Parameterized Complexity of DAG Partitioning

René van Bevern[1⋆], Robert Bredereck[1⋆⋆], Morgan Chopin[2⋆⋆⋆], Sepp Hartung[1], Falk Hüffner[1†], André Nichterlein[1], and Ondřej Suchý[3‡]

[1] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
{rene.vanbevern, robert.bredereck, sepp.hartung, falk.hueffner, andre.nichterlein}@tu-berlin.de
[2] LAMSADE, Université Paris-Dauphine, France
chopin@lamsade.dauphine.fr
[3] Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic.
ondrej.suchy@fit.cvut.cz

Abstract. The goal of tracking the origin of short, distinctive phrases (*memes*) that propagate through the web in reaction to current events has been formalized as DAG Partitioning: given a directed acyclic graph, delete edges of minimum weight such that each resulting connected component of the underlying undirected graph contains only one sink. Motivated by NP-hardness and hardness of approximation results, we consider the parameterized complexity of this problem. We show that it can be solved in $O(2^k \cdot n^2)$ time, where k is the number of edge deletions, proving fixed-parameter tractability for parameter k. We then show that unless the Exponential Time Hypothesis (ETH) fails, this cannot be improved to $2^{o(k)} \cdot n^{O(1)}$; further, DAG Partitioning does not have a polynomial kernel unless NP $\subseteq$ coNP/poly. Finally, given a tree decomposition of width w, we show how to solve DAG Partitioning in $2^{O(w^2)} \cdot n$ time, improving a known algorithm for the parameter pathwidth.

1 Introduction

The motivation of our problem comes from a data mining application. Leskovec et al. [6] want to track how short phrases (typically, parts of quotations) show up on different news sites, sometimes in mutated form. For this, they collected from 90 million articles phrases of at least four words that occur at least ten times. They then created a directed graph with the phrases as vertices and draw an arc from phrase p to phrase q if p is shorter than q and either p has small edit distance from q (with words as tokens) or there is an overlap of at least 10

⋆ Supported by DFG project DAPA (NI 369/12-1)
⋆⋆ Supported by DFG project PAWS (NI 369/10-1)
⋆⋆⋆ Supported by DAAD.
† Supported by DFG project PABI (NI 369/7-2).
‡ A major part of this work was done while with the TU Berlin, supported by the DFG project AREG (NI 369/9).

To appear in *Proceedings of the 8th International Conference on Algorithms and Complexity (CIAC '13)*, Barcelona, Spain, May, 2013.

consecutive words. Thus, an arc (p, q) indicates that p might originate from q. Since all arcs are from shorter to longer phrases, the graph is a directed acyclic graph (DAG). The arcs are weighted according to the edit distance and the frequency of q. A vertex with no outgoing arc is called a *sink*. If a phrase is connected to more than one sink, its ultimate origin is ambiguous. To resolve this, Leskovec et al. [6] introduce the following problem.

DAG Partitioning [6]
Input: A directed acyclic graph $D = (V, A)$ with positive integer edge weights $\omega : A \to \mathbb{N}$ and a positive integer $k \in \mathbb{N}$.
Output: Is there a set $A' \subseteq A$, $\sum_{a \in A'} \omega(a) \leq k$, such that each connected component in $D' = (V, A \setminus A')$ has exactly one sink?

While the work of Leskovec et al. [6] had a large impact (for example, it was featured in the New York Times), there are few studies on the computational complexity of DAG Partitioning so far. Leskovec et al. [6] show that DAG Partitioning is NP-hard. Alamdari and Mehrabian [1] show that moreover it is hard to approximate in the sense that if P $\neq$ NP, then for any fixed $\varepsilon > 0$, there is no $(n^{1-\varepsilon})$-approximation, even if the input graph is restricted to have unit weight arcs, maximum outdegree three, and two sinks.

In this paper, we consider the parameterized complexity of DAG Partitioning. (We assume familiarity with parameterized analysis and concepts such as problem kernels (see e. g. [4, 7])). Probably the most natural parameter is the maximum weight k of the deleted edges; edges get deleted to correct errors and ambiguity, and we can expect that for sensible inputs only few edges need to be deleted.

Unweighted DAG Partitioning is similar to the well-known Multiway Cut problem: given an undirected graph and a subset of the vertices called the *terminals*, delete a minimum number k of edges such that each terminal is separated from all others. DAG Partitioning in a connected graph can be considered as a Multiway Cut problem with the sinks as terminals and the additional constraint that not all edges going out from a vertex may be deleted, since this creates a new sink. Xiao [8] gives a fixed-parameter algorithm for solving Multiway Cut in $O(2^k \cdot n^{O(1)})$ time. We show that a simple branching algorithm solves DAG Partitioning in the same running time (Theorem 3). We also give a matching lower bound: unless the Exponential Time Hypothesis (ETH) fails, DAG Partitioning cannot be solved in $O(2^{o(k)} \cdot n^{O(1)})$ time (Corollary 1). We then give another lower bound for this parameter by showing that DAG Partitioning does not have a polynomial kernel unless NP $\subseteq$ coNP/poly (Theorem 5).

An alternative parameterization considers the structure of the underlying undirected graph of the input. Alamdari and Mehrabian [1] show that if this graph has pathwidth ϕ, DAG Partitioning can be solved in $2^{O(\phi^2)} \cdot n$ time, and thus DAG Partitioning is fixed-parameter tractable with respect to pathwidth. They ask if DAG Partitioning is also fixed-parameter tractable with respect to the parameter treewidth. We answer this question positively by giving an algorithm based on dynamic programming that given a tree decomposition of width w solves DAG Partitioning in $O(2^{O(w^2)} \cdot n)$ time (Theorem 7).

Due to space constraints, we defer some proofs to a journal version.

2 Notation and Basic Observation

All graphs in this paper are finite and simple. We consider directed graphs $D = (V, A)$ with *vertex set* V and *arc set* $A \subseteq V \times V$, as well as undirected graphs $G = (V, E)$ with vertex set V and *edge set* $E \subseteq \{\{u, v\} \mid u, v \in V\}$. For a (directed or undirected) graph G, we denote by $G \setminus E'$ the subgraph obtained by removing from it the arcs or edges in E'. We denote by $G[V']$ the subgraph of G induced by the vertex set $V' \subseteq V$. The set of out-neighbors and in-neighbors of a vertex v in a directed graph is $N^+(v) = \{u : (v, u) \in A\}$ and $N^-(v) = \{u : (u, v) \in A\}$, respectively. Moreover, for a set of arcs B and a vertex v we let $N_B(v) := \{u \mid (u, v) \in B \text{ or } (v, u) \in B\}$ and $N_B[v] := N_B(v) \cup \{v\}$. The out-degree, the in-degree, and the degree of a vertex $v \in V$ are $d^+(v) = |N^+(v)|$, $d^-(v) = |N^-(v)|$, and $d(v) = d^+(v) + d^-(v)$, respectively. A vertex is *a sink* if $d^+(v) = 0$ and *isolated* if $d(v) = 0$. We say that u *can reach* v (v *is reachable from* u) in D if there is an oriented path from u to v in D. In particular, u is always reachable from u. Furthermore, we use *connected component* as an abbreviation for *weakly connected component*, that is, a connected component in the underlying undirected graph. The *diameter* of D is the maximum length of a shortest path between two different vertices in the underlying undirected graph of D.

The following easy to prove structural result about minimal DAG Partitioning solutions is fundamental to our work.

Lemma 1. *Any minimal solution for* DAG Partitioning *has exactly the same sinks as the input.*

Proof. Clearly, no sink can be destroyed. It remains to show that no new sinks are created. Let $D = (V, A)$ be a DAG and $A' \subseteq A$ a minimal set such that $D' = (V, A \setminus A')$ has exactly one sink in each connected component. Suppose for a contradiction that there is a vertex t that is a sink in D' but not in D. Then there exists an arc $(t, v) \in A$ for some $v \in V$. Let C_v and C_t be the connected components in D' containing v and t respectively and let t_v be the sink in C_v. Then, for $A'' := A' \setminus \{(t, v)\}$, $C_v \cup C_t$ is one connected component in $(V, A \setminus A'')$ having one sink t_v. Thus, A'' is also a solution with $A'' \subsetneq A'$, a contradiction. □

3 Classical Complexity

Since DAG Partitioning is shown to be NP-hard in general, we determine whether relevant special cases are efficiently solvable. Alamdari and Mehrabian [1] already showed that DAG Partitioning is NP-hard even if the input graph has two sinks. We complement these negative results by showing that the problem remains NP-hard even if the diameter or the maximum degree is a constant.

Theorem 1. DAG Partitioning *is solvable in polynomial time on graphs of diameter one, but NP-complete on graphs of diameter two.*

Theorem 1 can be proven by reducing from general DAG Partitioning and by adding a gadget that ensures diameter two.

Theorem 2. DAG Partitioning *is solvable in linear time if D has maximum degree two, but NP-complete on graphs of maximum degree three.*

Proof. Any graph of maximum degree two consists of cycles or paths. Thus, the underlying graph has treewidth at most two and we can therefore solve the problem in linear time using Theorem 7.

We prove the NP-hardness on graphs of maximum degree three. To this end, we use the reduction from Multiway Cut to DAG Partitioning by Leskovec et al. [6]. In the instances produced by this reduction, we then replace vertices of degree greater than three by equivalent structures of maximum degree three.

Multiway Cut
Input: An undirected graph $G = (V, E)$, a weight function $w : E \to \mathbb{N}$, a set of terminals $T \subseteq V$, and an integer k.
Output: Is there a subset $E' \subseteq E$ with $\sum_{e \in E'} w(e) \leq k$ such that the removal of E' from G disconnects each terminals from all the others?

We first recall the reduction from Multiway Cut to DAG Partitioning. Let $I = (G = (V, E), w, T, k)$ be an instance of Multiway Cut. Since Multiway Cut remains NP-hard for three terminals and unit weights [3], we may assume that $w(e) = 1$ for all $e \in E$ and $|T| = 3$. We now construct the instance $I' = (D = (V', E'), k')$ of DAG Partitioning from I as follows. Add three vertices r_1, r_2, r_3 forming the set V_1, a vertex v' for each vertex $v \in V$ forming the set V_2, and a vertex $e_{\{u,v\}}$ for every edge $\{u, v\} \in E$ forming the set V_3. Now, for each terminal $t_i \in T$ insert the arc (t'_i, r_i) in E'. For each vertex $v \in V \setminus T$, add the arcs (v', r_i) for $i = 1, 2, 3$. Finally, for every edge $\{u, v\} \in E$ insert the arcs $(e_{\{u,v\}}, u')$ and $(e_{\{u,v\}}, v')$. Set $k' = k + 2(n - 3)$. We claim that I is a yes-instance if and only if I' is a yes-instance.

Suppose that there is a solution $S \subseteq E$ of size at most k for I. Then the following yields a solution of size at most k' for I': If a vertex $v \in V$ belongs to the same component as terminal t_i, then remove every arc (v', r_j) with $j \neq i$. Furthermore, for each edge $\{u, v\} \in S$ remove one of the two arcs $(e_{\{u,v\}}, u')$ and $(e_{\{u,v\}}, v')$. One can easily check that we end up with a valid solution for I'. Conversely, suppose that we are given a minimal solution of size at most k' for I'. Notice that one has to remove at least two of the three outgoing arcs of each vertex $v' \in V_2$ and that we cannot discard all three because, contrary to Lemma 1, this would create a new sink. Thus, we can define the following valid solution for I: remove an edge $\{u, v\} \in E$ if and only if one of the arcs $(e_{\{u,v\}}, u')$ and $(e_{\{u,v\}}, v')$ is deleted. Again the correctness can easily be verified.

It remains now to modify the instance I' to get a new instance I'' of maximum degree three. For each vertex $v \in V'$ with $|N^-(v)| = |\{w_1, \ldots, w_{d^-(v)}\}| \geq 2$, do the following: For $j = 2, \ldots, d^-(v)$ remove the arc (w_j, v) and add the vertex w'_j together with the arc (w_j, w'_j). Moreover, add the arcs (w_1, w'_2), $(w_{d^-(v)}, v)$, and (w'_j, w'_{j+1}) for each $j = 2, \ldots, d^-(v) - 1$. Now, every vertex has maximum degree four. Notice that, by Lemma 1, among the arcs introduced so far, only the arcs (w_j, w'_j) can be deleted, as otherwise we would create new sinks. The correspondence between deleting the arc (w_j, w'_j) in the modified instance and

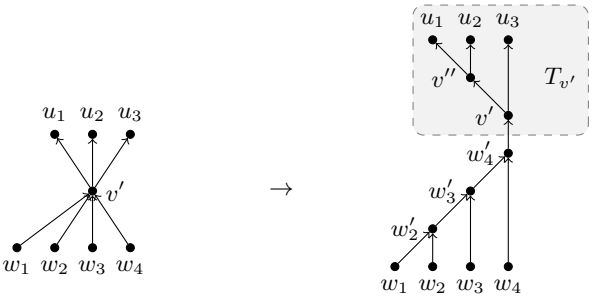

Fig. 1. Construction of the "tree structure" for a vertex $v' \in V_2$.

deleting the arc (w_j, v) in I' is easy to see. Thus the modified instance is so far equivalent to I. Notice also that the vertices in I'' that have degree larger than three are exactly the degree-four vertices in V_2. In order to decrease the degree of these vertices, we carry out the following modifications. For each vertex $v' \in V_2$, with $N^+(v') = \{u_1, u_2, u_3\}$, remove (v', u_1) and (v', u_2), add a new vertex v'', and insert the three arcs (v', v''), (v'', u_1), and (v'', u_2) (see Figure 1). This concludes the construction of I'' and we now prove the correctness. Let $T_{v'}$ be the subgraph induced by $\{v', v'', u_1, u_2, u_3\}$ where $v' \in V_2$. It is enough to show that exactly two arcs from $T_{v'}$ have to be removed in such a way that there remains only one path from v' to exactly one of the u_i. Indeed, we have to remove at least two arcs: otherwise, two sinks will belong to the same connected component. Next, due to Lemma 1, it is not possible to remove more than two arcs from $T_{v'}$. Moreover, using again Lemma 1, the two discarded arcs leave a single path from v' to exactly one of the u_i. This completes the proof. □

4 Parameterized Complexity: Bounded Solution Size

In this section, we investigate the influence of the parameter solution size k on the complexity of DAG Partitioning. To this end, notice that in Lemma 1 we proved that any minimal solution does not create new sinks. Hence, the task is to separate at minimum cost the existing sinks by deleting arcs without creating new ones. Note that this is very similar to the Multiway Cut problem: In Multiway Cut the task is to separate at minimum cost the given terminals. Multiway Cut was shown by Xiao [8] to be solvable in $O(2^k \min(n^{2/3}, m^{1/2})nm)$ time. However, the algorithm relies on minimum cuts and is rather complicated. In contrast, by giving a simple search tree algorithm running in $O(2^k \cdot n^2)$ time for DAG Partitioning, we show that the additional constraint to not create new sinks makes the problem arguably simpler.

Our search tree algorithm exploits the fact that no new sinks are created in the following way: Assume that there is a vertex v with only one outgoing arc pointing to a vertex u. Then, for any minimal solution, u and v are in the same connected component. This leads to the following data reduction rule, which enables us to provide the search tree algorithm. It is illustrated in Figure 2.

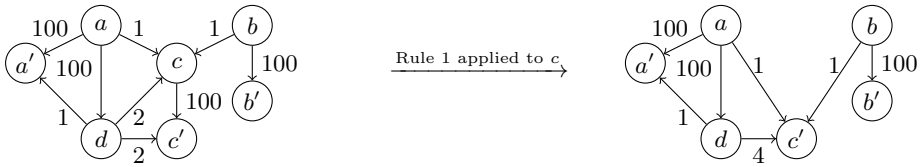

Fig. 2. Exemplary DAG Partitioning instance. Reduction Rule 1 transforms the instance into an equivalent instance: All paths from any vertex to c' can be disconnected with the same costs as before. In every solution with costs less than 100, a must be in the same component as a' and b must be in the same component as b'. Let $S = \{a', b', c'\}$ be the sinks. Furthermore, d is a sink in $D[V \setminus S]$ which must be disconnected from all but one sink. However, since a is adjacent to d removing the cheapest arc (d, a') does not lead to a solution, because a' and c' would remain in the same component. In contrast, by removing (a, c'), (b, c') and (d, c') one obtains the unique solution, which has cost 6.

Reduction Rule 1 *Let $v \in V$ be a vertex with outdegree one and u its unique out-neighbor. Then for each arc $(w, v) \in A$, add an arc (w, u) with the same weight. If there already is an arc (w, u) in A, then increase the weight of (w, u) by $\omega(w, v)$. Finally, delete v.*

The correctness of this rule follows from the discussion above. Clearly, to a vertex $v \in V$, it can be applied in $O(n)$ time. Thus, in $O(n^2)$ time, the input graph can be reduced until no further reduction by Reduction Rule 1 is possible. Thereafter, each vertex has at least two outgoing arcs and, thus, there is a vertex that has at least two sinks as out-neighbors. Exactly this fact we exploit for a search tree algorithm, yielding the following theorem:

Theorem 3. DAG Partitioning *can be solved in $O(2^k \cdot n^2)$ time.*

Proof. Since each connected component can be treated independently, we assume without loss of generality that the input graph D is connected. Moreover, we assume that Reduction Rule 1 is not applicable to D.

Let S be the set of sinks in D. If all vertices are sinks, then D has only one vertex and we are done. Otherwise, let r be a sink in $D[V \setminus S]$ (such a sink exists, since any subgraph of an acyclic graph is acyclic). Then the d arcs going out of r all end in sinks, that is, r is directly connected to d sinks. Since Reduction Rule 1 is not applicable, $d > 1$, and at most one of the d arcs may remain. We recursively branch into d cases, each corresponding to the deletion of $d - 1$ arcs. In each branch, k is decreased by $d - 1$ and the recursion stops as soon as it reaches 0. Thus, we have a branching vector (see e. g. [7]) of $(\underbrace{d-1, d-1, \ldots, d-1}_{d})$, which yields a branching number not worse than 2, since $2^{d-1} \geq d$ for every $d \geq 2$. Therefore, the size of the search tree is bounded by $O(2^k)$. Fully executing Reduction Rule 1 and finding r can all be done in $O(n^2)$ time. □

Limits of Kernelization and Parameterized Algorithms. In the remainder of this section we investigate the theoretical limits of kernelization and

parameterized algorithms with respect to the parameter k. Specifically, we show that unless the exponential time hypothesis (ETH) fails, DAG Partitioning cannot be solved in subexponential time, that is, the running time stated in Theorem 3 cannot be improved to $2^{o(k)}\text{poly}(n)$. Moreover, by applying a framework developed by Bodlaender et al. [2] we prove that DAG Partitioning, unless NP $\subseteq$ coNP/poly, does not admit a polynomial kernel with respect to k.

Towards both results, we first recall the Karp reduction (a polynomial-time computable many-to-one reduction) from 3-SAT into DAG Partitioning given by Alamdari and Mehrabian [1]. The 3-SAT problem is, given a formula F in conjunctive normal form with at most three literals per clause, to decide whether F admits a satisfying assignment.

Lemma 2 ([1, Sect. 2]). *There is a Karp reduction from* 3-SAT *to* DAG Partitioning.

Proof. We briefly recall the construction given by Alamdari and Mehrabian [1].

Let φ be an instance of 3-SAT with the variables $x_1, \ldots, x_n$ and the clauses $C_1, \ldots C_m$. We construct an instance (D, ω, k) with $k := 4n + 2m$ for DAG Partitioning that is a yes-instance if and only if φ is satisfiable. Therein, the weight function ω will assign only two different weights to the arcs: A *normal arc* has weight one and a *heavy arc* has weight $k + 1$ and thus cannot be contained in any solution.

Construction: We start constructing the DAG D by adding the special vertices f, f', t and t' together with the heavy arcs (f, f') and (t, t'). The vertices f' and t' will be the only sinks in D. For each variable x_i, introduce the vertices x_i^t, x_i^f, x_i and $\overline{x_i}$ together with the heavy arcs (t, x_i^t) and (f, x_i^f) and the normal arcs (x_i^t, x_i), $(x_i^t, \overline{x_i})$, (x_i^f, x_i), $(x_i^f, \overline{x_i})$, (x_i, f'), $(\overline{x_i}, f')$, (x_i, t'), and $(\overline{x_i}, t')$. For each clause C, add a vertex C together with the arc (t', C). Finally, for each clause C and each variable x_i, if the positive (or negative) literal of x_i appears in C, then add the arc (C, x_i) ($(C, \overline{x_i})$, resp.). This completes the construction of D.

Correctness: One can prove that (D, ω, k) is a yes-instance for DAG Partitioning if and only if φ is satisfiable.

Limits of Parameterized Algorithms. The Exponential Time Hypothesis (ETH) was introduced by Impagliazzo et al. [5] and states that 3-SAT cannot be solved in $2^{o(n)}\text{poly}(n)$ time, where n denotes the number of variables.

Corollary 1. *Unless the ETH fails,* DAG Partitioning *cannot be solved in* $2^{o(k)}\text{poly}(n)$ *time.*

Proof. The reduction provided in the proof of Lemma 2 reduces an instance of 3-SAT consisting of a formula with n variables to an equivalent instance (D, ω, k) of DAG Partitioning with $k = 4n + 2m$. In order to prove Corollary 1, it remains to show that we can upper-bound k by a linear function in n. Fortunately, this is done by the so-called *Sparsification Lemma* [5], which allows us to assume that the number of clauses in the 3-SAT instance that we reduce from is linearly bounded in the number of variables. □

Limits of Problem Kernelization. We first recall the basic concepts and the main theorem of the framework introduced by Bodlaender et al. [2].

Theorem 4 ([2, Corollary 10]). *If some set $L \subseteq \Sigma^*$ is NP-hard under Karp reductions and L cross-composes into the parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$, then there is no polynomial-size kernel for Q unless NP $\subseteq$ coNP/poly.*

Here, a problem L *cross-composes* into a parameterized problem Q if there is a polynomial-time algorithm that transform the instances $I_1, \ldots, I_s$ of Q into an instance (I, k) for L such that k is bounded by a polynomial in $\max_{i=1}^{s} |I_j| + \log s$ and $(I, k) \in L$ if and only if there is an instance $I_j \in Q$, where $1 \leq j \leq s$. Furthermore, it is allowed to assume that the input instances $I_1, \ldots, I_s$ belong to the same equivalence class of a *polynomial equivalence relation* $R \subseteq \Sigma^* \times \Sigma^*$, that is, an equivalence relation such that it can be decided in polynomial time whether two inputs are equivalent and each set $S \subseteq \Sigma^*$ is partitioned into at most $\max_{x \in S}(|x|)^{O(1)}$ equivalence classes.

In the following we show that 3-SAT cross-composes into DAG Partitioning parameterized by k. In particular, we show how the reduction introduced in Lemma 2 can be extended to a cross-composition.

Lemma 3. 3-SAT *cross-composes to* DAG Partitioning *parameterized by k.*

Proof. Let $\varphi_1, \ldots, \varphi_s$ be instances of 3-SAT. We assume that each of them has n variables and m clauses. This is obviously a polynomial equivalence relation. Moreover, we assume that s is a power of two, as otherwise we can take multiple copies of one of the instances. We construct a DAG $D = (V, A)$ that forms together with an arc-weight function ω and $k := 4n + 2m + 4 \log s$ an instance of DAG Partitioning that is a yes-instance if and only if φ_i is satisfiable for at least one $1 \leq i \leq s$.

Construction: For each instance φ_i let D_i be the DAG constructed as in the proof of Lemma 2. Note that (D_i, k') with $k' := 4n + 2m$ is a yes-instance if and only if φ_i is satisfiable. To distinguish them between multiple instances, we denote the special vertices f, f', t, and t' in D_i by f_i, f_i', t_i, and t_i'. For all $1 \leq i \leq s$, we add D_i to D and we identify the vertices $f_1, f_2, \ldots, f_t$ to a vertex f and, analogously, we identify the vertices $f_1', f_2', \ldots, f_t'$ to a vertex f'. Furthermore, we add the vertices t, t', and t'' together with the heavy arcs (t, t'') and (t, t') to D. As in the proof of Lemma 2, a heavy arc has weight $k + 1$ and thus cannot be contained in any solution. All other arcs, called normal, have weight one.

Add a balanced binary tree O with root in t'' and the leaves $t_1, \ldots, t_s$ formed by normal arcs which are directed from the root to the leaves. For each vertex in O, except for t'', add a normal arc from f. Moreover, add a balanced binary tree I with root t' and the leaves $t_1', \ldots, t_s'$ formed by normal arcs that are directed from the leaves to the root. For each vertex in I, except for t', add a normal arc to f'. This completes the construction of D.

Correctness: One can prove that (D, ω, k) is a yes-instance if and only if φ_i is satisfiable for at least one $1 \leq i \leq s$. □

Lemma 3 and Theorem 4 therefore yield:

Theorem 5. DAG Partitioning *does not have a polynomial-size kernel with respect to k, unless NP $\subseteq$ coNP/poly.*

We note that Theorem 5 can be strengthened to graphs of constant diameter or unweighted graphs.

5 Partitioning DAGs of Bounded Treewidth

In the meme tracking application, edges always go from longer to shorter phrases by omitting or modifying words. It is thus plausible that the number of phrases of some length on a path between two phrases is bounded. Thus, the underlying graphs are tree-like and in particular have bounded treewidth. In this section, we investigate the computational complexity of DAG Partitioning measured in terms of distance to "tree-like" graphs. Specifically, we show that if the input graph is indeed a tree with uniform edge weights, then we can solve the instance in linear time by data reduction rules (see Theorem 6). Afterwards, we prove that this can be extended to weighted graphs of constant treewidth and, actually, we show that DAG Partitioning is fixed-parameter tractable with respect to treewidth. This improves the algorithm for pathwidth given by Alamdari and Mehrabian [1], as the treewidth of a graph is at most its pathwidth.

Warm-Up: Partitioning Trees

Theorem 6. DAG Partitioning *is solvable in linear time if the underlying undirected graph is a tree with uniform edge weights.*

To prove Theorem 6, we employ data reduction on the tree's leaves. Note that Reduction Rule 1 removes all leaves of a tree that have outdegree one and leaves that are the only out-neighbor of their parent. In this case, Reduction Rule 1 can be realized by merely deleting such leaves. In cases where Reduction Rule 1 is not applicable to any leaves, we apply the following data reduction rule:

Reduction Rule 2 *Let $v \in V$ be a leaf with in-degree one and in-neighbor w. If w has more than one out-neighbor, then delete v and decrement k by one.*

We can now prove that as long as the tree has leaves, one of Reduction Rule 1 and Reduction Rule 2 applies, thus proving Theorem 6.

Partitioning DAGs of Bounded Treewidth. We note without proof that DAG Partitioning can be characterized in terms of monadic second-order logic (MSO), hence it is FPT with respect to treewidth (see e. g. [7]). However, the running time bound that this approach yields is far from practical. Therefore, we give an explicit dynamic programming algorithm.

Theorem 7. *Given a tree decomposition of the underlying undirected graph of width w,* DAG Partitioning *can be solved in $O(2^{O(w^2)} \cdot n)$ time. Hence, it is fixed-parameter tractable with respect to treewidth.*

Suppose we are given $D = (V, A)$, ω, k and a tree decomposition (T, β) for the underlying undirected graph of D of width w. Namely, T is a tree and β is a mapping that assigns to every node x in $V(T)$ a set $V_x = \beta(x) \subseteq V$ called *a bag* (for more details on treewidth see e. g. [7]). We assume without loss of generality, that the given tree decomposition is nice and is rooted in a vertex r with an empty bag. For a node x of $V(T)$, we denote by U_x the union of V_y over all y descendant of x, A_x^V denotes $A(D[V_x])$, and A_x^U denotes $A(D[U_x])$.

Furthermore, for a DAG G we define the transitive closure with respect to the reachability as $\mathrm{Reach}^*(G) := (V(G), A(G) \cup \{(u, v) \mid u, v \in V(G), u \neq v$, and there is an oriented path from u to v in $G\})$.

Solution Patterns. Our algorithm is based on leaf to root dynamic programming. The behavior of a partial solution is described by a structure which we call a pattern. Let x be a node of T and V_x its bag. Let P be a DAG with $V(P)$ consisting of V_x and at most $|V_x|$ additional vertices such that each vertex in $V(P) \setminus V_x$ is a non-isolated sink. Let $\mathcal{Q}$ be a partition of $V(P)$ into at most $|V_x|$ sets $Q_1, \ldots, Q_q$, such that each connected component of P is within one set of $\mathcal{Q}$ and each Q_i contains at most one vertex of $V(P) \setminus V_x$. Let R be a subgraph of $P[V_x]$. We call $(P, \mathcal{Q}, R)$ a *pattern* for x. In the next paragraphs, we give an account of how the pattern $(P, \mathcal{Q}, R)$ describes a partial solution.

Intuitively, the DAG P stores the vertices of a bag and the sinks these vertices can reach in the graph modified by the partial solution. The partition $\mathcal{Q}$ refers to a possible partition of the vertices of the graph such that each part is a connected component and each connected component contains exactly one sink. Finally, the graph R is the intersection of the partial solution with V_x.

Formally, a pattern describes a partial solution as follows. Let $A' \subseteq A_x^U$ be a set of arcs such that no connected component of $D_x(A') = D[U_x] \setminus A'$ contains two different sinks in $U_x \setminus V_x$ and every sink in a connected component of $D_x(A')$ which contains a vertex of V_x can be reached from some vertex of V_x in $D_x(A')$. A sink in $U_x \setminus V_x$ is called *interesting* in $D_x(A')$ if it is reachable from at least one vertex of V_x. Let $P_x(A')$ be the DAG on $V_x \cup V'$, where V' is the set of interesting sinks in $D_x(A')$ such that there is an arc (u, v) in $P_x(A')$ if the vertex u can reach the vertex v in the DAG $D_x(A')$. Let $\mathcal{Q}_x(A')$ be the partition of $V_x \cup V'$ such that the vertices u and v are in the same set of $\mathcal{Q}_x(A')$ if and only if they are in the same connected component of $D_x(A')$. Finally, by $R_x(A')$ we denote the DAG $D[V_x] \setminus A'$.

Let $(P, \mathcal{Q}, R)$ be a pattern for x. We say that A' *satisfies* the pattern $(P, \mathcal{Q}, R)$ at x if no connected component of $D_x(A')$ contains two different sinks in $U_x \setminus V_x$, every sink in a connected component of $D_x(A')$ which contains a vertex of V_x can be reached from some vertex of V_x in $D_x(A')$, $P = P_x(A')$ (there is an isomorphism between P and $P_x(A')$ which is identical on V_x, to be precise), $\mathcal{Q}$ is a coarsening of $\mathcal{Q}_x(A')$, and $R = R_x(A')$. Formally, $\mathcal{Q}$ is a coarsening of $\mathcal{Q}_x(A')$ if for each set $Q \in \mathcal{Q}_x(A')$ there exists a set $Q' \in \mathcal{Q}$ such that $Q \subseteq Q'$. Note, that some coarsenings of $\mathcal{Q}_x(A')$ may not form a valid pattern together with $P_x(A')$ and $R_x(A')$.

For each node x of $V(T)$ we have a single table Tab_x indexed by all possible patterns for x. The entry $Tab_x(P, \mathcal{Q}, R)$ stores the minimum weight of a set A' that satisfies the pattern $(P, \mathcal{Q}, R)$ at x. If no set satisfies the pattern, we store ∞. As the root has an empty bag, Tab_r has exactly one entry containing the minimum weight of set A' such that in $D_r(A') = D \setminus A'$ no connected component contains two sinks. Obviously, such a set forms a solution for D. Hence, once the tables are correctly filled, to decide the instance (D, ω, k), it is enough to test whether the only entry of Tab_r is at most k.

The Algorithm. Now we show how to fill the tables. First we initialize all the tables by ∞. By updating the entry $Tab_x(P, \mathcal{Q}, R)$ with m we mean setting $Tab_x(P, \mathcal{Q}, R) := m$ if $m < Tab_x(P, \mathcal{Q}, R)$. For a leaf node x we try all possible subsets $A' \subseteq A_x^U = A_x^V$, and for each of them and for each coarsening $\mathcal{Q}$ of $\mathcal{Q}_x(A')$ we update $Tab_x(P_x(A'), \mathcal{Q}, R_x(A'))$ with $\omega(A')$. Note that in this case, as there are no vertices in $P_x(A') \setminus V_x$, every coarsening of $\mathcal{Q}_x(A')$ forms a valid pattern together with $P_x(A')$ and $R_x(A')$. In the following we assume that by the time we start the computation for a certain node of T, the computations for all its children are already finished.

Consider now the case, where x is a forget node with a child y, and assume that $v \in V_y \setminus V_x$. For each pattern $(P, \mathcal{Q}, R)$ for y we distinguish several cases. In each of them we set $R' = R \setminus \{v\}$. If v is isolated in P and there is a set $\{v\}$ in $\mathcal{Q}$ (case (i)), then we let $P' = P \setminus \{v\}$, $\mathcal{Q}'$ be a partition of $V(P')$ obtained from $\mathcal{Q}$ by removing the set $\{v\}$, and update $Tab_x(P', \mathcal{Q}', R')$ with $Tab_y(P, \mathcal{Q}, R)$. If v is a non-isolated sink and $v \in Q_i \in \mathcal{Q}$ such that $Q_i \subseteq V_y$ (case (ii)), then we update $Tab_x(P, \mathcal{Q}, R')$ with $Tab_y(P, \mathcal{Q}, R)$ ($P' = P, \mathcal{Q}' = \mathcal{Q}$ in this case). If v is not a sink in P and there is no sink in $V(P) \setminus V_y$ such that v is its only in-neighbor (case (iii)), then let $P' = P \setminus \{v\}$ and $\mathcal{Q}'$ be a partition of $V(P')$, obtained from $\mathcal{Q}$ by removing v from the set it is in, and update $Tab_x(P', \mathcal{Q}', R')$ with $Tab_y(P, \mathcal{Q}, R)$. If there is a sink $u \in V(P) \setminus V_y$ such that v is its only in-neighbor and $\{u, v\}$ is a set of $\mathcal{Q}$ (case (iv)), then let $P' = P \setminus \{u, v\}$ and $\mathcal{Q}'$ be a partition of $V(P')$, obtained from $\mathcal{Q}$ by removing the set $\{u, v\}$, and update $Tab_x(P', \mathcal{Q}', R')$ with $Tab_y(P, \mathcal{Q}, R)$. We don't do anything for the patterns $(P, \mathcal{Q}, R)$ which do not satisfy any of the above conditions.

Next, consider the case, where x is an introduce node with a child y, and assume that $v \in V_x \setminus V_y$ and B is the set of arcs of A_x^V incident to v. For each $B' \subseteq B$ and for each pattern $(P, \mathcal{Q}, R)$ for y such that there is a Q_i in $\mathcal{Q}$ with $N_{B'}(v) \subseteq Q_i$ we let $R' = (V_x, A(R) \cup B')$, $D' = (V(P) \cup \{v\}, A(P) \cup B')$, $P' = \text{Reach}^*(D')$ and we distinguish two cases. If $B' = \emptyset$, then for every $Q_i \in \mathcal{Q}$ we let $\mathcal{Q}'$ be obtained from $\mathcal{Q}$ by adding v to the set Q_i and update $Tab_x(P', \mathcal{Q}', R')$ with $Tab_y(P, \mathcal{Q}, R) + \omega(B)$. Additionally, for $\mathcal{Q}'$ obtained from $\mathcal{Q}$ by adding the set $\{v\}$ we also update $Tab_x(P', \mathcal{Q}', R')$ with $Tab_y(P, \mathcal{Q}, R) + \omega(B)$. If B' is non-empty, then let Q_i be the set of $\mathcal{Q}$ with $N_{B'}(v) \subseteq Q_i$ and let $\mathcal{Q}'$ be obtained from $\mathcal{Q}$ by adding v to the set Q_i. We update $Tab_x(P', \mathcal{Q}', R')$ with $Tab_y(P, \mathcal{Q}, R) + \omega(B) - \omega(B')$.

Finally, consider the case that x is a join node with children y and z. For each pair of patterns $(P_y, \mathcal{Q}_y, R)$ for y and $(P_z, \mathcal{Q}_z, R)$ for z such that $\mathcal{Q}_y$ and $\mathcal{Q}_z$

partition the vertices of $V_y = V_z = V_x$ in the same way we do the following. Let D' be the DAG obtained from the disjoint union of P_y and P_z by identifying the vertices in V_x and $P' = \text{Reach}^*(D')$. Let $\mathcal{Q}'$ be a partition of $V(P')$ such that it partitions V_x in the same way as $\mathcal{Q}_y$ and $\mathcal{Q}_z$ and for each $u \in V(P') \setminus V_x$ add u to a set Q_i which contain a vertex v with (v, u) being an arc of P'. It is not hard to see, that there is always exactly one such set Q_i, as there are no arcs between different sets in P_y and P_z. If some set $Q \in \mathcal{Q}'$ contains more than one vertex of $V(P') \setminus V_x$, then continue with a different pair of patterns. Otherwise, we update $Tab_x(P', \mathcal{Q}', R)$ with $Tab_y(P_y, \mathcal{Q}_y, R) + Tab_z(P_z, \mathcal{Q}_z, R) - \omega(A_x^V) + \omega(A(R))$.

6 Outlook

We have presented two parameterized algorithms for DAG Partitioning, one with parameter solution size k and one with parameter treewidth w. In particular the algorithm for the parameter k seems suitable for implementation; in combination with data reduction, this might allow to solve optimally instances for which so far only heuristics are employed [6].

On the theoretical side, one open question is whether we can use the Strong Exponential Time Hypothesis (SETH) to show that there is no $O((2-\varepsilon)^k \text{poly}(n))$ time algorithm for DAG Partitioning. Another question is whether there is an algorithm solving the problem in $O(2^{O(w \log w)} \cdot n)$ or even $O(2^{O(w)} \cdot n)$ time, where w is the treewidth, as the $O(2^{O(w^2)} \cdot n)$ running time of our algorithm still seems to limit its practical relevance.

References

[1] S. Alamdari and A. Mehrabian. On a DAG partitioning problem. In *Proc. 9th WAW*, volume 7323 of *LNCS*, pages 17–28. Springer, 2012.

[2] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Cross-composition: A new technique for kernelization lower bounds. In *Proc. 28th STACS*, volume 9 of *LIPIcs*, pages 165–176. Dagstuhl Publishing, 2011.

[3] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23 (4):864–894, 1994.

[4] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[5] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001.

[6] J. Leskovec, L. Backstrom, and J. M. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *Proc. 15th ACM SIGKDD*, pages 497–506. ACM, 2009.

[7] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[8] M. Xiao. Simple and improved parameterized algorithms for multiterminal cuts. *Theory Comput. Syst.*, 46:723–736, 2010.