On Digraph Width Measures in Parameterized Algorithmics

(extended abstract)

Robert Ganian[1], Petr Hliněný[1], Joachim Kneis[2], Alexander Langer[2], Jan Obdržálek[1], and Peter Rossmanith[2]

[1] Faculty of Informatics, Masaryk University, Brno, Czech Republic
{xganian1,hlineny,obdrzalek}@fi.muni.cz
[2] Theoretical Computer Science, RWTH Aachen University, Germany
{kneis,langer,rossmani}@cs.rwth-aachen.de

Abstract. In contrast to undirected width measures (such as tree-width or clique-width), which have provided many important algorithmic applications, analogous measures for digraphs such as DAG-width or Kelly-width do not seem so successful. Several recent papers, e.g. those of Kreutzer–Ordyniak, Dankelmann–Gutin–Kim, or Lampis–Kaouri–Mitsou, have given some evidence for this. We support this direction by showing that many quite different problems remain hard even on graph classes that are restricted very beyond simply having small DAG-width. To this end, we introduce new measures K-width and DAG-depth. On the positive side, we also note that taking Kanté's directed generalization of rank-width as a parameter makes many problems fixed parameter tractable.

1 Introduction

The very successful concept of graph *tree-width* was introduced in the context of the Graph Minors project by Robertson and Seymour [RS86,RS91], and it turned out to be very useful for efficiently solving graph problems. Tree-width is a property of *undirected graphs*. In this paper we will be interested in *directed graphs* or *digraphs*.

Naturally, a width measure specifically tailored to digraphs with all the nice properties of tree-width would be tremendously useful. The properties of such a measure should include at least the following:

i) The width measure is small on many interesting instances.
ii) Many hard problems become easy if the width measure is bounded.

Obviously, there is a conflict between these goals, and consequently we can expect some trade-off. On the search for such a digraph measure, several suggestions were made, starting with directed tree-width [JRST01], and being complemented recently with several new approaches including DAG-width [Obd06,BDHK06], Kelly-width [HK08], entanglement [BG04], D-width [Saf05], directed path-width [Bar06] (defined by Reed, Seymour, and Thomas), and — although quite different — bi-rank-width [Kan08] (see Section 2).

Some positive results were encouraging: The Hamiltonian path problem can be solved in polynomial time (XP) if the directed tree width, the DAG-width, or the Kelly-width are bounded by a constant [JRST01]. More recently, it has been shown that parity games can be solved in polynomial time on digraphs of bounded DAG-width [BDHK06] and Kelly-width [HK08].

Are more results just waiting around the corner and do we just have to wait until we get more familiar with these digraph measures? It is the aim of this paper to answer this question, at least partially.

Unfortunately, as encouraging as the first positive results are, there is also the negative side. Hamiltonian path is W[2]-hard on digraphs of bounded DAG-width [LKM08], and some other natural problems even remain NP-hard on digraphs of low widths [KO08,DGK08,LKM08]. One of the main goals of this paper is to show that not only many problems are hard on DAGs, but rather that they remain hard even if we very severely further restrict the graphs structure.

We introduce two digraph measures for this purpose: K-width and DAG-depth. While K-width (Section 2.3) restricts the number of different simple paths between pairs of vertices, DAG-depth (Definition 2.6) is the directed analog of tree-depth [NdM06]. K-width and DAG-depth are very restrictive digraph measures; at least as high as DAG-width, and often much higher.

The problems we consider in this paper (and formally define in Section 3) are Hamiltonian path (HAM), Disjoint paths (k-Path), Directed Dominating Set (DiDS), unit cost Directed Steiner Tree (DiSTP), Directed Feedback Vertex Set (DFVS), Kernel (Kernel), Maximum Directed Cut (MaxDiCut), Oriented Colouring (OCN), MSO_1 model checking (ϕ-MSO_1MC), solving Parity Games (Parity) and LTL-model checking (ϕ-LTLMC). See Table 1 in Section 3.

It turns out that most of the aforementioned problems are not only hard for DAG-width, but even for constant K-width and DAG-depth, or on DAGs. This can be seen as a strong indication that DAG-width or related measures are not yet the right parameters for dealing with standard digraph problems.

On the other hand, one width measure that fares much better in Table 1 is bi-rank-width (Definition 2.4), a width measure generalizing the rank-width of undirected graphs [Kan08]. Nearly all of our problems are fixed parameter tractable or at least in XP with respect to this parameter. Even better, unlike as for DAG-width or Kelly-width, finding an optimal bi-rank-decomposition is known to be in FPT [HO08,Kan08].

2 Digraph Width Measures

The first wave of directed measures to appear shared the following features:

i) On bidirected orientations of graphs they coincided with the tree-width.
ii) These measures were strongly based on some variant of the directed *cops-and-robber game* on a digraph: There are k cops and a robber. Each cop can either occupy a vertex, or move around in a helicopter, and the robber occupies a vertex. The robber can, however, see the helicopter landing, and

can move at a great speed along a cop-free directed path to another vertex. The objective of the cops is to capture the robber by landing on the vertex currently occupied by him, the objective of the robber is to avoid capture.

iii) Point (ii) implied that DAGs and other graphs where vertices could be ordered in such a way that edges between them point mainly in one direction, and only a few point backwards, have a very low width.

iv) The last feature (iii) also made the algorithms to be XP, instead of FPT, because of the need to remember the partial results for all vertices with incoming edges from the outside, of which there could be $|V|$.

Directed tree-width. The first explicit directed measure was that of *directed tree-width (dtw)* [JRST01]. In the related cops-and-robber game the robber has to stay in the same cop-free strongly connected component, however the relationship between the number of cops needed and the directed tree-width is not strict. [JRST01] also contains XP algorithms for solving the Hamiltonian cycle, k-path, and related problems on graphs of bounded directed tree-width.

DAG-width. First defined in [Obd06] and, independently, in [BDHK06], *DAG-width (dagw)* was the next attempt to come up with a directed tree-width counterpart. This time the robber does not have to stay in the SCC, but the cop strategy has to be monotone, i.e., a cop cannot be placed on a previously vacated vertex. This game fully characterizes DAG-width. Note that monotone and non-monotone strategies are not equivalent [KO08].

Theorem 2.1 ([Obd06,BDHK06]). *For any graph G, there is a DAG-decomposition of G of width k if, and only if, the cop player has a monotone winning strategy in the k-cops-and-robber game on G.*

Kelly-width. Defined a year later, *Kelly-width (kellyw)* [HK08] aimed to solve an existing problem with DAG-decompositions: the number of nodes can be polynomially larger then the number of vertices in the original graph (the size depends on the width). The idea of Kelly-decompositions is based on the elimination ordering for tree-width, and therefore the size of the decomposition is linear in the size of the graph. The game characterizing Kelly-width is as for DAG-width, but with two important differences: 1) the cops cannot see the robber, and 2) the robber can move only when a cop is about to land on his vertex.

Cycle rank. This is perhaps the oldest definition of a digraph connectivity measure, given in 60's by Eggan and Büchi [Egg63].

Definition 2.2 (Cycle rank). The *cycle rank* $cr(G)$ of a digraph G is defined inductively as follows: For DAGs, $cr(G) = 1$. If G is strongly connected and $E(G) \neq \emptyset$, then $cr(G) = 1 + \min\{\, cr(G - v) : v \in V(G) \,\}$. Otherwise, $cr(G)$ is the maximum over the cycle rank of the strongly connected components of G.

Measure comparison. All the measures presented above are closely related to each other. The following theorem in a summary shows that if a problem is hard for graphs of bounded cycle rank, then it is hard for all the other measures.

Theorem 2.3. *Let G be a digraph. Then (dpw [Bar06] is the directed pathwidth):*

$$1/3(dtw(G)-1) \leq_{[\text{BDHK06}]} dagw(G) \leq dpw(G) \leq_{[\text{Gru08}]} cr(G)$$
$$1/6(dtw(G)+2) \leq_{[\text{HK08}]} kellyw(G) \leq dpw(G) \leq_{[\text{Gru08}]} cr(G)$$

Moreover, when DAG-width is bounded, so is Kelly-width [HO06].

2.1 Directed rank-width

The rank-width of undirected graphs was introduced by Oum and Seymour in relation to graph clique-width. While the definition of clique-width works "as is" also on digraphs, the following straightforward generalization of rank-width to digraphs (related to clique-width again) has been proposed by Kanté [Kan08].

Definition 2.4 (Bi-rank-width). Consider a digraph G, and vertex subsets $X \subseteq V(G)$ and $Y = V(G) \setminus X$. Let $\boldsymbol{A}^+_X$ denote the $X \times Y$ 0,1-matrix with the entries $a_{i,j} = 1$ ($i \in X$, $j \in Y$) iff $(i,j) \in E(G)$, and let $\boldsymbol{A}^-_X = (\boldsymbol{A}^+_Y)^T$. The *bi-cutrank* function of G is defined as the sum of the ranks of these two matrices $brk_G(X) = rk(\boldsymbol{A}^+_X) + rk(\boldsymbol{A}^-_X)$ over the binary field $GF(2)$. The *bi-rank-width* $brwd(G)$ of G then equals the branch-width of this bi-cutrank function brk_G.

We remind the readers that the *branch-width* [RS91] of an arbitrary symmetric submodular function $\lambda\colon 2^E \to \mathbb{N}$ is defined as the minimum width over all branch-decompositions of λ over E, where a *branch-decomposition* is a pair T, τ satisfying the following: T is a tree of degree at most three, and τ is a bijection from E to the leaves of T. If f is an edge of T, then let $X_f \subseteq V(T)$ be the vertex set of one of the two connected components of $T - f$, and let the *width* of f be $\lambda(\tau^{-1}(X_f))$. The width of T, τ is the largest width over all edges of T.

Importantly, as proved by Kanté [Kan08], the rank-decomposition algorithm of [HO08] can also be used to find an optimal bi-rank-decomposition of a digraph.

Theorem 2.5 ([HO08] and [Kan08]). *Let $t \in \mathbb{N}$ be constant. There exists an algorithm that in time $O(n^3)$, for a given n-vertex graph (digraph) G, either outputs a rank-decomposition (bi-rank-decomposition, respectively) of G of width at most t, or certifies that the rank-width (bi-rank-width) is more than t.*

A rank-decomposition is, actually, not so suitable for designing dynamic programming algorithms. Yet, there is an efficient alternative characterization of a rank-decomposition via algebraic terms (or *parse trees*) over the bilinear graph product, which has been proposed by Courcelle and Kanté [CK07] and further extended towards algorithmic applications by [GH08] (see also an independent similar approach of [BXTV08]). As shown in [Kan08], an analogous "dynamic programming friendly" parse-tree view (of bi-rank-width) exists for digraphs, and we will apply this later, e.g. in Theorems 3.7 and 3.12.

2.2 DAG-depth

This part is inspired by the tree-depth notion of Nešetřil and Ossona de Mendez. [NdM06, Lemma 2.2] gives an inductive definition of the *tree-depth* $td(G)$ of

undirected G as follows (compare to Def. 2.2). If G has one vertex, then $td(G) = 1$. If G is connected, then $td(G) = 1 + \min\{\, td(G - v) : v \in V(G) \,\}$. Otherwise, $td(G)$ equals the maximum over the tree-depth of the components of G.

We propose a new "directed" generalization of this definition. For a digraph G and any $v \in V(G)$, let G_v denote the subdigraph of G induced by the vertices reachable from v. The maximal elements of the poset $\{\, G_v : v \in V(G) \,\}$ in the graph-inclusion order are called *reachable fragments* of G. Notice that reachable fragments in the undirected case coincide with connected components.

Definition 2.6 (DAG-depth). The *DAG-depth* $ddp(G)$ of a digraph G is inductively defined: If $|V(G)| = 1$, then $ddp(G) = 1$. If G has a single reachable fragment, then $ddp(G) = 1 + \min\{\, ddp(G - v) : v \in V(G) \,\}$. Otherwise, $ddp(G)$ equals the maximum over the DAG-depth of the reachable fragments of G.

Comparing Definitions 2.2 and 2.6, one can see that DAG-depth equals cycle rank on bidirected orientations of graphs. Furthermore, the following useful game characterization of this new measure can be proved along Definition 2.6.

Theorem 2.7. *The DAG-depth of a digraph G is at most k if, and only if, the cop player has a "lift-free" winning strategy in the k-cops and robber game on G, i.e., a strategy that never moves a cop from a vertex once he has landed.*

Corollary 2.8 (cf. Theorem 2.1, Def. 2.2). *For any digraph G, the DAG-depth of G is greater than or equal to the DAG-width and the cycle rank of G.* □

Another claim tightly relates our new measure to directed paths in a digraph.

Proposition 2.9. *Consider a digraph G of DAG-depth t, and denote by ℓ the number of vertices of a longest directed path in G. Then $\lfloor \log_2 \ell \rfloor + 1 \leq t \leq \ell$.*

2.3 K-width

Moreover, applications in various "directed path" problems, see e.g. Section 3.1, inspired the following width measure: The *K-width* (a shortcut of "Kenny width") of a digraph G is the maximum number of distinct (not necessarily disjoint) simple s–t paths in G over all pairs of distinct vertices $s, t \in V(G)$.

Similarly to DAG-depth in Proposition 2.9, K-width can be arbitrarily large on DAGs. By giving a suitable search strategy for the cop player in a digraph G based on a DFS tree of G, we show that K-width is lower-bounded by DAG-width, but K-width is generally incomparable with cycle-rank which is unbounded on bidirected paths.

Theorem 2.10 (cf. Theorem 2.1). *For any digraph G, the K-width of G is greater or equal to the DAG-width of G minus one.*

Furthermore, an easy algorithm enumerating all paths leads to:

Proposition 2.11. *The K-width k of a given digraph G can be computed in time $k \cdot \mathrm{poly}(|V(G)|)$.*

3 Summary of Complexity Results

Table 1. Old and new (in boldface) complexity results on digraph measures (*-marked results assume a decomposition is given in advance; p-NPC is a shortcut for the complexity class para-NPC; and c and ϕ are fixed parameters of the respective problems).

Problem	K-width	DAG-depth	DAG-width	Cycle-rank	DAG	Bi-rank-width
HAM	**FPT**	**FPT**	XP[a] *	XP[a] *	P	XP[b]
			W[2]-hard[c]			W[2]-hard[d]
c-PATH	**FPT**	**FPT**	XP[a] *	XP[a] *	P[a]	**FPT**
k-PATH	**p-NPC**	**p-NPC**	**NPC**	**NPC**	**NPC**	p-NPC[e]
DIDS	**p-NPC**	**p-NPC**	**NPC**	**NPC**	**NPC**	**FPT**
DISTP	**p-NPC**	**p-NPC**	**NPC**	**NPC**	**NPC**	**FPT**
MAXDICUT	p-NPC[c]	p-NPC[c]	NPC[c]	NPC[c]	NPC[c]	**XP**
c-OCN	**p-NPC**	**p-NPC**	NPC[f]	NPC[f]	NPC[f]	**FPT**
DFVS	*open*	*open*	p-NPC[g]	p-NPC[g]	P	**FPT**
KERNEL	p-NPC[h]	p-NPC[h]	p-NPC[g,h]	p-NPC[g,h]	P	**FPT**
ϕ-MSO$_1$MC	**p-NPH**	**p-NPH**	NPH	NPH	NPH	**FPT**[i]
ϕ-LTLMC	**p-coNPH**	**p-coNPH**	**coNPH**	**coNPH**	**coNPC**	**p-coNPH**
PARITY	XP[j]	XP[j]	XP[j] *	XP[j] *	P	XP[k]

References [a][JRST01] [b][GH09] [c][LKM08] [d][FGLS09] [e][GW06] [f][CD06] [g][KO08] [h][vL76] [i][CMR00] [j][BDHK06] [k][Obd07] . Refer to the respective following sections for details and the new results.

3.1 Hamiltonian Path (HAM) and Disjoint Paths (k-PATH)

The classical NP-hard *Hamiltonian Path* (HAM) problem [GJ79] is to find a directed path that visits each vertex of a digraph exactly once. A natural generalization of HAM is the *Longest Path* problem (LONGEST PATH), where one is asked to find the longest simple path in a given digraph.

It is easy to see that HAM can be solved on DAGs in polynomial time. When using the parameter DAG-width, HAM belongs to XP [JRST01], but was also proven to be W[2]-hard [LKM08]. We prove our new FPT results for the parameters K-width and DAG-depth on more general LONGEST PATH. Using a simple enumeration of all distinct paths in the case of bounded K-width, or applying Proposition 2.9 and any FPT-algorithm for LONGEST PATH in the standard parameterization (e.g. [CKL+09]) when DAG-depth is bounded, we get:

Theorem 3.1. *There is a fixed parameter tractable algorithm solving the* LONGEST PATH *problem on a digraph G*

a) in time $O\big(t \cdot |V(G)| \cdot |E(G)|\big)$ if G is of K-width at most t;

b) in time $O\big(4^{2^t+O(t^3)} \cdot |V(G)| \cdot |E(G)|\big)$ if G is of DAG-depth at most t.

Another well-known problem is *Disjoint Paths* (k-PATH); given a digraph and k pairs of nodes (s_i, t_i), $1 \leq i \leq k$, the task is to find pairwise disjoint directed paths from each s_i to the respective t_i. This problem is NP-complete [FHW80] even when k is bounded by any constant $c \geq 2$ (c-PATH). Moreover, a

"mixed" generalization of c-PATH remains NP-complete [BJK09] even on DAGs, and k-PATH is NP-complete [GW06] even on digraphs of bounded bi-rank-width.

If the digraph of an instance of k-PATH has K-width ≤ 2, then it can be expressed as a 2-SAT formula, and if DAG-depth is ≤ 2, then it is equivalent to an SDR instance (system of distinct representatives). If, however, we slightly relax the restrictions as follows, the problem becomes NP-complete again.

Theorem 3.2. *The k-PATH problem (with k as part of input)*
a) can be solved in polynomial time on graphs of K-width or DAG-depth 2;
b) is NP-complete on DAGs of K-width 3 and DAG-depth 4.

Finally, since one can express an instance of c-PATH for any fixed c in MSO_1 logic (Section 3.6), it follows from Theorem 3.12 that this problem is fixed parameter tractable on digraphs of bi-rank-width t with parameters c and t. The c-PATH problem however also becomes easier for the other new measures:

Theorem 3.3. *There is a fixed parameter tractable algorithm (for constant c) solving the c-PATH problem on a digraph G*
a) in time $O(t^c \cdot |E(G)|)$ if G is of K-width at most t;
b) in time $O\big((2c)^{ct4^t} \cdot |E(G)|^2\big)$ if G is of DAG-depth at most t.

3.2 Directed Dominating Set (DiDS) and Steiner Tree (DiSTP)

The well-known NP-hard *Dominating Set* (DS) and *Steiner Tree* (STP) problems both allow for natural directed counterparts. We consider them in their unweighted variants for simplicity. The *Directed Dominating Set* problem (DiDS) asks for a minimum cardinality vertex set X in a digraph G such that every vertex of G not in X is an outneighbour of X. The *Directed Steiner Tree* problem (DiSTP) [HRW92], given a digraph G and $T \subseteq V(G)$, $r \in V(G)$, asks for a minimum size tree in G spanning $\{r\} \cup T$ with all arcs oriented away from r.

While it is folklore that both of these problems are NP-hard in general, we show (with a simple reduction from VERTEX COVER) that the same holds even on very restricted graph classes.

Theorem 3.4. DiDS *and* DiSTP *problems are NP-complete on a digraph G even if G is restricted to be a DAG of K-width 2 and DAG-depth 3.*

Applying the MSO_1 optimization framework described in Section 3.6 we get:

Proposition 3.5 (Theorem 3.12). *The (unit cost)* DiDS *and* DiSTP *problems are fixed parameter tractable when parameterized by bi-rank-width.*

3.3 Maximum directed cut (MaxDiCut)

Maximum directed cut (MAXDICUT) is an extensively studied problem on digraphs. Given a digraph G, the goal is to partition the vertex set $V(G)$ into V_0 and V_1 such that the cardinality of $\{\,(u,v) \in E(G) : u \in V_0, v \in V_1\,\}$ is

maximized. This problem is often stated with edge weights, but we consider only the unweighted (cardinality MaxDiCut) variant in our paper.

It is well known that the MaxDiCut optimization problem is NP-hard, and it has been shown that MaxDiCut stays NP-hard even on DAGs [LKM08]. A closer, yet quite nontrivial look, at the reduction reveals the resulting graph to have also bounded DAG-depth and K-width.

Theorem 3.6 ([LKM08]). *The* MaxDiCut *problem is NP-hard on a digraph G even if G is restricted to be a DAG of K-width* 4608 *and DAG-depth* 11*.*

The only new efficiently solvable case among our measures is the following:

Theorem 3.7. *The unweighted* MaxDiCut *problem on a digraph G of bi-rank-width t is polynomially solvable for every fixed t (i.e. it belongs to the class XP).*

3.4 Oriented Colouring (OCN)

A natural directed generalization of the ordinary graph colouring problem can be obtained as follows: The chromatic number $\chi(G)$ of a graph G equals the minimum c such that G has a homomorphism into the complete graph K_c. The *Oriented Chromatic Number* (OCN) $\chi_o(G)$ of a digraph G is defined as the minimum c such that G has a homomorphism into some(!) orientation of K_c.

In other words, $\chi_o(G)$ equals minimum c such that the vertex set of G can be partitioned into c independent sets such that, between each pair of the sets, all arcs have the same direction. For instance, $\chi_o = 5$ for the directed 5-cycle.

It has been shown [KM04] that checking $\chi_o(G) \leq 3$ is easy, but determining whether $\chi_o(G) \leq 4$ is already NP-complete. Subsequently, [CD06] have shown that the problem $\chi_o(G) \leq 4$ remains NP-complete even on acyclic digraphs. Using a simpler and more powerful reduction than [CD06], we prove:

Theorem 3.8. *The problem (*4-OCN*) to decide whether a digraph G satisfies $\chi_o(G) \leq 4$ is NP-complete even if G is a DAG of K-width* 3 *and DAG-depth* 5*.*

On the other hand, it follows from the general framework of Theorem 3.12:

Proposition 3.9. *The problem (c-*OCN*) to decide $\chi_o(G) \leq c$ on an input digraph G of bi-rank-width t is fixed parameter tractable with parameters c and t.*

3.5 Directed Feedback Vertex Set (DFVS) and Kernel (Kernel)

The *directed feedback vertex set* (DFVS) problem is to find a minimum cardinality set S of vertices of a digraph G whose removal leaves $G \setminus S$ acyclic. This problem is trivial for acyclic digraphs, and it is FPT with the parameter $k = |S|$. We hence consider only the optimization variant of DFVS with unbounded k.

Kreutzer and Ordyniak [KO08] gave a reduction showing NP-hardness of the DFVS optimization problem on digraphs of DAG-width 4. A closer look at this reduction reveals that all the produced graphs are moreover of cycle rank 4, but they have unbounded K-width and DAG-depth.

The *kernel* of a digraph G is defined as an independent set $S \subseteq V(G)$ such that for every $x \in V(G) \setminus S$ there is an arc from x into S. Notice that a kernel may not always exist. However, on acyclic digraphs, a kernel can be easily found. Having a closer look at the NP-completeness reduction of van Leeuwen [vL76], one discovers the following claim (cf. also [KO08]).

Theorem 3.10 (van Leeuwen [vL76]). *It is NP-complete to decide whether a digraph G has a kernel, even if G is restricted to have (all at once) DAG-width and K-width* 2*, cycle rank also* 2*, and DAG-depth* 4*.*

Finally, by Example 3.11 and Theorem 3.12, both the KERNEL and DFVS problems are fixed parameter tractable on digraphs of bounded bi-rank-width.

3.6 $\mathbf{MSO_1}$ Model Checking (ϕ-$\mathrm{MSO_1MC}$)

Monadic second order (MSO) logic is a language often used for description of combinatorial algorithmic problems. When applied to a one-sorted relational graph structure (i.e. to a set V with a symmetric relation $edge(u, v)$), this language is abbreviated as MSO_1. We use the same abbreviation MSO_1 also for digraphs with a relation $arc(u, v)$.

Example 3.11. The following properties are expressible in MSO_1 on digraphs

- a directed dominating set X as $\forall z\big(z \in X \vee \exists x \in X\, arc(x, z)\big)$,
- the existence of a kernel S as $\exists S \forall x\big[x \notin S \leftrightarrow \big(\exists y \in S\, arc(x, y)\big)\big]$, or
- a feedback vertex set Z as $\forall X\big[X \cap Z = \emptyset \rightarrow \big(\exists x \in X \forall y \in X\, \neg arc(x, y)\big)\big]$.

On the other hand, MSO_1 cannot express Hamiltonian cycle, for instance.

The MSO_1 *model checking problem* (ϕ-$\mathrm{MSO_1MC}$), where ϕ is a fixed formula, is FPT on (undirected) graphs of bounded clique-width or rank-width [CMR00,CK07]. Not surprisingly, this extends to digraphs parameterized by bi-rank-width. More generally, the *LinEMSO$_1$ optimization framework* includes all problems which can be expressed as maximization of a linear evaluational term over all tuples of sets $X_1, \ldots, X_j$ satisfying $\psi(X_1, \ldots, X_j)$ where ψ is an MSO_1 formula — see [CMR00] for details. Analogously to [CMR00] (or [GH08]) we get:

Theorem 3.12 (cf. [CMR00], and [Kan08,GH08]).
Every ψ-LinEMSO$_1$ optimization problem is fixed parameter tractable when restricted to digraphs of bi-rank-width t, with parameters t and ψ.

Theorem 3.12 particularly implies that the problems listed in Example 3.11 (and many others) are FPT on digraphs of bi-rank-width t. No analogous results, however, seem possible for our other directed width measures since one can interpret ϕ-$\mathrm{MSO_1MC}$ of arbitrary undirected graphs via subdividing each edge and giving the two new edges opposite orientations, leading to:

Proposition 3.13. *The ϕ-$\mathrm{MSO_1MC}$ problem is NP-hard even when restricted to DAGs that are of K-width* 1 *and DAG-depth* 2*.*

3.7 LTL Model Checking (ϕ-LTLMC) and Parity Games (PARITY)

Another useful language that allows to express properties of digraphs is *Linear Temporal Logic* (LTL) — see, e.g., [BK08]. LTL model checking remains hard for a fixed formula ϕ and all of the directed width measures we considered here, including bi-rank-width (as opposed to MSO_1 model checking).

Theorem 3.14. *The ϕ-LTLMC problem is coNP-hard even when the input digraph is restricted to have K-width 1, DAG-depth 4, and bi-rank-width 2.*

Theorem 3.15. *The ϕ-LTLMC problem is coNP-complete on DAGs.*

Parity games — see e.g. [GTW02] for a reference, play an important role in the field of model-checking and formal verification. There are many reasons for this. First, solving parity games is equivalent to model-checking the modal μ-calculus, an important modal logic subsuming many other logics (e.g. CTL). Moreover, the modal μ-calculus is a bisimulation invariant fragment of MSO_1.

Second, the exact complexity of solving a parity game is a long-standing open problem. It is known to be in NP $\cap$ co-NP, and widely believed to be in P. It is trivially in P for acyclic digraphs. Moreover, it was shown that solving a parity game is in XP for digraphs of bounded tree-width [Obd03], bounded DAG-width [BDHK06] (hence also on bounded K-width, DAG-depth, and cycle rank) and bounded Kelly-width [HK08], and of bounded clique-width [Obd07] (implying the same for bi-rank-width).

4 Conclusion

Table 1, and the related results in this paper, have left several interesting open problems and questions. Just to specifically mention a few:

1) We suggest there exist FPT algorithms solving the DFVS problem for bounded K-width or DAG-depth (two of the open table entries).
2) For some entries in the table, we neither expect an FPT algorithm, nor have an NP-hardness estimate. E.g., MAXDICUT or k-PATH for bi-rank-width, or c-PATH for cycle rank. Can we then, at least, show a W-hardness result?
3) While we have given FPT and XP, respectively, algoritms solving the unit-cost variants of DISTP and MAXDICUT, these problems are usually considered in their weighted variants and then we expect their complexity to be higher. We, however, have no further results in this direction.
4) Some suggest that the DFVS number (see in Section 3.5) perhaps can be a good directed width measure. However, since majority of our sample problems in Table 1 remain hard even on DAGs, there is not much room left for applications of the DFVS parameter. Interestingly though, KERNEL becomes FPT when parametrized by DFVS.

Theorem 4.1. *If a digraph G is given with a directed feedback vertex set of size k, then the* KERNEL *problem can be solved in time $O(2^k \cdot |V(G)|^2)$.*

Finally, we try to formulate the overall impression coming from Table 1: Robber-and-cops based width measures do not seem to be very useful for parameterized algorithms on digraphs. One reason might be that cops "give" good graph separators in the undirected case, but that does not work any more for digraphs. Considering the DFVS number as a width parameter does not seem to help either. We perhaps need something new to move on. At this moment, bi-rank-width seems like a good alternative.

Acknowledgments. This work has been supported by a Czech–German bilateral grant of GAČR and DFG (201/09/J021 and RO 927/9). Moreover, P. Hliněný has been supported by the Czech research grant GAČR 201/08/0308.

References

[Bar06] J. Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2):161–172, 2006.

[BDHK06] D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. DAG-width and parity games. In *STACS'06*, volume 3884 of *LNCS*, pages 524–536. Springer, 2006.

[BG04] D. Berwanger and E. Grädel. Entanglement – a measure for the complexity of directed graphs with applications to logic and games. In *LPAR 2004*, volume 3452 of *LNCS*, pages 209–223. Springer, 2004.

[BJK09] J. Bang-Jensen and M. Kriesell. Disjoint directed and undirected paths and cycles in digraphs. Technical Report PP-2009-03, University of South Denmark, 2009.

[BK08] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, Cambridge, Massachusetts, 2008.

[BXTV08] B.-M. Bui-Xuan, J. Telle, and M. Vatshelle. H-join and algorithms on graphs of bounded rank-width. submitted, November 2008.

[CD06] J.-F. Culus and M. Demange. Oriented coloring: Complexity and approximation. In *SOFSEM'06*, volume 3831 of *LNCS*, pages 226–236. Springer, 2006.

[CK07] B. Courcelle and M. Kanté. Graph operations characterizing rank-width and balanced graph expressions. In *WG'07*, volume 4769 of *LNCS*, pages 66–75. Springer, 2007.

[CKL+09] J. Chen, J. Kneis, S. Lu, D. Mölle, S. Richter, P. Rossmanith, S. Sze, and F. Zhang. Randomized divide-and-conquer: Improved path, matching, and packing algorithms. *SIAM Journal on Computing*, 38(6):2526–2547, 2009.

[CMR00] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[DGK08] P. Dankelmann, G. Gutin, and E. Kim. On complexity of minimun leaf out-branching. arXiv:0808.0980v1, August 2008.

[Egg63] L. Eggan. Transition graphs and the star-height of regular events. *Michigan Mathematical Journal*, 10(4):385–397, 1963.

[FGLS09] F. Fomin, P. Golovach, D. Lokshtanov, and S. Saurab. Clique-width: On the price of generality. In *SODA'09*, pages 825–834. SIAM, 2009.

[FHW80] S. Fortune, J. E. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980.

[GH08] R. Ganian and P. Hliněný. Automata approach to graphs of bounded rank-width. In *IWOCA'08*, pages 4–15, 2008.

[GH09] R. Ganian and P. Hliněný. Better polynomial algorithms on graphs of bounded rank-width. In *IWOCA'09*, LNCS. Springer, 2009. to appear.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.

[Gru08] H. Gruber. Digraph complexity measures and applications in formal language theory. In *MEMICS'08*, pages 60–67, 2008.

[GTW02] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.

[GW06] F. Gurski and E. Wanke. Vertex disjoint paths on clique-width bounded graphs. *Theor. Comput. Sci.*, 359(1-3):188–199, 2006.

[HK08] P. Hunter and S. Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theor. Comput. Sci.*, 399(3):206–219, 2008.

[HO06] P. Hliněný and J. Obdržálek. Escape-width: Measuring "width" of digraphs. Presented at Sixth Czech-Slovak International Symposium on Combinatorics, Graph Theory, Algorithms and Applications, 2006.

[HO08] P. Hliněný and S. Oum. Finding branch-decomposition and rank-decomposition. *SIAM J. Comput.*, 38:1012–1032, 2008.

[HRW92] F. Hwang, D. Richards, and P. Winter. *The Steiner Tree Problem*. Annals of Discrete Mathematics. Noth-Holland, 1992.

[JRST01] T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

[Kan08] M. Kanté. The rank-width of directed graphs. arXiv:0709.1433v3, March 2008.

[KM04] W. Klostermeyer and G. MacGillivray. Homomorphisms and oriented colorings of equivalence classes of oriented graphs. *Discrete Mathematics*, 274:161–172, 2004.

[KO08] S. Kreutzer and S. Ordyniak. Digraph decompositions and monotonicity in digraph searching. In *WG'08*, volume 5344 of *LNCS*, pages 336–347. Springer, 2008.

[LKM08] M. Lampis, G. Kaouri, and V. Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. In *ISAAC*, volume 5369 of *LNCS*, pages 220–231. Springer, 2008.

[NdM06] J. Nešetřil and P. Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European J. Combin.*, 27(6):1024–1041, 2006.

[Obd03] J. Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In *CAV 2003*, volume 2725 of *LNCS*, pages 80–92. Springer, 2003.

[Obd06] J. Obdržálek. DAG-width – connectivity measure for directed graphs. In *SODA'06*, pages 814–821. ACM-SIAM, 2006.

[Obd07] Jan Obdržálek. Clique-width and parity games. In *CSL'07*, volume 4646 of *LNCS*, pages 54–68. Springer, 2007.

[RS86] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, September 1986.

[RS91] N. Robertson and P. D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory B*, 52(2):153–190, 1991.

[Saf05] M. Safari. D-width: A more natural measure for directed tree-width. In *MFCS'05*, volume 3618 of *LNCS*, pages 745–756. Springer, 2005.

[vL76] J. van Leeuwen. Having a Grundy-numbering is NP-complete. Technical Report 207, The Pennsylvania State University, September 1976.

APPENDIX

Notes on Section 2.1 (bi-rank-width)

A rank-decomposition is, actually, not so suitable for designing dynamic programming algorithms. Yet, there is an efficient alternative characterization of a rank-decomposition via algebraic terms (or *parse trees*) over the bilinear graph product, which has been proposed by Courcelle and Kanté [CK07] and further extended towards algorithmic applications by [GH08] (see also an independent similar approach of [BXTV08]). As shown in [Kan08], an analogous "dynamic programming friendly" parse-tree view (of bi-rank-width) exists for digraphs, and we will apply this later, e.g. in Theorem 3.12. in Section 3.6.

Following Section 2.1, we describe Kanté's *bi-labeling parse trees* [Kan08, Section 4] (thereafter called "algebraic expressions for bin-rank-width"), which characterize bi-rank-width of digraphs up to a multiplicative factor 2 (Lemma 5.3).

A *t-labeled digraph* is a pair $\bar{G} = (G, lab)$ of a digraph G and a vertex *labeling* $lab\colon V(G) \to 2^{1,\dots,t}$ into subsets of t labels, or equivalently in linear algebra terms a mapping $lab\colon V(G) \to GF(2)^t$ into the points of a t-dimensional binary vector space. For technical reasons, we analogously define a *t-bi-labeled* digraph $\tilde{H} = (H, lab^+, lab^-)$. A *t-relabeling* is a linear mapping $f\colon GF(2)^t \to GF(2)^t$, or in other words a binary $t \times t$ matrix f.

Definition 5.1 (Bi-labeling join). Considering a t-labeled digraph $\bar{G} = (G, lab)$ and a t-bi-labeled digraph $\tilde{H} = (H, lab^+, lab^-)$, a *t-bi-labeling join* $\bar{G} \otimes \tilde{H}$ is defined on a disjoint union of G and H by adding, where $u \in V(G), v \in V(H)$; all arcs (u, v) such that $|lab(u) \cap lab^+(v)|$ is odd, and all arcs (v, u) such that $|lab(u) \cap lab^-(v)|$ is odd. The resulting digraph is unlabeled.

Definition 5.2 (Bi-labeling parse trees). Considering t-labeled digraphs $\bar{G}_i = (G_i, lab_i)$, $i = 1, 2$, and relabelings $f_1, f_2, h^+, h^-\colon GF(2)^t \to GF(2)^t$, we define a *t-bi-labeling composition* operator $\otimes[h^+, h^-;\ f_1, f_2]$ as follows. $\bar{G}_1 \otimes [h^+, h^-;\ f_1, f_2]\, \bar{G}_2 = \bar{G}_3$ where $G_3 = \bar{G}_1 \otimes (G_2, h^+ \cdot lab_2^T, h^- \cdot lab_2^T)$ and the labeling of $v \in V(G_i)$ in $\bar{G}_3$ is $lab_3(v) = f_i \cdot lab_i^T$, $i = 1, 2$.

A *t-bi-labeling parse tree* T, see also [GH08], is a finite rooted ordered subcubic tree (with the root degree at most 2) such that

- the leaves of T contain a $\odot$ symbol creating a new graph vertex of label $\{1\}$,
- each internal node of T contains one of the t-bi-labeling composition symbols.

A parse tree T then *generates* (parses) the digraph $\bar{G}$ which is obtained by successive leaves-to-root applications of the operators in the nodes of T.

Lemma 5.3 (Kanté [Kan08]). *Let G be a digraph of bi-rank-width t. If m is the smallest integer such that (some labeling of) G is produced by some m-bi-labeling parse tree, then $t \geq m \geq t/2$.*

Having now the bi-labeling parse tree machinery at hand, it is straightforward to translate the formal tools of [GH08,GH09] to digraphs of bounded bi-rank-width, see e.g. the proof of Theorem 3.7. In this way, for instance, the XP algorithm for undirected Hamiltonian path [GH09] directly translates to an XP algorithm for Hamiltonian path in digraphs of bounded bi-rank-width.

On the other hand, we remind the readers that one cannot use an undirected rank-decomposition (or parse tree) of a digraph G to design a dynamic programming algorithm for a problem referring to the direction of arcs of G. That is because the parse tree produces large bipartite cliques, and one cannot exhaustively process all possible orientations of those.

For sake of completeness, we lastly remark that Kanté [Kan08] considers also another directed generalization of rank-width, the so called $GF(4)$-rank-width. Since these two are within a constant factor, there is no need to consider the latter in our paper.

Proofs for Section 2.2 (DAG-depth)

Theorem 2.7. *The DAG-depth of a digraph G is at most t if, and only if, the cop player has a "lift-free" winning strategy in the k-cops and robber game on G, i.e., a strategy that never moves a cop from a vertex once he has landed.*

Proof. (sketch) We proceed by induction along the definition of DAG-depth. That is trivial if $|V(G)| = 1$. Let $F_1, \ldots, F_d$ be all the reachable fragments of G. If $d > 1$, then the robber may start in any vertex of any $F_i \subseteq G$, $i \in \{1, \ldots, d\}$, and so the cop player needs as many moves in G as in such most expensive reachable fragment which is $\max\{\, ddp(F_i) : i = 1, \ldots, d \,\}$ by inductive assumption.

Now assume G has a single reachable fragment. Hence there is $v \in V(G)$ which can reach whole G, and so whenever another cop is to land at $s \in V(G)$, the robber may move to any vertex of $G-s$. It follows from inductive assumption that the cop player needs another $ddp(G-s)$ moves after landing at s. Therefore, the cop player needs at least $1 + \min\{\, ddp(G-v) : v \in V(G) \,\}$ moves on G, and this is also sufficient. □

Proposition 2.9. *Consider a digraph G of DAG-depth t, and denote by ℓ the number of vertices of a longest directed path in G. Then $\lfloor \log_2 \ell \rfloor + 1 \leq t \leq \ell$.*

Proof. Firstly, we show that the DAG-depth of an ℓ-vertex path P is at least $\lfloor \log_2 \ell \rfloor + 1$. This is trivial if $\ell = 1$. Since a path has a single reachable fragment, we have from the definition $ddp(P) = 1 + ddp(Q)$ where Q a path of length $\lceil (\ell-1)/2 \rceil$. If ℓ is even, then $ddp(P) = 1 + \lfloor \log_2(\ell/2) \rfloor + 1 = \lfloor \log_2 \ell \rfloor + 1$. If ℓ is odd, then $ddp(P) = 1 + \lfloor \log_2((\ell-1)/2) \rfloor + 1 = \lfloor \log_2(\ell-1) \rfloor + 1 = \lfloor \log_2 \ell \rfloor + 1$.

On the other hand, we describe a simple ℓ-move lift-free winning strategy for the cop player on any such digraph G. The first cop lands on the initial position s_1 of the robber. In cop move $i > 1$, the cop number i lands on a vertex s_i of G which is the out-neighbour of s_{i-1} on some directed path from s_{i-1} to the current robber position. Since all directed paths starting in s_1 have $\leq \ell$ vertices, the robber is finally caught after $\leq \ell$ cop moves. □

Notice that Proposition 2.9 provides efficient computation of DAG-depth:

Corollary 5.6. *The DAG-depth of a digraph G can be approximated by an FPT algorithm, and computed exactly by an XP algorithm.*

Proof. We first compute the longest directed path length ℓ in G, which can be done by an FPT algorithm of e.g., [CKL$^+$09]. This ℓ is already a good estimate of the DAG-depth of G by Proposition 2.9.

In the second part, we carry a brute-force recursive computation of the DAG-depth of G according to the definition. Since the depth of recursion is bounded by ℓ, and each call branches into $O(|V(G)|)$ subproblems, we get an XP algorithm. □

Furthermore, notice that the approach of Corollary 5.6 also gives an efficient way to construct a bounded DAG-decomposition for G if the DAG-depth is bounded (of course, with no matching lower bound).

Proofs for Section 2.3 (K-width)

Theorem 2.10. *For any digraph G, the K-width of G is greater or equal to the DAG-width of G minus one.*

Proof. Let T be any depth first search tree of G. Based on T, we outline a monotone search strategy for the cop player on G, in which the player is to use a cop number $k+1$ only if there are at least k paths between a pair of vertices.

(i) In the first move a cop is placed at the root of T.
(ii) In each subsequent cop-placing move, the cop player chooses the (unique) vertex v of G such that; v is an outneighbour of a cop-occupied vertex, and v reaches the robber along a cop-free path in T.
(iii) Whenever a cop-occupied vertex u is no longer reachable from the current robber position, the cop from u is lifted back.

This strategy is clearly monotone. Consider the vertex v in rule (ii). If there was a cop-occupied vertex w in G which is not an ancestor of v, then w must no longer be reachable by the robber since T is a DFS tree. So (iii) for $u = w$ applies before (ii). Therefore, our strategy maintains an invariant that all vertices occupied by cops belong to one directed path of T.

Consider now a situation when there is a set U of k cop-occupied vertices in G, and rule (ii) applies again. Then there is a path $P \subseteq T$ such that $U \subseteq V(P)$. Let s be the last cop-occupied vertex of P. By (iii), each vertex $w \in U$ is reachable in G from the robber vertex r along a cop-free path Q_w. So $P \cup Q_w$ contains a path from r to s, and these k paths are pairwise distinct for distinct w. □

Notice that the proof of Theorem 2.10, together with Proposition 2.11, give an efficient way to construct a bounded DAG-decomposition for G if the K-width is bounded. Furthermore, the following simple claim will be useful in algorithmic applications of K-width.

Lemma 5.8. *If G is a digraph of K-width t, then all (at most $t|V(G)|$) directed paths starting at a vertex $u \in V(G)$ can be enumerated in time $O(t \cdot |E(G)|)$.*

Proof. Enumerate paths in G starting at u by backtracking and prune the search whenever finding a vertex that is already on the current path.

The resulting search tree has at most $t|V(G)|$ nodes: Each node in the search tree corresponds to a simple path in G starting at u. There can be at most t such paths with the same terminal vertex.

The time spent in each node of the search tree is $O(d)$, where d is the out-degree of the terminal vertex of the corresponding simple path. Overall this amounts to a running time of $O(t|E(G)|)$. □

Finally, we can easily compute the K-width, if it is not too big.

Proposition 2.11. *The K-width k of a given digraph G can be computed in time $k \cdot \text{poly}(|V(G)|)$.*

Proof. Enumerate all (up to k each) simple paths starting from every vertex in G. Count how many path to each other vertex. The maximum number you encounter is exactly k. □

Proofs for Section 3.1 (HAM, *k*-Path)

Note that FPT-membership for Longest Path implies membership for HAM.

Theorem 3.1(a). *Given a digraph G with K-width at most t, one can solve* Longest Path *in time $O(t|V(G)| \cdot |E(G)|)$.*

Proof. For all $u \in V(G)$ enumerate all simple paths starting at u according to Lemma 5.8 while keeping track of their lengths. □

Theorem 3.1(b). *Given a digraph G with DAG-depth at most t, one can solve* Longest Path *in time $4^{2^t+O(t^3)} \cdot |V(G)| \cdot |E(G)|$.*

Proof. We know by Proposition 2.9 that $\lfloor \log_2 \ell \rfloor + 1 \leq t \leq \ell$, or in other words, $\ell \leq 2^t$, where ℓ is the length of the longest path. We can hence use an arbitrary FPT-algorithm for the Longest Path decision problem in the standard parameterization (e.g., [CKL+09] with running time $4^{\ell+O(\log^3 \ell)}|V(G)| \cdot |E(G)|$): We begin with $\ell = 1$ and subsequently increase ℓ until a "no"-instance is found. This yields an FPT-algorithm for parameter t even if t is unknown to the algorithm. □

Theorem 3.2(a). *The k-Path problem can be solved in polynomial time on graphs of K-width at most 2 or DAG-depth at most 2.*

Proof. Given a digraph G with K-width ≤ 2 and k pairs of nodes (s_1, t_1), $\ldots, (s_k, t_k)$, we first for every $1 \leq i \leq k$ compute by Lemma 5.8 the (wlog) two possible paths $p_{i,1}$ and $p_{i,2}$ from s_i to t_i. Then we construct a 2-SAT formula

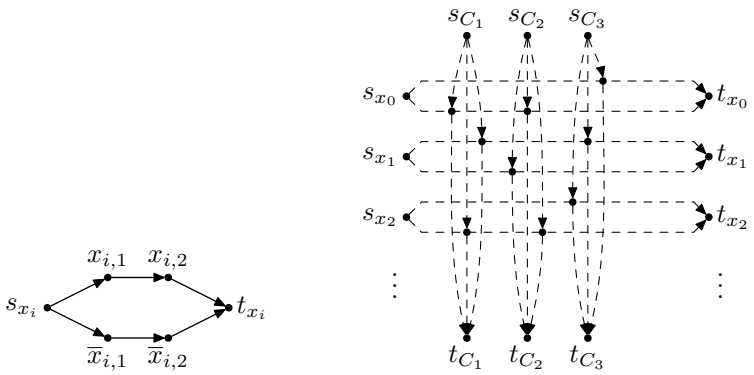

Fig. 1. Left: gadget for variable x_i; right: schematic of the construction

as follows: For each pair (s_i, t_i), $1 \leq i \leq k$, there is a clause over the two alternative paths, $C_i = \{p_{i,1}, p_{i_2}\}$. Furthermore, for each pair of non-disjoint paths $p_1, p_2 \in \{\, p_{i,j} : 1 \leq i \leq k, 1 \leq j \leq 2 \,\}$, such that $p_1 \neq p_2$ and $V(p_1) \cap V(p_2) \neq \emptyset$, there is a clause excluding each other, $C_{p_1,p_2} = \{\neg p_1, \neg p_2\}$. We omit the simple proof that the formula is satisfiable if and only if there is a solution to the k-paths instance at hand.

Similarly, given a digraph G with $ddp(G) \leq 2$ and k pairs of nodes (s_1, t_1), $\ldots, (s_k, t_k)$, we proceed as follows. In the first step, for each pair (s_i, t_i) such that $(s_i, t_i) \in E(G)$, we simple remove both vertices s_i, t_i from the instance.

Hence we may assume that every s_i–t_i path in G is formed by a pair of arcs $(s_i, x), (x, t_i) \in E(G)$, cf. Proposition 2.9. We denote by X_i the set of all such x in G for the pair (s_i, t_i). Then, clearly, the k-Path instance has a solution if and only if $X_1, \ldots, X_k$ admit a system of distinct representatives, which can be decided in P. □

Theorem 3.2(b). *The k-Path problem is NP-complete on DAGs with K-width 3 and DAG-depth 4.*

Proof. We reduce from the well-known NP-complete 3-SAT problem, where each clause contains exactly three literals. Let φ be a 3-SAT-formula with m clauses $C_1, \ldots, C_m$ over n variables. Without loss of generality, we may assume that every variable occurs in at most three literals (cf. the proof of Theorem 3.6), and that no variable has all three literals positive or all three negated (otherwise we set it true or false, respectively). Hence every literal occurs at most two times in the whole formula φ. We create a digraph G as follows.

For every variable x_i, we add a gadget as depicted in Figure 1. The "upper" path in the gadget corresponds to a negative assignment of the variable since it leaves the nodes $\overline{x}_{i,1}$ and $\overline{x}_{i,2}$ available for clauses, while similarly the "lower" path corresponds to a positive assignment. Then, for every clause $C_i = \{\ell_1, \ell_2, \ell_3\}$, we add two nodes s_{C_i} and t_{C_i}. Then, for every literal l_j,

$1 \leq j \leq 3$, such that l_j is the kth occurrence in the formula, we add the edges $(s_{C_i}, \ell_{j,k})$ and $(\ell_{j,k}, t_{C_i})$.

For example, if $l_3 = \overline{x}_5$, and $\overline{x}_5$ occurred already in some $C_{i'}$ with $i' < i$, then we add the edges $(s_{C_i}, x_{5,2})$ and $(x_{5,2}, t_{C_i})$. See Figure 1 for a schematic view.

It is easy to see that the resulting digraph is a DAG, the longest path contains at most four nodes (which bounds the DAG-depth by Proposition 2.9), and between any two nodes there are at most three paths. Furthermore, φ is satisfiable if and only if G is a "yes"-instance to the k-path problem with pairs (s_{x_i}, t_{x_i}) for all $1 \leq i \leq n$ and pairs (s_{C_j}, t_{C_j}) for all $1 \leq j \leq m$:

Let C be a satisfying assignment of the variables. For the path between a pair (s_{x_i}, t_{x_i}), we use the path corresponding to the assignment of the variable x_i, i.e., if x_i is assigned 0, we use the path through the nodes labeled with $x_{i,1}$ and $x_{i,2}$, and the path through $\overline{x}_{i,1}$ and $\overline{x}_{i,2}$ otherwise. If a clause C_j is satisfied by some literal l_i, then by construction the path between s_{x_i} and t_{x_i} is not using the node v labeled with $\bar{l}_i$, which means we can use the path $s_{C_j} v t_{C_j}$ for the pair (s_{C_j}, tC_j). Hence, all pairs can be connected by disjoint paths.

If otherwise there is a solution to the k-path problem on the constructed instance, then first note that a path between each s_{x_i} and t_{x_i} for every variable x_i either has to use the "positive" or the "negative" path through its corresponding gadget. We choose an assignment C of the variables, where each variable is assigned 0 if the path between s_{x_i} and t_{x_i} uses the path through the nodes labeled with $x_{i,1}$ and $x_{i,2}$, and is assigned 1 else. Then each clause $C_j = \{l_1, l_2, l_3\}$ is satisfied: The path between s_{C_j} and t_{C_j} has to use one of the three nodes corresponding to l_1, l_2, and l_3, say l_k for some variable x_i. Since all paths are disjoint, the path between s_{x_i} and t_{x_i} is not using l_k, and therefore the variable is assigned a value such that l_k has the value 1 and C_j is satisfied. □

Lemma 5.14. *Let G be a digraph, and let c pairs of vertices $s_i, t_i \in V(G)$, $i = 1, \ldots, c$ be given. There is an MSO$_1$ formula expressing (c-*Path*) the existence of c pairwise disjoint directed s_i–t_i paths, $i = 1, \ldots, c$, in G.*

Proof. We write

$$\exists X_1, \ldots, X_c \left[\bigwedge_{i \neq j \in \{1,\ldots,c\}} X_i \cap X_j = \emptyset \ \wedge \bigwedge_{i \in \{1,\ldots,c\}} s_i, t_i \in X_i \ \wedge \right.$$

$$\left. \bigwedge_{i \in \{1,\ldots,c\}} \forall Z \subseteq X_i \big((s_i \in Z \wedge t_i \notin Z) \rightarrow \exists x \in Z, y \in X_i \setminus Z \ arc(x,y) \big) \right]$$

which means that there exist pairwise disjoint sets $X_1, \ldots, X_c \subseteq V(G)$ such that $s_i, t_i \in X_i$, and each X_i induces a subdigraph of G in which t_i is reachable from s_i. Notice that s_i, t_i are not variables, but constants in this sentence. □

Theorem 3.3. *There is a fixed parameter tractable algorithm solving the c-*Path* problem (for fixed c) on a digraph G*

a) in time $O(t^c \cdot |E(G)|)$ if G is of K-width at most t;

b) in time $O\big((2c)^{ct4^t} \cdot |E(G)|^2\big)$ if G is of DAG-depth at most t.

Proof. (sketch) Notice that we can, without loss of generality of the c-PATH problem, assume that G is a simple digraph (while 2-cycles are permitted).

a) We, for each $i = 1, \ldots, c$, use Lemma 5.8 to list all $\leq t$ distinct directed paths from s_i to t_i. Then, using brute force over all t^c possibilities, we check whether there is a selection of pairwise disjoint ones.

b) This algorithm uses part (a) and recursive calls in a clever way. By Proposition 2.9, the longest directed path in G has length $\ell < 2^t$ (which is the only extra information we use about G). We are actually going to recursively solve a more general problem to find a collection of c pairwise disjoint directed s_i–t_i paths Q_i in G such that $E(Q_i) \subseteq E_i \subseteq E(G)$. Initially $E_1 = \cdots = E_c = E(G)$.

Let $\mathcal{P}_i$ be the collection of all s_i–t_i paths with arcs from E_i. If $|\mathcal{P}_i| \leq (c\ell)^{\ell^2}$ for all $i = 1, \ldots, c$, then we may actually use (a) to solve the problem in time $O\big((c\ell)^{c\ell^2} \cdot |E(G)|\big)$. Otherwise, $|\mathcal{P}_i| > (c\ell)^{\ell^2}$ for some i, and we may apply the following for $u = s_i$, $v = t_i$, and (loosely) $m = c\ell$, $k = \ell$:

Claim. Let H be a simple digraph, and u, v two vertices of H such that the longest path starting in u has length $k+1$ and there exist $1 + (m-1)^{k^2}$ distinct directed u–v paths in H. Then there exist vertices u', v' in H such that there are m pairwise internally disjoint u'–v' paths.

To prove the claim, we may assume that every arc of H is on some u–v path. By the pigeon-hole priciple, there exists a vertex u' in H having outdegree $\geq 1 + (m-1)^k$, and this u' is not an in-neighbour of v (otherwise, we would have only $\leq \big((m-1)^k\big)^k$ distinct u–v paths). Let u'_i, $i = 1, \ldots, p \geq 1 + (m-1)^k$ be the out-neighbours of u' in H, and let H' be an inclusion-wise minimal subgraph of H such that H' contains some u'–v path S_i passing through u'_i for all $i = 1, \ldots, p$. By a symmetric application of the pigeon-hole priciple, there exists a vertex v' having indegree $\geq m$ in H'. Let v'_j, $j = 1, \ldots, q \geq m$ be the in-neighbours of v' in H'. It follows from minimality of H' that the u'–v' paths S'_j passing through appropriate u'_{i_j} and v'_j are pairwise internally disjoint.

In other words, there exist vertices s', t' in G such that $c\ell$ suitable fragments of paths from $\mathcal{P}_i$ form pairwise internally disjoint s'–t' paths $R_1, \ldots, R_{c\ell}$. These paths can be found in time $O\big((c\ell)^{\ell^2} \cdot |E(G)|\big)$ using an approach similar to Lemma 5.8. Now, we make a new arc set E'_i from E_i by removing all arcs of $R_1 \cup \cdots \cup R_{c\ell}$, and adding a new arc $f' = (s', t')$. We call the same algorithm recursively on $E_1, \ldots, E'_i, \ldots, E_c$.

This algorithm clearly stops after $O(c|E(G)|)$ recursive calls since each call decreases $|E_1| + \cdots + |E_c|$. Hence the overall run-time is $O\big((c\ell)^{c\ell^2} \cdot |E(G)|^2\big)$. It remains to prove that there is a solution with constrains to $E_1, \ldots, E_i, \ldots, E_c$ if, and only if, there is a solution to $E_1, \ldots, E'_i, \ldots, E_c$. The "only if" direction is trivial since we can simply use the arc $f' = (s', t')$ when needed.

In the "if" direction, when f' is not used in the path, we are done. If f' is used in the s_i–t_i path Q'_i, then we notice the following: By the pigeon-hole principle, some of the paths $R_1, \ldots, R_{c\ell}$ must be disjoint from all other $c-1$ paths of $\leq \ell$ vertices in the solution, and hence we can use the path $(Q_i - f') \cup R_j$ with all arcs in E_i instead. □

Proofs for Section 3.2 (DiDS and DiSTP)

Theorem 3.4. DiDS *and* DiSTP *are NP-complete on DAGs that are of K-width* 2 *and DAG-depth* 3*.*

Proof. We use a reduction from Vertex Cover to show hardness. Let a graph $G = (V, E)$ and $k \in \mathbb{N}$ be an input instance for Vertex Cover. Wlog, we can assume that $|V| \geq k + 2$.

We construct $G' = (V', E')$ as follows. We set $V' = V \cup E \cup \{v_0\}$ and define the set of edges as follows:

$$E' = \{\, (v_0, v) : v \in V \,\} \cup \{\, (v, e) \in V \times E : v \in e \,\}.$$

Now, $G = (V, E)$ has a vertex cover of size k iff $G' = (V', E')$ has a directed dominating set of size $k + 1$.

Assume that there is some k vertex cover $C \subseteq V$ in G. Then $v_0 \cup C$ is a directed dominating set in G', because v_0 dominates itself as well as all $v \in V$, and since each $e \in E$ is incident to some $v \in C$, C dominates E in G'.

Now let D be a directed dominating set in G' of size $k+1$. Since $|V| \geq k+2$, $v_0 \in D$, because otherwise a node in V would not be dominated. Moreover, we can assume that $D \cap E = \emptyset$, because each $e \in E$ can only dominate itself in G'. It is thus always safe to pick a predecessor of e instead. But then, each $e \in E$ is dominated by some $v \in D \cap V$, and thus $D \cap V$ is a vertex cover in G.

Finally, G' is a DAG with K-width two, since there are only two paths from v_0 to each $e \in E$, only one path from v_0 to each $v \in V$ and only one path from each $v \in V$ to each $e \in E$. Likewise, the DAG-depth of G' is at most three.

Note that the same construction also can be used to prove hardness for the DiSTP problem. The dominating set implied by a vertex cover forms a Steiner tree of size $1 + k + n$, by connecting all $e \in E$ via nodes in D to the root v_0.

Moreover, any Steiner tree T that connects v_0 to all $e \in E$ implies a vertex cover $V(T) \cap V$, since each node in E must be connected by a node $v \in V$ with $v \in e$. Moreover, any such Steiner tree T of cost at most $k + |E|$ contains at most k nodes from V, and thus $V(T) \cap V$ is a vertex cover of size k in G. □

Proposition 3.5. *The (unit cost)* DiSTP *problem can be formulated as a* $\mathit{LinEMSO}_1$ *optimization problem, and hence* DiSTP *is fixed parameter tractable when parameterized by bi-rank-width.*

Proof. Let G be a digraph, and $T \subseteq V(G)$, $r \in V(G) \setminus T$. Though DiSTP problem optimizes over the number of edges (recall unit cost!) of a Steiner tree $S \subseteq G$ rooted from r and spanning T, there is a simple equality; $|E(S)| = |V(S)| - 1$. Hence we can, instead, minimize the cardinality of $X = V(S)$ such that X induces in G directed paths from r to all vertices of T.

Similarly to Lemma 5.14, we can thus write (with constants r and T)

$$\forall t \in T \, \forall Z \subseteq X \big((r \in Z \wedge t \notin Z) \rightarrow \exists x \in Z, y \in X \setminus Z \; \mathit{arc}(x, y) \big). \qquad \square$$

Proofs for Section 3.3 (MaxDiCut)

Theorem 3.6. *The* MaxDiCut *problem is NP-hard on a digraph G even if G is restricted to be acyclic (implying directed tree-width, DAG-width and Kelly-width, and cycle rank* 1*) of K-width* 4608 *and DAG-depth* 11*.*

Proof. To verify that the digraph which is the result of the [LKM08] reduction from not-all-equal (NAE) 3SAT has bounded DAG-depth and K-width, we need to slightly modify the construction.

First we may assume the the input instance ϕ of NAE-3SAT contains no clause with both positive and negative occurrence of the same variable. If this is not so, we can remove all such clauses, as they are always satisfied in the NAE-SAT sense. Moreover, we can also assume that each variable occurs at most 4 times in the input formula. If not, we can replace the k different occurrences of a variable x with k fresh variables $x_1, \ldots, x_k$ and add the following clauses to the formula: $(x_1 \vee \neg x_2 \vee \neg x_2) \wedge (x_2 \vee \neg x_3 \vee \neg x_3) \wedge \ldots \wedge (x_k \vee \neg x_1 \vee \neg x_1)$. It is not hard to see that the new formula is satisfied (in the NAE sense) iff the original formula was satisfied, and every variable occurs at most four times. Moreover, the size of this new formula is linear in the size of ϕ.

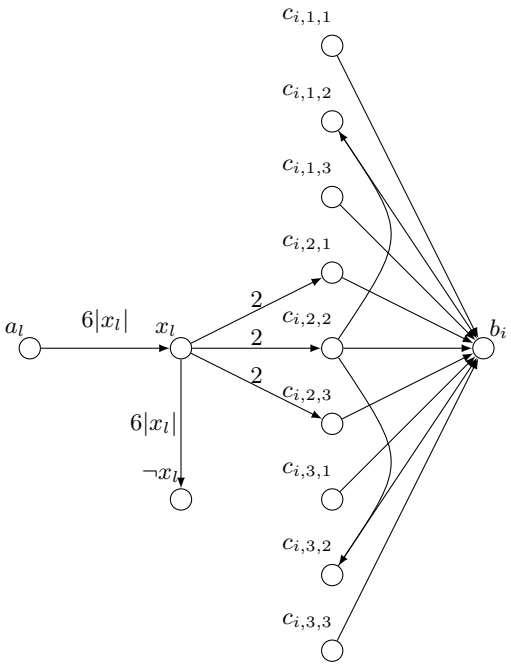

Fig. 2. MaxDiCut reduction gadget

Fig. 2 shows a part of the resulting graph for formula ϕ, a variable x_l and a clause c_i, which contains a positive occurrence of x_l on the second position. The labels on the edges show the weight of the given edge (unlabelled edges have weight 1). $|x_l|$ is the number of occurrences of the variable x_l in the formula ϕ (at most four, as argued above).

To obtain an unweighted graph we use the construction from [LKM08, Theorem 3]. This construction first replaces each edge (u, v) of weight k by k parallel edges, and then replaces each parallel edge by a path of length 3 (two fresh vertices are added for each such edge).

To compute K-width we notice that the highest number of paths between some a_i and b_j and can be at most $6|x_i| * 6|x_i| * 2(1 + 2 + 1) \leq 4608$, since $|x_i| \leq 4$. Finally, the DAG-depth is bounded by the length of longest path, which is $3 + 3 + 3 + 1 + 1 = 11$. □

Theorem 3.7. *The unweighted* MaxDiCut *problem on a digraph G of bi-rank-width t is polynomially solvable for every fixed t (i.e. it belongs to the class XP).*

Proof. (sketch) We give an XP dynamic programming algorithm running on a bi-rank-width parse tree (cf. the appendix of Section 2.1 for the terminology).

We use shortcut notation $arcs(G; V_0, V_1) = \{\, (u, v) \in E(G) : u \in V_0, v \in V_1 \,\}$. Given two t-labeled digraphs $\bar{G}_1, \bar{G}_2$ and mappings $\varphi_i \colon V(G_i) \to \{0, 1\}$ where $i = 1, 2$ (here φ_i gives a partition of $V(G_i)$ into $V_0 = \varphi_i^{-1}(0)$ and $V_1 = \varphi_i^{-1}(1)$), we define an equivalence relation: $(\bar{G}_1, \varphi_1) \approx (\bar{G}_2, \varphi_2)$ if, and only if, the following holds for all t-bi-labeled digraphs $\tilde{H}$ and all mappings $\psi \colon V(H) \to \{0, 1\}$

$$\begin{aligned} &\left|arcs\left(\bar{G}_1 \otimes \tilde{H};\, \varphi_1^{-1}(0), \psi^{-1}(1)\right)\right| + \left|arcs\left(\bar{G}_1 \otimes \tilde{H};\, \psi^{-1}(0), \varphi_1^{-1}(1)\right)\right| = \\ = &\left|arcs\left(\bar{G}_2 \otimes \tilde{H};\, \varphi_2^{-1}(0), \psi^{-1}(1)\right)\right| + \left|arcs\left(\bar{G}_2 \otimes \tilde{H};\, \psi^{-1}(0), \varphi_2^{-1}(1)\right)\right|. \end{aligned}$$

In informal words, the relation $\approx$ captures "all necessary information from $\bar{G}_1$" needed to find an optimal solution to MaxDiCut on any (bigger) $\bar{G}_1 \otimes \tilde{H}$.

Let T be a t-bi-labeling (bi-rank-width) parse tree of the input digraph G, constructed from Theorem 2.5 (Lemma 5.3). Our algorithm processes T in the leaves to root direction. At every node s of T, parsing a t-labeled subdigraph $\bar{G}_s$, and for every equivalence class $\mathcal{C}$ of $\approx$, we remember a mapping $\varphi \colon V(G_s) \to \{0, 1\}$ achieving maximum cardinality of $arcs\big(G_s; \varphi^{-1}(0), \varphi^{-1}(1)\big)$ among all $(\bar{G}_s, \varphi) \in \mathcal{C}$. This information can be easily combined from the two descendants in our parse tree processing. The maximum value (over all classes of $\approx$) recorded at the root of T is then the optimal solution.

It remains to bound the number of classes of $\approx$. From the definition of t-bi-labeling join operator $\otimes$ (cf. the appendix of Section 2.1), we straightforwardly derive the following claim: Let a *signature* of $(\bar{G}, \varphi)$, where $\bar{G} = (G, lab)$ is a t-labeled digraph, be the pair of multisets $\langle \{\, lab(x) : x \in \varphi^{-1}(0) \,\}, \{\, lab(x) : x \in \varphi^{-1}(1) \,\} \rangle$. If $(\bar{G}_1, \varphi_1)$ and $(\bar{G}_2, \varphi_2)$ have the same signature, then $(\bar{G}_1, \varphi_1) \approx (\bar{G}_2, \varphi_2)$.

The total number of signatures for t-labeled n-vertex digraphs is clearly at most $n^{2 \cdot 2^t}$ since every $lab(x) \in GF(2)^t$ and we record the multiplicities of all labels. Hence our above outlined algorithm runs in time $n^{O(2^t)}$ which is polynomial in n for every fixed t. □

Proofs for Section 3.4 (OCN)

Theorem 3.8. *The problem (4-OCN) to decide whether a digraph G satisfies $\chi_o(G) \leq 4$ is NP-complete even if G is acyclic of K-width 3 and DAG-depth 5.*

Proof. We use the following easy claim from [CD06] as the starting point of our reduction: Let R be the digraph on the right-hand side of Fig. 3, and Q be the acyclic digraph on the left-hand side. Then every oriented 4-colouring of Q must induce a homomorphism into R such that b is mapped to B and f_1, f_2 are both mapped to F.

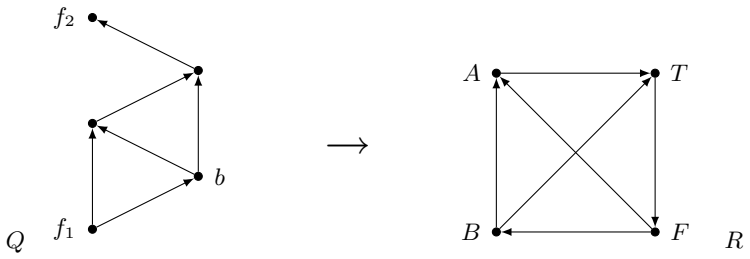

Fig. 3. Forcing a 4-colouring homomorphism

We reduce from NP-complete not-all-equal (NAE) 3SAT problem, which has an input CNF formula φ with exactly three literals in each clause, and the question is whether φ has a satisfying assignment such that no clause receives three times true. We replace each variable x of φ with a gadget depicted in Fig. 4 left, consisting of a copy of Q, two arcs leaving the copy of vertex b into new vertices p and n, a new path of length 5 from p to n, and the necessary number of terminals for the x and $\neg x$ literals occurring in φ, each adjacent from p or n, respectively. Then we replace each clause C of φ with a gadget depicted in Fig. 4 right, consisting of three directed paths of lengths $3, 4, 5$, with a common source. The ends of these paths are the terminals for the literals of C.

Let G_φ be the digraph obtained from all these variable and clause gadgets (pairwise disjoint so far) by identifying all the pairs of corresponding (in φ) literal terminals. We claim that φ is NAE satisfiable if and only if the oriented chromatic number of G_φ is 4. It follows from the following sequence of claims:

- Any oriented 4-colouring of G_φ is a homomorphism into the above digraph R.
- In any homomorphism of the variable gadget into R, the vertices p, n are mapped into $\{A, T\}$. Furthermore, the colours of p and n must be distinct.

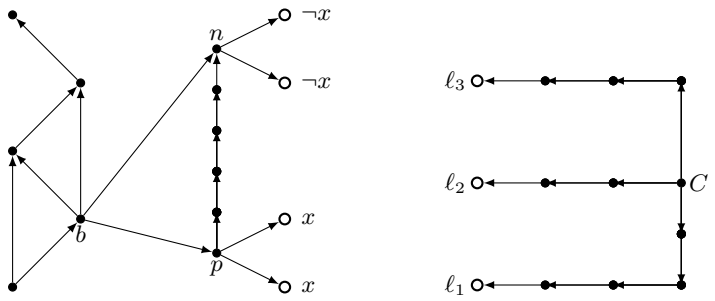

Fig. 4. Variable and clause gadgets for 4-OCN reduction

Hence all the x-terminals of the gadget are mapped to T and all the $\neg x$-terminals are mapped to F (meaning x is valued *true*), or vice versa (meaning x is valued *false*).

- A simple case-analysis shows that any homomorphism of the clause gadget into R such that ℓ_1, ℓ_2, ℓ_3 are mapped into $\{T, F\}$ has an additional property that not all three colours of ℓ_1, ℓ_2, ℓ_3 are the same (meaning that this clause is *NAE satisfied*).
- On the other hand, for both possible surjective mappings $p\colon \{x, \neg x\} \to \{T, F\}$ there exist homomorphisms of the variable gadget into R extending p. Similarly for all surjective mappings $q\colon \{\ell_1, \ell_2, \ell_3\} \to \{T, F\}$ there exist homomorphisms of the clause gadget into R extending q.

Secondly, we claim that G_φ has K-width 3 and DAG-depth 5. Since all the terminals in our construction of the acyclic digraph G_φ are sinks, it is enough to verify the claimed properties for each gadget separately. The K-width bound is easy; we get up to three distinct paths between two vertices in a copy of Q (Fig. 3). We now show a winning strategy for the cops on a variable gadget in 5 moves. In the first two moves, cops land on b and p, and then the robber is easily caught on one of the remaining directed paths of length ≤ 6. For a clause gadget, just 4 moves suffice when the first cop lands on C. □

Proposition 3.9. *The problem (c-*OCN*) to decide $\chi_o(G) \leq c$ on an input digraph G of bi-rank-width t is fixed parameter tractable with parameters c and t.*

Proof. We write an MSO_1 formula

$$\exists X_1, \ldots, X_c \quad \Bigg[\bigwedge_{i=1,\ldots,c} \forall x, y \in X_i \big(\neg \mathit{arc}(x, y)\big) \\ \wedge \bigwedge_{i,j=1,\ldots,c} \forall x, y \in X_i, z, t \in X_j \big(\mathit{arc}(x, z) \to \neg \mathit{arc}(t, y)\big) \Bigg]$$

which describes valid oriented colouring classes $X_1, \ldots X_c$ in a graph G. Hence the result follows from Theorem 3.6. □

At last we remark that, although there is an XP algorithm computing the chromatic number of a given graph of bounded rank-width, it is open whether such an algorithm could exist for computing the oriented chromatic number of a digraph of bounded bi-rank-width. It seems that the known "undirected" algorithm does not extend in this way.

Proofs for Section 3.6 (ϕ-MSO_1MC)

Theorem 3.12 (cf. [CMR00], and [Kan08,GH08]).
Every LinEMSO$_1$ optimization problem is fixed parameter tractable when restricted to digraphs of bi-rank-width t, with a parameter t.

Proof. (sketch) Given an input digraph G of bi-rank-width t, we first use Theorem 2.5 to compute a width-t bi-rank-decomposition of G, and then construct an expression X_G of clique-width $\leq 2^{t+1}-1$ for G using [Kan08, Proposition 5.3]. Now, although the formulations in [CMR00] speak only about FPT solvability of LinEMSO$_1$ problems on undirected graphs (their τ_1 graph structure has a symmetric adjacency relation) of bounded clique-width, there is no apparent mathematical reason why not to extend the whole interpretation scheme there to digraphs. Therefore, [CMR00] (indirectly) implies our theorem.

Alternative proof. Nevertheless, the indirect interpretability approach (based on [CMR00]) to Theorem 3.12 has some disadvantages. First, there is no apparent explicit algorithm behind it, and no "nice" estimate of run-time dependency on t for particular problems except a generic "tower of exponents". Second, the clique-width parameter in the above reduction may grow up to exponentially in t which is not good in applications.

We propose another, more explicit approach to proving Theorem 3.12, based on the bi-labeling parse trees of [Kan08] and the proof method of [GH08, Theorem 4.2] (which has been an alternative to [CMR00] on undirected graphs of bounded rank-width) Given an input digraph G of bi-rank-width t, we first use Theorem 2.5 to compute a width-t bi-rank-decomposition of G, and then we translate this decomposition into a t-bi-labeling parse tree, e.g. in quadratic time using the method of [GH08, Theorem 2.2].

Now, with the same "automata–regularity" tools as used in [GH08, Theorem 4.2], we prove (constructively) the following: For every MSO_1 formula φ and fixed t, there is a finite tree automaton accepting exactly those t-bi-labeling parse trees T giving a digraph $\bar{G}_T$ such that $G_T \models \varphi$ (when φ has free variables, we naturally consider $\bar{G}_T$ equipped with interpretations for these free variables).

In a dynamic processing of the input labeling parse tree, we then keep track only of suitable "optimal" representatives of all possible interpretations of the free variables in φ, indexed by the states of the automaton. The overall running time is $O(f(t)\cdot n^3)$ for some computable f depending on the problem (on φ). □

Proposition 3.13. *The MSO$_1$ model checking problem is NP-hard even when restricted to acyclic digraphs of K-width* 1 *and DAG-depth* 2.

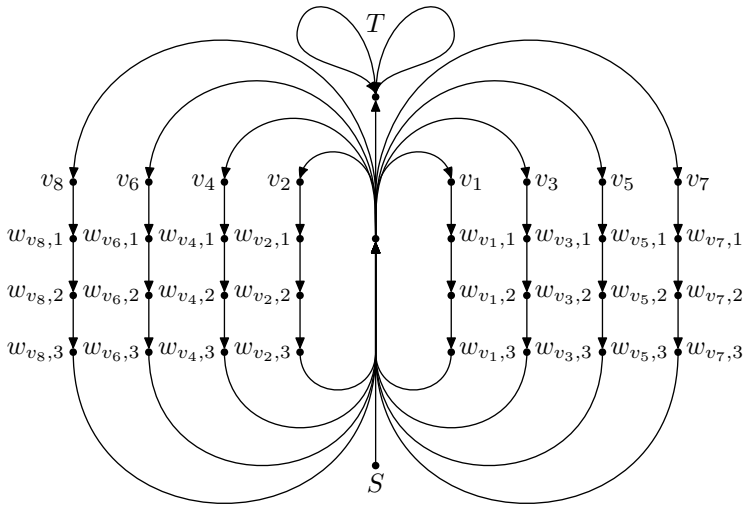

Fig. 5. The construction used in Theorem 3.14

Proof. We show an MSO_1 interpretation of undirected graphs in suitable digraphs. Given a graph H without isolated vertices, we construct an acyclic digraph G of K-width 1 and DAG-depth 2: For every edge $e = uv$ of H, we add a new vertex x_e and replace e with two arcs ux_e, vx_e. Notice that G has no directed path on 3 vertices. A vertex v of H can be then identified in G with $\exists x(\mathit{arc}(v, x)$, and a predicate $\mathit{edge}(u, v)$ can be written as $\exists x(\mathit{arc}(u, x) \wedge \mathit{arc}(v, x))$. The claim follows since MSO_1 model checking is NP-hard on undirected graphs. □

Proofs for Section 3.7 (ϕ-LTLMC)

We briefly fix the LTL-notation used in the following proofs. Atomic properties do hold on vertices (states). The Boolean connectors are as usual, the operators *next* and *eventually (in the future)* are denoted by capital letters X and F. We assume that if a run reaches a sink, it repeats its symbol infinitely often to avoid the existence of finite runs. For clarity, this will be made explicit with self-loops in the figures.

Theorem 3.14. *The ϕ-LTLMC problem is coNP-hard even when the input digraph is restricted to have K-width 1, DAG-depth 4, and bi-rank-width* 2.

Proof. We use a reduction from the DS (undirected dominating set) problem, which is a folklore NP-complete problem even when the input is restricted to cubic graphs. Let $G = (V, E)$, $k \in \mathbb{N}$ be an input instance for DS such that the graph G has all degrees 3. We construct the following instance of ϕ-LTLMC with an underlying digraph $G' = (V', E')$:

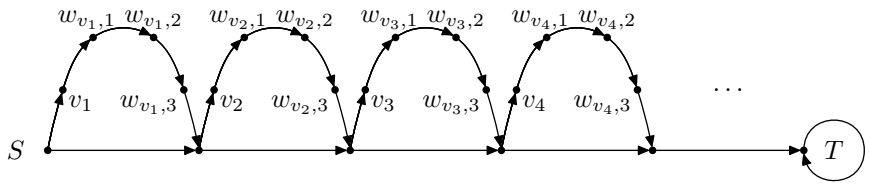

Fig. 6. The construction used in Theorem 3.15

For each $v \in V$, we define $V'_v = \{u_v, u_{v,1}, u_{v,2}, u_{v,3}\}$. Then $V' = \{s, t, c\} \cup \bigcup_{v \in V} V'_v$. We let properties S hold in s, and T in t. If the neighbours of each v in G are $w_{v,1}$, $w_{v,2}$, $w_{v,3}$, then let v hold in u_v, and $w_{v,i}$ hold in $u_{v,i}$ for $i = 1, 2, 3$. Edges are added as follows.

$$E' = \{(s,c),(c,t)\} \cup \{\,(c,u_v),(u_{v,3},c) : v \in V\,\} \\ \cup \{\,(u_v,u_{v,1}),(u_{v,1},u_{v,2}),(u_{v,2},u_{v,3}),: v \in V\,\}.$$

Then

$$\mathcal{F} = \neg\,(X^{1+5k+1}\,T \,\wedge \bigwedge\nolimits_{v \in V} Fv)$$

holds on G' iff G does not contain a dominating set of size k. See Figure 5 for an illustration.

Assume there is any dominating set D of size k in G. Then the run (i.e. directed walk in G') R starting in s, following the cycle through each $V'_v \cup \{c\}$ for all $v \in D$, and visiting only t afterwards, does not satisfy $\mathcal{F}$: After $1 + 5k + 1$ steps, the sink t is reached (where T holds), and since C is a dominating set, each $v \in V$ holds at some point in P – namely when V'_w is traversed for some $w \in D$ that dominates v.

Now assume that G contains no dominating set of size k. Then, no run R of length at most $1 + 5k$ can satisfy $\bigwedge_{v \in V} Fv$: Since at most k sets $V'_{v_1}, \ldots, V'_{v_k}$ are visited by R and since the corresponding nodes $v_1, \ldots, v_k \in V$ cannot form a dominating set in G, there is some $w \in V$ such that w does not hold on R. Hence, any path that satisfies $\bigwedge_{v \in V} Fv$ cannot satisfy $X^{1+5k+1}\,t$ at the same time. Therefore, $\mathcal{F}$ holds on G'.

Finally, the digraph G' has clearly K-width 1, and it can be shown to have DAG-depth 4. To prove that the bi-rank-width of G' is at most 2 (recall Section 2.1), we look at the set $Y = \{s, t, c\}$. Then both matrices $\boldsymbol{A}^+_Y$ and $\boldsymbol{A}^-_Y$ have only one nonzero row each, and so $brk_{G'}(Y) = 2$. Furthermore, the subdigraph $G' - Y$ is a collection of paths, and so $brk_{G'}$ can easily be 2-branched on $V' \setminus Y$. Hence the bi-rank-width of G' is at most 2. □

Theorem 3.15. *LTL model checking is coNP-complete on DAGs.*

Proof. For containment in coNP, note that the NTM just has to guess a run on which the formula does not hold. Since there are no loops (except self-loops), the length of this run is polynomially bounded.

For the hardness part, we use a similar construction to the one used in Theorem 3.14. See in Figure 6. Let $G = (V, E)$, $k \in \mathbb{N}$ be an input instance for DS such that the graph G has all degrees 3. We construct the following instance of ϕ-LTLMC with an underlying digraph $G' = (V', E')$:

Again, $V'_v = \{u_v, u_{v,1}, u_{v,2}, u_{v,3}\}$ for each $v \in V$. Then $V' = \{s_0, s_1, \ldots, s_{|V|}, t\} \cup \bigcup_{v \in V} V'_v$. We let S hold in s_0 and T hold in t. If the neighbours of each v in G are $w_{v,1}$, $w_{v,2}$, $w_{v,3}$, then let v hold in u_v, and $w_{v,i}$ hold in $u_{v,i}$ for $i = 1, 2, 3$. Edges are added as follows, assuming $V = \{v_1, \ldots, v_n\}$ (in arbitrary order).

$$E' = \{\, (s_{i-1}, s_i) : 1 \le i \le |V| \,\} \cup \{(s_{|V|}, t)\} \cup \{\, (s_{i-1}, u_{v_i}), (u_{v_i,3}, s_i) : 1 \le i \le |V| \,\}$$
$$\cup \{(u_{v_i}, u_{v_i,1}), (u_{v_i,1}, u_{v_i,2}), (u_{v_i,2}, u_{v_i,3}) : 1 \le i \le |V|\}$$

Then

$$\mathcal{F} = \neg \left(X^{4k+|V|}\, t \wedge \bigwedge\nolimits_{v \in V} Fv \right)$$

holds on G' iff G does not contain a dominating set of size k. The rest follows in the same way as in the proof of Theorem 3.14. □

Proofs for Section 4 (DFVS)

Theorem 4.1. *If a digraph G is given with a directed feedback vertex set of size k, then the* Kernel *problem can be solved in time $O(2^k \cdot |V(G)|^2)$.*

Proof. (sketch) We are actually going to solve the annotated Kernel problem, in which the input is a digraph G and a set $U \subseteq V(G)$, and the task is to find a kernel which is a subset of U.

For a given DAG G and set U, we easily solve annotated Kernel using the following reduction rules:

- The set $K \subseteq V(G)$ of all sink vertices of G (which is acyclic) must be included in the kernel. If $K \not\subseteq U$, then no such kernel exists.
- Hence all the in-neighbours $N^-(K)$ already have an arc into the kernel, and so $N^-(K)$ can be ignored further on. We call the procedure recursively on $G - (K \cup N^-(K))$ and $U \setminus (K \cup N^-(K))$.

We now consider an arbitrary digraph G with a feedback set $S \subseteq V(G)$ of size k, and solve annotated Kernel for G and $U \subseteq V(G)$ as follows. We cycle through all 2^k subsets $Z \subseteq S$, looking for a kernel that intersects S in Z:

- If $Z \not\subseteq U$ or Z is not independent in G, then this iteration fails.
- We include Z into the kernel. Hence all the in-neighbours $N^-(Z)$ already have an arc into the kernel, and so $N^-(Z)$ can be ignored further on. All the out-neighbours $N^+(Z)$ must stay out of the kernel, and so they can simply be removed from the set U.
- Since all the vertices of $S \setminus Z$ are not in the kernel, the incoming edges of $S \setminus Z$ have no influence on the problem, and so they can be removed from the digraph G, making a new digraph G'.

– Finally, the digraph $G' - Z$ is acyclic. Therefore, we call the previous procedure on $G' - (Z \cup N^-(Z))$ and $U \setminus (Z \cup N^-(Z) \cup N^+(Z))$.

It remains to straightforwardly verify that this procedure gives a correct answer. □