

DAG-width – Connectivity Measure for Directed Graphs

Jan Obdržálek*

Laboratory for Foundations of Computer Science
The University of Edinburgh
j.obdrzalek@ed.ac.uk

Abstract

Tree-width is a very useful connectivity measure for undirected graphs. We propose a new definition, called DAG-width, for directed graphs which measures how close a graph is to a directed acyclic graph. In addition we define a cops-and-robber game and show that this game characterises exactly the class of graphs of bounded DAG-width. A comparison of DAG-width with tree-width and directed tree-width follows. Finally we show that NP-complete problems can be solved in polynomial time on graphs of bounded DAG-width.

1 Introduction

Tree-width is a graph theoretic concept first introduced by Robertson and Seymour [9] in their work on graph minors. Roughly speaking, tree-width measures how close a graph is to a tree. Interesting problems can be solved faster on graphs with bounded tree-width than on graphs in general using decomposition of the problem into subproblems. Some NP-complete problems such as graph colouring or Hamiltonian cycle [1], or even PSPACE-complete problems such as some path-forming games [2] are solvable in linear or polynomial time on these graphs. See [3] for an introduction to tree-width and [7] for more in-depth coverage.

It has taken a long time for a similar connectivity measure to be defined for directed graphs [4]. The main reason for this has been the problem of defining a right “separation lemma” for directed graphs (more or less a canonical way of dividing a graph using smallest possible cutsets). The notion defined in [4] is called *directed tree-width*, where again the decomposition structure of the graph is a tree – but this time directed (i.e. with a designated root). See also [8] for some background on directed tree-width.

However, the notion of directed tree-width has had

less impact than tree-width. We try to identify reasons why this is so. It appears as though in the attempt to capture as broad a class of graphs as possible, the definition is too general. The main problem seems to be that the separator sets are not monotone with respect to decomposition (called arboreal decomposition). This makes reasoning about directed tree-width complicated and error prone (if you want to read the [4] paper, check also the addendum [5]).

Our goal is therefore to find a measure with nice algorithmic and graph theoretical properties, which is simpler to use and reason about and retains generality. We introduce a new connectivity measure called DAG-width, whose decomposition structure of which is a DAG. The rest of the paper is organised as follows: In the following section we present the notions of tree-width and directed tree-width, as well as some notation. Section 3 then contains the definition of DAG-width and some of its properties including a normal form. In the next section we present a variant of cops-and-robber games related to the definition of DAG-width. After that in Section 5 we prove that this measure is a little stricter than directed tree-width and show also a relationship to tree-width of undirected graphs. In Section 6 we discuss ways of solving NP-complete problems in polynomial time on graphs of bounded DAG-width. We conclude with extra observations and sketch some future work.

2 Tree-width and Directed Tree-width

We start by first presenting the definitions of tree-width and directed tree-width, which will be used throughout the paper. Unless said otherwise all graphs in this paper are directed and finite. We also need to introduce some notation: For a graph G let $V(G)$ be its vertices, $E(G)$ its edges and $G \setminus X$ the graph obtained from G by deleting vertices in X . Similarly $G|X$ is the graph $G \setminus (V(G) \setminus X)$. We say that directed edge (u, v) which goes from u to v has *tail* u and *head* v . Finally we denote $[V]^{<k}$ the set of all subsets of V of cardinality $< k$.

*The author has been partially supported by the research centre ‘Institute for Theoretical Computer Science (ITI)’, project No. 1M0021620808.

2.1 Tree-width

DEFINITION 2.1. (TREE DECOMPOSITION) A tree decomposition of an (undirected) graph G is a pair (T, \mathcal{X}) , where T is a tree (its vertices are called nodes throughout this paper) and $\mathcal{X} = \{X_t \mid t \in T\}$ is a family of subsets of $V(G)$ satisfying the following three conditions:

- (T1) $V(G) = \bigcup_{t \in V(T)} X_t$,
- (T2) for every edge $\{v, w\} \in E(G)$ there exists $t \in V(T)$ s.t. $\{v, w\} \subseteq X_t$, and
- (T3) for all $t, t', t'' \in V(T)$ if t' is on the (unique) path from t to t'' in T , then $X_t \cap X_{t''} \subseteq X_{t'}$.

The width of a tree decomposition (T, \mathcal{X}) is $\max_{t \in T} |X_t| - 1$. The tree-width of a graph G (written as $tw(G)$) is the minimum width over all possible tree decompositions of G . Trees have tree-width one. One obtains an equivalent definition if the third condition is replaced by:

- (T3') For all $v \in V$, the set of nodes $\{t \in V(T) \mid v \in X_t\}$ is a connected subtree of T .

Tree-width generalises easily to directed graphs: For a directed graph G we put $tw(G) = tw(G')$ where G' is obtained from G by forgetting the orientation of the edges. We can therefore freely use the term “tree-width” when talking about directed graphs. For more information about tree-width we refer the reader to [7].

2.2 Directed tree-width Directed tree-width was introduced by Johnson, Robertson, Seymour and Thomas in [4] (also see [8]) as a counterpart of tree-width for directed graphs. The decomposition structure is still a tree, though this time directed (i.e. with a designated root). We will see that the definition looks different from that of tree-width. Before we present the definition we need to introduce some more notation.

For a directed acyclic graph R we use the following notation: If $r, r' \in V(R)$ we write $r < r'$ iff there is a directed path with initial vertex r and terminal vertex r' . We write $r \leq r'$ iff $r < r'$ or $r = r'$. Finally if $e \in E(R)$ then $e \sim r$ iff e is incident with r . For a graph G a set $S \subseteq V \setminus Z$ is Z -normal if there is no directed path in $G \setminus Z$ with first and last vertices in S that uses a vertex of $G \setminus (S \cup Z)$. I.e. no path can leave S and then return back to S without passing through a vertex in Z .

DEFINITION 2.2. (ARBOREAL DECOMPOSITION) An arboreal decomposition of a graph G is a triple $(R, \mathcal{X}, \mathcal{W})$ where R is a directed tree, and $\mathcal{X} = \{X_e \mid e \in E(R)\}$, $\mathcal{W} = \{W_r \mid r \in V(R)\}$ are sets of vertices of G satisfying:

(R1) \mathcal{W} is a partition of $V(G)$ into nonempty sets

(R2) for $e \in E(R)$, $e = (r_1, r_2)$ the set $\bigcup \{W_r \mid r \in V(R) \text{ and } r \geq r_2\}$ is X_e -normal.

The width of $(R, \mathcal{X}, \mathcal{W})$ is the least integer w such that for all $r \in V(R)$, $|W_r \cup \bigcup_{e \sim r} X_e| \leq w$. The directed tree-width of a graph G (written as $dtw(G)$) is the minimum width over all possible tree decompositions of G . You can see an example of arboreal decomposition of width 1 in Fig. 1 (b). Sets W_r are drawn in the nodes and edges are annotated by sets X_e .

3 DAG-width

In this section we present a definition of our new connectivity measure called *DAG-width*. The main difference with the previous two measures is that we use a DAG instead of a (directed) tree as the basis for the decomposition. Note that the (D1)-(D3) properties closely correspond to (T1)-(T3) in the Definition 2.1.

DEFINITION 3.1. (DAG DECOMPOSITION) A DAG-decomposition of a (directed) graph G is a pair (D, \mathcal{X}) where D is a DAG and $\mathcal{X} = \{X_d \mid d \in V(D)\}$ is a family of subsets of $V(G)$ satisfying:

(D1) $V(G) = \bigcup_{d \in V(D)} X_d$

(D2) If $(d, d') \in E(D)$, then for each $(v, w) \in E$ s.t. $v \in X_{\geq d'} \setminus X_d$ we have $w \in X_{\geq d'}$, where $X_{\geq c} = \bigcup_{c' \geq c} X_{c'}$. If d' is a root we replace X_d with \emptyset .

(D3) for all $d, d', d'' \in D$ if d' lies on (some) path from d to d'' , then $X_d \cap X_{d''} \subseteq X_{d'}$.

The width of a DAG decomposition (D, \mathcal{X}) is $\max_{d \in D} |X_d| - 1$. The DAG-width of a graph G (written as $dgw(G)$) is the minimum width over all possible DAG decompositions of G . DAGs have DAG-width zero. It is easy to see that we can turn every DAG decomposition into one with a single root d_0 , and in the rest of the paper we assume this to be the case. To get a better intuition check the Fig. 1 (c).

Node $d \in V(D)$ is a *root node* of vertex $v \in V(G)$ if $v \in X_d$ and $\forall d' \in V(D). d' < d \implies v \notin X_{d'}$. We denote $X_R(v)$ the set of all root nodes for a vertex v . (Obviously for each vertex of G this set contains at least one element.) In the rest of this paper we will also use $X_{>d}$ to denote the set $X_{\geq d} \setminus X_d$.

From the definition of DAG decomposition it is apparent that root nodes play an important role - and there is an associated normal form. We say that a DAG decomposition (D, \mathcal{X}) is in *normal form*, if in addition to (D1)-(D3) the following two properties hold:

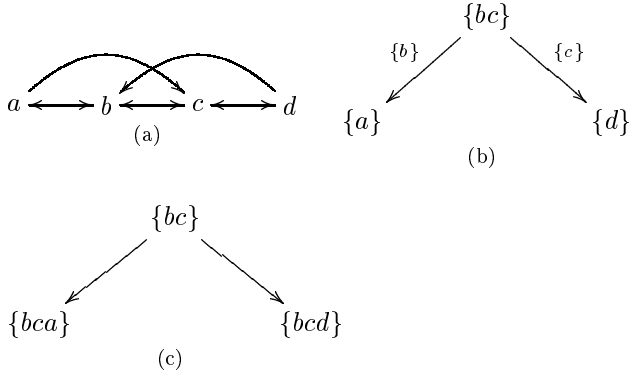


Figure 1: Graph G_X , its arboreal decomposition and DAG decomposition

- (i) for every edge $(d, d') \in E(D)$ there is no other path from d to d' in D .
- (ii) every node $d \in D$ is a root node of some vertex $v \in V(G)$

LEMMA 3.1. (NORMAL FORM) *Let G be a graph and (D, \mathcal{X}) its DAG decomposition of width k . Then there is an algorithm which converts (D, \mathcal{X}) into (D', \mathcal{X}') of width k in normal form in linear time in the size of D .*

Proof. We will construct a sequence $(D, \mathcal{X}) = (D_0, \mathcal{X}_0), \dots, (D_f, \mathcal{X}_f) = (D', \mathcal{X}')$ of pairs (D_i, \mathcal{X}_i) by iteratively removing (one at a time) edges violating (i) and nodes violating (ii) in the definition of normal form above, while maintaining the invariant that (D_i, \mathcal{X}_i) is a DAG decomposition of G .

$D_0 = D$ satisfies the DAG decomposition axioms by definition. Let D_i be the DAG after the i -th iteration. For case (i) we put $V(D_{i+1}) = V(D_i), E(D_{i+1}) = E(D_i) \setminus \{(d, d')\}, \mathcal{X}_{i+1} = \mathcal{X}_i$. It is clear that removing an edge (d, d') s.t. there is some other path from d to d' in D_i does not violate any of (D1)-(D3).

For case (ii) assume there is a node $d \in V(D_i)$ s.t. d is not a root node for any vertex. Let s_1, \dots, s_k be its predecessors and t_1, \dots, t_l its successors. We put $V(D_{i+1}) = V(D_i) \setminus \{d\}, \mathcal{X}_{i+1} = \mathcal{X}_i$ and $E(D_{i+1}) = \{e \in E(D_i) \mid e \not\sim d\} \cup E'$ where $E' = \{(s_i, t_j) \mid 1 \leq i \leq k, 1 \leq j \leq l\}$. A quick check reveals that (D1)-(D3) remain satisfied.

A natural question would be whether we get an equivalent class of graphs if (ii) above is replaced by “every node $d \in D$ is a root node of exactly one vertex $v \in V(G)$ ”, giving us a bijection between vertices of G and nodes of D . It can be shown that this is impossible in general, however the subclass of graphs satisfying this property is very interesting and deserves further study.

We also could have made the root nodes more prominent in the definition of DAG decomposition. The following property (D2') can be used instead of (D2) in Definition 3.1 without changing the class of the DAG decompositions:

- (D2') if $(v, w) \in E(G)$ then for each $d \in X_R(v)$ either $w \in X_d$ or there exist $d' \in X_R(w)$ s.t. $d < d'$.

In [4] the authors mention two other attempts at defining a version of tree-width for directed graphs which did not work because they were not closed under directed unions. (A graph G is a directed union of graphs G_1 and G_2 if G_1 and G_2 are induced subgraphs of G , $V(G_1) \cup V(G_2) = V(G)$ and no edge of G has a head in $V(G_1)$ and tail in $V(G_2)$.) DAG decompositions are indeed closed under directed unions:

LEMMA 3.2. *Let G, G_1, G_2 be as above with $dgw(G_1) = k_1$ and $dgw(G_2) = k_2$. Then $dgw(G) = \max\{k_1, k_2\}$.*

Proof. Let $(D_1, \mathcal{X}_1) [(D_2, \mathcal{X}_2)]$ be the decomposition of G_1 (G_2) of width k_1 (k_2), and let d_2 be the root of D_2 . Then (D, \mathcal{X}) where $V(D) = V(D_1) \cup V(D_2)$, $E(D) = E(D_1) \cup E(D_2) \cup \{(d, d_2) \mid d \in D_1\}$ and $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$ is a DAG decomposition of width $\max\{k_1, k_2\}$.

4 Cops and Robber Games

Both tree-width and directed tree-width are closely related to certain cops-and-robber games on graphs. These games provide a valuable insight into the inner workings of the decompositions and therefore we will also introduce them here. We also show that there is a version of this game strongly related to DAG decompositions.

The original game first appeared in [10]. The robber stands on a vertex of the graph, and can at any time run at great speed to any other vertex along a path of the graph. He is not permitted to run through a cop, however. There are k cops, each of whom at any time either stands on a vertex or is in a helicopter. The goal of the player controlling the cops is to land a cop via a helicopter onto a vertex currently occupied by the robber, and the robber's objective is to elude capture. (The point of helicopters is that cops are not constrained to move along the paths of the graph.) The robber can see the helicopter landing and may run to a new vertex before it actually lands. If k cops can capture the robber in G we say that k cops can *search* G . Moreover if they can do so without revisiting a vertex then they can *monotonely search* G . The following theorem comes from [10]:

THEOREM 4.1. ([10]) *Let G be a (undirected) graph. Then the following are equivalent:*

- (1) k cops can search G
- (2) k cops can monotonely search G
- (3) G has tree-width at most $k - 1$

In [4] the authors presented a similar game for directed graphs. There are only two differences to the above mentioned game: 1. the robber must respect the orientation of edges, and 2. when the robber moves he must stay in the same strong component of $G \setminus Z$, where Z is the set of vertices currently occupied by the cops. Unfortunately, there is not such a nice theorem as Theorem 4.1 for directed tree-width. In particular, the monotone and non-monotone search strategies are not equivalent – there is an example in [4] of a graph which can be searched by 4 cops, but only if some vertex is revisited twice. Moreover only the (directed tree-width) equivalent of (3) \implies (1) was proved, the reverse being conjectured.

For our purposes we will use a variant of this game where the restriction “robber must stay in the same SCC” is dropped. We want to argue that this game is indeed the right counterpart to our definition of DAG-width. The following theorem is a directed version of (3) \implies (1) above:

THEOREM 4.2. *Let G be a directed graph of DAG-width k . Then it can be monotonely searched by $k + 1$ cops.*

Proof. The winning monotone strategy for the cops is as follows. In the first move cops will occupy X_{d_0} , where d_0 is the root of D . The robber now must be in some vertex $v \in X_{>d_0}$. Since D is a rooted DAG with every node accessible from the root, there must be an edge $(d_0, d_1) \in E(D)$ s.t. some $d \in X_R(v)$ is reachable from d_1 . In the next move cops in $X_{d_0} \setminus X_{d_1}$ will take off, leaving only $X_{d_0} \cap X_{d_1}$ occupied. By (D2) the robber must stay in $X_{\geq d_1}$. The cops now occupy the remaining vertices of X_{d_1} , forcing the robber into $X_{>d_1}$ and so on. Continuing in this way the robber will be eventually captured. The number of cops is limited by $\max_{d \in D} |X_d| = dgw(G) + 1$.

Unlike in the case of directed tree-width we can prove that the opposite is true as well:

THEOREM 4.3. *Let G be a directed graph which can be monotonely searched by $k + 1$ cops. Then G has DAG-width at most k .*

Proof. Let π be a monotone search strategy for $k + 1$ cops. It is not hard to see that we can always restrict ourselves to the case where at most one cop moves at a time. We will construct a DAG decomposition (D, \mathcal{X}) of width k as follows:

The nodes of our decomposition will be pairs (Y, C) , where $Y \in [V]^{<k+1}$ and C is a strongly connected component of $G \setminus Y$. Here Y stands for the set of vertices currently occupied by the cops and C is the strongly connected component of $G \setminus Y$ containing the robber.

Now fix a node $(Y, C) \in V(D)$, and let Y' be the next position of the cops given by the strategy π . Then for each strongly connected component C' of $G \setminus Y'$ s.t. C' is reachable from a vertex of C in the graph $G \setminus Y'$ we put $((Y, C), (Y', C')) \in E(D)$. Finally we put $X_{(Y,C)} = Y$ for every node $(Y, C) \in V(D)$.

Obviously the number of vertices in each node is at most $k + 1$. It remains to be checked that the properties (D1) to (D3) hold, which is not hard: (D1) is obvious, and (D2) together with (D3) hold since π is monotonous.

Note that the Theorem 4.3 provides us with an upper bound on the minimal number of nodes in a DAG decomposition. The bound is polynomial in n and exponential in k , which is in contrast to both tree and arboreal decomposition, where this bound is obviously linear.

It remains an open problem whether monotonous and general strategies are equivalent - it indeed looks quite likely. As a step in this direction, the next theorem provides us with a way of describing a winning strategy for a robber:

THEOREM 4.4. *A graph G cannot be searched by $< k$ cops iff there is a function σ mapping each $X \in [V(G)]^{<k}$ to a nonempty union $\sigma(X)$ of strongly connected components of $G \setminus X$ s.t. if $X \subseteq Y \in [V(G)]^{<k}$ then:*

1. $\forall S \in \sigma(X). \exists T \in \sigma(Y)$ s.t. there is a directed path from S to T in $G \setminus X$
2. $\forall S \in \sigma(Y). \exists T \in \sigma(X)$ s.t. there is a directed path from S to T in $G \setminus X$

Proof. If such a function σ exists, the robber can remain uncaptured by choosing the corresponding element of $\sigma(X)$ in each step. Conversely, suppose that $< k$ cops cannot search the graph. Then for each $X \in [V(G)]^{<k}$ let $\sigma(X)$ be a union of all strongly connected components of $G \setminus X$ s.t. the cop player can guarantee a win from those components. Then σ satisfies the theorem. (To keep the size of $\sigma(X)$ small we can always chose the greatest subset s.t. there is not a (directed) path between any two strongly connected components in this set.)

In [10] there is a similar definition of σ for the games on undirected graphs. In that paper it is also shown that

there is always a special σ called a *haven*, which assigns to X a single connected component (remember we are dealing with undirected graphs). However, in our case we really need $\sigma(X)$ to be a union of unconnected sets of vertices. Consider the graph G_X in Fig. 1, which cannot be searched by less than three cops. It is easy to see that $\sigma(\{b, c\}) = \{a\} \cup \{d\}$, since two cops can force the robber from a to d and back again.

5 Relationship Between DAG-width and Tree-width

Here we present the relationship between DAG-width and both tree-width and directed tree-width. As one could expect, DAG-width falls in between the two measures.

THEOREM 5.1. *Let G be a directed graph. Then $dgw(G) \leq tw(G)$. Moreover there is a graph G for which the inequality is sharp.*

Proof. Every tree decomposition of a graph G can be turned into a DAG decomposition of the same width by selecting one vertex as a root of the decomposition and orienting all edges away from this root. To see that this inequality can be sharp take G to be a directed clique of size n , i.e. a graph created by taking an (undirected) clique of n vertices and orienting all edges so they form a DAG (with a single source and single sink). Since cliques of size n have tree-width n the result follows.

The next lemma confirms that DAG-width is “correct” counterpart of tree-width on directed graphs. (Directed tree-width also enjoys this property.)

LEMMA 5.1. *Let G be an undirected graph of tree-width k . Then the directed graph G' created by replacing each edge of G with a pair of oppositely oriented edges has DAG-width k .*

Proof. By Theorem 5.1 (2) we already know that $dgw(G') \leq tw(G)$. Because $tw(G) = k$ there must be a haven σ in G of size $k + 1$. But this σ also satisfies the assumptions from Theorem 4.4 and therefore by Theorem 4.2 $dgw(G') > k$, which finishes the proof.

The relationship to directed tree-width is a little bit more complicated. An obvious approach is to use the following theorem:

THEOREM 5.2. ([4]) *Let G be a directed graph, and let $k > 0$ be an integer. Then either G has tree-width at most $3k - 2$, or it has a haven of order k .*

It remains to observe that a haven of order k in directed graph gives a winning strategy for a robber

against $k - 1$ cops in the (directed tree-width version of) cops-and-robber game, which easily translates to a winning strategy against the same number of cops in the DAG-width version of the game. Therefore directed tree-width and DAG-width are within a constant factor of each other. For an example of a graph G s.t. $dtw(G) < dgw(G)$ take the graph G_X in Fig. 1. It is easy to see that $dtw(G) = 1$ and $dgw(G) = 2$.

The natural question is whether we can convert a DAG decomposition into an arboreal decomposition of the same width. We will show how to do that for a subclass of DAG-decompositions.

DEFINITION 5.1. *A DAG decomposition (D, \mathcal{X}) of a graph G is called simple, if in addition to (D1) - (D3) it satisfies the following:*

(D4) $\forall v \in V(G)$ there exist a single $d \in V(D)$ s.t. $v \in X_d$ and if $v \in X_{d'}$ for $d'' \in V(D), d'' \neq d$ then $(d', d'') \in E(D) \implies d < d'$.

LEMMA 5.2. *Let (D, \mathcal{Y}) be a simple DAG decomposition of graph G of width k in normal form and d_0 its root. Then $(R, \mathcal{X}, \mathcal{W})$ computed by the Alg. 1 is an arboreal decomposition of G of the same width as (D, \mathcal{Y}) .*

Proof. Note that this algorithm is a standard DFS topological sort. We write $d \prec d'$ for $d, d' \in V(R)$ iff $\gamma[d] < \gamma[d']$. Then \prec is a linearisation of $<$ on D .

Algorithm 1: DAGtoTree

input : DAG decomposition (D, \mathcal{Y})
output: arboreal decomposition $(R, \mathcal{X}, \mathcal{W})$
 $i:=0$;
 $V(R):=V(D); E(R):=\emptyset$;
 $\mathcal{X}:=\emptyset; \mathcal{W}:=\bigcup_{d \in V(R)} W_d = \emptyset$;
 $W_{d_0}:=Y_{d_0}; \text{DFS}(d_0)$

Procedure DFS(d)

for each d' s.t. $e = (d, d') \in E(D)$ **do**
 if have not seen d' before then
 $W_{d'}:=Y_{d'} \setminus Y_d$;
 $E(R):=E(R) \cup \{e\}$;
 $X_e:=Y_d \cap Y_{d'}; \mathcal{X}:=\mathcal{X} \cup \{X_e\}$;
 DFS(d');
 $\gamma[d]:=i++$;

To begin \mathcal{W} is clearly a partition of $V(G)$ (the vertices left in each W_d are exactly those vertices of Y_d for which d is the root node). Moreover for $(d, d') \in E(R)$ the set $W_{d'}$ is empty only when $Y_{d'} \subseteq Y_d$, which is not possible since the DAG decomposition is in normal

form. We will prove (R2) by induction on $\gamma[d]$. In the rest of the proof let $e_d \in E(R)$ be the only edge of R with head d . We claim that (R2) holds for e_d .

For d with $\gamma[d] = 0$ we have that d is a leaf (i.e. vertex with no successors) in D . Then (R2) holds for e_d by construction and (D2). Similarly in the inductive step we are done if all the successors of d in D are also successors of d in R (using the induction hypothesis for these successors). For contradiction assume there is a cycle violating (R2). Therefore there must be some $v \in W_d$, $d' \in V(R)$ s.t. $(d, d') \in E(D)$, $(d, d') \notin E(R)$, and $w \in X_{>d}$ s.t. this cycle starts with the edge (v, w) .

Since the edge (d, d') was not included in $E(R)$, then d' must have been finished before d was ($d' \prec d$), and therefore $e_{d'}$ satisfies induction hypothesis. However by (D4) we must have $X_{e_{d'}} \subseteq X_e \cup W_d$ which completes the proof. Finally note that the width of $(R, \mathcal{X}, \mathcal{W})$ is at most k , since $W_d \cup \bigcup_{d \sim e} X_e \subseteq Y_d$.

For general DAG decompositions we run into technical difficulties. It seems that the wording of (R1) is too restrictive - the main problem being the requirement for the sets W_d to be non-empty. However if we drop this requirement, the algorithm above can be modified to work on general DAG decompositions.

6 Algorithms

The natural question to ask is how hard is to compute DAG-width of a given graph G . Unfortunately the situation is no better than for the case of tree-width. Also similarly to the case of tree-width, the situation is better if we fix k .

THEOREM 6.1. *Let G be a directed graph and k a natural number. The problem of determining whether the DAG-width of G is at most k is NP-complete.*

Proof. We know that the problem of determining tree-width of an undirected graph G is NP-complete. But by Lemma 5.1 we can reduce this problem to the problem of determining the DAG-width of G' , and the theorem follows.

THEOREM 6.2. *Let G be a directed graph and k a natural number. Then there is a algorithm which in time $\mathcal{O}(|G|^{\mathcal{O}(k)})$ decides whether the DAG-width of G is at most k , and if so, computes a DAG decomposition of this width.*

The main significance of hierarchical graph decompositions for computer science is that they allow us to solve many problems which are hard in general (e.g. NP-complete [1] or even PSPACE-complete [2]) in linear or polynomial time. In this section we give a generic

algorithm for solving problems on graphs of bounded DAG-width.

A general technique for solving problems on graphs with hierarchical decompositions can be described as: “computing tables of (characterisations of) partial solutions for nodes of the decomposition in bottom-up order”. I.e. in the case of DAG-width we would compute a table of partial solutions for graph $G|X_{>d}$ in the node d . However, since our decomposition is a DAG, we must be a bit more careful: Assume that $d \in V(D)$ has two successors d_1 and d_2 . To compute the table for d we need to combine tables for d_1 and d_2 . However $X_{>d_1}$ and $X_{>d_2}$ may not be disjoint, so only some of the partial solutions can be combined. Therefore some kind of “compatibility checking” is needed.

One thing we can do is to use the existing algorithms for graphs of bounded directed tree-width from [4]. In the rest of this section we will use simple DAG decompositions, the reason being two-fold: First the algorithms are easier to present. And second, the arboreal decompositions gained by running the Alg. 1 have nice properties like the one given by the following lemma. Note that this property is not necessarily true for arbitrary arboreal decompositions, as claimed in the original paper – see [5].

LEMMA 6.1. *Let G be a graph, (D, \mathcal{Y}) its simple DAG decomposition in normal form and $(R, \mathcal{X}, \mathcal{W})$ its arboreal decomposition obtained by running Alg. 1 on (D, \mathcal{Y}) . Then for each $d \in V(R)$ we can number its successors d_1, \dots, d_l in such a way that for $1 \leq i < j \leq l$ no edge has head in $W_{\geq d_i}$ and tail in $W_{\geq d_j}$.*

Proof. First note that \prec satisfies the following property P: Let d_1, d_2 be successors of d in R and w.l.o.g. assume that $d_1 \prec d_2$. Then $\forall d \in W_{\geq d_1} \forall d' \in W_{\geq d_2} . d \prec d'$. It is not hard to check that P holds, as it follows from the fact that Alg. 1 is depth-first search.

Now for every node $d \in V(R)$ we number its successors d_1, \dots, d_l in such a way that $d_1 \prec \dots \prec d_l$. Let $(v, w) \in E(G)$, $v \in W_d$, $w \in W_{d'}$. By (D2') either $d' \prec d$ (in R), or there is a path in D with source d and target d' . In the latter case by (D2') and the fact that \prec is a linearisation of $<$ we have $d' \prec d$. Together with P this concludes the proof.

Here is the general setting. (Due to space constraints we will not go into details here and point the reader to [4].) For the algorithm to run in polynomial time we require that for every integer k there is a real number α such that the two following properties are true about the tables of partial solutions (here called *itineraries*) we compute.

Axiom 1 Let G be a graph, $A, B \subseteq V(G)$ disjoint sets such that no edge of G has head in A and tail in B . Then the itinerary for $A \cup B$ can be computed from itineraries for A and B in time $\mathcal{O}((|A| + |B|)^\alpha)$.

Axiom 2 Let G be a graph, $A, B \subseteq V(G)$ disjoint sets such that A is Z -normal for some $Z \subseteq V(G)$ and $|B| \leq k$. Then the itinerary for $A \cup B$ can be computed from itineraries for A and B in time $\mathcal{O}((|A| + 1)^\alpha)$.

The construction of a polynomial algorithm for a given itinerary is obvious: We go through nodes in the order given by \prec . For a node d with multiple successors we compute first the information for $G|W'$, where $W' = \bigcup_{(d,d') \in E(R)} W_{>d'}$. This we can do by iterative application of Axiom 1, which is possible thanks to Lemma 6.1. Then we apply Axiom 2 to $G|W'$ and W_d .

In addition to the Lemma 6.1 above for the case of bounded DAG-width we also get easier proofs and better complexity estimates for the class of problems shown to be solvable in polynomial time (than having just bounded directed tree-width). These problems all are expressible in the terms of linkage. A *linkage* L in a graph G is a subgraph L s.t. every component of this graph is a directed path. Let $\sigma = \{(s_1, t_1), (s_2, t_2), \dots, (s_k, t_k)\}$ be a set of pairs of vertices of G . We say that subgraph L is σ -linkage if L consists precisely of k paths from s_i to t_i .

The following is an adaptation of (4.8) in [4]:

THEOREM 6.3. *For all fixed integers $k, w \geq 0$ there exists a polynomial-time algorithm to solve the following problem:*

INSTANCE: A graph G on n vertices, a DAG decomposition (D, \mathcal{X}) of G of width at most $w - 1$, a sequence of $2k$ vertices of G , and a set $M \subseteq \{1, 2, \dots, n\}$.

QUESTION: Does there exist a σ -linkage L in G s.t. $|V(L)| \in M$?

Note that this includes the Hamiltonian path/cycle problem, even cycle problem through specified vertex etc. Here we want to point out that thanks to using simple DAG decompositions we get an improvement in the complexity. While for general arboreal decompositions the estimate is $\mathcal{O}(n^{4(k+2w)+3})$, for simple DAG decompositions we get $\mathcal{O}(n^{4(k+w)+3})$. An open problem is whether we can do even better. For example take the Hamiltonian cycle problem. Note that for a node d the linkage must consist of at most w paths, all of them disjoint. By definition of DAG decompositions we also know that these paths must all end in vertices of X_d . So all such partial solutions can be described by vectors of length $|X_d| \leq w$, mentioning only the starting vertices of these paths, which suggests a complexity of $\mathcal{O}(n^{2w})$.

7 Conclusions

It seems that DAG-width is a useful connectivity measure for directed graphs. While being a bit more restrictive than directed tree-width, our measure seems to be general enough and the class of graphs of bounded DAG-width is very large. In comparison with directed tree-width it is much easier to reason about and hopefully allows for more effective algorithms. There is also a close connection to tree-width for undirected graphs.

There is much work left to do. First, there is the unsettled question whether monotonous and non-monotonous search strategies for cops are equivalent. We conjecture that a DAG-width version of Theorem 4.1 holds and we have done some work in this direction. Another problem is whether every DAG-decomposition can be converted into one of size linear in the size of G .

We have also started to examine (possibly more) effective algorithms on graphs of bounded DAG-width, other than the adapted algorithms for directed tree-width. A recent, not yet published result is that parity games can be solved in polynomial time on graphs of bounded DAG-width. (Thus improving the result for tree-width [6].)

Finally there are many questions already intensively studied for the notion of tree-width - a typical challenge being characterising graphs of bounded DAG-width in terms of graph minors.

References

- [1] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, 23(1):11–24, 1989.
- [2] H. Bodlaender. Complexity of path-forming games. *Theoretical Computer Science*, 110(1):215–245, 1993.
- [3] Hans L. Bodlaender. Treewidth: Algorithmic techniques and results. In *MFCS'97*, volume 1295 of *LNCS*, pages 19–36, 1997.
- [4] T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.
- [5] T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Addendum to "Directed Tree-Width". <http://www.math.gatech.edu/~thomas/PAP/diradd.pdf>, 2002.
- [6] J. Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In *CAV'03*, volume 2725 of *LNCS*, pages 80–92. Springer-Verlag, 2003.
- [7] B. Reed. Tree width and tangles, a new measure of connectivity and some applications. In R. Bailey, editor, *Surveys in Combinatorics 1997*, pages 87–162. Cambridge University Press, 1997.
- [8] B. Reed. Introducing directed tree width. *Electronic Notes in Discrete Mathematics*, 3:222–229, May 1999.

- [9] N. Robertson and P. D. Seymour. Graph Minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36:49–63, 1984.
- [10] P. D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993.