Digraph Measures: Kelly Decompositions, Games, and Orderings

Paul Hunter Stephan Kreutzer*†

Abstract

We consider various well-known, equivalent complexity measures for graphs such as elimination orderings, k-trees and cops and robber games and study their natural translations to digraphs. We show that on digraphs all these measures are also equivalent and induce a natural connectivity measure. We introduce a decomposition for digraphs and an associated width, Kelly-width, which is equivalent to the aforementioned measure. We demonstrate its usefulness by exhibiting a number of potential applications including polynomial-time algorithms for NP-complete problems on graphs of bounded Kelly-width, and complexity analysis of asymmetric matrix factorization. Finally, we compare the new width to other known decompositions of digraphs.

1 Introduction

An important and active field of algorithm theory is to identify natural classes of structures or graphs which are algorithmically well-behaved, i.e. on which efficient solutions to otherwise NP-complete problems can be found. A particularly rich source of tractable cases comes from graph structure theory in the form of graph decompositions and associated measures of structural complexity such as tree-width or rank-width. For instance, Courcelle's celebrated theorem [8] shows that every property of undirected graphs that can be formulated in monadic second-order logic can be decided in linear time on any class of graphs of bounded tree-width. This result immediately implies linear time algorithms for a huge class of problems on such graphs. Since then, hundreds of papers have been published describing efficient algorithms for graph problems on classes of graphs of bounded tree-width. (See e.g. [5] and references therein.) Similarly, efficient algorithms can sometimes be found for planar graphs [13, 2] or more general classes of graphs, for instance classes of graphs of bounded local tree-width [14], or graph classes excluding a minor (see e.g. [10] and references therein). Another interesting example are classes of graphs of bounded clique- or rank-width [9, 22, 21].

All the examples mentioned above are defined by imposing restrictions on the underlying undirected graph structure. However, there are many applications where the input structures – networks, state transition systems, dependency graphs as in database theory, or the arenas of combinatorial games such as parity games – are more naturally modelled as directed rather than undirected graphs. In these cases, the notion of tree-width is unsatisfactory as it does not take the direction of edges into account. This information loss may be crucial, as demonstrated by the problem of finding a Hamiltonian cycle in a digraph: an acyclic orientation of a grid has very high tree- or clique-width, but the Hamiltonian cycle problem on the digraph is trivial.

As a consequence, several authors have tried to generalise notions like tree- or path-width from undirected to directed graphs (See e.g. [23, 18, 3, 25, 20, 4]). In [18], Johnson, Robertson, Seymour, and Thomas concentrate on the connectivity aspect of tree-width, generalising this to strong connectivity in the directed case, to define directed tree-width. In the same paper, the authors give an algorithmic application of directed tree-width by showing that a number of NP-complete problems such as Hamiltonicity or the k-disjoint paths problems become tractable on graphs of small directed tree-width. Berwanger, Dawar, Hunter and Kreutzer [4] and, independently, Obdržálek [20] introduce the notion of DAG-width. DAG-width is a slightly weaker notion than directed tree-width, in the sense that more graphs have small directed tree-width than small DAG-width, but it has a cleaner characterisation in terms of cops and robber games and gives more control over the graph, as the guarding condition used is stricter. In [4], this has been used to show that the winner of a parity game – a form of combinatorial games played on digraphs – can be decided in polynomial time provided the game graph has bounded DAG-width. The analagous question remains open for graphs of bounded directed tree-width.

Both directed tree-decompositions and DAG-decompositions provide a natural and interesting connectivity measure for directed graphs. Both, however, also suffer from some difficulties. As Adler shows [1], directed tree-decompositions are not closed under even mild forms of directed minors, the corresponding games are neither cop- nor robber monotone, and there is no

*Paul Hunter, Stephan Kreutzer, Logic and Discrete Systems, Institute for Computer Science, Humboldt-University Berlin, {hunter, kreutzer}@informatik.hu-berlin.de

†Part of this work was carried out during the Logic and Algorithms programme at the Isaac Newton Institute, Cambridge.

precise characterisation of directed tree-width by these games (only up to a constant factor). Although this is not really a problem for algorithmic applications, it suggests that the notion of directed tree-width may not be as well-behaved as undirected tree-width. Furthermore, being a measure based on strong connectivity makes it difficult to develop algorithms outside of those provided in [18]. DAG-decompositions, on the other hand, suffer from the fact that the best upper bound for the size of DAG-decompositions of graphs of width k known so far is $\mathcal{O}(n^k)$. This is a significant problem, as the space consumption of algorithms is often more problematic than running time. Hence, it is not known whether deciding that a digraph has DAG-width at most k is in NP, contrary to the authors' claims. (It is NP-hard. This follows easily from the NP-completeness of the corresponding question for tree-width.)

Whereas for undirected graphs it is widely accepted that tree-width is the "right" notion, the problems described above suggest that more research is needed to decide what the "right" notion for digraphs is – if there is any. A natural way to search for practical generalisations of undirected tree-width is to look at useful equivalent characterisations of it and translate them to digraphs.

In this paper we consider three characterisations of tree-width: partial k-trees, elimination orders and a graph searching game in which an invisible robber attempts to avoid capture by a number of cops, subject to the restriction that he may only move if a cop is about to occupy his position. Partial k-trees are the historical forerunner of tree-width and are therefore associated with graph structure theory [24], elimination orders have found application in the analysis of symmetric matrix factorization, such as Cholesky decomposition [19], and graph searching problems have recently been used to explore and generate robust measures of graph complexity (see e.g. [11, 15]). We generalise all of these to directed graphs, resulting in partial k-DAGs, directed elimination orderings, and an inert robber game on digraphs. We show that all of these generalisations are equivalent on digraphs and are also equivalent to the width-measure associated to a new kind of decomposition we introduce. As the game is reminiscent of capturing hideout-based outlaws, we propose the name Kelly-decompositions, after the infamous Australian bushranger Ned Kelly. The fact that all these notions are equivalent on digraphs as they are on undirected graphs suggests that this might be a robust measure of complexity/connectivity of digraphs.

In addition to being equivalent to the natural generalisations of the above characterisations, we believe that Kelly-decompositions have many advantages over DAG-decompositions and directed tree-decompositions. Unlike the former, the size of these decompositions can be made linear in the size of the graph it decomposes. On the other hand, their structure and strict guarding condition make them suitable for constructing dynamic programming algorithms which can lead to polynomial-time algorithms for NP-complete problems on graphs of bounded Kelly-width. We also show how they are applicable to asymmetric matrix factorization by relating them to the elimination DAGs of [16].

The paper is organised as follows. In Section 3 we formally define elimination orderings, inert robber games, and partial k-DAGs and show the equivalence of the associated width measures. In Section 4, we introduce Kelly-decompositions and Kelly-width. In Section 5, we present applications: Algorithms for Hamiltonian cycle, weighted disjoint paths and parity games that all run in polynomial time on graphs of bounded Kelly-width, and details of the connection between Kelly-decompositions and asymmetric matrix factorization. Finally, we compare our new width measure to other known measures on digraphs, in particular to directed tree-width and DAG-width. Due to space restrictions, most of the proofs appear in the full version of the paper.

2 Preliminaries

We use standard graph theory notation. See e.g. [12]. Let $\mathcal{G}$ be a digraph. We write $V(\mathcal{G})$ for its set of vertices and $E(\mathcal{G})$ for its edge set. For $X \subseteq V(\mathcal{G})$ we write $\mathcal{G}[X]$ for the subgraph of $\mathcal{G}$ induced by X and $\mathcal{G} \setminus X$ for $\mathcal{G}[V(\mathcal{G}) \setminus X]$. If $X := \{v\}$ is a singleton set, we simply write $\mathcal{G} \setminus v$. Finally, we sometimes write $\mathcal{G}[v_1, \ldots, v_k]$ for $\mathcal{G}[\{v_1, \ldots, v_k\}]$. For every $v \in V(\mathcal{G})$ and $X \subseteq V(\mathcal{G})$ such that $v \notin X$ we write $\mathit{Reach}_{\mathcal{G}\setminus X}(v)$ for the set of vertices in $V(\mathcal{G}) \setminus X$ reachable from v by a directed walk in $\mathcal{G} \setminus X$. If $\mathcal{G}$ is a directed, acyclic graph (DAG), we write $\preceq_{\mathcal{G}}$ for the reflexive, transitive closure of the edge relation.

3 Elimination Orderings, Inert Robber Games, and Partial k-DAGs

In this section we formally define directed elimination orderings, inert robber games, and partial k-DAGs and show that the associated width-measures of digraphs are equivalent.

Our first definition extends the idea of vertex elimination to digraphs. Vertex elimination is the process of removing vertices from a graph but adding edges to preserve reachability. The complexity measure we are interested in is the maximum out-degree of eliminated vertices.

DEFINITION 3.1. *Let $\mathcal{G}$ be a digraph. An* (directed) elimination ordering $\lhd$ *is a linear ordering on $V(\mathcal{G})$.*

Given an elimination ordering $\lhd := (v_0, v_1, \ldots, v_{n-1})$ *of* $\mathcal{G}$*, we define:* $\mathcal{G}_0^{\lhd} := \mathcal{G}$*; and* $\mathcal{G}_{i+1}^{\lhd}$ *is obtained from* $\mathcal{G}_i^{\lhd}$ *by deleting* v_i *and adding new edges (if necessary)* (u, v) *if* $(u, v_i), (v_i, v) \in E(\mathcal{G}_i^{\lhd})$ *and* $u \neq v$*.* $\mathcal{G}_i^{\lhd}$ *is the* directed elimination graph at step i according to $\lhd$*. The* width *of an elimination ordering is the maximum over all* i *of the out-degree of* v_i *in* $\mathcal{G}_i^{\lhd}$*. For convenience we also define the* support of v_i with respect to $\lhd$ *as* $supp_{\lhd}(v_i) := \{v_j : (v_i, v_j) \in E(\mathcal{G}_i^{\lhd})\}$*. Note that the width of an elimination ordering* $\lhd$ *is the maximum cardinality of all supports.*

Immediately from the definitions, we have this simple lemma relating the support of an element in an elimination ordering to the set of vertices reachable from that node.

LEMMA 3.1. *Let* $\lhd$ *be a directed elimination ordering of a graph* $\mathcal{G}$ *and let* $v \in V(\mathcal{G})$*. Let* $R := \{u : v \lhd u\}$*. Then* $supp_{\lhd}(v) = \{u : v \lhd u$ *and there is* $v' \in Reach_{\mathcal{G} \setminus R}(v)$ *such that* $(v', u) \in E(\mathcal{G})\}$*.*

We proceed with defining inert robber games on digraphs. Intuitively, a robber occupies some vertex of a graph $\mathcal{G}$. k cops attempt to capture this robber by occupying the same vertex as the robber. The robber evades capture by being able to run from his position along any directed path which does not pass through a cop. Any number of cops can move anywhere on the graph but they do so by removing themselves completely from the graph and then announcing where they are moving. It is during this transition that the robber moves. In the inert robber game, the robber may only move if a cop is about to land on his current position, however he is not visible to the cops and he knows the cops' strategy in advance. More formally,

DEFINITION 3.2. (INERT ROBBER GAME) *The* (k-cop) inert robber game *on a digraph* $\mathcal{G}$ *is the set of all plays, where a* play *is a sequence* $(X_0, R_0), (X_1, R_1), \ldots (X_n, R_n)$*, such that* $(X_0, R_0) = (\varnothing, V(\mathcal{G}))$ *and for all i:* $X_i, R_i \subseteq V(\mathcal{G})$*;* $|X_i| \leq k$*; and*

$$R_{i+1} = \left(R_i \cup \bigcup_{v \in R_i \cap X_{i+1}} Reach_{\mathcal{G} \setminus (X_i \cap X_{i+1})}(v) \right) \setminus X_{i+1}.$$

Intuitively, the X_i *represent the cop locations, and the* R_i *represent the set of potential robber locations (also known as contaminated vertices). The sequence* $X_0, X_1, \ldots$ *is the* strategy *for the cops. Note that given a strategy we can reconstruct the play. A strategy* $X_0, X_1, \ldots, X_n$ *is* winning *if* $R_n = \varnothing$ *in the associated play. Finally, a strategy is* monotone *if* $R_i \supseteq R_{i+1}$ *for all* i *in the associated play.*

The last characterisation we consider is a generalisation of partial k-trees, called partial k-DAGs. k-trees can be viewed as a class of graphs generated by a generalisation of how one might construct a tree. In the same way, k-DAGs are a class of digraphs generated by a generalisation of how one might construct a DAG in a top-down manner.

DEFINITION 3.3. (PARTIAL GRAPH) *Given two digraphs* $\mathcal{G}$ *and* $\mathcal{H}$*, we say* $\mathcal{H}$ *is a partial graph of* $\mathcal{G}$ *if* $V(\mathcal{H}) = V(\mathcal{G})$ *and* $E(\mathcal{H}) \subseteq E(\mathcal{G})$*, i.e.* $\mathcal{G}$ *is a spanning subgraph of* $\mathcal{H}$*.*

DEFINITION 3.4. ((PARTIAL) k-DAG) *The class of* k*-DAGs is defined recursively as follows:*

- *A* k*-clique (that is, a complete digraph with* k *vertices) is a* k*-DAG.*
- *A* k*-DAG with* $n+1$ *vertices can be constructed from a* k*-DAG* $\mathcal{H}$ *with* n *vertices by adding a vertex* v *and edges satisfying the following:*
  - *At most* k *edges from* v *to* $\mathcal{H}$
  - *If* X *is the set of endpoints of the edges added in the previous subcondition, an edge from* $u \in V(\mathcal{H})$ *to* v *if* $(u, w) \in E(\mathcal{H})$ *for all* $w \in X \setminus \{u\}$*. Note that if* $X = \varnothing$*, this condition is true for all* $u \in V(\mathcal{H})$*.*

A partial k-DAG *is a partial graph of a* k*-DAG.*

The second condition on the edges provides a method to add as many edges as possible going to the new vertex without introducing cycles. Note that this definition generalises k-trees, for if the vertices (X) adjacent to the new vertex (v) form a clique, we will add edges back from X to v, effectively creating undirected edges between v and X (and possibly some additional edges from $\mathcal{H} \setminus X$ to v). Note that a partial 0-DAG is a DAG.

Our main result of this section is that the three measures introduced are equivalent on digraphs.

THEOREM 3.1. *Let* $\mathcal{G}$ *be a digraph. The following are equivalent:*

1. $\mathcal{G}$ *has a directed elimination ordering of width* $\leq k$*.*
2. $k+1$ *cops have a monotone winning strategy to capture an inert robber.*
3. $\mathcal{G}$ *is a partial* k*-DAG.*

It follows from this theorem that the minimal width over all directed elimination orderings of $\mathcal{G}$ and the minimal number of cops required to capture an inert robber (less one) coincide, and this class of digraphs is characterised by partial k-DAGs. This leads to the following definition:

DEFINITION 3.5. (ELIMINATION WIDTH) *Let* $\mathcal{G}$ *be a digraph. The* (directed) elimination width *of* $\mathcal{G}$ *is the minimal width over all directed elimination orderings of* $\mathcal{G}$*.*

4 Decompositions

With a robust measure for digraph complexity defined, we now turn to the problem of finding a closely related digraph decomposition. The decomposition we introduce is a partition of the vertices, arranged as a directed acyclic graph, together with sets of vertices which guard against paths in the graph that do not respect this arrangement. We have an additional restriction to avoid trivial decompositions – vertices in the guard sets must appear either to the left or earlier in the decomposition. More precisely,

DEFINITION 4.1. (GUARDING) *Let $\mathcal{G}$ be a digraph. We say $W \subseteq V(\mathcal{G})$* guards *$X \subseteq V(\mathcal{G})$ if $W \cap X = \varnothing$ and for all $(u, v) \in E(\mathcal{G})$ with $u \in X$, we have $v \in X \cup W$.*

DEFINITION 4.2. (KELLY-DECOMPOSITIONS) *A* Kelly-decomposition *of a digraph $\mathcal{G}$ is a triple $\mathfrak{D} := (\mathcal{D}, (B_t)_{t\in V(\mathcal{D})}, (W_t)_{t\in V(\mathcal{D})})$ so that*

- *$\mathcal{D}$ is a DAG and $(B_t)_{t\in V(\mathcal{D})}$ partitions $V(\mathcal{G})$,*
- *for all $t \in V(\mathcal{D})$, $W_t \subseteq V(\mathcal{G})$ guards $\mathcal{B}_t^{\downarrow} := \bigcup_{t' \succeq_{\mathcal{D}} t} B_{t'}$, and*
- *for all $s \in V(\mathcal{D})$ there is a linear order on its children $t_1, \dots, t_p$ so that for all $1 \le i \le p$, $W_{t_i} \subseteq B_s \cup W_s \cup \bigcup_{j<i} \mathcal{B}_{t_j}^{\downarrow}$. Similarly, there is a linear order on the roots such that $W_{r_i} \subseteq \bigcup_{j<i} \mathcal{B}_{r_j}^{\downarrow}$.*

The width *of $\mathfrak{D}$ is $\max\{|B_t \cup W_t| : t \in V(\mathcal{D})\}$. The* Kelly-width *of $\mathcal{G}$ is the minimal width of any of its Kelly-decompositions.*

Our main result of this section is that Kelly-decompositions do in fact correspond with the complexity measure defined at the end of the previous section.

THEOREM 4.1. *$\mathcal{G}$ has directed elimination width $\le k$ if, and only if, $\mathcal{G}$ has Kelly-width $\le k + 1$.*

The proof of Theorem 4.1 is constructive in that given an elimination ordering of width k it constructs a Kelly-decomposition of width $k + 1$, and conversely. In fact, the proof establishes a slightly stronger statement.

COROLLARY 4.1. *Every digraph $\mathcal{G}$ of Kelly-width k has a Kelly-decomposition $(\mathcal{D}, (B_t)_{t\in V(\mathcal{D})}, (W_t)_{t\in V(\mathcal{D})})$ of width k such that for all $t \in V(\mathcal{D})$:*

- *$|B_t| = 1$,*
- *W_t is the minimal set which guards $\mathcal{B}_t^{\downarrow}$, and*
- *every vertex $v \in \mathcal{B}_t^{\downarrow}$ is reachable in $\mathcal{G} \setminus W_t$ from the unique $w \in B_t$.*

Further, if $\mathcal{G}$ is strongly connected, then $\mathcal{D}$ has only one root.

We call such a decomposition *special*.

5 Applications

5.1 Computing Kelly-decompositions In this section we mention several algorithms for computing Kelly-width and Kelly-decompositions. The proofs of Theorems 3.1 and 4.1 show that Kelly-decompositions can easily (i.e. polynomial time) be constructed from directed elimination orderings or monotone winning strategies, so we concern ourselves with the problem of finding any of the equivalent characterisations.

In a recent paper [6] Bodlaender et al. study exact algorithms for computing the (undirected) tree-width of a graph. Their algorithms are based on dynamic programming to compute an elimination ordering of the graph. In the same paper, the authors remark on actual experiments with these algorithms. Using some preprocessing techniques, the dynamic programming approach seems to perform reasonably well (in particular for not too large instances). The algorithms translate to directed elimination orderings and can therefore be used to compute Kelly-width. Hence, we get the following theorem.

THEOREM 5.1. *The Kelly-width of a graph with n vertices can be determined in time $\mathcal{O}^*(2^n)$ and space $\mathcal{O}^*(2^n)$, or in time $\mathcal{O}^*(4^n)$ and polynomial space.*

Here, $\mathcal{O}^*(f(n))$ means that polynomial factors are suppressed. For a given k, the problem whether a digraph $\mathcal{G}$ has Kelly-width $\le k$ is decided in exponential time with the above algorithms. Further, as the size of Kelly-decompositions is linear in the size of the graphs they decompose, the problem, given a graph $\mathcal{G}$ and $k \in \mathbb{N}$, to decide if $\mathcal{G}$ has Kelly-width at most k, is in NP. We can simply guess a Kelly-decomposition. As the minimization problem is NP-complete (it generalises the NP-complete problem of deciding the tree-width of an undirected graph), we cannot expect polynomial time algorithms to exist. It seems plausible though that, as in the case of DAG-width, studying strategies in the inert robber game will lead to a polynomial time algorithm when k is fixed. This is part of ongoing research.

5.2 Algorithms on graphs of small Kelly-width In this section we present algorithmic applications of the decomposition introduced above, including a general scheme that can be used to construct algorithms based on Kelly-decompositions. We assume that a Kelly-decomposition (or even an elimination ordering) has been provided or pre-computed. We give two example algorithms based on this which run in polynomial time on graphs of bounded Kelly-width. The first is an algorithm for the NP-complete optimization problem of computing disjoint paths of minimal weight in weighted graphs. The second is an algorithm to compute the winner of certain forms of combinatorial games. We conclude the section with remarks about how these algorithms compare with similar algorithms previously presented in [18] and [4].

Algorithms using Kelly-decompositions often follow a common pattern. Similar to algorithms on graphs of small tree-width, the algorithms start with computing a special Kelly-decomposition $(\mathcal{D}, (B_t)_{t\in V(\mathcal{D})}, (W_t)_{t\in V(\mathcal{D})})$ and then work bottom up to compute for each node $t \in V(\mathcal{D})$ a data set containing information on the set $\mathcal{B}_t^{\downarrow} := \bigcup_{t'\succeq t} B_t$. The general pattern is therefore described by the following steps (after the special Kelly-decomposition has been computed):

Leaves: Compute the data set for all leaves.

Combine: If $t \in V(\mathcal{D})$ is an inner node with children $t_1, \dots, t_p$ ordered by the ordering guaranteed by the Kelly-decomposition (we observe that such an ordering can be computed easily with a greedy algorithm), combine the data sets computed for $\mathcal{B}_{t_1}^{\downarrow}, \dots, \mathcal{B}_{t_p}^{\downarrow}$ to a data set for the union $\bigcup_{1\le i\le p} \mathcal{B}_{t_i}^{\downarrow}$.

Update: Update the data set computed in the previous step so that the new vertex u with $B_t = \{u\}$ is taken into account. Usually, the vertex u will have been part of at least some guard sets W_{t_i}. As $u \notin W_t$, it can now be used freely.

Expand: Finally, expand the data set to include guards in $W_t \setminus \bigcup_i W_{t_i}$ and also paths etc. starting at u.

We illustrate this pattern by presenting an algorithm for computing a Hamiltonian-cycle of minimal weight in a weighted digraph. We explain below how this algorithm extends to the much more general problem of finding disjoint paths of minimal weight.

5.2.1 Weighted Hamiltonian Cycle and Disjoint Paths.

A weighted digraph is a pair $(\mathcal{G}, \omega)$ where $\mathcal{G}$ is a digraph and $\omega : V(\mathcal{G}) \to \mathbb{R}$ is a weight function. The Kelly-width of $(\mathcal{G}, \omega)$ is the Kelly-width of $\mathcal{G}$.

THEOREM 5.2. *For any k, given a weighted digraph $(\mathcal{G}, \omega)$ and a Kelly-decomposition $(\mathcal{D}, (B_t)_{t\in V(\mathcal{D})}, (W_t)_{t\in V(\mathcal{D})})$ of $\mathcal{G}$ of width $\le k$, there exists a polynomial time algorithm which computes a Hamilton-cycle of $(\mathcal{G}, \omega)$ of minimal weight or determines that $\mathcal{G}$ is not Hamiltonian.*

Here, the weight of a Hamilton-cycle is the sum of the weights of the edges occurring on the cycle. To prove the theorem we first need some notation. Given a weighted digraph $(\mathcal{G}, \omega) \in \mathcal{C}$ and tuple $\boldsymbol{s} := \{(s_1, t_1), \dots, (s_r, t_r)\}$ of pairs of vertices, an *$\boldsymbol{s}$-linkage* is a sequence $\mathcal{P} := (P_1, \dots, P_r)$ of pairwise inner vertex disjoint paths so that P_i links s_i to t_i. The *order* of $\mathcal{P}$ is the order of $\mathcal{G}[\bigcup_i P_i]$. The *weight* of $\mathcal{P}$ is the sum of the weights of edges in P_i. Now, let $(\mathcal{G}, \omega)$ and $\boldsymbol{s}$ be given. For a set $U \subseteq V(\mathcal{G})$ and its guarding set W we write $\textsc{Link}(U, W)$ for the set of all tuples $((u_1, v_1), \dots, (u_r, v_r), l, w)$ such that a) $r \le k$, where k is the Kelly-width of $\mathcal{G}$, $u_i \in U$, $v_i \in W$, b) there are pairwise vertex disjoint paths $P_1, \dots, P_r$ in $U \cup W$ with all inner vertices in U so that P_i links u_i to v_i, c) the order of $(P_1, \dots, P_r)$ is l, and d) w is the minimal weight of any such sequence of paths of order l. For $t \in V(\mathcal{D})$ let $\mathcal{B}_t^{\downarrow} := \bigcup_{t'\succeq t} B_{t'}$ and define $\textsc{Link}(t)$ as $\textsc{Link}(\mathcal{B}_t^{\downarrow}, W_t)$. Now, beginning from the leaves, we carry out the four steps described above and compute for each node $t \in V(\mathcal{D})$ the set $\textsc{Link}(t)$ as follows. Once we have computed $\textsc{Link}(t)$ for all nodes, the theorem follows easily by examining linkages of order $|V(\mathcal{G})|$ during the *Combine* step at the (unique) root of $\mathcal{D}$.

Leaves: Clearly, for a leaf t, the set $\textsc{Link}(t)$ can be computed in constant time.

Now let t be an inner vertex and let $t_1, \dots, t_p$ be the children of t ordered according to the ordering guaranteed by the Kelly-decomposition. To compute $\textsc{Link}(t)$ we perform three steps.

Combine: In the first step we combine the sets $\textsc{Link}(t_i)$ to obtain $\textsc{Link}(\bigcup_i \mathcal{B}_{t_i}^{\downarrow}, \bigcup_i W_{t_i})$. Let $B_i := \bigcup_{j\le i} \mathcal{B}_{t_j}^{\downarrow}$ and $W_i := \bigcup_{j\le i} W_{t_j} \setminus B_{i-1}$. The sets $\textsc{Link}(B_i, W_i)$ are computed by induction on i. The crucial observation is that, by definition of Kelly-decompositions, there are no edges starting in B_{i-1} and ending in $\mathcal{B}_{t_i}^{\downarrow} \setminus B_{i-1}$. Hence, a path between $u \in \mathcal{B}_{t_i}^{\downarrow}$ and a guard either stays in $\mathcal{B}_{t_i}^{\downarrow}$, in which case we can read its existence from $\textsc{Link}(t_i)$, or it crosses over to B_{i-1}, in which case the first vertex in $B_{i-1} \setminus \mathcal{B}_{t_i}^{\downarrow}$ is in W_{t_i}. The first part of such paths can be computed from $\textsc{Link}(t_i)$ and second part, the part in B_{i-1}, can be computed from $\textsc{Link}(B_{i-1}, W_{i-1})$.

In the next two steps we compute the set $\textsc{Link}(t)$. Let $B_t = \{u\}$. Note that $W_p \subseteq W_t \cup B_t$ and $u \notin W_t$. Further, if $w \in W_t \setminus W_p$ then $(u, w) \in E(\mathcal{G})$ and there is no $v \in B_p$ with $(v, w) \in E(\mathcal{G})$. Now, consider a tuple $\boldsymbol{s} := ((u_1, v_1), \dots, (u_r, v_r))$ with $u_i \in \mathcal{B}_t^{\downarrow}$ and $v_i \in W_t$. Clearly, as $u \notin W_t$, $v_i \ne u$ for all i. There can only be an $\boldsymbol{s}$-linkage in $\mathcal{B}_t^{\downarrow}$ in one of the following cases.

Case 1: $u_i \in B_p$ and $v_i \in W_p \setminus \{u\}$ for all i.

Case 2: $u_i \in B_p$ for all i and there is at least one $v_i \in W_t \setminus W_p$.

Case 3: $u_i = u$ for some i.

The first case concerns tuples which have already been considered in the *Combine* step but now may have additional linkages containing u as an inner vertex. In this sense, we are merely updating information for tuples we have already processed. Hence, this case is dealt with in the *Update* step. The last two cases concern new tuples which have not been considered before. These cases are treated in the *Expand* step. The full details of these steps can be found in the full version of the paper.

An analysis of the algorithm shows that the *Update* and *Expand* steps can be implemented to run in time $\mathcal{O}(n^{2k}+1)$. Hence, as $|V(\mathcal{D})| = |V(\mathcal{G})|$, the overall running time of the algorithm is $\mathcal{O}(n^{2k+2})$. This slightly improves the running time of the Hamilton-cycle algorithm given in [18] for digraphs (without a weight function) of small directed tree-width.

The algorithm introduced above can easily be extended to solve the following, more general problem. The *weighted w-linkage problem* is the problem, given a weighted digraph $(\mathcal{G}, \omega)$, a tuple $\boldsymbol{s} := ((s_1,t_1),\ldots,(s_w,t_w))$, and a set $M \subseteq \{1,\ldots,|V(\mathcal{G})|\}$, to compute for each $l \in M$ an $\boldsymbol{s}$-linkage of order l of minimal weight (among all $\boldsymbol{s}$-linkages of order l).

THEOREM 5.3. *For every $w, k \in \mathbb{N}$, given a weighted digraph and a Kelly-decomposition of width $\leq k$, the weighted w-linkage problem can be solved in polynomial time.*

5.2.2 Parity Games.

Another example for an algorithm on graphs of bounded Kelly-width is an algorithm for solving parity games on game arenas of small Kelly-width. Parity games are a form of combinatorial games played on digraphs with many applications in the area of verification. See [17] for a definition. It is well known that deciding the winner of a parity game is in NP $\cap$ co-NP and it is a longstanding open problem if the problem is in P. In [4], Berwanger et al. describe an algorithm for computing the winner of a parity game of bounded DAG-width. This algorithm can be translated to arenas of small Kelly-width and, in some sense, becomes more transparent. Due to lack of space, the algorithm is given in the full version of the paper.

THEOREM 5.4. *For any k, given an arena $\mathcal{A}$ of a parity game and a Kelly-decomposition of $\mathcal{A}$ of width $\leq k$, the winning region of $\mathcal{A}$ can be computed in polynomial time.*

5.2.3 Remarks.

In the following section we show that the class of graphs of bounded Kelly-width is (strictly) smaller than the class of graphs of bounded directed tree-width. Consequently, the algorithms presented in [18] can be used on graphs of bounded Kelly-width, including the disjoint paths algorithm. This raises the question, what advantages does our algorithm enjoy over the one for directed tree-width? The first and obvious difference is that our algorithm computes a Hamilton-cycle of minimal weight. However, the main technical difference is the role the guards play in the algorithms. In the algorithm presented above, the guards W_t of a node t play an active role: We only consider paths from vertices $u \in \mathcal{B}_t^{\downarrow}$ to guards $v \in W_t$. In a directed tree-decomposition, a set $S \subseteq V(\mathcal{G})$ does not uniquely define its guards and these guards may only be reachable from S by a path that involves other vertices outside of S. Consequently, the guards only play an indirect role in the algorithm on directed tree-decompositions in that they give a bound on the size of tuples that have to be considered. Although this is enough for algorithms computing disjoint paths, Hamilton-cycles and similar problems, this forms a significant issue for other types of problems. An example of this is in the presented parity game algorithm, which, so far, has resisted attempts to translate it to directed tree-decompositions. Hence, if a problem requires to compute more complicated data structures than paths between vertices, Kelly-decompositions may be much easier to work with than directed tree-decompositions.

As for DAG-decompositions, it is their space consumption that forms a significant problem. Although our algorithm for parity games is similar to that of [4], ours requires only storing at most a linear number of data structures. Until the $\mathcal{O}(n^k)$ bound on the size of DAG-decompositions is reduced, such dynamic programming algorithms are only feasible for small values of k.

Finally, we believe that the presentation of algorithms on Kelly-decompositions is simpler and more understandable than on directed tree-decompositions or DAG-decompositions, again for the reasons that a) there is a strict separation of guards and vertices in the sets $\mathcal{B}_t^{\downarrow}$ (which is foreign to DAG-decompositions) and b) that the guards of a set $S \subseteq V(\mathcal{G})$ are uniquely defined and can therefore be used in more ways.

5.3 Asymmetric matrix factorization

The use of elimination orders and elimination trees to investigate symmetric matrix factorizations is well documented (see e.g. [19]). For example, the height of an elimination tree gives the parallel time required to factor a matrix [7]. In [16], Gilbert and Liu introduced a generalisation of elimination trees, called elimination DAGs, which can be similarly used to analyse factorizations in the asymmetric case. Kelly-decompositions are closely related to these structures, as illustrated by the following theorem.

DEFINITION 5.1. *Let $M = (a_{ij})$ be a square $n \times n$ matrix. We define G_M as the directed graph with $V(G_M) = \{v_1,\ldots,v_n\}$, and for $i \neq j$, $(v_i, v_j) \in E(G_M)$ if, and only if, $a_{ij} \neq 0$. We define $\lhd_M := (v_1,\ldots,v_n)$.*

THEOREM 5.5. *Let M be a square matrix that can be decomposed as $M = LU$ without pivoting. Let $(\mathcal{D}, (B_t)_{t\in V(\mathcal{D})}, (W_t)_{t\in V(\mathcal{D})})$ be the Kelly-decomposition of G_M obtained by applying the proof of Theorem 4.1 with elimination order $\lhd_M$. Then*

(a) $(\mathcal{D}, (B_t)_{t\in V(\mathcal{D})})$ *is equivalent to the lower elimination DAG (as defined in [16]), and*

(b) $G_U = (V(G_M), \{(v, w) : w \in W_v\})$*, which implies the upper elimination DAG is equivalent to the transitive reduction of the relation* $\{(v, w) : w \in W_v\}$.

We can use the results of [16] to make the following observation when we construct Kelly-decompositions on undirected graphs.

COROLLARY 5.1. *Let $\mathcal{G}$ be an undirected graph, $\lhd$ an elimination order on $\mathcal{G}$ and $(\mathcal{D}, (B_t)_{t\in V(\mathcal{D})}, (W_t)_{t\in V(\mathcal{D})})$ the Kelly-decomposition of $\mathcal{G}$ (considered as a bidirected graph) obtained by applying the proof of Theorem 4.1 with elimination order $\lhd$. Then $\mathcal{D}$ is a tree, and more precisely, $(\mathcal{D}, (B_t)_{t\in V(\mathcal{D})})$ is equivalent to the elimination tree associated with the (undirected) elimination order $\lhd$.*

6 Is it better to be invisible but lazy or visible and eager?

In this section we use graph searching games to compare Kelly-width to DAG-width and directed tree-width. In the undirected case, all games require the same number of searchers, however we show that in the directed case there are graphs on which all three measures differ by an arbitrary amount. Our results do imply that Kelly-width bounds directed tree-width within a constant factor, but the converse fails as there are classes of graphs of bounded directed tree-width and unbounded Kelly-width. We also provide evidence to suggest that Kelly-width and DAG-width are within a constant factor of each other. We begin by introducing the games associated with DAG-width and directed tree-width (see [4, 20, 18] for formal definitions).

DEFINITION 6.1. (VISIBLE ROBBER GAME) *The* visible robber game *is played as the inert robber game except that the robber's position is always known to the cops and the robber is free to move during a cop transition irrespective of where the cops intend to move (however, he still cannot run through a stationary cop). The* strong visible robber game *adds the further restriction that the robber can only move in the same strongly connected component (of the graph with the stationary cops' locations removed). A* strategy *for the cops is a function that, given the current locations of the cops and the robber, indicates the next location of the cops. A strategy is* winning *if it captures the robber, and it is* monotone *if the set of vertices which the robber can reach is non-increasing.*

The following theorem summarises the results of [4, 20, 18]:

THEOREM 6.1. *Let $\mathcal{G}$ be a digraph.*

1. *$\mathcal{G}$ has* DAG*-width k if, and only if, k cops have a monotone winning strategy in the visible robber game on $\mathcal{G}$.*
2. *$\mathcal{G}$ has directed tree-width $\leq 3k+1$ or k cops do not have a winning strategy in the strong visible robber game on $\mathcal{G}$.*

Our first result shows that a monotone winning strategy in the inert robber game can be translated to a (not necessarily monotone) winning strategy in the visible robber game.

THEOREM 6.2. *If k cops can catch an inert robber with a robber-monotone strategy, then $2k - 1$ cops can catch a mobile, visible robber.*

One consequence of this theorem is that Kelly-width bounds directed tree-width by a constant factor.

COROLLARY 6.1. *If $\mathcal{G}$ has Kelly-width $\leq k$ then $\mathcal{G}$ has directed tree-width $\leq 6k - 2$.*

Since it is not known whether monotone strategies are sufficient in the visible robber game, we cannot obtain a similar bound for DAG-width. We can, however, ask whether we can improve the bound, i.e. assuming that k cops have a robber-monotone winning strategy against an invisible, inert robber can we define a winning strategy for less than $2k - 1$ cops in the visible robber game? Although it might be possible to improve the result, the next theorem shows that we cannot do better than with $\frac{4}{3}k$ cops.

THEOREM 6.3. *For every $k \in \mathbb{N}$, there is a graph such that $3k$ cops have a robber-monotone winning strategy in the inert robber game but no fewer than $4k$ cops can catch a mobile visible robber.*

Proof. Consider the graph $\mathcal{G}$ with vertex set $\{v_1, \ldots, v_6\}$ so that $\{v_2, v_5, v_6\}$ as well as $\{v_3, v_5, v_6\}$ and $\{v_4, v_5, v_6\}$ form 3-cliques and there are edges $(v_4, v_3), (v_3, v_2), (v_1, v_2), (v_1, v_3), (v_3, v_1)$. It is easy to see that on $\mathcal{G}$, 3 cops do not have a (non-monotone winning) strategy to catch a visible robber, however 4 cops do. On the other hand, 3 cops suffice to capture an invisible, inert robber with a robber-monotone strategy. The result follows by taking the lexicographic product of this graph with the complete graph on k vertices. See the full version of the paper for details. □

In fact, 4 cops can capture a visible robber with a monotone strategy on the graph in the previous proof, giving us the following:

COROLLARY 6.2. *For all $k \geq 1$ there exists graphs of* DAG-*width $4k$ and Kelly-width $3k$.*

Despite this $\frac{4}{3}$ bound, for graphs of small Kelly-width we can do better.

THEOREM 6.4. *For $k = 1$ or 2, if $\mathcal{G}$ has Kelly-width k, $\mathcal{G}$ has DAG-width k.*

We now turn to the converse problem, what can be said about the Kelly-width of graphs given their directed tree-width or DAG-width? First, we consider the binary tree with back-edges example in [4], where it was shown this class of graphs has bounded directed tree-width but unbounded DAG-width. It is readily shown that this class of graphs also has unbounded Kelly-width.

THEOREM 6.5. *There exists classes of digraphs with bounded directed tree-width and unbounded Kelly-width.*

Our final result is a step towards relating Kelly-width to DAG-width by showing how to translate a monotone strategy in the visible robber game to a (not necessarily monotone) strategy in the inert robber game.

THEOREM 6.6. *If $\mathcal{G}$ has* DAG-*width $\leq k$, then k cops have a winning strategy in the inert robber game.*

Proof. Given a DAG-decomposition $(\mathcal{D}, (X_d)_{d \in V(\mathcal{D})})$ of $\mathcal{G}$ of width k, the strategy for k cops against an invisible, inert robber is to follow a depth-first search on the decomposition. $\square$

Again we observe that it is unknown if monotone strategies suffice in the inert robber game, so this result does not allow us to compare Kelly-width and DAG-width. However, we strongly believe that monotone strategies suffice in both the inert robber game and the visible robber game, giving us the following conjecture:

> **Conjecture.** The Kelly-width and DAG-width of a graph lie within constant factors of one another.

References

[1] I. Adler. Directed tree-width examples. To appear in *Journal of Combinatorial Theory, Series B*.

[2] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.

[3] J. Barát. Directed path-width and monotonicity in digraph searching. To appear in *Graphs and Combinatorics*.

[4] D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. DAG-width and parity games. In *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 524–536, 2006.

[5] H. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 19–36, 1997.

[6] H. Bodlaender, F. Fomin, A. Koster, D. Kratsch, and D. Thilikos. On exact algorithms for treewidth. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, 2006.

[7] H. Bodlaender, J. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995.

[8] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwan, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 193–242. 1990.

[9] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 1–3:77–114, 2000.

[10] E. Demaine, M. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 637–646, 2005.

[11] N. Dendris, L. Kirousis, and D. Thilikos. Fugitive-search games on graphs and related parameters. *Theoretical Computer Science*, 172(1-2):233–254, 1997.

[12] R. Diestel. *Graph Theory*. Springer, 3rd edition, 2005.

[13] F. Dorn, E. Penninkx, H. Bodlaender, and F.Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA)*, pages 95–106, 2005.

[14] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *Journal of Graph Algorithms and Applications*, 3(3):1–27, 1999.

[15] F. V. Fomin, P. Heggernes, and J. A. Telle. Graph searching, elimination trees, and a generalization of bandwidth. *Algorithmica*, 41(2):73–87, 2004.

[16] J. Gilbert and J. Liu. Elimination structures for unsymmetric sparse LU factors. *SIAM Journal of Matrix Analysis and Applications*, 14:334–352, 1993.

[17] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.

[18] T. Johnson, N. Robertson, P. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

[19] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM Journal of Matrix Analysis and Applications*, 11(1):134–172, 1990.

[20] J. Obdržálek. DAG-width: connectivity measure for directed graphs. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 814–821, 2006.

[21] S.-I. Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B*, 95:79 – 100, 2005.

[22] S.-I. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 2006. to appear.

[23] B. Reed. Introducing directed tree width. In *6th Twente Workshop on Graphs and Combinatorial Optimization*, volume 3 of *Electronic Notes in Discrete Mathematics*. Elsevier, 1999.

[24] D. J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32:597–609, 1970.

[25] M. Safari. D-width: A more natural measure for directed tree width. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 745–756, 2005.