely, suppose $x \in \mathcal{X}_{d'}$. We will show that $x \in X_d \cap X_{d'}$, or $x \notin X_d$. Let $x \in X_{d''}$ for $d' \preceq_\mathcal{D} d''$, and suppose $x \in X_d$. Then as $d \preceq_\mathcal{D} d' \preceq_\mathcal{D} d''$, $x \in X_d \cap X_{d''} \subseteq X_{d'}$. Hence $x \in X_d \cap X_{d'}$. Thus $\mathcal{X}_{d'} \setminus X_d \supseteq \mathcal{X}_{d'} \setminus (X_d \cap X_{d'})$. □

Lemma 3.13. *Let $\mathfrak{D} = (\mathcal{D}, (X_d)_{d \in V^\mathcal{D}})$ be a* DAG-*decomposition of a graph $\mathcal{G}$. For $W \subseteq V^\mathcal{G}$, $\mathfrak{D}|_W := (\mathcal{D}, (X_d \cap W)_{d \in V^\mathcal{D}})$ is a* DAG-*decomposition of $\langle W \rangle^\mathcal{G}$.*

Proof. Clearly (D1), (D2), and (D3) still hold for $\mathfrak{D}|_W$. For (D4) and (D5), we observe that if X guards Y in $\mathcal{G}$, then $X \cap W$ guards $Y \cap W$ in $\langle W \rangle^\mathcal{G}$. For if $v \in Y \cap W$, $w \in W \setminus Y$ and $(v, w) \in E^\mathcal{G}$, then $w \in X$ (as X guards Y), hence $w \in X \cap W$. (D4) and (D5) then follow immediately from the respective conditions for $\mathfrak{D}$. □

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 [3].

Definition 3.14. A DAG-decomposition $(\mathcal{D}, (X_d)_{d \in V^\mathcal{D}})$ is *nice* if

(N1) $\mathcal{D}$ has a unique root.
(N2) Every $d \in V^\mathcal{D}$ has at most two successors.
(N3) If d_1, d_2 are two successors of d_0, then $X_{d_0} = X_{d_1} = X_{d_2}$.
(N4) If d_1 is the unique successor of d_0, then $|X_{d_0} \Delta X_{d_1}| \leq 1$, where Δ is the symmetric set difference operator ($A \Delta B = (A \setminus B) \cup (B \setminus A)$).

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 3.15 (Unique root)**.** *Let $(\mathcal{D}, (X_d)_{d \in V^\mathcal{D}})$ be a* DAG-*decomposition of width k, and let $d_1, d_2, \ldots d_m$ be the roots of $\mathcal{D}$. The decomposition $(\mathcal{D}', (X'_d)_{d \in V^{\mathcal{D}'}})$ where*

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

is a DAG*-decomposition of width* k.

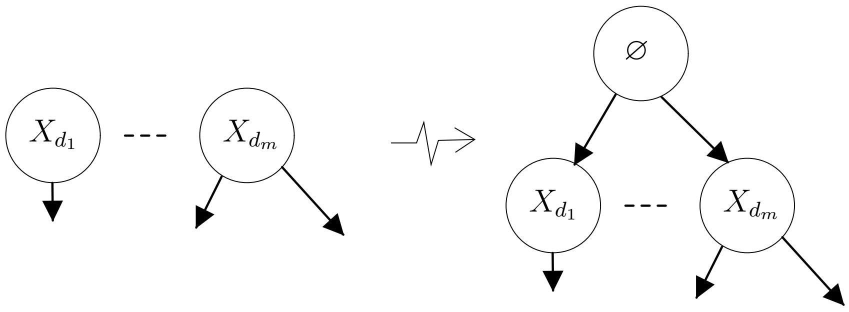

Fig. 1. Forming a unique root

Proof. As we have only added edges from $r \notin V^{\mathcal{D}}$, $\mathcal{D}'$ is acyclic. (D2) is trivially satisfied as we have only added a node. If $d \preceq_{\mathcal{D}'} d' \preceq_{\mathcal{D}'} d''$, then either $d = r$ in which case $X'_d \cap X'_{d''} = \varnothing \subseteq X'_{d'}$, or $d \in V^{\mathcal{D}}$, in which case $X'_d \cap X'_{d''} \subseteq X'_{d'}$ follows from the fact that $(\mathcal{D}, (X_d)_{d \in \mathcal{D}})$ is a DAG-decomposition. This establishes (D3). (D4) is again trivially satisfied, as $\mathcal{X}'_r = V^{\mathcal{G}}$. Finally, for $(r, d_i) \in E^{\mathcal{D}'}$, $X'_r \cap X'_{d_i} = \varnothing$ guards $\mathcal{X}_{d_i} = \mathcal{X}'_{d_i} \setminus X'_r$. Otherwise $(d, d') \in E^{\mathcal{D}}$ and (D5) follows from the fact that $(\mathcal{D}, (X_d)_{d \in \mathcal{D}})$ is a DAG-decomposition. As $|X'_r| = 0$, $(\mathcal{D}', (X'_d)_{d \in V^{\mathcal{D}'}})$ has width k.

Figure 1 gives a visual representation of the construction. □

Definition 3.16 (Splitting)**.** Let $\mathfrak{D} := (\mathcal{D}, (X_d)_{d \in V^{\mathcal{D}}})$ be a DAG-decomposition, and suppose $d_0 \in V^{\mathcal{D}}$ has $m > 1$ successors $d_1, d_2, \ldots, d_m$. The decomposition $\mathfrak{D}' := (\mathcal{D}', (X'_d)_{d \in V^{\mathcal{D}'}})$ obtained from $\mathfrak{D}$ by *splitting* d_0 is defined as

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

Figure 2 gives a visual representation of this transformation.

Lemma 3.17. *Let* $\mathfrak{D} = (\mathcal{D}, (X_d)_{d \in V^{\mathcal{D}}})$ *be a* DAG*-decomposition of a graph* $\mathcal{G}$ *of width* k*, and suppose* $d_0 \in V^{\mathcal{D}}$ *has* $m > 1$ *successors* $d_1, d_2, \ldots, d_m$*. Then* $\mathfrak{D}' := (\mathcal{D}', (X'_d)_{d \in V^{\mathcal{D}'}})$ *obtained from* $\mathfrak{D}$ *by splitting* d_0 *is a* DAG*-decomposition of* $\mathcal{G}$ *of width* k*.*

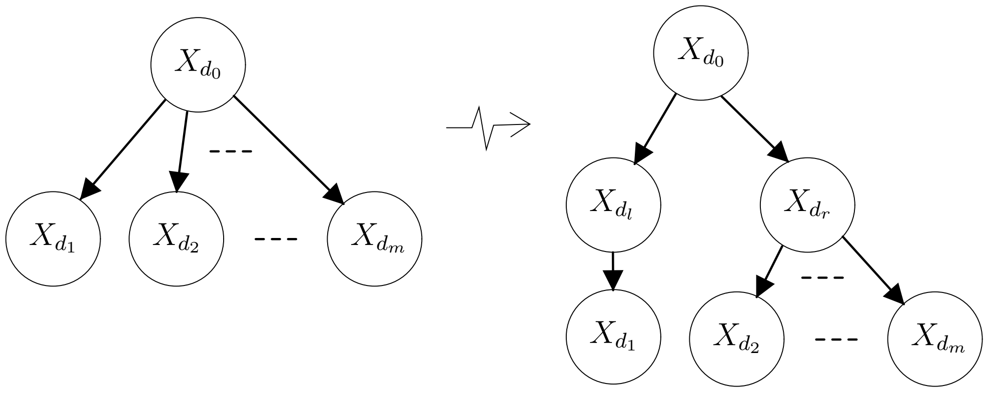

Fig. 2. Splitting at d_0

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

$$\mathcal{X}'_d = \bigcup_{d \preceq_{\mathcal{D}'} d'} X'_{d'} = \bigcup_{d \preceq_{\mathcal{D}} d'} X_{d'} = \mathcal{X}_d,$$

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 $1 \leq i \leq m$, $X_{d_0} \cap X_{d_i}$ guards $\mathcal{X}_{d_i} \setminus X_{d_0}$, so by Lemma 3.11(ii),

$$X_{d_0} \text{ guards } \mathcal{X}_{d_i} \setminus X_{d_0}. \tag{5}$$

It is easily seen that the edges added do not create any cycles, so $\mathcal{D}'$ is a DAG. Further, $\bigcup_{d \in V^{\mathcal{D}'}} X'_d = \bigcup_{d \in V^{\mathcal{D}}} X_d = V^{\mathcal{G}}$. To prove the connectivity condition (D3), let $d, d', d'' \in V^{\mathcal{D}'}$, be such that $d \preceq_{\mathcal{D}'} d' \preceq_{\mathcal{D}'} d''$. If $d' = d$ or d'' then trivially $X'_d \cap X'_{d''} \subseteq X'_{d'}$, so suppose $d \prec_{\mathcal{D}'} d' \prec_{\mathcal{D}'} d''$. We consider four cases:

- If none of d, d', d'' is d_l or d_r, then $d, d', d'' \in \mathcal{D}$, and (D3) follows from the fact that $\mathfrak{D}$ is a DAG-decomposition.
- If d is d_l or d_r then since all descendants of d are in $V^{\mathcal{D}}$, and $d_0 \in V^{\mathcal{D}}$ is the unique predecessor of d, we obtain the following chain of nodes in $\mathcal{D}$: $d_0 \prec_{\mathcal{D}} d' \prec_{\mathcal{D}} d''$. So $X'_d \cap X'_{d''} = X_{d_0} \cap X_{d''} \subseteq X_{d'} = X'_{d'}$.
- If d'' is d_l or d_r then from the comments at the start of the proof, $d \prec_{\mathcal{D}} d' \preceq_{\mathcal{D}} d_0$. Thus, $X'_d \cap X'_{d''} = X_d \cap X_{d_0} \subseteq X_{d'} = X'_{d'}$.
- Finally, if d' is d_l or d_r then by the same reasoning as the previous two cases, $d \preceq_{\mathcal{D}} d_0 \prec_{\mathcal{D}} d''$. So $X'_d \cap X'_{d''} = X_d \cap X_{d''} \subseteq X_{d_0} = X'_{d'}$.

Thus, in all cases, $X'_d \cap X'_{d''} \subseteq X'_{d'}$, showing (D3). (D4) follows from the fact that every root of $\mathcal{D}'$ is a root of $\mathcal{D}$ too. So $\varnothing$ guards $\mathcal{X}_d = \mathcal{X}'_d$. Finally, to show (D5), let $(d, d') \in E^{\mathcal{D}'}$. We consider three cases:

– $d' \in V^{\mathcal{D}}$ (i.e. $d' \neq d_l, d_r$). If $d = d_l$ or d_r, then $X'_d = X_{d_0}$. Otherwise $(d, d') \in E^{\mathcal{D}}$. In both cases, $X'_d \cap X'_{d'}$ guards $\mathcal{X}'_{d'} \setminus X'_d$.
– $d' = d_l$ (so $d = d_0$). Here $\mathcal{X}'_{d'} = X_{d_0} \cup \mathcal{X}_{d_1}$, so $\mathcal{X}'_{d'} \setminus X'_d = \mathcal{X}_{d_1} \setminus X_{d_0}$. Hence, by Equation 5, $X_{d_0} = X'_d \cap X'_{d'}$ guards $\mathcal{X}_{d_1} \setminus X_{d_0} = \mathcal{X}'_{d'} \setminus X'_d$.
– $d' = d_r$ (so $d = d_0$). Here $\mathcal{X}'_{d'} = X_{d_0} \cup \bigcup_{2 \leq i \leq m} \mathcal{X}_{d_i}$, and so $\mathcal{X}'_{d'} \setminus X'_d = (\bigcup \mathcal{X}_{d_i}) \setminus X_{d_0} = \bigcup(\mathcal{X}_{d_i} \setminus X_{d_0})$, where the unions are taken over i for $2 \leq i \leq m$. From Lemma 3.11(i) and Equation 5, $X'_d \cap X'_{d'} = X_{d_0}$ guards $\bigcup_{2 \leq i \leq m}(\mathcal{X}_{d_i} \setminus X_{d_0}) = \mathcal{X}'_{d'} \setminus X'_d$.

As $X'_{d_l} = X'_{d_r} = X_{d_0}$, $\max\{|X'_d| : d \in V^{\mathcal{D}'}\} = \max\{|X_d| : d \in V^{\mathcal{D}}\} = k$. So $(\mathcal{D}', (X'_d)_{d \in V^{\mathcal{D}'}})$ has width k. □

By the *decomposition resulting from splitting* d $m-1$ *times* we mean the decomposition resulting from splitting d, and then recursively splitting the successor with more than one successor until no such successor exists. A *complete split* of $\mathfrak{D}$ is the decomposition $\mathfrak{D}'$ obtained by recursively splitting every node with more than two children.

Definition 3.18 (Adding)**.** Let $\mathfrak{D} = (\mathcal{D}, (X_d)_{d \in V^{\mathcal{D}}})$ be a DAG-decomposition of a graph $\mathcal{G}$. If $(d_0, d_1) \in E^{\mathcal{D}}$ and $X \subseteq V^{\mathcal{G}}$ the *decomposition resulting from adding* X *to* (d_0, d_1) is the decomposition $(\mathcal{D}', (X'_d)_{d \in V^{\mathcal{D}'}})$ where

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

See Figure 3 for a visual interpretation.

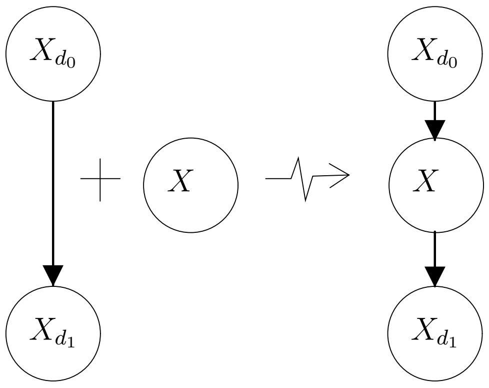

Fig. 3. Adding X to (d_0, d_1)

Lemma 3.19. *Let $\mathfrak{D} = (\mathcal{D}, (X_d)_{d \in V^{\mathcal{D}}})$ be a* DAG*-decomposition of a graph $\mathcal{G}$ of width k and let $\mathfrak{D}' := (\mathcal{D}', (X'_d)_{d \in V^{\mathcal{D}'}})$ be the decomposition resulting from adding $X \subseteq V^{\mathcal{G}}$ to (d_0, d_1). If either*

(i) $X_{d_0} \cap X_{d_1} \subseteq X \subseteq X_{d_0}$, *or*
(ii) $X_{d_0} \cap X_{d_1} \subseteq X \subseteq X_{d_1}$,

then $\mathfrak{D}'$ is a DAG*-decomposition of $\mathcal{G}$ of width k.*

Proof. We observe that for all $d \in V^{\mathcal{D}}$, if $d \prec_{\mathcal{D}'} d_X$, then as $d_0 \in V^{\mathcal{D}}$ is the unique predecessor of d_X, $d \preceq_{\mathcal{D}} d_0$, and if $d_X \prec_{\mathcal{D}'} d$, then as $d_1 \in V^{\mathcal{D}}$ is the unique successor of d_X, $d_1 \preceq_{\mathcal{D}} d$. This implies, for all $d \in V^{\mathcal{D}}$

$$\mathcal{X}'_d = \bigcup_{d \preceq_{\mathcal{D}'} d'} X'_{d'} = \bigcup_{d \preceq_{\mathcal{D}} d'} X_{d'} = \mathcal{X}_d,$$

since if X'_{d_X} is inluded 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 $\mathcal{X}_{d_1} \setminus X_{d_0} = \mathcal{X}_{d_1} \setminus (X_{d_0} \cap X_{d_1})$ from Lemma 3.12.

We now show that $\mathfrak{D}'$ satisfies the properties (D1) to (D5). It is easily seen that $\mathcal{D}'$ is a DAG and $\bigcup_{d \in V^{\mathcal{D}'}} X'_d = X \cup \bigcup_{d \in V^{\mathcal{D}}} X_d = V^{\mathcal{G}}$. Also, if d is a root of $\mathcal{D}'$, then d is a root of $\mathcal{D}$. Hence $\varnothing$ guards $\mathcal{X}_d = \mathcal{X}'_d$. This shows (D1), (D2), and (D4). Towards establishing condition (D3), suppose $d \preceq_{\mathcal{D}'} d' \preceq_{\mathcal{D}'} d''$. If $d' = d$ or $d' = d''$ then trivially $X'_d \cap X'_{d''} \subseteq X'_{d'}$, so suppose $d \prec_{\mathcal{D}'} d' \prec_{\mathcal{D}'} d''$. We consider four cases:

– If none of d, d', d'' is d_X then d, d', and d'' are all in $V^{\mathcal{D}}$, so (D3) follows from the fact that $\mathfrak{D}$ is a DAG-decomposition.
– Suppose $d = d_X$. From the observations made at the start of the proof, we get the following chain of nodes in $\mathcal{D}$: $d_0 \prec_{\mathcal{D}} d_1 \preceq_{\mathcal{D}} d' \prec_{\mathcal{D}} d''$. 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 (D3) of $\mathfrak{D}$. If $X \subseteq X_{d_1}$, then $X'_d \cap X'_{d''} = X \cap X_{d''} \subseteq X_{d_1} \cap X_{d''} \subseteq X_{d'} = X'_{d'}$.
– The other cases are similar. If $d'' = d_X$ then we obtain $d \prec_{\mathcal{D}} d' \preceq_{\mathcal{D}} d_0 \prec_{\mathcal{D}} d_1$. So if $X \subseteq X_{d_0}$, then $X'_d \cap X'_{d''} = X_d \cap X \subseteq X_d \cap X_{d_0} \subseteq X_{d'} = X'_{d'}$. If $X \subseteq X_{d_1}$, then $X'_d \cap X'_{d''} = X_d \cap X \subseteq X_d \cap X_{d_1} \subseteq X_{d'} = X'_{d'}$.
– Finally, assume $d' = d_X$. Then $d \preceq_{\mathcal{D}} d_0 \prec_{\mathcal{D}} d_1 \preceq_{\mathcal{D}} d''$. Hence $X_d \cap X_{d''} \subseteq X_{d_0}$ and $X_d \cap X_{d''} \subseteq X_{d_1}$. Thus $X'_d \cap X'_{d''} = X_d \cap X_{d''} \subseteq X_{d_0} \cap X_{d_1} \subseteq X = X'_{d'}$.

Finally, towards (D5), let $(d, d') \in E^{\mathcal{D}'}$. We consider three cases:

– $d_X \notin \{d, d'\}$, i.e. $(d, d') \in E^{\mathcal{D}}$. In this case, (D5) follows from the fact that $\mathfrak{D}$ 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

$$\mathcal{X}_{d_1} \setminus (X_{d_0} \cap X_{d_1}) \quad \supseteq \quad \mathcal{X}_{d_1} \setminus X \quad \supseteq \quad \mathcal{X}_{d_1} \setminus X_{d_0}.$$

Further, by Lemma 3.12, $\mathcal{X}_{d_1} \setminus (X_{d_0} \cap X_{d_1}) = \mathcal{X}_{d_1} \setminus X_{d_0}$. Therefore $\mathcal{X}_{d_1} \setminus X = \mathcal{X}_{d_1} \setminus X_{d_0}$. As $\mathfrak{D}$ is a DAG-decomposition, $X_{d_0} \cap X_{d_1}$ guards $\mathcal{X}_{d_1} \setminus X_{d_0}$, and as $X_{d_0} \cap X_{d_1} \subseteq X \cap X_{d_1}$, Lemma 3.11(ii) implies that $X'_d \cap X'_{d_1} = X \cap X_{d_1}$ guards $\mathcal{X}_{d_1} \setminus X_{d_0} = \mathcal{X}'_{d_1} \setminus X'_d$.

Otherwise we are in case (ii) and we have $X_{d_0} \cap X_{d_1} \subseteq X \subseteq X_{d_1}$. Let $Z = X \backslash (X_{d_0} \cap X_{d_1})$. We know $(X_{d_0} \cap X_{d_1})$ guards $\mathcal{X}_{d_1} \backslash (X_{d_0} \cap X_{d_1})$ from Lemma 3.12. Hence, from Lemma 3.11(iii) $X'_d \cap X'_{d_1} = X = (X_{d_0} \cap X_{d_1}) \cup Z$ guards

$$(\mathcal{X}_{d_1} \backslash (X_{d_0} \cap X_{d_1})) \backslash Z = \mathcal{X}_{d_1} \backslash ((X_{d_0} \cap X_{d_1}) \cup Z) = \mathcal{X}_{d_1} \backslash X = \mathcal{X}'_{d_1} \backslash X'_{d'}.$$

– Finally, suppose $d' = d_X$ (so $d = d_0$). Here we claim $\mathcal{X}'_{d_X} \setminus X'_{d_0} = \mathcal{X}_{d_1} \setminus X_{d_0}$. If $X \subseteq X_{d_0}$, then $\mathcal{X}'_{d_X} \setminus X'_{d_0} = (X \cup \mathcal{X}_{d_1}) \setminus X_{d_0} = (X \setminus X_{d_0}) \cup (\mathcal{X}_{d_1} \setminus X_{d_0}) = \mathcal{X}_{d_1} \setminus X_{d_0}$. If $X \subseteq X_{d_1}$, then since $d_X \preceq_{\mathcal{D}'} d_1$, $\mathcal{X}'_{d_X} = \mathcal{X}'_{d_1} = \mathcal{X}_{d_1}$. Now $X \supseteq X_{d_0} \cap X_{d_1}$, so by Lemma 3.11(ii), $X'_{d'} = X$ guards $\mathcal{X}_{d_1} \setminus X_{d_0} = \mathcal{X}'_{d_X} \setminus X'_{d_0}$.

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

If $X_1, X_2, \ldots, X_n$ is a sequence of subsets of $V^{\mathcal{G}}$, the *decomposition resulting from adding* $X_1, X_2, \ldots, X_n$ *to* (d_0, d_1) is the decomposition resulting from adding X_1 to (d_0, d_1) and then recursively adding X_{i+1} to (d_{X_i}, d_1).

We can now describe how to transform a DAG-decomposition into one which is nice with the same width.

Theorem 3.20. *If $\mathcal{G}$ has a* DAG*-decomposition of width k, it has a nice* DAG*-decomposition of width k.*

Proof. Let $\mathfrak{D} = (\mathcal{D}, (X_d)_{d \in \mathcal{D}})$ be a DAG-decomposition of width k. We carry out each of the following steps and reset $\mathfrak{D}$ to be the resulting decomposition.

1. We apply Lemma 3.15 to obtain a decomposition with a unique root, therefore satisfying (N1).
2. We apply a complete split on $\mathfrak{D}$ 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 $(d_0, d_1) \in E^{\mathcal{D}}$ with $X_{d_0} \neq X_{d_1}$, we add $X_{d_0} \cap X_{d_1}$ to (d_0, d_1) to obtain a DAG-decomposition such that for every $(d, d') \in E^{\mathcal{D}'}$, X_d is either a subset or a super-set of $X_{d'}$.
4. Secondly, for each $(d, d') \in E^{\mathcal{D}}$ with $|X_d| - |X_{d'}| = m > 1$ (or $|X_{d'}| - |X_d| = m > 1$), let $X_0 = X_d, X_1, \ldots, X_m = X_{d'}$ be a strictly decreasing (increasing) sequence of subsets. Such a sequence exists because at the previous step we finished with a DAG-decomposition such that $X_d \subseteq X_{d'}$ or $X_d \supseteq X_{d'}$. Add $X_1, X_2, \ldots, X_{m-1}$ to (d, d'). At this point we have a decomposition which satisfies (N1) to (N4), and is therefore nice.

Finally, from Lemmas 3.15, 3.17, and 3.19, at each step we have a DAG-decomposition of width k. □

Tree-width on undirected graphs also have a useful characterisation in terms of balanced separators. We are able to obtain one direction of a similar characterisation for DAG-width by showing that graphs of small DAG-width admit small balanced *directed separators*.

Definition 3.21. Let $\mathcal{G} := (V, E)$ be a digraph and $W \subseteq V$ be a set of nodes. 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 $W \setminus X = W_1 \dot{\cup} W_2$, $W_1 \cap W_2 = \varnothing$, $\frac{1}{3}|W| \leq |W_i| \leq \frac{2}{3}|W|$, $i = 1, 2$, and there is no path from a node W_2 to a node in W_1 in $\mathcal{G} \setminus X$.

A graph $\mathcal{G}$ has the *directed k-separator property* if every set $\subseteq V^{\mathcal{G}}$ has a directed separator of width k.

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

Theorem 3.22. *Every graph of* DAG-*width at most k has the directed k-separator property.*

Using Lemma 3.13, the proof of this theorem follows along the same lines as for the case of undirected graphs.

Finally, we show that the DAG-width of graphs is closed under directed unions, which is considered (see [8]) as an important property of a reasonable decomposition of directed graphs.

Definition 3.23. Let $\mathcal{G}$ and $\mathcal{H}$ be (disjoint) directed graphs. The *directed union* of $\mathcal{G}$ and $\mathcal{H}$, denoted $\mathcal{G} \overrightarrow{\cup} \mathcal{H}$ is defined as:

$$\mathcal{G} \overrightarrow{\cup} \mathcal{H} := \big(V^{\mathcal{G}} \cup V^{\mathcal{H}}, E^{\mathcal{G}} \cup E^{\mathcal{H}} \cup (V^{\mathcal{G}} \times V^{\mathcal{H}})\big).$$

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

Theorem 3.24. *If $\mathcal{G}$ and $\mathcal{H}$ are digraphs, then*

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

Proof. ($\leq$) Let $(\mathcal{D}_{\mathcal{G}}, (X_d)_{d \in V^{\mathcal{D}_{\mathcal{G}}}})$ and $(\mathcal{D}_{\mathcal{H}}, (Y_d)_{d \in V^{\mathcal{D}_{\mathcal{H}}}})$ be DAG-decompositions of $\mathcal{G}$ and $\mathcal{H}$ respectively. Let $\mathfrak{D} = (\mathcal{D}, (X_d)_{d \in V^{\mathcal{D}_{\mathcal{G}}}} \dot{\cup} (Y_d)_{d \in V^{\mathcal{D}_{\mathcal{H}}}})$, where $\mathcal{D}$ is the DAG obtained by putting an edge from every leaf of $\mathcal{D}_{\mathcal{G}}$ to every root of $\mathcal{D}_{\mathcal{H}}$. Then $\mathfrak{D}$ is DAG-decomposition for $\mathcal{G} \overrightarrow{\cup} \mathcal{H}$.

($\geq$) Conversely, if $\mathfrak{D}$ is a DAG-decomposition of $\mathcal{G} \overrightarrow{\cup} \mathcal{H}$, then by Lemma 3.13, $\mathfrak{D}|_{V^{\mathcal{G}}}$ is a DAG-decomposition of $\mathcal{G}$ and $\mathfrak{D}|_{V^{\mathcal{H}}}$ is a DAG-decomposition of $\mathcal{H}$, both of width less than or equal to the width of $\mathfrak{D}$. ☐

Note that this result extends to partial directed unions as well.

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 an optimal decomposition. The following is a direct consequence of the similar result for tree width.

Theorem 3.25. *Given a digraph $\mathcal{G}$ and a natural number k, deciding if the* DAG-*width of $\mathcal{G}$ is at most k is* NP-*complete.*

However, for any fixed k, it is possible, in polynomial time, to decide if a graph has DAG-width at most k and to compute a DAG-decomposition of this width if it has. We give an algorithm for this that is based on computing monotone winning strategies in the k-cops and robber game.

Theorem 3.26. *Let $\mathcal{G}$ 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 $\mathcal{G}$ and for computing such a strategy.*

Proof. Given a graph $\mathcal{G}$ 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, $\mathcal{H}(\mathcal{G}) = (V_0 \dot\cup V_1, E)$ 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 nodes reachable from r in $\mathcal{G} \setminus X_1$ contains the set of nodes reachable from r in $\mathcal{G} \setminus (X_1 \cap X_2)$.
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 $\mathcal{G} \setminus (X_1 \cap X_2)$.
Finally, there is an edge from v_0 to every node $(\varnothing, r) \in W_1$, where $r \in V$.

Note that $\mathcal{H}(\mathcal{G})$ 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 $\mathcal{H}(\mathcal{G})$ is polynomial in the size of the input, and we can compute the winner of the simpler game in polynomial time, the theorem follows.

□

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 3.27. *Given a graph $\mathcal{G}$ of DAG-width k, a DAG-decomposition of $\mathcal{G}$ of width k can be computed in time $\mathcal{O}(|\mathcal{G}|^{\mathcal{O}(k)})$.*

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 5.3), 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 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. So if the problem was solvable in polynomial time on graphs of bounded DAG-width then for every given undirected graph we could simple direct the edges so that it becomes acyclic, i.e. of DAG-width 1, and solve the problem then. This shows that 3-colourability is not solvable efficiently on graphs of bounded DAG-width unless PTIME $=$ NP. It also implies that Courcelle's theorem does fail for DAG-width, as 3-colourability is easily seen to be MSO-definable.

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 $\mathcal{P}$ is a tuple (V, V_0, E, Ω) where (V, E) is a directed graph, $V_0 \subseteq V$ and $\Omega : V \to \omega$ is a function assigning a priority to each node. There is no loss of generality in assuming that the range of Ω 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 $\mathcal{P}$ 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 π is winning for Even if $\liminf_{i\to\infty} \Omega(v_i)$ is even and π is winning for Odd otherwise.

A *strategy* is a map $f : V^{<\omega} \to V$ such that for any sequence $(v_0 \cdots v_i) \in V^{<\omega}$, $(v_i, f(v_0 \cdots v_i)) \in E$. A play $\pi = (v_i \mid i \in \omega)$ is consistent with Even playing f if whenever $v_i \in V_0$, $v_{i+1} = f(v_0 \cdots v_i)$. Similarly, π is consistent with Odd playing f if whenever $v_i \notin V_0$, $v_{i+1} = f(v_0 \cdots v_i)$. A strategy f is winning for Even if every play consistent with Even playing f is winning for Even. A strategy is *memoryless* if whenever $u_0 \cdots u_i$ and $v_0 \cdots v_j$ are two sequences in $V^{<\omega}$ with $u_i = v_j$, then $f(u_0 \cdots u_i) = f(v_0 \cdots v_j)$. 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 starting node, which player has a winning strategy. The complexity of this problem in general remains a major open question, as explained in Section 1. We demonstrate that parity games are solvable on arenas of bounded DAG-width by an algorithm similar in spirit to that of Obdržàlek [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 4.1, 4.2 and 4.3.

For a parity game $\mathcal{P} = (V, V_0, E, \Omega)$ 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 $\pi = (v_i : i \in \omega)$ that is consistent with Even playing f and Odd playing g. Let π' be the maximal initial segment of π that is contained in U. The *outcome* of the pair of strategies (f, g) (given U and v) is defined as follows.

$$\text{out}_{f,g}(U, v) := \begin{cases} \text{winEven} & \text{if } \pi' = \pi \text{ and } \pi \text{ is winning for Even;} \\ \text{winOdd} & \text{if } \pi' = \pi \text{ and } \pi \text{ is winning for Odd;} \\ (v_{i+1}, p) & \text{if } \pi' = v_0 \cdots v_i \text{ and } p = \min\{\Omega(v_j) \mid 0 \leq j \leq i+1\}. \end{cases}$$

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 π 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 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 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) winOdd $\trianglelefteq o$ for all outcomes o;
(ii) $o \trianglelefteq$ 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 $o \trianglelefteq o'$, 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 $\trianglelefteq$-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 (pot-out$(W), \trianglelefteq)$, where W is a set of guards for U. We write pot-res(W) for the set of *potential results* in W. To be precise, pot-res(W) is the set of all anti-chains in the partial order (pot-out$(W), \trianglelefteq)$. By definition of the order $\trianglelefteq$, if either of winEven or winOdd is in the set 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 $(w, p) \in$ result$_f(U, v)$ so the number of distinct values that result$_f(U, v)$ can take is at most $(|U| + 1)^{|W|} + 2$ (in fact, $(d + 1)^{|W|}$, where d is the number of different priorities in U). This is the cardinality of the set pot-res(W).

We also abuse notation and extend the order $\trianglelefteq$ to the set pot-res(W) pointwise. That is, for $r, s \in$ pot-res(W) we write $r \trianglelefteq s$ if, for each $o \in s$, there is an $o' \in r$ with $o' \trianglelefteq o$. With this definition, the order $\trianglelefteq$ on 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 $\trianglelefteq$ minimal elements in the set of outcomes $r \cup s$. One further piece of notation we use is that we write Res(U, v) for the set $\{$result$_f(U, v) : f$ is a strategy$\}$.

Suppose now that $\mathcal{P} = (V, V_0, E, \Omega)$ is a parity game and we are given a DAG decomposition $(\mathcal{D}, (X_d)_{d \in V^{\mathcal{D}}})$ of (V, E) of width k that is nice in the sense of Definition 3.14. For each $d \in V^{\mathcal{D}}$, we write V_d for the set $\mathcal{X}_d \setminus X_d$. The key to the algorithm is that we construct the set of results Res(V_d, v) for each $v \in V_d$. Since V_d is guarded by X_d, $|X_d| \leq k$ and $|V_d| \leq n$, the number of distinct values of 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 result$_f(U, v)$ and Frontier(d), f and g range over *all* strategies and not just memoryless ones. The bound on the number of possible values of result$_f(V_d, v)$ guarantees that $|\text{Frontier}(d)| \leq n((n + 1)^k + 2)$. We aim to show how 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 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 Frontier$(d) =$ Frontier$(e_1) \cup$ Frontier(e_2).

To see this, suppose first that $(v, r) \in$ Frontier(e_1) (the case of Frontier(e_2) is symmetrical) and in particular $r =$ result$_f(V_{e_1}, v)$. Now, if $o \in r$ there is a g such that $o =$ out$_{f,g}(V_{e_1}, v)$. If o is winEven or winOdd it is clear that $o =$ out$_{f,g}(U, v)$ for any $U \supset V_{e_1}$ and in particular $o =$ out$_{f,g}(V_d, v)$. If $o = (w, p)$ then the play π 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 $o =$ out$_{f,g}(V_d, v)$. We conclude that the set of available outcomes is the same and therefore the set of $\trianglelefteq$-minimal outcomes is the same. That is, $r =$ result$_f(V_d, v)$ and therefore $(v, r) \in$ Frontier(d).

In the other direction, suppose $(v, r) \in$ 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 $r =$ result$_f(V_d, v)$. Suppose $o =$ out$_{f,g}(V_d, v)$ for some strategy g and let π be the play starting at v determined by f and g. We claim that $o = \mathrm{out}_{f,g}(V_{e_1}, v)$. If this is not the case, then the first occurrence in π 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 $\mathrm{out}_{f,g}(V_d, v) = \mathrm{out}_{f,g}(V_{e_1}, v)$ and therefore $r = \mathrm{result}_f(V_{e_1}, v)$.

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

Case 2: d has one successor e and $X_d = X_e$. In this case, Frontier(d) = Frontier(e).

Case 3: d has one successor e and $X_d \setminus X_e = \{u\}$. Then, by (D3), $u \notin V_e$. 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 3.11(ii)), there is no path from any element of V_e to u except through X_e. Thus, if $(w, p) \in \mathrm{result}_f(V_d, v)$ for some v and f, it must be the case that $w \in X_e$. Hence, Frontier(d) = 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 Frontier(d) we must determine the results of all plays beginning at u.

Consider the set of vertices v in $\mathcal{X}_d$ such that $(u, v) \in E^{\mathcal{G}}$. These fall into two categories. Either $v \in X_d$ or $v \in V_e$. Let $x_1, \ldots, x_s$ enumerate the first category and let $v_1, \ldots, v_m$ enumerate the second. Let $O = \{(x_i, \min\{\Omega(x_i), \Omega(u)\}) : 1 \le i \le s\}$. 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 $\trianglelefteq$, $O \in$ 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 Frontier(e), the set Res(V_e, v_i). For each result $r \in$ Res(V_e, v_i), we write 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 $\trianglelefteq$-minimal elements from the resulting set. This is mod(r). Note that mod$(r) \in$ pot-res(X_d). The intuition is that mod$(\mathrm{result}_f(V_e, v_i))$ 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 $1 \le i \le m$, let $M_i = \{\mathrm{mod}(r) : r \in \mathrm{Res}(V_e, v_i)\}$.

We now wish to use the sets of results M_i, O and $\mathcal{O}$ to construct the Res(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 4.1. *If $u \in V_0$, then Res$(V_d, u) = \bigcup_i M_i \cup \mathcal{O}$.*

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

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

Lemma 4.2. *If $u \notin V_0$, then $r \in$ Res(V_d, u) if, and only if, there is a function c on the set $[m]$ with $c(i) \in M_i$ such that $r = O \sqcap \bigsqcap_{i\in[m]} c(i)$.*

Proof.

$\Rightarrow$ Let $r \in \text{Res}(V_d, u)$, 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 v_i, it is clear that $r \trianglelefteq c(i)$ for each i. Odd can also move to any of the x_i and therefore $r \trianglelefteq O$. 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 $O \sqcap \bigsqcap_{i\in[m]} c(i) \trianglelefteq r$.

$\Leftarrow$ 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 $\text{result}_f(V_d, u) = O \sqcap \bigsqcap_{[m]} c(i)$.

□

Lemma 4.2 suggests constructing Res(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 Res(V_d, u). Recall that Res$(V_d, u) \subseteq$ pot-res(X_d) and the latter set has at most $(n+1)^k+2$ elements. We check, for each $r \in$ pot-res(X_d), in polynomial time, whether there is a choice function c as in Lemma 4.2 that yields r. In particular, we take the following alternative characterisation of Res(V_d, u).

Lemma 4.3. *If $u \notin V_0$, then $r \in$ Res(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) *$r = O \sqcap \bigsqcap_{i\in D} d(i)$; and*
(ii) *for each $i \notin D$ there is an $r_i \in M_i$ with $r \trianglelefteq r_i$.*

Proof.

$\Rightarrow$ Assume $r \in \text{Res}(V_d, u)$ and let c be the choice function given by Lemma 4.2. 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

$$c(i) = \begin{cases} d(i) & i \in D \\ r_i & i \notin D \end{cases}$$

Now, since by hypothesis $r \trianglelefteq r_i$ and $r = O \sqcap \bigsqcap_{i\in D} d(i)$, it is easily seen that $r = O \sqcap \bigsqcap_{i\in[m]} c(i)$.

□

Now, any $r \in$ pot-res(X_d) has at most k elements. Thus, to check whether such an r is in Res(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 $r = O \sqcap \bigsqcap_D d(i)$, 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 Res(V_d, u). One barrier remains to completing the construction of Frontier(d). Elements (v, r) of 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 Res(V_d, u). To be precise, let $r \in$ Res(V_e, v) for some $v \in V_e$ and $s \in$ Res(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 $(w, \min\{p, q\})$. $c(r, s) = r \sqcap s'$.

Intuitively, if $r =$ result$_f(V_e, v)$ and $s =$ result$_{f'}(V_d, u)$ then $c(r, s)$ is the set of $\trianglelefteq$-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 4.4. *For any $v \in V_e$,*

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

Proof. It is clear that, for any strategy f, result$_f(V_d, v) = c($result$_f(V_e, v),$ result$_f(V_d, u))$. Thus, Res(V_d, v) is included in the set on the right hand side. For the converse, suppose first that $r =$ 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. Thus result$_f(V_e, v) =$ result$_f(V_d, v) = c(r, s)$ for all s. Now, let $r =$ result$_{f_1}(V_e, v)$ include an outcome (u, p) and $s =$ 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 result$_f(V_d, v) = c(r, s)$. □

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

Theorem 4.5. *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 with* DAG*-width at most k.*

Proof. By Proposition 3.27, 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 $(\mathcal{D}', (X_d)_{d \in V^{\mathcal{D}'}})$ in time at most quadratic (in the size of the decomposition). Let a be the root of $\mathcal{D}'$ and let $X_a = \{x_1, \ldots, x_l\}$ where $l \leq k$. Consider the dag $\mathcal{D}$ formed by adding l new elements $a_0, \ldots, a_{l-1}$ to $\mathcal{D}'$ in a simple directed path ending in a. Further, for each i define X_{a_i} to be the set $\{x_1, \ldots, x_i\}$. In particular, the new root a_0 is labelled by $\varnothing$. It is easily seen that the new labelled DAG $(\mathcal{D}, (X_d)_{d \in V^{\mathcal{D}}})$ still meets the definition of a nice decomposition. We then use the above construction to obtain Frontier(d) for each d in $\mathcal{D}$, starting from the leaves and working our way to the root. Since the size of $\mathcal{D}$ is at most $n^{2k} + k$, the total time taken is bounded by a polynomial. Now, for the root a_0 of $\mathcal{D}$ $\mathcal{X}_a = V_a = V$. Thus, if $(v, r) \in$ Frontier(a_0) then $r \subseteq \{$winEven, winOdd$\}$. If winEven $\in r$, this means that player Even has a strategy to win the parity game beginning at vertex v, and if 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. □

5 Relation to other graph connectivity measures

As a structural measure for undirected graphs, the concept of tree-width is of unrivaled 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. [8], directed path-width [1], and entanglement proposed by Berwanger and Grädel [2].

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 $\mathcal{G}$ is defined as the tree-width of the undirected graph obtained from $\mathcal{G}$ by forgetting the orientation of the edges.

Proposition 5.1.

(i) *If a directed graph $\mathcal{G}$ 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 $(\mathcal{T}, (W_t)_{t \in V^\mathcal{T}})$ is a tree decomposition of $\mathcal{G}$ of width k. Choose some $r \in V^\mathcal{T}$ and orient the edges of $\mathcal{T}$ away from r. That is if $\{s,t\} \in E^\mathcal{T}$ and s is on the unique path from r to t then change $\{s,t\}$ to (s,t). Since $\mathcal{T}$ is a tree every edge has a unique orientation in this manner. Let $\mathcal{D}$ be the resulting DAG. For all $d \in V^\mathcal{D}$, set $X_d := W_t$ where t is the node of $\mathcal{T}$ corresponding to d. We claim that $(\mathcal{D}, (X_d)_{d \in V^\mathcal{D}})$ is a DAG-decomposition of $\mathcal{G}$ of width $k{+}1$. (D1) and (D2) are trivial. (D3) follows from the connectivity condition of tree decompositions. The orientation ensures $\mathcal{D}$ has one root, r, so $\mathcal{X}_r = V^\mathcal{G}$ and (D4) follows. Finally we need to check (D5). This follows from a similar condition for tree decompositions. Let $(d,d') \in E^\mathcal{D}$ and suppose $v \in \mathcal{X}_{d'} \setminus X_d$. Suppose also that $(v,w) \in E^\mathcal{G}$ and $w \notin \mathcal{X}_{d'} \setminus X_d$. We will show that $w \in X_d \cap X_{d'}$. Since $v \notin X_d$ and $v \in \mathcal{X}_{d'}$, any d'' such that $v \in X_{d''}$ must satisfy $d' \preceq_\mathcal{D} d''$ by the connectivity condition of tree decompositions. As $(v,w) \in E^\mathcal{G}$, there exists $d'' \in V^\mathcal{D}$ such that $\{v,w\} \subseteq X_{d''}$. Thus $w \in \mathcal{X}_{d'}$. As $w \notin \mathcal{X}_{d'} \setminus X_d$, it follows that $w \in X_d$. By (D3), $w \in X_{d'}$ also, as $w \in \mathcal{X}_{d'}$. Thus $w \in X_d \cap X_{d'}$ and (D5) holds.

(ii). For any 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 complete and has tree-width $n-1$.

□

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

Proposition 5.2. *$\mathcal{G}$ has tree-width $k-1$ if, and only if $\overleftrightarrow{\mathcal{G}}$ has* DAG*-width k.*

Proof. It is easily seen that the k-cops and robber game for undirected graphs on $\mathcal{G}$ is equivalent to the k-cops and robber game for directed graphs on $\overleftrightarrow{\mathcal{G}}$. The result follows from the correspondence between the measures and existence of monotone winning strategies. □

Directed tree-width. Aiming to reproduce the success of tree-decompositions in allowing divide-and-conquer algorithms, directed tree-width is associated to 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 robber-and-cops 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 cop-monotone strategies are known to not be sufficient.

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, as illustrated in the following proposition.

Proposition 5.3.

(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 $\mathcal{G}$ has DAG-width k then k cops can win the k-cops and robber game on $\mathcal{G}$. Thus k cops can win the game in [8], and so $\mathcal{G}$ does not have a (directed) haven of size k. Therefore $\mathcal{G}$ has a directed tree decomposition of width $3k + 1$ [8].

(ii). Consider the family $\{(\mathcal{T}_k^1)^{op} : k \geq 2\}$ of graphs defined in Proposition 3.5. Note that $(\mathcal{T}_k^1)^{op}$ is a binary branching tree of height k with back-edges from every node to its ancestors. We have shown that $(\mathcal{T}_k^1)^{op}$ has game-width k, and it is clear that the strategy described for k cops is monotone, so $(\mathcal{T}_k^1)^{op}$ has DAG-width k. On the other hand, if we let $\mathcal{T}$ be the directed tree obtained from $(\mathcal{T}_k^1)^{op}$ by removing back-edges; define for all $t' \in V^{\mathcal{T}}$ $B_{t'} := \{t, s\}$ where t is the corresponding vertex in $(\mathcal{T}_k^1)^{op}$ and s is the predecessor of t; and $X_{(s',t')} = \{s\}$ for all $(s', t') \in E^{\mathcal{T}}$, it is easy to show that $(\mathcal{T}, (B'_t)_{t' \in V^{\mathcal{T}}}, (X_e)_{e \in E^{\mathcal{T}}})$ is a directed tree decomposition of $(\mathcal{T}_k^1)^{op}$ of width 1. For $k \geq 2$, $(\mathcal{T}_k^1)^{op}$ is not acyclic and therefore has directed tree-width exactly 1.

□

An 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) nicely generalises DAG-width without using games.

Proposition 5.4. *If a graph $\mathcal{G}$ has* DAG*-width k its extended directed tree-width is at most $k-1$.*

Proof. Let $(\mathcal{D}, (X_d)_{d\in\mathcal{D}})$ be a DAG-decomposition of $\mathcal{G}$ of width k. From Lemma 3.15 we can assume $\mathcal{D}$ has a unique root. Let $\mathcal{T}$ be a spanning tree of $\mathcal{D}$. Let $t_1, t_2, \ldots$, be the sequence of nodes of $\mathcal{T}$ visited in a depth-first traversal of $\mathcal{T}$, where a node is only added after all its children have been. Starting from $i = 1$, we define $B_{t_i} := X_{d_i} \setminus \bigcup_{j<i} B_{t_i}$, where d_i is the node of $\mathcal{D}$ corresponding to t_i. For $(s,t) \in E^{\mathcal{T}}$ we set $W_{(s,t)} := X_{s'} \cap X_{t'}$ where s', t' are the nodes of $\mathcal{D}$ corresponding to s and t respectively. It follows easily that $(\mathcal{T}, (B_t)_{t\in V^{\mathcal{T}}}, (W_e)_{e\in E^{\mathcal{T}}})$ is a pre-decomposition of $\mathcal{G}$ of width at most $k-1$. □

Directed path-width. Directed path-width was introduced by Thomas [16] as a generalisation of path-width to directed graphs. Formally, a directed path decomposition of a directed graph $\mathcal{G}$ is a sequence $W_1, W_2, \ldots, W_n$ such that

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

The width of $W_1, W_2, \ldots, W_n$ is $\max\{|W_i| : 1 \leq i \leq n\} - 1$, and the *directed path-width* of $\mathcal{G}$ 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 5.5.

(i) *If a graph $\mathcal{G}$ 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 $W_1, W_2, \ldots, W_n$ be a directed path decomposition of $\mathcal{G}$ of width k. Let $\mathcal{D}_n$ be the directed path with n vertices. That is $V^{\mathcal{D}_n} = \{d_1, \ldots, d_n\}$ and $(d_i, d_j) \in E^{\mathcal{D}_n}$ if and only if $j = i+1$. Set $X_{d_i} := W_i$ for all $d_i \in V^{\mathcal{D}_n}$. We claim $(\mathcal{D}_n, (X_d)_{d\in V^{\mathcal{D}_n}})$ is a DAG-decomposition of $\mathcal{G}$ of width $k+1$. (D1) and (D4) are obvious. (D2) follows from (P1) and (D3) follows from (P2). To show (D5), for $1 \leq i < n$ suppose $v \in \mathcal{X}_{d_{i+1}} \setminus X_{d_i}$ and $(v,w) \in E^{\mathcal{G}}$. 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 $w \in \mathcal{X}_{d_{i+1}}$. If $w \notin \mathcal{X}_{d_{i+1}} \setminus X_{d_i}$ 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 $\mathcal{X}_{d_{i+1}} \setminus X_{d_i}$.

(ii). Let $\mathcal{T}_k$ be the (undirected) binary tree of height $k \geq 2$. From Proposition 5.2, $\overleftrightarrow{\mathcal{T}_k}$ has DAG-width 2. It is known that $\mathcal{T}_k$ has path-width exactly $k-1$, and it is straightforward to show that $\overleftrightarrow{\mathcal{T}_k}$ must therefore have directed path-width exactly $k-1$. Thus the family $\{\mathcal{T}_k : k \geq 2\}$ satisfies the proposition.

□

In [1], 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.

Entanglement. The notion of entanglement measures the nesting depth of directed cycles in a graph. In terms of robber-and-cop 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 can only move to a successor of his current residence.

Unlike the other graph widths considered here, entanglement is not associated to an efficient tree-shaped graph representation. Nevertheless, it was shown that parity games on arenas of of bounded entanglement can be solved in polynomial time. 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.

The following proposition shows that having bounded DAG-width is more general than having bounded entanglement. On the other hand, the gap between DAG-width and entanglement can be at most logarithmic in the number of graph vertices.

Proposition 5.6.

(i) *If a graph has entanglement k, its* DAG*-width is at most $k+1$.*
(ii) *There are graphs with arbitrarily large entanglement but with* DAG*-width 2.*
(iii) *If a graph $\mathcal{G}$ has* DAG*-width k, its entanglement is at most $(k+1) \cdot \log |V^{\mathcal{G}}|$.*

Proof. (i). Let $\mathcal{G}$ be a graph of entanglement k. We trace a generic play of the DAG-width game with $k+1$ cops on $\mathcal{G}$ alongside with 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. Basically, the moves of the k cops will just be copied to the first game; on account of the robber's ability in the DAG-width game to move along several edges at a time, we employ an additional cop, called *chaser*, who follows the itinerary of the robber, but moving slowly, one edge per round. Therefore, each round in the entanglement game will be associated to two half-rounds in the DAG-width game, one for copying the cop's move and another for posting the chaser.

We keep a record of the following data at the beginning of a round: the set of positions occupied by cops in the entanglement game $X_i \subseteq [V]^{\leq k}$, the position r_i of the robber in this game which coincides with the position of the chaser in the DAG-width game, and the position s_i of the robber in the DAG-with game. During the play, r_i will always be on the trajectory of the robber in the DAG-width game. Let σ_i be the simple path from r_i to s_i following this trajectory but avoiding cycles. We will maintain the invariant that beyond its first position σ_i is cop-free.

At the beginning of a play, in the DAG-game, the cop player chooses the empty set $X_0 := \varnothing$ and the robber chooses the initial position r_0. In the second phase of this first round, the cop player then posts the chaser $X_0 := \{r_0\}$ while the robber may moves to some position s_0. Passing to the auxiliary entanglement game, we set the cops idle, and the robber to r_0. Thus, the plays are left in the configurations $(\{r_0\}, s_0)$ and respectively $(\varnothing, r_0)$.

Let (X_i, r_i) and $X_i \cup \{r_i\}, s_i$ be the starting configuration for round i, and let σ_i be the (shortcut) link along the trajectory of the robber from r_i to s_i. We assume that the segment σ_i avoids X_i from the second position onwards.

To decide on the next move of the cop player, we first look at the entanglement game. There his winning strategy may indicate either

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

Notice that in the DAG-game the position r_i is already guarded by the chaser. To transfer the cop's move from the entanglement game, we might hence skip the first phase in the first two cases listed above. In the latter case, when the cop needs to be removed from position z, we choose the set $(X_i \setminus z) \cup \{r_i\}$ for the cop player. Meanwhile, the robber moves to some position s that he can reach from s_i avoiding X_{i+1}. Hence, by our assumption, the (cycle-simplified) path from r_i via σ_i to s following the robber's trajectory will also avoid X_{i+1}. In particular, it implies that the successor of r_i along this path is cop-free. We choose this successor r_{i+1} as a new position for the the chaser. Hence, in the second phase of the round, the chaser is removed from r_i and sent to to r_{i+1}. During this cop-move, the robber will prolong his trajectory to some position s_{i+1}. However notice that this prolongation does either not go through r_i or it closes a cycle which will be discarded so that the new segment σ_{i+1} which links r_{i+1} to s_{i+1} still fulfills our invariant. Finally, we interpret the chaser's choice as a move of the robber in the entanglement game which thus assumes the new configuration (X_{i+1}, r_{i+1}).

Thanks to our invariant, the robber can thus freely move in the entanglement game as long as the robber in the DAG-width game was able to move. But since the cop player has a winning strategy for k cops this cannot go well forever. Hence, it must happen that eventually the robber cannot prolong his trajectory and loses. This shows that $k + 1$ cops have a winning strategy in the DAG-width game on $\mathcal{G}$, or equivalently that the DAG-width of $\mathcal{G}$ is at most $k + 1$.

(ii). Let $\mathcal{T}_k^{\downarrow}$ be the full binary tree of depth k with edges oriented downwards, and let $\mathcal{T}_k^{\uparrow}$ be the same tree with edges oriented upwards. Every node $v^{\downarrow} \in \mathcal{T}_k^{\downarrow}$ has a *double* $v^{\uparrow} \in \mathcal{T}_k^{\uparrow}$, and vice versa. The graph $G(2, k)$ is constructed by taking the union $\mathcal{T}_k^{\downarrow} \cup \mathcal{T}_k^{\uparrow}$, 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 $\mathcal{T}_k^{\uparrow}$. It is easy to see that $G(2, k)$ DAG-width 2.

We claim that ent$(G(2, k)) > k$. To prove this we describe a strategy by which the thief escapes against k detectives. We call a path in $G(2, k)$ *free* if all nodes on the path and all their doubles are unguarded by the detectives. We say that a node is *blocked* if both the node and its double are guarded. The thief moves according to the following strategy: *at a leaf $w^{\uparrow}$, she selects an ancestor $u^{\downarrow}$ of $w^{\downarrow}$ from which there is a free path to a leaf $v^{\downarrow}$. She goes to $v^{\downarrow}$ by moving upwards through $\mathcal{T}_k^{\uparrow}$, stepping over to $u^{\downarrow}$ and moving downwards through $\mathcal{T}_k^{\downarrow}$. Finally she steps over to $v^{\uparrow}$.*

With this strategy, the thief is never below a blocked node. A leaf has (including itself) $k + 1$ ancestors in $\mathcal{T}_k^{\downarrow}$, so there is always an ancestor with a free path to a leaf. Thus, the thief can maintain this strategy and escape forever.

(iii). This follows from the k-separator property for graphs of DAG-width k. In the entanglement game on such a graph, a winning strategy for the cops can be described as follows: choose a balanced separator S and place a cop whenever the robber passes through a vertex in S. Intuitively, we intend to cut the graph into two partitions using at most k cops, so that the robber is trapped in either one of them. The cut is produced when all vertices in S are occupied by cops. While waiting for this event, we apply the procedure recursively on the partition where the robber currently resides. Whenever he moves to the other partition, we may remove any cop from the original one. In case the robber again returns to the previous partition, on more vertex in the separator will be blocked, so that the robber is finally trapped in a partition. During a play, these partitions decrease up to size one, at which point the robber loses. □

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 result of this section.

$$\begin{aligned} \text{directed tree-width}(\mathcal{G}) \leq \text{DAG-width}(\mathcal{G}) &\leq \text{tree-width}(\mathcal{G}) \\ \text{DAG-width}(\mathcal{G}) &\leq \text{directed path-width}(\mathcal{G}) \\ \text{ent}(\mathcal{G})/\log|V^{\mathcal{G}}| \leq \text{DAG-width}(\mathcal{G}) &\leq \text{ent}(\mathcal{G}). \end{aligned}$$

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

References

1. J. BARÁT, *Directed path-width and monotonicity in digraph searching.* To appear in *Graphs and Combinatorics*.
2. D. BERWANGER AND E. GRÄDEL, *Entanglement – a measure for the complexity of directed graphs with applications to logic and games*, in LPAR, 2004, pp. 209–223.
3. H. L. BODLAENDER, *Treewidth: Algorithmic techniques and results*, in MFCS, 1997, pp. 19–36.
4. B. COURCELLE, *Graph rewriting: An algebraic and logic approach*, in Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), J. van Leeuwan, ed., 1990, pp. 193–242.
5. N. D. DENDRIS, L. M. KIROUSIS, AND D. M. THILIKOS, *Fugitive-search games on graphs and related parameters*, TCS, 172 (1997), pp. 233–254.
6. E. EMERSON, C. JUTLA, AND A. SISTLA, *On model checking for the μ-calculus and its fragments*, TCS, 258 (2001), pp. 491–522.
7. G. GOTTLOB, N. LEONE, AND F. SCARCELLO, *Robbers, marshals, and guards: Game theoretic and logical characterizations of hypertree width*, in PODS, 2001, pp. 195–201.
8. T. JOHNSON, N. ROBERTSON, P. D. SEYMOUR, AND R. THOMAS, *Directed tree-width*, Journal of Combinatorial Theory, Series B, 82 (2001), pp. 138–154.
9. M. JURDZIŃSKI, *Deciding the winner in parity games is in UP $\cap$ co-UP*, Information Processing Letters, 68 (1998), pp. 119–124.
10. D. KOZEN, *Results on the propositional mu-calculus*, TCS, 27 (1983), pp. 333–354.

11. J. Obdržálek, *Fast mu-calculus model checking when tree-width is bounded*, in Proceedings of 15th International Conference on Computer Aided Verification, vol. 2725 of LNCS, Springer, 2003, pp. 80–92.
12. 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.
13. N. Robertson and P. Seymour, *Graph Minors. III. Planar tree-width*, Journal of Combinatorial Theory, Series B, 36 (1984), pp. 49–63.
14. M. Safari, *D-width: A more natural measure for directed tree width*, in MFCS 2005, vol. 3618 of LNCS, Springer, 2005, pp. 745–756.
15. P. Seymour and R. Thomas, *Graph searching, and a min-max theorem for tree-width*, Journal of Combinatorial Theory, Series B, 58 (1993), pp. 22–33.
16. R. Thomas, *Directed tree-width*. Slides from lecture at the Regional NSF-CBMS Conference, 2002. Available at http://www.math.gatech.edu/ thomas/SLIDE/CBMS/dirtrsl.pdf.