The DAG-Width of Directed Graphs

Dietmar Berwanger [a], Anuj Dawar [b], Paul Hunter [c],
Stephan Kreutzer [d], Jan Obdržálek [e]

[a] *LSV, CNRS & ENS de Cachan, France*

[b] *University of Cambridge Computer Laboratory, UK*

[c] *Department of Computer Science, University of Oxford, UK*

[d] *Department of Electrical Engineering and Computer Science, Technical University Berlin, Germany*

[e] *Faculty of Informatics, Masaryk University, Brno, Czech Republic*

Abstract

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

Email addresses: dwb@lsv.ens-cachan.fr (Dietmar Berwanger), anuj.dawar@cl.cam.ac.uk (Anuj Dawar), paul.hunter@cs.ox.ac.uk (Paul Hunter), stephan.kreutzer@tu-berlin.de (Stephan Kreutzer), obdrzalek@fi.muni.cz (Jan Obdržálek).

sures of graph connectivity such as tree-width [24]. Apart from their interest in graph-structure theory, these notions have also proved very useful in algorithm design. 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 small 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 [8] shows that any property definable in monadic second-order logic is solvable in linear time on graphs of bounded tree-width.

The idea of designing algorithms that work on tree decompositions has been generalised from graphs to other 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 [23,16,5,25,15]) which extend notions of tree decompositions and tree-width to directed graphs. In particular, Johnson et al. [16] 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 new algorithms.

We are especially interested in one particular problem on directed graphs, that of determining the winner of a *parity game*. This is an infinite two-player game played on a directed graph where the vertices are labelled by priorities. The players take turns pushing a token along edges of the graph. The winner is determined by the parity of the least priority occurring infinitely often in this infinite play. Parity games have proved useful in the development of model-checking algorithms used in the verification of concurrent systems. The modal μ-calculus, introduced in [18], 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 [13]). The exact complexity of solving parity games is an open problem that has received a large amount of attention. It is known [11] that the problem is in NP $\cap$ co-NP but no polynomial time algorithm is known. From the general result of Courcelle [8], it follows that there is a linear time algorithm that solves parity games with a fixed number of priorities on graphs of bounded

tree-width. Obdržálek [21] exhibited a polynomial time algorithm for games with an arbitrary number of priorities on graphs of bounded tree-width. He points out that the algorithm would not give good bounds, for instance, on directed acyclic graphs even though solving the games on such graphs is easy. He asks whether there is a structural property of directed graphs that would allow a fast algorithm on both bounded tree-width structures and on DAGs. In this article, we give just such a generalisation.

We introduce a measure of the connectivity of graphs that we call DAG-width. It is intermediate between tree-width and directed tree-width, in that for any graph G, the directed tree-width of G is no greater than its DAG-width which, in turn, is no greater than its tree-width. Thus, the class of structures of DAG-width $k+1$ or less includes all structures of tree-width k and more (in particular, DAGs of arbitrarily high tree-width all have DAG-width 1). This measure was introduced independently in two conference papers [22,4] to which the present paper is a follow-up.

The notion of DAG-width can be understood as a simple adaptation of the *cops and robber game* (which characterises tree-width) to directed graphs. The game is played by two players, one of whom controls a set of k cops attempting to catch a robber controlled by the other player. The cop player can move any set of cops to any vertices on the graph, while the robber can move along any path in the graph as long as there is no cop currently on the path. Such games have been extensively studied (see [26,9,14,3,5]). It is known [26] that the cop player has a winning strategy on an undirected graph G using $k+1$ cops if, and only if, G has tree-width k. We consider the natural adaptation of this game to directed graphs, by constraining the robber to move along directed paths. We show that the class of directed graphs where there is a monotone (in a sense we make precise in Section 3.2) strategy for k cops to win is characterised by its width in a decomposition that is a generalisation of tree decompositions. We are then able to show that the problem of determining the winner of a parity game is solvable in polynomial time on the class of graphs of DAG-width k, for any fixed k.

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

2 Preliminaries

We first fix some notation used throughout the article. All graphs used are finite and simple (i.e., no self-loops and no multiple edges) unless otherwise stated. Also we

will use the term "digraph" when refering to directed graphs.

We write ω for the set of finite ordinals, i.e., natural numbers (including 0). For every $n \in \omega$, we write $[n]$ for the set $\{1, \ldots, n\}$. For a set V and a number $k \in \omega$, we write $[V]^{\leq k}$ for the set of all $X \subseteq V$ with $|X| \leq k$. Given sets A, B, we denote their symmetric difference $(A \setminus B) \cup (B \setminus A)$ by $A \triangle B$.

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

The following definition is standard (see [10]).

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

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

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

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

3 Games, Strategies and Decompositions

This section contains the graph-theoretical part of this article. We define DAG-width and its relation to graph searching games. As mentioned in the introduction, the notion of tree-width has a natural characterisation in terms of a cops and robber game. Directed tree-width has also been characterised in terms of such games [16], but these games appear to be less intuitive. In this article, we consider the straightforward extension of the cops and robber game from undirected graphs to digraphs. We show that these games give a characterisation of the graph connectivity measure

that we call DAG-width and introduce in Section 3.1. We comment on algorithmic properties in Section 3.5.

3.1 DAG-*decompositions and* DAG-*width*

In this section, we present a decomposition of digraphs that is somewhat similar in style to tree decompositions of undirected graphs. This leads to the definition of DAG-width, which can be seen as a measure of how close a given digraph is to being acyclic. We also present some properties enjoyed by DAG-width.

The concept of DAG-width we introduce is based on the following concept of *guarding*.

Definition 2. Let $G := (V, E)$ be a digraph and $W, V' \subseteq V$. Then, W *guards* V' if, for all $(u, v) \in E$, if $u \in V'$ then $v \in V' \cup W$.

The next lemma lists some simple properties of guarding used throughout the paper.

Lemma 3. *Let $G = (V, E)$ be a digraph and $X, X_0, X_1, Y, Y_0, Y_1, Z \subset V$. Then the following holds:*

(1) *If X_0 guards Y_0 and X_1 guards Y_1, then $X_0 \cup X_1$ guards $Y_0 \cup Y_1$.*
(2) *If X guards Y and $Z \supseteq X$, then Z guards Y.*
(3) *If X guards Y then $X \cup Z$ guards $Y \setminus Z$.*

Proof. (1) Suppose $(v, w) \in E(G)$, $v \in Y_0 \cup Y_1$ and $w \notin Y_0 \cup Y_1$. Let $v \in Y_i$, then $w \in X_i$, as X_i guards Y_i. Hence $w \in X_0 \cup X_1$, and $X_0 \cup X_1$ guards $Y_0 \cup Y_1$.
(2) Suppose $(v, w) \in E(G)$, $v \in Y$ and $w \notin Y$. As X guards Y, $w \in X$. As $Z \supseteq X$, $w \in Z$. Therefore, Z guards Y.
(3) Suppose $(v, w) \in E(G)$, $v \in Y \setminus Z$ and $w \notin Y \setminus Z$. Thus, $w \notin Y$ or $w \in Z$. For the first case, $w \in X$ as X guards Y. Hence $w \in X \cup Z$. □

We are now ready to define the concepts of DAG-width.

Definition 4 (DAG-decomposition)**.** Let $G := (V, E)$ be a digraph. A DAG-*decomposition* of G is a pair $(D, \mathcal{X})$ where D is a DAG and $\mathcal{X} = (X_d)_{d \in V(D)}$ is a family of subsets of V such that

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

The width of a DAG-decomposition $(D, \mathcal{X})$ is defined as $\max\{|X_d| : d \in V(D)\}$. The DAG-*width* of a digraph is defined as the minimal width of any of its DAG-decompositions.

Next, we show that the class of digraphs of DAG-width at most k is closed under directed unions, which is considered (see [16]) to be an important property of a reasonable decomposition of digraphs.

Definition 5. Let G and H be vertex-disjoint digraphs. A *partial directed union* of G and H is a digraph $(V(G) \cup V(H), E(G) \cup E(H) \cup E)$ where $E \subseteq V(G) \times V(H)$.

Theorem 6. *If G and G' are vertex-disjoint digraphs and H is a partial directed union of G and G', then*

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

Proof. For DAG-decompositions $(D^G, \mathcal{X}^G)$ and $(D^{G'}, \mathcal{X}^{G'})$ of G and G' respectively, the DAG D obtained by putting an edge from every sink of D^G to every source of $D^{G'}$ together with $\mathcal{X} := (X^G_d)_{d \in V(D^G)} \dot\cup (X^{G'}_d)_{d \in V(D^{G'})}$ forms a DAG-decomposition of H. Conversely, any DAG-decomposition $(D, \mathcal{X})$ of H can be restricted to G and G' yielding DAG-decompositions for these digraphs, according to Lemma 8. □

Finally we present two useful observations about DAG-decompositions. The first one tells us more about guarding subgraphs in the decomposition, and the second explains how to obtain, for a digraph G and a set $W \subseteq V(G)$, a DAG-decomposition of $G[W]$ from a DAG-decomposition of G.

Lemma 7. *Let $(D, \mathcal{X})$ be a* DAG*-decomposition of a digraph. For all $(d, d') \in E(D)$,*

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

Proof. It is clear that $X_{\succeq d'} \setminus X_d \subseteq X_{\succeq d'} \setminus (X_d \cap X_{d'})$. Conversely, let $x \in X_{\succeq d'} \cap X_d$. Then $x \in X_{d''}$ for some d'' s.t. $d' \preceq_D d''$. Since $d \preceq_D d' \preceq_D d''$, (D2) implies $x \in X_{d'}$ and therefore $x \in (X_d \cap X_{d'})$. □

Lemma 8. *Let $(D, \mathcal{X})$ be a* DAG*-decomposition of a digraph G. For $W \subseteq V(G)$, $(D, \mathcal{X}|_W)$ with $\mathcal{X}|_W := (X_d \cap W)_{d \in V(D)}$ is a* DAG*-decomposition of $G[W]$.*

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

3.2 Cops and robber games

The cops and robber game on a digraph is a game where k cops try to catch a robber. 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 9 (Cops and robber game)**.** Given a digraph $G := (V, E)$, the *k-cops and robber* game on G is played between two players, the *cop* and the *robber* player. Positions of this game are pairs (X, r), where $X \in [V]^{\leq k}$ are the vertices occupied by the cops and $r \in V$ is the vertex occupied by the robber. The game is played as follows:

- At the beginning, the cop player chooses $X_0 \in [V]^{\leq k}$, and the robber player chooses a vertex $r_0 \in V$, giving position (X_0, r_0).
- From position (X_i, r_i), if $r_i \notin X_i$ then the cop player chooses $X_{i+1} \in [V]^{\leq k}$, and the robber player chooses a vertex r_{i+1} such that there is a directed path from r_i to r_{i+1} in the digraph $G \setminus (X_i \cap X_{i+1})$.
- A *play* in the game is a maximal (finite or infinite) sequence $\pi := (X_0, r_0), (X_1, r_1), \ldots$ of positions given by the rules above.
- A play π is *winning for the cop player* if, and only if, it is finite. (Note that, by the rules above, this implies that $r_m \in X_m$ for the last position (X_m, r_m) of this play.) A play π is *winning for the robber player* if, and only if, it is infinite.
- 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.
- The *cop number* of a digraph G is the least k such that the cop player has a strategy to win the k-cops and robber game on G.

Variants of the game where the robber moves first, or only one cop can be moved at a time, or the cops are lifted and placed in separate moves are all easily seen to be equivalent in that the cop number of a digraph does not depend on the variant.

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

Lemma 10. *The cop number of any non-empty, digraph G, is at least 1 and it is exactly 1 if, and only if, G is acyclic.*

Proof. Against zero cops, a strategy by the robber player where the robber remains stationary is winning. If G is acyclic, then one cop can catch the robber by always playing to the robber's current position, chasing the robber towards a sink. Eventually, the robber will not be able to move and the cop will capture him. Conversely, if G has a cycle, then the robber can win against one cop by forever staying in the

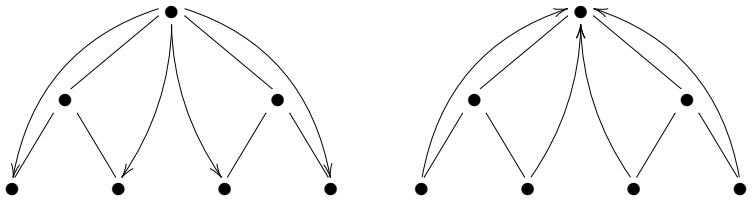

Fig. 1. The digraphs T_3^2 (cop number 2) and $(T_3^2)^{op}$ (cop number 3)

cycle. □

Games similar to the one defined above have been used to give game characterisations of measures such as tree-width [26] and directed tree-width [16]. In Section 5 we investigate this in more detail. One property of these measures is that they are invariant under edge reversal, that is, they do not change if the directions of the edges of the digraph are reversed. As we see below, this is not the case for the game we have defined. One exception is digraphs of cop number 1, that is, acyclic graphs.

Proposition 11. *The cop number of a digraph G is 1 if, and only if, the cop number of G^{op} is 1.*

Proof. This follows from Lemma 10 by observing that G is acyclic if, and only if, G^{op} is acyclic. □

Proposition 12. *For any j, k with $2 \leq j \leq k$, there exists a digraph T with cop number j such that the cop number of T^{op} is k.*

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

To show that the cop number of T_k^j is j, we describe a winning strategy for j cops and a strategy for the robber to defeat $j - 1$ cops. For simplicity in strategy descriptions, we refer to features such as the "root", "subtrees", "ancestors" and "descendents" of the underlying binary branching tree (with edges oriented away from the root). First we see that j cops have a winning strategy by initially playing on the root then following the robber down whichever subtree he plays in. This is achieved by moving (if there are j cops already in play) the cop on the most distant ancestor of the robber's current position to the root of the subtree he has moved to. To defeat $j - 1$ cops, the robber chooses any leaf. Whenever a cop moves to that leaf, a simple counting argument shows that there must be at least one unoccupied ancestor with at least one clear path to a leaf below. The robber then plays to that ancestor and along that path to the leaf.

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

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

Recall that a *strongly connected component* in a digraph is a maximal set of vertices S such that for any $u, v \in S$ there is a directed path from u to v.

Proposition 13. *A robber can defeat k cops on a digraph G if, and only if, there exists a function σ mapping each $X \in [V(G)]^{\leq k}$ to a non-empty union of strongly connected components of $G \setminus X$ such that if $X \subseteq Y \in [V(G)]^{\leq k}$ then:*

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

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

The function σ above plays the same role as havens for tree-width or directed tree-width (see [26,16]). The main difference is that $\sigma(X)$ need not be connected. Indeed, as Figure 2 shows, there are examples where $\sigma(X)$ is not connected. A robber can defeat 2 cops on this digraph, yet for any appropriate function σ, $\sigma(\{b, c\})$ must be precisely $\{a, d\}$.

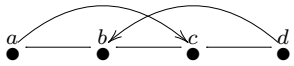

Fig. 2. Digraph showing havens may not be connected

Winning strategies for the cop player, introduced in Definition 9 play a central role throughout this article. Here we present two definitions for monotonicity and show that they are, for our purposes, equivalent.

For a digraph $G := (V, E)$ and a set of vertices $X \subseteq V$ and $r \in V$ we write $\mathit{Reach}_{G \setminus X}(r)$ to denote the set of vertices v such that there is a directed path from r to v in G that does not visit any vertex in X.

Definition 14. Let $G := (V, E)$ be a digraph.

(i) 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 $v \in X_i \setminus X_{i+1}$, we have $v \notin X_j$ for all $j > i$.

(ii) 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. That is, if $(X_0, r_0), (X_1, r_1), \ldots$ is a play consistent with the strategy, then $\mathit{Reach}_{G \setminus X_{i+1}}(r_{i+1}) \subseteq \mathit{Reach}_{G \setminus X_i}(r_i)$ for all i.

Lemma 15. (1) *If the cop player has a robber-monotone strategy then he also has a cop-monotone strategy.*

(2) *Any cop-monotone strategy is also robber-monotone.*

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

Proof. (1) 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 $X_i \setminus X_{i+1} \neq \varnothing$ (otherwise we are just placing extra cops, and therefore the set of vertices reachable by the robber cannot increase) and let $v \in X_i \setminus X_{i+1}$. As $v \in X_i$, the robber is unable to reach v when the cops are on X_i. As the strategy is robber-monotone, the robber is unable to reach v at any further stage, in particular, he cannot reach v when the cops are on X_{i+1} (i.e., not on v). Thus, no cop needs to revisit v in order to prevent the robber from reaching v. Thus, we can remove v from all X_j, $j > i$. Proceeding in this way results in a sequence $(X_0, r_0), (X'_1, r_1), \ldots$. 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 he also has a cop-monotone winning strategy.

(2) 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 $(X_0, r_0), (X_1, r_1), \ldots, (X_n, r_n)$ 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 $j > i$, the robber is unable to reach v otherwise he could play to v and sit there indefinitely, contradicting the assumption that cop player is playing a winning strategy. Thus, the robber is unable to reach any of the vertices in $X_i \setminus X_{i+1}$ and is therefore unable to reach any new vertices. Hence $R_i \supseteq R_{i+1}$, so the strategy is robber-monotone. □

It follows that whenever the cop player has either a cop-monotone or a robber-monotone strategy, he has one that is both cop- and robber-monotone. With this lemma in mind we define a *monotone winning strategy* in the obvious way.

3.3 Games and DAG-width equivalence

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

Theorem 16. *For any digraph G, there is a* DAG*-decomposition of G of width at most k if, and only if, the cop player has a monotone winning strategy in the k-cops and robber game on G.*

Proof. For the "if"-direction, let $G = (V, E)$ be a digraph and suppose the cop player has a monotone winning strategy $f : [V]^{\leq k} \times V \to [V]^{\leq k}$ in the k-cops and robber game on G. Without loss of generality, we assume that the first move defined by f is to place no cops. Furthermore, as f is monotone, we assume that cops are only ever placed on vertices that are reachable by the robber. That is,

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

To define a DAG-decomposition of width at most k, we will first present the strategy in a slightly different form, called the *strategy DAG*. Let D' be a digraph with vertex set $[V]^{\leq k} \times V$ and an edge from (X, r) to (X', r') if $X' = f(X, r)$ and $r' \in \mathit{Reach}_{G \setminus (X \cap X')}(r)$. That is, nodes (X, r) in D' correspond to game positions with the cops being on X and the robber being on r and an edge from (X, r) to (X', r') corresponds to the round in the game where the cops follow their strategy f to $X' = f(X, r)$ and the robber chooses r' as next position. In particular, paths in D' correspond to plays where the cops follow their winning-strategy f. Recall that we assume that the cops' first move is to place no cops, i.e., every game on G consistent with f starts in a position $(\varnothing, r)$. We therefore define the sub-digraph $D = (V(D), E(D))$ of D' induced by the set of nodes (X, r) in D' reachable from a node $(\varnothing, r)$, with $r \in V$. We call D the *strategy DAG* for the strategy f on G.

By construction, for very node (X, r) in D the robber has a strategy in the game on G against the cops playing f that guarantees that the position (X, r) will be attained.

Claim. D is acyclic.

Proof. Suppose there was a cycle $C := (v_1, e_1, \ldots, v_n, e_n)$ in D with $e_i := (v_i, v_{i+1})$, for all $i < n$, and $e_n := (v_n, v_1)$. Let $v_i := (X_i, r_i)$, for all $1 \leq i \leq n$. As explained above, the robber has a strategy against f to force the game into the

position $v_1 = (X_1, r_1)$. By construction of D, the cops' response from this position is to play to $X_2 = f(X_1, r_1)$ and the robber can then reply by moving to $r_2 \in \mathit{Reach}_{G\setminus(X\cap X')}(r)$. More generally, whenever the game is at position (X_i, r_i) the cops, following f, will move to X_{i+1} (or X_1 if $i = n$) and the robber can move to r_{i+1} (or r_1 resp.). Hence, the robber can evade capture forever and wins the play, contradicting the assumption that f is a winning strategy. $\dashv$

Let $\mathcal{X} := (X_d)_{d\in V(D)}$ with $X_d := f(X, r)$, where $d = (X, r) \in V(D)$. We claim that $(D, \mathcal{X})$ is a DAG-decomposition of G of width at most k. It is clear from the construction that the width of $(D, \mathcal{X})$ is at most k, as $\max\{|X_d| : d \in V(D)\} = \max\{|f(d)| : d \in V(D)\} \leq k$. Hence, it remains to show the properties (D1) - (D3) in Definition 4.

Towards (D1), suppose there was a vertex $v \in V \setminus \bigcup_{d\in V(D)} X_d$. But then the robber wins the game on G against the strategy f by initially going to the vertex v. As $v \notin \bigcup_{d\in V(D)} = \bigcup_{d=(X,r)\in V(D)} f(X, r)$, the cops will never place a cop on v and hence will fail to capture the robber, contradicting the fact that f is a winning strategy.

Towards (D2), let $d, d', d'' \in V(D)$ with $d \preceq_D d' \preceq_D d''$ and let P be a path from d to d'' containing d'. Recall that such a path corresponds to a play from d where the cops follow f. Hence, if $X_d \cap X_{d''} \not\subseteq X_{d'}$ this means that there is a vertex $v \in X_d$ occupied by a cop at position d but which is released by the cops between d and d' in the play following the path P and later reoccupied between d' and d''. But this would contradict the assumption that f is a cop-monotone strategy.

It remains to show (D3), i.e. we have to show that $X_d \cap X_{d'}$ guards $X_{\succeq d'} \setminus X_d$, for all edges $(d, d') \in E(D)$, where $X_{\succeq d'}$ stands for $\bigcup_{d'\preceq_D d''} X_{d''}$. Furthermore, for any source d, $X_{\succeq d}$ must be guarded by $\varnothing$.

Towards this aim, we first show the following claim.

Claim. For all $d = (X, r) \in V(D)$

$$\Big(\bigcup_{d\preceq_D d'} f(d')\Big) \setminus X \quad = \quad \mathit{Reach}_{G\setminus(X\cap f(X,r))}(r). \tag{2}$$

Proof. Note first that $\mathit{Reach}_{G\setminus(X\cap f(X,r))}(r) = \mathit{Reach}_{G\setminus X}(r)$. That $\mathit{Reach}_{G\setminus X}(r) \subseteq \mathit{Reach}_{G\setminus(X\cap f(X,r))}(r)$ is clear (see Lemma 3). Conversely, if there was a vertex $v \in \mathit{Reach}_{G\setminus(X\cap f(X,r))}(r) \setminus \mathit{Reach}_{G\setminus X}(r)$, that would imply that at position (X, r) in the game the vertex v was not reachable for the robber but when the cops make their move to $f(X, r)$, then the robber can reach v. But this would contradict robber-monotonicity of f. Hence, $\mathit{Reach}_{G\setminus(X\cap f(X,r))}(r) = \mathit{Reach}_{G\setminus X}(r)$.

Now, as f is robber-monotone, whenever $d := (X, r) \preceq_D d' := (X', r')$ then $\mathit{Reach}_{G\setminus X'}(r') \subseteq \mathit{Reach}_{G\setminus X}(r)$. Furthermore, by (1), $f(d') \subseteq X' \cup \mathit{Reach}_{G\setminus X'}(r')$.

A simple induction on the distance between d and d' in D then shows that $X' \subseteq X \cup \mathit{Reach}_{G\setminus X}(r)$ and therefore $f(d') \subseteq X \cup \mathit{Reach}_{G\setminus X}(r)$. Hence, $\Big(\bigcup_{d \preceq_D d'} f(d')\Big) \setminus X \subseteq \mathit{Reach}_{G\setminus X}(r)$.

Conversely, the same argument as in (D1) shows that $\mathit{Reach}_{G\setminus X}(r) \subseteq \Big(\bigcup_{d \preceq_D d'} f(d')\Big) \setminus X$ as otherwise the robber would win from (X, r).

Taken together, we get

$$\Big(\bigcup_{d \preceq_D d'} f(d')\Big) \setminus X = \mathit{Reach}_{G\setminus X}(r) = \mathit{Reach}_{G\setminus(X \cap f(X,r))}(r). \qquad \dashv$$

To establish (D3), let $d = (X, r)$ be a source of D. By assumption on f, $X := \varnothing$ and therefore, by (2), $X_{\succeq d} = \Big(\bigcup_{d \preceq_D d'} f(d')\Big) = \mathit{Reach}_G(r)$ is guarded by $\varnothing$.

On the other hand, if $(d, d') \in E(D)$ is any edge in D, where $d = (X, r)$ and $d' = (X', r')$ and $X' = f(X, r)$, then by (2),

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

Therefore, $X_d \cap X_{d'} = X' \cap f(X', r')$ guards $X_{\succeq d'} \setminus X_d$. This concludes the proof of (D3) and therefore we have shown that $(D, \mathcal{X})$ is a DAG-decomposition of G of width at most k.

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

(1) Let the robber choose a vertex $v \in V$. From (D1), there exists $d_v \in V(D)$ such that $v \in X_{d_v}$. Let d be a source of D which lies above d_v.
(2) Place cops on X_d.
(3) From (D3) and Lemma 3(2), X_d guards $X_{\succeq d} \setminus X_d$. Therefore, the robber can only move to vertices in $X_{\succeq d} \setminus X_d$. Suppose the robber moves to $v' \in X_{d''}$. Let d' be a sucessor of d which lies above d''.
(4) Remove cops on $X_d \setminus X_{d'}$ (leaving cops on $X_d \cap X_{d'}$)
(5) As $X_d \cap X_{d'}$ guards $X_{\succeq d'} \setminus X_d$, the robber can only move to vertices in $X_{\succeq d'}$ – that is, the robber must remain in the sub-DAG whose source is d'.
(6) Return to step (2) with d' as d.

As D is a DAG, at some point the robber player will not be able to move (since $X_{\succeq d} \setminus X_d$ is empty when d is a sink). Hence, this is a winning strategy for k cops. To show that it is monotone, observe that (D2) ensures that at no point does a cop return to a vacated vertex. This concludes the proof of Theorem 16. □

The above game characterization allows us to easily show some interesting prop-

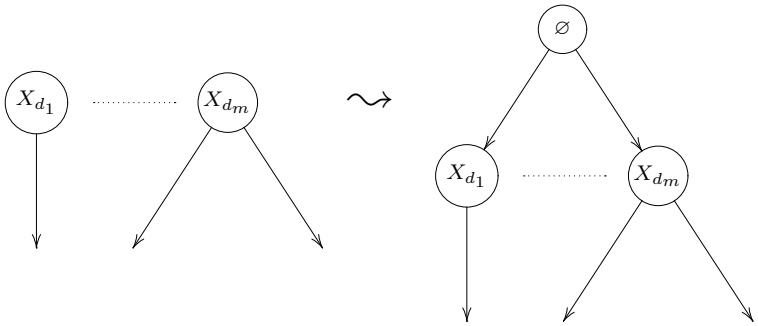

Fig. 3. Forming a unique source

erties of DAG-width. For instance, observe that the winning strategies for the cop player in Lemma 10 and Proposition 12 are monotone. Then we get the following:

Corollary 17. *Let G be a digraph. Then G has* DAG*-width 1 if, and only if, it is acyclic (indeed, the digraph itself will suffice as a decomposition). Moreover, for any $j, k \in \omega$ with $2 \leq j \leq k$, there exists a digraph G of* DAG*-width j such that G^{op} is of* DAG*-width k.*

3.4 Nice DAG-decompositions

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 [6].

Definition 18. A DAG-decomposition $(D, (X_d)_{d \in V(D)})$ of a digraph G is *nice* if

(N1) D has a unique source.
(N2) Every $d \in V(D)$ has at most two successors.
(N3) For $d_0, d_1, d_2 \in V(D)$, if d_1, d_2 are two successors of d_0, then $X_{d_0} = X_{d_1} = X_{d_2}$.
(N4) For $d_0, d_1 \in V(D)$, if d_1 is the unique successor of d_0, then $|X_{d_0} \triangle X_{d_1}| = 1$.

We show next that every digraph 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 19 (Unique source)**.** *Let $(D, \mathcal{X})$ be a* DAG*-decomposition of width k of a digraph G, and let $d_1, d_2, \ldots, d_m$ be the sources of D. Then, the pair $(D', \mathcal{X}')$ with*

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

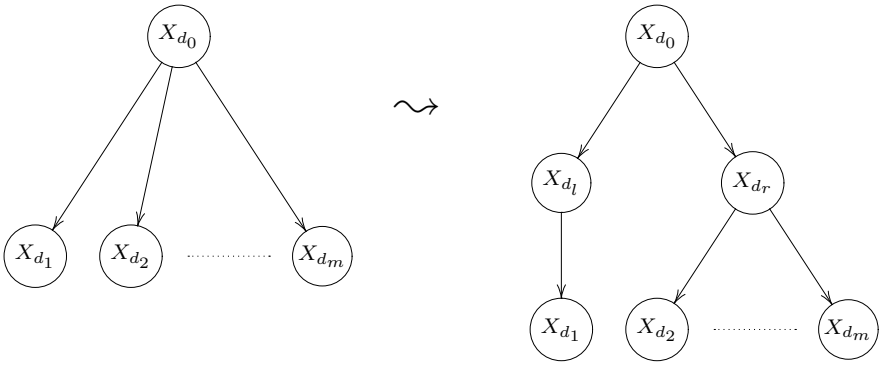

Fig. 4. Splitting at d_0

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

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

Figure 3 gives a visual representation of the construction. □

Definition 20 (Splitting)**.** Let $(D, \mathcal{X})$ be a DAG-decomposition, and suppose $d_0 \in V(D)$ has $m > 1$ successors $d_1, d_2, \ldots, d_m$. The decomposition $(D', \mathcal{X}')$ obtained from $(D, \mathcal{X})$ by *splitting* d_0 is defined as follows:

(i) $V(D') = V(D) \dot\cup \{d_l, d_r\}$,

(ii) $E(D') = \Big(E(D) \setminus \{(d_0, d_i) : 1 \leq i \leq m\}\Big)$
$\cup \{(d_0, d_l), (d_0, d_r), (d_l, d_1)\}$
$\cup \{(d_r, d_i) : 2 \leq i \leq m\}$, and

(iii) $X'_d = X_d$, for all $d \in V(D)$, and $X'_{d_l} = X'_{d_r} = X_{d_0}$.

Figure 4 gives a visual representation of this transformation.

Lemma 21. *Let* $(D, \mathcal{X})$ *be a* DAG*-decomposition of a digraph* G *of width* k*, and suppose* $d_0 \in V(D)$ *has* $m > 1$ *successors* $d_1, d_2, \ldots, d_m$*. Then,* $(D', \mathcal{X}')$ *obtained from* $(D, \mathcal{X})$ *by splitting* d_0 *is a* DAG*-decomposition of* G *of width* k*.*

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

for all $d \in V(D)$,

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

since if X_{d_l} or X_{d_r} is included in the union on the left, then so is X_{d_0}, and so neither X_{d_l} nor X_{d_r} contribute to the overall union.

It is easily seen that the edges added do not create any cycles, so D' is a DAG. Further, $\bigcup_{d \in V(D')} X'_d = \bigcup_{d \in V(D)} X_d = V(G)$. To prove the connectivity condition (D2), let $d, d', d'' \in V(D')$, be such that $d \preceq_{D'} d' \preceq_{D'} d''$. If $d' = d$ or $d' = d''$ then trivially $X'_d \cap X'_{d''} \subseteq X'_{d'}$, so suppose $d \prec_{D'} d' \prec_{D'} d''$. We consider four cases:

- If none of d, d', d'' is d_l or d_r, then $d, d', d'' \in V(D)$, and (D2) follows from the fact that $(D, \mathcal{X})$ is a DAG-decomposition.
- If d is d_l or d_r then $d', d'' \in V(D)$ and, since $d_0 \in V(D)$ is the unique predecessor of d, we obtain the following chain of nodes in D: $d_0 \prec_D d' \prec_D d''$. So $X'_d \cap X'_{d''} = X_{d_0} \cap X_{d''} \subseteq X_{d'} = X'_{d'}$.
- If d'' is d_l or d_r then from the comments at the beginning of the proof, $d \prec_D d' \preceq_D d_0$. Thus, $X'_d \cap X'_{d''} = X_d \cap X_{d_0} \subseteq X_{d'} = X'_{d'}$.
- Finally, if d' is d_l or d_r then by the same reasoning as the previous two cases, $d \preceq_D d_0 \prec_D d''$. So $X'_d \cap X'_{d''} = X_d \cap X_{d''} \subseteq X_{d_0} = X'_{d'}$.

Thus, in all cases, $X'_d \cap X'_{d''} \subseteq X'_{d'}$, showing that (D2) holds. To see that condition (D3) also holds, first note that for all i such that $1 \leq i \leq m$, it is true that $X_{d_0} \cap X_{d_i}$ guards $X_{\succeq d_i} \setminus X_{d_0}$. Therefore, by Lemma 3(2),

$$X_{d_0} \text{ guards } X_{\succeq d_i} \setminus X_{d_0}. \tag{3}$$

Now every source r of D' is also a source of D and therefore $\varnothing$ guards $X_{\succeq r} = X'_{\succeq r}$. So let $(d, d') \in E(D')$. We consider three cases:

- $d' \in V(D)$. If $d = d_l$ or $d = d_r$, then $X'_d = X_{d_0}$. Otherwise $(d, d') \in E(D)$. In both cases, $X'_d \cap X'_{d'}$ guards $X'_{\succeq d'} \setminus X'_d$.
- $d' = d_l$ (so $d = d_0$). Here $X'_{\succeq d'} = X_{d_0} \cup X_{\succeq d_1}$, so $X'_{\succeq d'} \setminus X'_d = X_{\succeq d_1} \setminus X_{d_0}$. Hence, by (3), $X_{d_0} = X'_d \cap X'_{d'}$ guards $X_{\succeq d_1} \setminus X_{d_0} = X'_{\succeq d'} \setminus X'_d$.
- $d' = d_r$ (so $d = d_0$). Here $X'_{\succeq d'} = X_{d_0} \cup \bigcup_{2 \leq i \leq m} X_{\succeq d_i}$, and so $X'_{\succeq d'} \setminus X'_d = (\bigcup_{2 \leq i \leq m} X_{\succeq d_i}) \setminus X_{d_0} = \bigcup_{2 \leq i \leq m} (X_{\succeq d_i} \setminus X_{d_0})$. From Lemma 3(1) and (3), $X'_d \cap X'_{d'} = X_{d_0}$ guards $\bigcup_{2 \leq i \leq m} (X_{\succeq d_i} \setminus X_{d_0}) = X'_{\succeq d'} \setminus X'_d$.

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

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

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

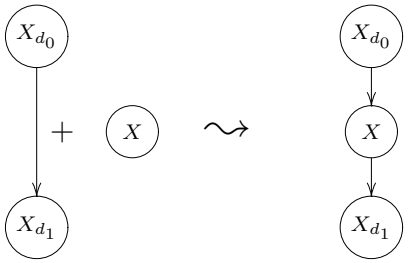

Fig. 5. Adding X to (d_0, d_1)

By the *decomposition resulting from splitting d for $m-2$ times* we mean the decomposition resulting from splitting d, and then recursively splitting the successor of d with more than one successor until no such successor exists. A *complete split* of $(D, \mathcal{X})$ is the decomposition $(D', \mathcal{X}')$ obtained by recursively splitting every node with more than two sucessors.

Definition 22 (Adding)**.** Let $(D, \mathcal{X})$ be a DAG-decomposition of a digraph G. If $(d_0, d_1) \in E(D)$ and $X \subseteq V(G)$ the *decomposition resulting from adding X to (d_0, d_1)* is the pair $(D', \mathcal{X}')$ with

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

See Figure 5 for a visual interpretation.

Lemma 23. *Let $(D, \mathcal{X})$ be a DAG-decomposition of a digraph G of width k and let $(D', \mathcal{X}')$ be the decomposition resulting from adding $X \subseteq V(G)$ to (d_0, d_1). If either*

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

then $(D', \mathcal{X}')$ is a DAG-decomposition of G of width k.

Proof. Note that for all $d, d' \in V(D)$ such that $d \prec_{D'} d_X \prec_{D'} d'$ we have that $d \preceq_{D'} d_0 \prec_{D'} d_X \prec_{D'} d_1 \preceq_{D'} d'$. This implies, for all $d \in V(D)$

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

since if X'_{d_X} is included in the union on the left, then both X'_{d_0} and X'_{d_1} are, and so in either case of the lemma $X'_{d_X} = X$ does not contribute to the overall union.

Further, $X_{d_0} \cap X_{d_1}$ guards $X_{\succeq d_1} \setminus X_{d_0} = X_{\succeq d_1} \setminus (X_{d_0} \cap X_{d_1})$ from Lemma 7.

Clearly, D' is a DAG. We now show that $(D', \mathcal{X}')$ satisfies the properties (D1) to (D3). It is easily seen that $\bigcup_{d \in V(D')} X'_d = X \cup \bigcup_{d \in V(D)} X_d = V(G)$. This

shows (D1). Towards establishing condition (D2), suppose $d \preceq_{D'} d' \preceq_{D'} d''$. If $d' = d$ or $d' = d''$ then trivially $X'_d \cap X'_{d''} \subseteq X'_{d'}$, so suppose $d \prec_{D'} d' \prec_{D'} d''$. We consider four cases:

- If none of d, d', d'' is d_X then d, d', and d'' are all in $V(D)$, so (D2) follows from the fact that $(D, \mathcal{X})$ is a DAG-decomposition.
- Suppose $d = d_X$. From the observations made at the beginning of the proof, we get the following chain of nodes in D: $d_0 \prec_D d_1 \preceq_D d' \prec_D d''$. So if $X \subseteq X_{d_0}$, i.e., we are in case (i) of the lemma, then $X'_d \cap X'_{d''} = X \cap X_{d''} \subseteq X_{d_0} \cap X_{d''} \subseteq X_{d'} = X'_{d'}$, by condition (D2) of $(D, \mathcal{X})$. If $X \subseteq X_{d_1}$, then $X'_d \cap X'_{d''} = X \cap X_{d''} \subseteq X_{d_1} \cap X_{d''} \subseteq X_{d'} = X'_{d'}$.
- Now assume $d'' = d_X$. Then $d \prec_D d' \preceq_D d_0 \prec_D d_1$ and the rest of the proof is symmetric to the previous case.
- Finally, assume $d' = d_X$. Then $d \preceq_D d_0 \prec_D d_1 \preceq_D d''$. Hence $X_d \cap X_{d''} \subseteq X_{d_0}$ and $X_d \cap X_{d''} \subseteq X_{d_1}$. Thus, $X'_d \cap X'_{d''} = X_d \cap X_{d''} \subseteq X_{d_0} \cap X_{d_1} \subseteq X = X'_{d'}$.

Finally, we need to show that (D3) holds as well. First, if d is a source of D', then d is a source of D. Hence $\varnothing$ guards $X_{\succeq d} = X_{\succeq d'}$. So let $(d, d') \in E(D')$. We consider three cases:

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

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

  Further, by Lemma 7, $X_{\succeq d_1} \setminus (X_{d_0} \cap X_{d_1}) = X_{\succeq d_1} \setminus X_{d_0}$. Therefore $X_{\succeq d_1} \setminus X = X_{\succeq d_1} \setminus X_{d_0}$. As $(D, \mathcal{X})$ is a DAG-decomposition, $X_{d_0} \cap X_{d_1}$ guards $X_{\succeq d_1} \setminus X_{d_0}$, and as $X_{d_0} \cap X_{d_1} \subseteq X \cap X_{d_1}$, Lemma 3(2) implies that $X'_d \cap X'_{d_1} = X \cap X_{d_1}$ guards $X_{\succeq d_1} \setminus X_{d_0} = X'_{\succeq d_1} \setminus X'_d$.

  Otherwise we are in case (ii) and we have $X_{d_0} \cap X_{d_1} \subseteq X \subseteq X_{d_1}$. Let $Z = X \setminus (X_{d_0} \cap X_{d_1})$. We know $(X_{d_0} \cap X_{d_1})$ guards $X_{\succeq d_1} \setminus (X_{d_0} \cap X_{d_1})$, due to Lemma 7. Hence, by Lemma 3(3), $X'_d \cap X'_{d_1} = X = (X_{d_0} \cap X_{d_1}) \cup Z$ guards

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

- Finally, suppose $d' = d_X$ (so $d = d_0$). Here we claim $X'_{\succeq d_X} \setminus X'_{d_0} = X_{\succeq d_1} \setminus X_{d_0}$. If $X \subseteq X_{d_0}$, then $X'_{\succeq d_X} \setminus X'_{d_0} = (X \cup X_{\succeq d_1}) \setminus X_{d_0} = (X \setminus X_{d_0}) \cup (X_{\succeq d_1} \setminus X_{d_0}) = X_{\succeq d_1} \setminus X_{d_0}$. If $X \subseteq X_{d_1}$, then since $d_X \preceq_{D'} d_1$, $X'_{\succeq d_X} = X'_{\succeq d_1} = X_{\succeq d_1}$. Now $X \supseteq X_{d_0} \cap X_{d_1}$, so by Lemma 3(2), $X'_{d'} = X$ guards $X_{\succeq d_1} \setminus X_{d_0} = X'_{\succeq d_X} \setminus X'_{d_0}$.

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

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

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

Theorem 24. *If G has a* DAG*-decomposition of width k, it also has a nice* DAG*-decomposition of width k.*

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

(1) We apply Lemma 19 to obtain a decomposition with a unique source, therefore satisfying (N1).
(2) We apply a complete split on $(D, \mathcal{X})$ to obtain a DAG-decomposition such that every node has at most two successors, and if d has two successors d_1 and d_2, then $X_d = X_{d_1} = X_{d_2}$. This establishes (N2) and (N3).
(3) To satisfy (N4), we require two stages. First, for each $(d_0, d_1) \in E(D)$ with $X_{d_0} \neq X_{d_1}$, we add $X_{d_0} \cap X_{d_1}$ to (d_0, d_1) to obtain a DAG-decomposition such that for every $(d, d') \in E(D')$, X_d is either a subset or a super-set of $X_{d'}$.
(4) In the second step for each edge $(d, d') \in E(D)$ such that $|X_d \bigtriangleup X_{d'}| = m \geq 2$ we do the following: If $X_d \supset X_{d'}$ then let $X_0 = X_d, X_1, \ldots, X_m = X_{d'}$ be a strictly decreasing sequence of sets. We then add $X_1, X_2, \ldots, X_{m-1}$ to (d, d'). The case $X_d \subset X_{d'}$ is symmetric.

At this point we have a decomposition which satisfies (N1) to (N4), and is therefore nice. Finally, from Lemmas 19, 21, and 23, at each step we have a DAG-decomposition of width k. □

3.5 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 digraph and computing a DAG-decomposition.

3.5.1 Computing DAG*-width and decompositions*

In Proposition 34 we will show that the DAG-width of a digraph G is equal to the tree-width of its underlying undirected graph plus one. The following is then a direct consequence of the fact that it is NP-hard to determine the tree-width of a graph [2].

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

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

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

Proof. Given a digraph G with vertex set V 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, bipartite digraph, or arena, $H(G) = (V_0 \dot\cup V_1, E)$ which is defined as follows. Let $W_1 := [V]^{\leq k} \times V$ and $W_2 := ([V]^{\leq k} \times [V]^{\leq k} \times V)$.

(i) $V_0 := W_1$,
(ii) $V_1 := W_2 \dot\cup \{v_0\}$, and
(iii) From each node $(X, r) \in W_1$ there is an edge to every node $(X_1, X_2, r') \in W_2$ such that $r = r'$, $X = X_1$, and $\mathit{Reach}_{G\setminus(X_1\cap X_2)}(r) \subseteq \mathit{Reach}_{G\setminus X_1}(r)$. 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 $r' \in \mathit{Reach}_{G\setminus X_1\cap X_2}(r)$. Finally, there is an edge from v_0 to every node $(\varnothing, r) \in W_1$.

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

The game starts with a token at the node v_0. Player 0 moves the token whenever it is on a node in V_0, and Player 1 moves the token whenever it is on a node in V_1. The token may only be moved along an out-edge, on a path of length 1. If a player cannot move he loses. If the game lasts forever, Player 1 wins. Computing which player wins is thus an example of alternating reachability and is therefore decidable in polynomial time (with respect to the size of the arena) (see [7]).

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

Note also that the translation of strategies into decompositions is computationally easy, i.e., can be done in polynomial time. Since winning strategies can be computed in polynomial time in the size of the graph, we get the following.

Proposition 27. *Given a digraph G of* DAG*-width at most k, a* DAG*-decomposition of G of width at most k can be computed in time $\mathcal{O}(|G|^{\mathcal{O}(k)})$.*

3.5.2 Algorithms on digraphs of bounded DAG*-width*

In Section 5, we will show that the directed tree-width of a digraph is bounded above by a constant factor of its DAG-width (see Proposition 35). Therefore any graph property that can be decided in polynomial time on classes of digraphs of bounded directed tree-width can also be decided on classes of graphs of bounded DAG-width. This implies that properties such as Hamiltonicity that are known to be polynomial-time solvable on digraphs of bounded directed tree-width can be solved efficiently on digraphs of bounded DAG-width too. We give a nontrivial application of DAG-width in Section 4 where we show that parity games can be solved efficiently on digraphs 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 digraphs of bounded DAG-width. For instance, the 3-colourability problem is known to be decidable in polynomial time on graphs of bounded tree-width. However, the problem does not depend on the direction of edges. For any given undirected graph, we can simply direct the edges in such a way that it becomes acyclic. Thus, arbitrary instances are polynomial-time reducible to instances of DAG-width 1. As 3-colourability over arbitrary undirected graphs is NP-hard, it follows that the problem cannot be solved in polynomial time on digraphs of bounded DAG-width, unless P $=$ NP. Furthermore, as 3-colourability is MSO-definable, this also implies that Courcelle's theorem [8] does fail for DAG-width.

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

4 Parity Games on Digraphs of Bounded DAG-Width

We are interested in the problem of determining, given a parity game and a starting vertex v, which player has a winning strategy from v. The complexity of this problem in general remains a major open question, as explained in Section 1. We demonstrate that parity games are tractable on arenas of bounded DAG-width by an algorithm similar in spirit to that of Obdržálek [21]. That algorithm relies on the fact that in a tree decomposition (of the underlying undirected graph), the set

of k nodes in any vertex of the decomposition guards all entries and exits to the part of the graph below this vertex. In the case of a DAG-decomposition, while the k-element set guards all exits from the subgraph below it, there may be an unlimited number of edges going into this subgraph. This is the main challenge that our algorithm addresses, and is specifically solved in Lemmas 28, 29 and 30.

A *parity game* is a tuple (V, V_0, E, Ω) where (V, E) is a digraph, $V_0 \subseteq V$ and $\Omega : V \to \omega$ is a function assigning a priority to each vertex. As we shall see, 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 digraph with Even playing when the token is on a vertex in V_0 and Odd playing otherwise. Formally, a *play* of the game is an infinite sequence $\pi = (v_i \mid i \in \omega)$ such that $(v_i, v_{i+1}) \in E$ for all i. We say π is *winning* for Even if $\liminf_{i\to\infty} \Omega(v_i)$ is even and π is winning for Odd otherwise. That is to say, π is winning for Even if the least value that occurs infinitely often in the sequence $(\Omega(v_i))_{i\in\omega}$ is even and it is winninf for Odd otherwise.

A *strategy* is a map $f : V^{<\omega} \to V$ such that, for any sequence $(v_0, \ldots, v_i) \in V^{<\omega}$, we have $(v_i, f(v_0, \ldots, v_i)) \in E$. A play $\pi = (v_i \mid i \in \omega)$ is *consistent* with Even playing f if whenever $v_i \in V_0$, $v_{i+1} = f(v_0, \ldots, v_i)$. Similarly, π is consistent with Odd playing f if whenever $v_i \notin V_0$, $v_{i+1} = f(v_0, \ldots, v_i)$. A strategy f is winning for Even from a vertex v if every play beginning at v that is consistent with Even playing f is winning for Even. A strategy is *memoryless* if whenever $u_0, \ldots, u_i$ and $v_0, \ldots, v_j$ are two sequences in $V^{<\omega}$ with $u_i = v_j$, then $f(u_0, \ldots, u_i) = f(v_0, \ldots, v_j)$. 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 [12]. 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, i.e. an odd priority is always better than an even one; among odd priorities smaller ones are better; and among even ones larger priorities are better.

For a parity game (V, V_0, E, Ω), consider $U \subseteq V$ and a set W that guards U. Fix a pair of strategies f and g. For any $v \in U$, there is exactly one play $\pi = (v_i : i \in \omega)$ starting at v that is consistent with Even playing f and Odd playing g. Let 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.

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

That is to say that, if the play that results from Even playing f and Odd playing g leads to a cycle contained entirely within U, then the outcome simply records which player wins the game. However, if the winner is not determined entirely within U, the outcome records the vertex w in W in which the play emerges from U and the lowest priority that is seen in the play 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 $(\text{pot-out}(W), \trianglelefteq)$, 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 $(\text{pot-out}(W), \trianglelefteq)$. By definition of the order $\trianglelefteq$, if either of winEven or winOdd is in the set $\text{result}_f(U, v)$, then it is the sole element of the set. Also, for each $w \in W$, there is at most one p such that $(w, p) \in \text{result}_f(U, v)$ so the number of distinct values that $\text{result}_f(U, v)$ can take is at most $(n+1)^{|W|} + 2$. This is an upper bound on the cardinality of the set 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 result(U, v) for the set $\{\text{result}_f(U, v) : f \text{ is a strategy}\}$.

Suppose now that (V, V_0, E, Ω) is a parity game and $(D, \mathcal{X})$ is a DAG-decomposition of (V, E) of width k that is *nice* in the sense of Definition 18. For each $d \in V(D)$, we write V_d for the set $X_{\succeq d} \setminus X_d$. The key to the algorithm is that we construct the set of results result(V_d, v) for each $v \in V_d$. Since V_d is guarded by X_d and $|X_d| \leq k$, the number of distinct values of 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 the key data structure used in our algorithm:

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

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 is a sink. In this case, V_d is empty and so is Frontier(d).

Case 2: d has two successors e_1 and e_2. In this case, $X_d = X_{e_1} = X_{e_2}$ by the definition of a nice decomposition. Thus, $V_d = V_{e_1} \cup V_{e_2}$. Moreover, each of the three sets V_d, V_{e_1} and V_{e_2} is guarded by X_d so, in particular, each path from a vertex in $V_{e_1} \setminus V_{e_2}$ to a vertex in $V_{e_2} \setminus V_{e_1}$ (or vice versa) contains a vertex from 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 $o =$ 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 $\text{out}_{f,g}(V_d, v) = \text{out}_{f,g}(V_{e_1}, v)$ and therefore $r = \text{result}_f(V_{e_1}, v)$.

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

Case 3: d has one successor e and $X_d \setminus X_e = \{u\}$. Then, by (D2), $u \notin V_e$. Also, by definition of V_d, $u \notin V_d$. We conclude that $V_d = V_e$. Moreover, since X_e guards V_e (by Lemma 3(2)), there is no path from any element of V_e to u except through X_e. Thus, if $(w, p) \in \text{result}_f(V_d, v)$ for some v and f, it must be the case that $w \in X_e$. Hence, $\text{Frontier}(d) = \text{Frontier}(e)$.

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

Consider the set of vertices v in $X_{\trianglerighteq d}$ such that $(u, v) \in E(G)$. These fall into two categories. Either $v \in X_d$ or $v \in V_e$. Let $x_1, \ldots, x_s$ enumerate the first category and let $v_1, \ldots, v_m$ enumerate the second. Let $O = \{(x_i, \min\{\Omega(x_i), \Omega(u)\}) : 1 \leq i \leq s\}$. This is the set of outcomes obtained if a 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 $\trianglelefteq$, $O \in \text{pot-res}(X_d)$. We write $\mathcal{O}$ for $\{\{o\} : o \in O\}$. That is $\mathcal{O}$ is the set of singleton results obtained from O. For each v_i we know, from $\text{Frontier}(e)$, the set $\text{result}(V_e, v_i)$. For each result $r \in \text{result}(V_e, v_i)$, we write $\text{mod}(r)$ for the set of outcomes defined by modifying r as follows. First, if r contains an outcome (u, p), we replace it by winEven if $\min\{p, \Omega(u)\}$ is even and winOdd if it is odd. Secondly, for any pair $(w, p) \in r$ where $w \neq u$, we replace it with $(w, \min\{p, \Omega(u)\})$. Finally, we take the set of $\trianglelefteq$-minimal elements from the resulting set. This is $\text{mod}(r)$. Note that $\text{mod}(r) \in \text{pot-res}(X_d)$. The intuition is that $\text{mod}(\text{result}_f(V_e, v_i))$ defines the set of best possible outcomes for player Odd, if starting at u, the play goes to v_i and from that point on, player Even plays according to strategy f. For each $1 \leq i \leq m$, let $M_i = \{\text{mod}(r) : r \in \text{result}(V_e, v_i)\}$.

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

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

Lemma 28. *If $u \in V_0$, then result$(V_d, u) = M_1 \cup \ldots \cup M_m \cup \mathcal{O}$.*

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

$r = \text{mod}(\text{result}_f(V_e, v_i))$, then $r = \text{result}_{f'}(V_d, u)$ where f' is the strategy that moves from u to v_i and then follows the strategy f from that point on. □

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

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

Proof. $\Rightarrow$ Let $r \in \text{result}(V_d, u)$, i.e. there is a strategy f such that $r = \text{result}_f(V_d, u)$. We define the function c by $c(i) = \text{mod}(\text{result}_f(V_e, v_i))$. Since player Odd can move to any of the vertices v_i, it is clear that $r \trianglelefteq c(i)$, 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 29 suggests constructing $\text{result}(V_d, u)$ by considering all possible choice functions c. However, as each set M_i may have as many as $(n+1)^k + 2$ elements, there are $m^{(n+1)^k+2}$ possibilities for c and our algorithm would be exponential. We consider an alternative way of constructing $\text{result}(V_d, u)$. Recall that $\text{result}(V_d, u) \subseteq \text{pot-res}(X_d)$ and the latter set has at most $(n+1)^k + 2$ elements. We check, for each $r \in \text{pot-res}(X_d)$, in polynomial time, whether there is a choice function c as in Lemma 29 that yields r. In particular, we take the following alternative characterisation of $\text{result}(V_d, u)$.

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

(i) *$r = O \sqcap \bigsqcap_{i \in D} d(i)$; and*
(ii) *for each $i \in [m] \setminus D$ there is an $r_i \in M_i$ with $r \trianglelefteq r_i$.*

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

$\Leftarrow$ Given D, d and the collection of r_i as specified, we define the choice function c

by

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

Since by hypothesis $r \trianglelefteq r_i$ and $r = O \sqcap \bigsqcap_{i \in D} d(i)$, it is then 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 result(V_d, u) we cycle through all sets $D \subseteq [m]$ with k or fewer elements (and there are $\mathcal{O}(n^k)$ such sets) and for each one consider all candidate functions d (of which there are $\mathcal{O}(n^{k^2})$). Having found a d which gives $r = O \sqcap \bigsqcap_D d(i)$, 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 result(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 result(V_d, u). To be precise, let $r \in$ result(V_e, v) for some $v \in V_e$ and $s \in$ result(V_d, u). Define the combined result $c(r, s)$ as follows:

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

Intuitively, if $r =$ 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 31. *For any $v \in V_e$,*

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

Proof. Clearly, for any strategy f, result$_f(V_d, v) = c($result$_f(V_e, v),$ result$_f(V_d, u))$. Thus, result(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. Therefore, 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 set $s =$ 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$(d) = \{(v, r) : r \in$ result$(V_d, v)\}$.

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

Proof. By Proposition 27, there is a polynomial-time algorithm that will produce a DAG-decomposition of the game graph of width at most k. This can be converted into a nice decomposition $(D', \mathcal{X}')$ in time at most quadratic (in the size of the decomposition). Let a be the source of D' and let $X_a = \{x_1, \ldots, x_l\}$ where $l \leq k$. Consider the DAG D formed by adding l new vertices $a_0, \ldots, a_{l-1}$ to D' in a simple directed path ending in a. Further, for each i define X_{a_i} to be the set $\{x_1, \ldots, x_i\}$. In particular, the new source a_0 is labelled by $\varnothing$. It is easily seen that the new labelled DAG $(D, \mathcal{X})$ with $\mathcal{X} = (X_d)_{d \in V(D)}$ still meets the definition of a nice decomposition. We then use the above construction to obtain Frontier(d) for each d in D, starting from the sinks and working our way to the source. Since the size of D is at most $n^{2k} + k$, the total time taken is bounded by a polynomial. Now, for the source a_0 of D it is true that $X_{\geq a_0} = V_{a_0} = V$. Thus, if $(v, r) \in$ Frontier(a_0) then $r \subseteq \{\text{winEven}, \text{winOdd}\}$. 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 unrivalled robustness. On the realm of digraphs, however, its heritage seems to be split among several different concepts. In the remainder of the article, we compare DAG-width with other connectivity measures for digraphs, particularly with directed tree-width introduced by Johnson et al. [16] and directed path-width [3].

5.1 Undirected tree-width

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

The tree-width of a digraph G is defined as the tree-width of the underlying undirected graph, that is, the graph obtained from G by replacing each directed edge (u, v) with an undirected edge $\{u, v\}$ and removing duplicates.

Proposition 33.

(i) *If a digraph G has tree-width k, then its* DAG*-width is at most $k + 1$.*

(ii) *There exists a family of digraphs with arbitrarily large tree-width and* DAG*-width* 1.

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

(ii). For any number n, let K_n be the (undirected) complete graph with n vertices $v_1, v_2, \ldots, v_n$. Orient the edges of K_n such that (v_i, v_j) is an edge if and only if $i < j$. The resulting digraph is acyclic and therefore has DAG-width 1, but the underlying undirected graph is a complete graph of n vertices and therefore has tree-width $n - 1$. □

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

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

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

5.2 Directed tree-width

With the aim of recovering the effectiveness of tree decompositions in allowing divide-and-conquer algorithms, directed tree-width is associated with a tree-shaped representation of the input digraph. It was proved that this representation leads to

efficient algorithms for solving a particular class of NP-complete problems, including, e.g., Hamiltonicity, when directed tree-width is bounded. Unfortunately this generic method does not cover many interesting problems. In particular, the efficient solution of parity games on bounded tree-width has failed so far to generalise to directed tree-width.

In terms of games, directed tree-width is characterised by a restriction of the cops and robber games for DAG-width, in which the robber is only permitted to move to vertices where there exists a directed cop-free path from his intended destination back to the current position. In contrast to the case of undirected tree-width, for these games cop-monotonicity and robber-monotonicity differ and both cop- and robber-monotone strategies are known to not be sufficient [1]. However, the difference is within a constant factor.

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

Proposition 35.

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

Proof. (i). The argument is based on the duality theorem for directed tree width proved in [16], which relates the notions of havens and arboreal decompositions, i.e., tree decompositions, in our terminology. The idea is as follows. If G has DAG-width k then k cops can win the k-cops and robber game on G. Thus, k cops can win the game defined in [16], and so G does not have a (directed) haven of size k. By the duality result of [16], this implies that G has an directed tree decomposition of width at most $3k + 1$.

(ii). Consider the family $\{(T_k^2)^{op} : k \geq 2\}$ of digraphs defined in Proposition 12. Note that $(T_k^2)^{op}$ is a binary branching tree of height k with back-edges from every vertex to each of its ancestors. We have shown that $(T_k^2)^{op}$ has cop number k, and it is clear that the strategy described for k cops is monotone, so $(T_k^2)^{op}$ has DAG-width k. On the other hand, consider the directed tree T obtained from $(T_k^2)^{op}$ by removing back-edges. For each $t' \in V(T)$, let $B_{t'} := \{t, s\}$ where t is the vertex corresponding to t' in $(T_k^2)^{op}$ and s is the predecessor of t (if t' is not the root of T), and let $X_{(s',t')} := \{s\}$ for all $(s', t') \in E(T)$. Then, it is easy to show that $(T, (B'_t)_{t' \in V(T)}, (X_e)_{e \in E(T)})$ is a directed tree decomposition of $(T_k^2)^{op}$ of width 1. For $k \geq 2$, $(T_k^2)^{op}$ is not acyclic and therefore has directed tree-width exactly 1. □

5.3 Directed path-width

Directed path-width was introduced by Reed, Seymour and Thomas as a generalisation of path-width to digraphs (see [27,3]). Formally, a *directed path decomposition* of a digraph G is a sequence $W_1, W_2, \ldots, W_n$ such that

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

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

It is worth noting that for undirected graphs, path-width readily generalises to tree-width as a path decomposition is also a tree decomposition. We show that DAG-width generalise directed path-width in the same way.

Proposition 36.

(i) *If a digraph G has directed path-width k, its* DAG*-width is at most $k + 1$.*
(ii) *There exists a family of digraphs 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 G of width k. Let D_n be the directed path with n vertices. That is, $V(D_n) = \{d_1, \ldots, d_n\}$ and $E(()D_n) = \{(d_1, d_2), \ldots, (d_{n-1}, d_n)\}$. Set $X_{d_i} := W_i$, for all $d_i \in V(D_n)$. We claim that $(D_n, (X_d)_{d \in V(D_n)})$ is a DAG-decomposition of G of width $k + 1$. Condition (D1) follows from (P1) and (D2) follows from (P2). To show (D3) for $1 \leq i < n$, suppose $v \in X_{\succeq d_{i+1}} \setminus X_{d_i}$ and $(v, w) \in E(G)$. From (P3) there exist $i' \leq j'$ such that $v \in W_{i'}$ and $w \in W_{j'}$. If $i' \leq i$, then by (P2) $v \in X_{d_i}$, contradicting the choice of v. Thus, $i < i' \leq j'$ and $w \in X_{\succeq d_{i+1}}$. If $w \notin X_{\succeq d_{i+1}} \setminus X_{d_i}$ then $w \in X_{d_i}$ and therefore $w \in X_{d_{i+1}}$ by (P2). Thus, $X_{d_i} \cap X_{d_{i+1}}$ guards $X_{\succeq d_{i+1}} \setminus X_{d_i}$.

(ii). Let T_k be the (undirected) complete ternary tree of height $k \geq 2$. According from Proposition 34, $\overleftrightarrow{T_k}$ has DAG-width 2. On the other hand, it is known from [17] that T_k has path-width exactly k, and it is straightforward to show that $\overleftrightarrow{T_k}$ must therefore have directed path-width exactly $k - 1$. Thus, the family $\{T_k : k \geq 2\}$ witnesses the statement. □

In [3], Barát showed that directed path-width corresponds to the number of cops required to catch an invisible robber on a digraph. It should therefore not be surprising that our measure generalises directed path-width.

We conclude that, despite their conceptual affinity, directed tree-width, directed path-width, and DAG-width are rather different measures. The following inequalities summarise, up to constant factors, the results of this section.

$$\text{directed tree-width}(G) \leq \text{ DAG-width}(G) \leq \begin{cases} \text{tree-width}(G) \\ \text{directed path-width}(G). \end{cases}$$

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

6 Further Remarks

We conclude with a comment on a few recent and relevant results. In [19], Kreutzer and Ordyniak show that monotonicity is not sufficient in the cops and robber game on digraphs. In particular, for any $n \in \mathbb{N}$ there exist digraphs which require n more cops to capture the robber with a monotone strategy than with a non-monotone strategy. Their examples do not preclude the possibility of bounded monotonicity cost; that is, the existence of a function f such that if k cops have a winning strategy then $f(k)$ cops have a monotone winning strategy. We believe that these games have linearly bounded monotonicity cost, however the problem remains an active area of research.

Another measure for the connectivity of directed graphs is entanglement, proposed by Berwanger and Grädel [5]. Unlike the other measures considered here, entanglement is not associated with an efficient tree-shaped graph representation. Nevertheless, it was shown that parity games on graphs of bounded entanglement can be solved in polynomial time [5]. In fact, just a bound on the minimal entanglement of a subgraph induced by any winning strategy rather than of the input graph is required. It is difficult to compare entanglement with DAG-width as the latter measure requires monotone strategies whereas the former does not.

The class of digraphs for which the winner of a parity game can be efficiently decided has been extended by Obdržálek [20] to include digraphs of bounded clique-width. As there are DAGs of arbitrary clique-width and digraphs of fixed clique-width but arbitrary DAG-width, this result is incomparable with our own. Whether there exists a measure which generalises both clique-width and DAG-width, particularly with regard to efficiently solving parity games, remains an open problem.

Toward investigating other characterisations of tree-width and their extension to digraphs, Hunter and Kreutzer [15] show that the natural generalisations of partial k-trees and elimination orderings result in a measure different from DAG-width

which they call *Kelly-width*. The measures are similar in that the cop number of a digraph bounds its Kelly-width and likewise, a non-monotone version of Kelly-width bounds DAG-width, up to constant factors. As a consequence, the authors conjecture that Kelly-width lies within a constant factor of DAG-width. Resolving this and similar questions would provide insight into the structure theory of digraphs associated with DAG-width and is part of ongoing work.

Acknowledgements. This work has been partially supported by the ESF Networking Programme GAMES and the Czech research grant GAČR 201/09/J021.

References

[1] I. Adler, Directed tree-width examples, J. Comb. Theory Ser. B 97 (5).

[2] S. Arnborg, D. Corneil, A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Matrix Anal. Appl. 8 (2) (1987) 277–284.

[3] J. Barát, Directed path-width and monotonicity in digraph searching, Graphs and Combin. 22 (2) (2006) 161–172.

[4] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, DAG-width and parity games, in: STACS'06, vol. 3884 of LNCS, Springer, 2006, pp. 524–536.

[5] D. Berwanger, E. Grädel, Entanglement – A measure for the complexity of directed graphs with applications to logic and games, in: LPAR'04, vol. 3452 of LNCS, Springer, 2005, pp. 209–223.

[6] H. Bodlaender, Treewidth: Algorithmic techniques and results, in: MFCS'97, vol. 1295 of LNCS, Springer, 1997, pp. 19–36.

[7] A. K. Chandra, D. Kozen, L. J. Stockmeyer, Alternation, J. ACM 28 (1981) 114–133.

[8] B. Courcelle, The monadic second order logic of graphs I: Recognizable sets of finite graphs, Inform. and Comput. 85 (1990) 12–75.

[9] N. Dendris, L. Kirousis, D. Thilikos, Fugitive-search games on graphs and related parameter, Theoret. Comput. Sci. 172 (1997) 233–254.

[10] R. Diestel, Graph Theory, 3rd ed., Springer, 2005.

[11] E. Emerson, C. Jutla, The complexity of tree automata and logics of programs, in: FOCS'88, IEEE, 1988, pp. 328–337.

[12] E. Emerson, C. Jutla, Tree automata, mu-calculus and determinacy, in: FOCS'91, IEEE, 1991, pp. 368–377.

[13] E. Emerson, C. Jutla, A. Sistla, On model checking for the μ-calculus and its fragments, Theoret. Comput. Sci. 258 (1-2) (2001) 491–522.

[14] G. Gottlob, N. Leone, F. Scarcello, Robbers, marshals, and guards: Game theoretic and logical characterizations of hypertree width, in: PODS'01, ACM, 2001, pp. 195–201.

[15] P. Hunter, S. Kreutzer, Digraph measures: Kelly decompositions, games, and orderings, Theor. Comput. Sci. 399 (3) (2008) 206–219.

[16] T. Johnson, N. Robertson, P. D. Seymour, R. Thomas, Directed tree-width, J. Combin. Theory Ser. B 82 (1) (2001) 138–154.

[17] L. M. Kirousis, C. H. Papadimitriou, Searching and pebbling, Theor. Comput. Sci. 47 (3) (1986) 205–218.

[18] D. Kozen, Results on the propositional mu-calculus, Theoret. Comput. Sci. 27 (1983) 333–354.

[19] S. Kreutzer, S. Ordyniak, Digraph decompositions and monotonicity in digraph searching, in: WG'08, vol. 5344 of LNCS, Springer, 2008, pp. 336–347.

[20] J. Obdržálek, Clique-width and parity games, in: CSL'07, vol. 4646 of LNCS, Springer, 2007, pp. 54–68.

[21] J. Obdržálek, Fast mu-calculus model checking when tree-width is bounded, in: CAV 2003, vol. 2725 of LNCS, Springer, 2003, pp. 80–92.

[22] J. Obdržálek, DAG-width – connectivity measure for directed graphs, in: SODA'06, ACM-SIAM, 2006, pp. 814–821.

[23] B. Reed, Introducing directed tree width, in: 6th Twente Workshop on Graphs and Combinatorial Optimization, vol. 3 of Electron. Notes Discrete Math., Elsevier, 1999, pp. 222–229.

[24] N. Robertson, P. D. Seymour, Graph minors. III. Planar tree-width, J. Combin. Theory Ser. B 36 (1) (1984) 49–63.

[25] M. Safari, D-width: A more natural measure for directed tree-width, in: MFCS'05, vol. 3618 of LNCS, Springer, 2005, pp. 745–756.

[26] P. D. Seymour, R. Thomas, Graph searching and a min-max theorem for tree-width, J. Combin. Theory Ser. B 58 (1) (1993) 22–33.

[27] R. Thomas, Directed tree-width, slides from a lecture at the Regional NSF-CBMS Conference on Graph Structure and Decomposition, 2002. Available at `http://www.math.gatech.edu/~thomas/SLIDE/CBMS/dirtrsl.pdf` (2002).