On a DAG Partitioning Problem

Soroush Alamdari[1] and Abbas Mehrabian[2]

[1] David R. Cheriton School of Computer Science
University of Waterloo
s26hosse@uwaterloo.ca

[2] Department of Combinatorics and Optimization
University of Waterloo
amehrabi@uwaterloo.ca

Abstract. We study the following DAG Partitioning problem: given a directed acyclic graph with arc weights, delete a set of arcs of minimum total weight so that each of the resulting connected components has exactly one sink. We prove that the problem is hard to approximate in a strong sense: If $\mathcal{P} \neq \mathcal{NP}$ then for every fixed $\epsilon > 0$, there is no $(n^{1-\epsilon})$-approximation algorithm, even if the input graph is restricted to have unit weight arcs, maximum out-degree three, and two sinks. We also present a polynomial time algorithm for solving the DAG Partitioning problem in graphs with bounded pathwidth.

Keywords: DAG Partitioning, Inapproximability, Reduction, 3-SAT, pathwidth, fixed parameter tractable.

1 Introduction

Tracking ideas and memes as they spread and evolve through the web has been studied extensively in recent years. Adar, Zhang, Adamic, and Lukose [2] studied the influence of blogs by analyzing the linking behavior of posts. They asked the question of finding a single source for each topic and assigning a topic to each post. Leskovec, Backstrom, and Kleinberg [9] formulated this question as the following DAG Partitioning problem: Given a directed acyclic graph with arc weights and n nodes, delete a set of arcs of minimum total weight so that each of the resulting connected components has a single sink. Here a *sink* refers to a vertex with no outgoing arc, and by connected components we mean weakly connected components.

In the information retrieval literature, there is a large interest on analyzing the spread of influence and topics throughout objects in the Web (see, e.g., [1,2,4,8,9,10]). Such objects can be blog posts, quotes by people, news headlines, or almost anything that appears on the Web. An immediate question is to assign a source of influence to each of these objects. We can model these objects by a directed graph, in which the vertices represent the objects and the arcs, which are weighted, represent traces of possible influence. These arcs can be extracted from the Web using different methods, such as studying linkage structure or studying occurrences of similar phrases; in the latter case, the weight of an arc is the degree of similarity between phrases. These objects influence each other in an acyclic manner in reality, but this might not be the case in the retrieved graph of influence. Yet, there are ways to find an acyclic subgraph of a given graph without dramatically distorting the structure of the graph (see [3] for instance). Once a directed acyclic graph is formed, the most natural way to assign a source to the objects, seems to be the DAG Partitioning problem we study here.

Leskovec et al. [9] proved the $\mathcal{NP}$-hardness of the DAG Partitioning problem, by reducing the Multiway Cut problem to it (see [6] for the definition of the latter problem). Their reduction converts an instance of the Multiway Cut Problem with k terminals to an instance of the DAG Partitioning problem with k sinks. The Multiway Cut problem can be solved efficiently for 2 terminals, and when there are k terminals, a $(\frac{3}{2} - \frac{1}{k})$-approximation algorithm is known [6]. Thus one might expect similar approximation algorithms for the DAG Partitioning problem as well. In section 2 we show that this is far from being true, and even the simplest case of this problem is hard to approximate. Indeed, assuming $\mathcal{P} \neq \mathcal{NP}$, for every fixed $\epsilon > 0$, there is no $(n^{1-\epsilon})$-approximation algorithm for the DAG Partitioning problem. It is a standard assumption to restrict the graph to have vertices with only constant number of outgoing arcs, since it is true for real instances of the problem. We show that our hardness result holds, even if the input graph is restricted to have unit weights, maximum out-degree three, and just two sinks. We use reduction from the 3-SAT problem.

The *pathwidth* of a directed graph is simply the pathwidth of its undirected underlying graph. In section 3 we study the parameterized version of the problem, with pathwidth chosen as the parameter. We show that the problem is fixed parameter tractable. More precisely, we present an algorithm that given a path decomposition of the graph with width k, solves the DAG Partitioning problem optimally and has running time $2^{O(k^2)}n$. Thus, for graphs with bounded pathwidth, the problem is solvable in linear time. We conclude with an open problem in section 4.

2 The Hardness Result

For a directed acyclic graph D, a *good cut* is a subset C of arcs such that when deleted from D, each of the resulting connected components has a single sink. So the DAG Partitioning problem is just the problem of finding a good cut with minimum weight. The *size* of an instance of a 3-SAT problem is simply the number of clauses in the instance, and in the following we will assume that this number is sufficiently large.

The basic construction we use is the following.

Definition. Let $\mathcal{I}$ be an instance of the 3-SAT problem, and M be a positive integer. Arcs of weight M are called the *heavy* arcs, and the rest of the arcs are called the *light* arcs. The directed graph $D = D(\mathcal{I}, M)$ is the following:

– There are four special vertices t, t', f, f' in D, where (t', t) and (f', f) are heavy arcs.

– For each variable x of $\mathcal{I}$, there are
  - four vertices $x, \overline{x}, x_t, x_f$,
  - two heavy arcs (t', x_t) and (f', x_f), and
  - eight arcs (x_t, x), $(x_t, \overline{x})$, (x, t), $(\overline{x}, t)$, (x_f, x), $(x_f, \overline{x})$, (x, f), and $(\overline{x}, f)$ of weight one (see Figure 1).

– For each clause $L = (a \vee b \vee c)$, there is
  - a vertex L,
  - a heavy arc (t', L), and
  - three arcs $(L, a), (L, b), (L, c)$ of weight one (see Figure 2).

  Note that a, b, c denote literals here, and a vertex is already associated with each of them.

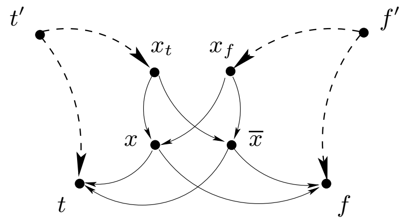

Fig. 1. The gadget corresponding to a variable: heavy arcs are dashed and light arcs are solid.

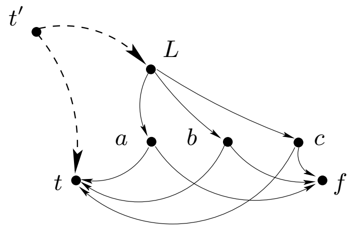

Fig. 2. The gadget corresponding to a clause: heavy arcs are dashed and light arcs are solid.

It is easy to verify that if $\mathcal{I}$ has size $s > 3$, then $D(\mathcal{I}, M)$ has at most $14s$ vertices, $27s$ light arcs and $8s$ heavy arcs. Moreover, $D(\mathcal{I}, M)$ is acyclic and has just two sinks t and f.

Lemma 1. *The instance $\mathcal{I}$ is satisfiable if and only if $D = D(\mathcal{I}, M)$ has a good cut that does not contain any heavy arcs.*

Proof. ($\Rightarrow$) Consider a satisfying assignment of the variables. Build the subset T of vertices of D as following: Put t, t' in T. For each clause L, put L in T. For each variable x, put x_t in T. If x is true then put x in T, otherwise put $\overline{x}$ in T. Do not put any other vertex in T.

Let C be the set of arcs with exactly one endpoint in T. Then C does not contain any heavy arcs. Deleting C results in a directed acyclic graph H with two connected components, with one of them containing the sink t and the other one containing the sink f. For each variable x, exactly one of x and $\overline{x}$ is in T, so none of x_t and x_f is a sink in H. If x is true, then $(x,t),(\overline{x},f)$ are arcs in H, and if x is false, then $(x,f),(\overline{x},t)$ are arcs in H, so none of $x,\overline{x}$ is a sink in H. For each clause $L=(a \vee b \vee c)$, at least one of a,b,c is in T, so L is not a sink in H. Thus it can be verified that C is a good cut that does not contain any heavy arcs.

($\Leftarrow$) Let C be a good cut of minimum size that does not contain any heavy arcs. Let H be the directed acyclic graph obtained from deleting C. We claim that H has two connected components. First, it has at least two components as t and f are sinks in H. If it has a third component (other than the components containing t and f), then let r be a sink in the third component. Let e be an arbitrary outgoing arc from r in G. Then $C \setminus \{e\}$ is a good cut with a smaller size that does not contain any heavy arcs, contradicting the choice of C. Hence H has two connected components and so t and f are the only sinks in H. Denote the components containing t and f by T and F, respectively.

For each variable x, if $x \in T$ then let x be true, and let x be false otherwise. Observe that:

– Since C has no heavy arcs, $t' \in T$ and $f' \in F$.
– For each variable x, we have $x_t \in T$ and $x_f \in F$.
– Since for each variable x, none of x_t or x_f is a sink in H, exactly one of $x,\overline{x}$ is in T and the other one is in F (see Figure 1).
– Since C has no heavy arc, for each clause L, vertex L is in T.
– As vertex L is not a sink in H, at least one of the vertices a,b,c is in T (see Figure 2).

Therefore, this is a satisfying assignment, and the proof is complete. □

Corollary 1. *Let $\mathcal{I}$ be an instance of 3-SAT of size s, where $s > 3$. If $\mathcal{I}$ is satisfiable then the optimal value of $D(\mathcal{I},M)$ is at most $27s$. Otherwise, the optimal value of $D(\mathcal{I},M)$ is at least M.*

Now, we alter the construction so that we just have unit weights and out-degrees at most 3.

Definition. Let $\mathcal{I}$ be an instance of the 3-SAT problem, and M be a positive integer. Note that the only arcs of non-unit weight, are those going out from t' and f'. Also, the only vertices with out-degree more than three are t' and f'. The directed graph $\overline{D}(\mathcal{I},M)$ is obtained from $D(\mathcal{I},M)$ as follows. For each heavy arc (t',v), add M new vertices $x_1,x_2,\ldots,x_M$, and add the arcs $(x_1,v),(x_1,t),(x_2,v),(x_2,t),\ldots,(x_M,v),(x_M,t)$. Perform the same procedure for all heavy outgoing arcs from f'. Finally, delete t',f', and all adjacent arcs.

Note that the directed graph $\overline{D}(\mathcal{I},M)$ has $\Theta(sM)$ vertices and $\Theta(sM)$ arcs, and can be constructed in time polynomial in s and M. Moreover, it is acyclic and all of its arcs have unit weights. Also, all vertices have out-degree at most three. It is easy to check that Corollary 1 remains true for $\overline{D}(\mathcal{I}, M)$.

Corollary 2. *Let $\mathcal{I}$ be an instance of 3-SAT of size s, where $s > 3$. If $\mathcal{I}$ is satisfiable then the optimal value of $\overline{D}(\mathcal{I}, M)$ is at most $27s$. Otherwise, the optimal value of $\overline{D}(\mathcal{I}, M)$ is at least M.*

Definition. The Restricted DAG Partitioning problem is the following problem. The input is a directed acyclic graph G with arc weights, such that the weight of each arc is one, the out-degree of each vertex is at most three, and the graph has exactly two sinks. The output is a set C of arcs of minimum total weight such that each of the connected components of $G - C$ has a single sink.

We are now ready to prove our hardness result.

Theorem 1. *Assume that $\epsilon > 0$ is fixed and there is a polynomial-time $(n^{1-\epsilon})$-approximation algorithm for the Restricted DAG Partitioning problem. Then the 3-SAT problem is in $\mathcal{P}$.*

Proof. Let $\mathcal{I}$ be an instance of 3-SAT of size s, where $s > 3$. Take $M = \Theta(s^{2/\epsilon})$, so that $s^{2-\epsilon} = o(M^\epsilon)$, and construct the instance $\overline{D}(\mathcal{I}, M)$ of the Restricted DAG Partitioning problem, which has size $\Theta(sM)$. Let w be the weight of the solution found by the approximation algorithm. If $\mathcal{I}$ is satisfiable, then the optimal value is $O(s)$, so we would have $w = O\left(s(sM)^{1-\epsilon}\right) = o(M)$. Otherwise, the optimal value is at least M, so $M \leq w$. Hence one can decide whether $\mathcal{I}$ is satisfiable in polynomial time. $\square$

3 Linear-Time Algorithm for Bounded-Pathwidth Graphs

A *tree decomposition* of a directed graph G is a pair (T, W), where T is a tree and $W = (W_t : t \in V(T))$ is a family of (not necessarily induced) directed subgraphs of G such that

(i) $\bigcup_{t \in V(T)} V(W_t) = V(G)$, and every arc of G has both endpoints in some W_t, and
(ii) For every $v \in V(G)$, the set $\{t : v \in V(W_t)\}$ induces a subtree of T.

The *width* of (T, W) is

$$\max\{|V(W_t)| - 1 : t \in V(T)\},$$

and the *treewidth* of G is the minimum width of a tree decomposition of G. If each $t \in V(T)$ has degree at most 2, then (T, W) is called a *path decomposition* of G. The *pathwidth* of a graph G, written $pw(G)$, is the minimum width of a path decomposition of G. For clarity we will refer to the vertices of T as the *nodes*. W_t is called the *bag* of the decomposition corresponding to the node t. Note that sometimes the bags are simply defined as subsets of vertices of G, but here we assume that each bag also contains a subset of arcs of G. The value of pathwidth is the same in either definition.

In this section we present an algorithm that, given a DAG and a path decomposition of width k, solves the DAG Partitioning problem optimally, and has running time $2^{O(k^2)}n$. The algorithm first chooses an arbitrary degree one node of the decomposition as its root, and then (to simplify the main part) augments the decomposition so that it has the following properties:

1. Every arc appears in exactly one bag, and every bag has at most one arc.
2. If a bag W_t has an arc, then the corresponding node t has a child t_2. Let t_1 be the parent of t. Then W_{t_1}, W_t and W_{t_2} have the same vertex set, and W_{t_1} and W_{t_2} do not have an arc.
3. If node t has a child t', then the vertex sets of W_t and $W_{t'}$ differ by at most one vertex. That is,
$$|(V(W_t) \cup V(W_{t'})) \setminus (V(W_t) \cap V(W_{t'}))| \leq 1.$$
4. The bags corresponding to the root and the (only) leaf of the decomposition have empty vertex sets.

It is not hard to see that given a path decomposition with $O(n)$ bags, it is possible to augment it to a desired one with $O(k^2 n)$ bags.

An important observation is that in a DAG G, if for every $v \in V(G)$ there is a unique sink $s \in V(G)$ such that v has a directed path to s, then every connected component of G has a unique sink. Hence, if C is any solution for the DAG Partitioning problem for DAG G, then in $G - C$, to every vertex v is assigned a unique sink s, which we will sometimes call "the sink" of v in this solution.

We use dynamic programming on the path decomposition. Let us define the subproblems. Let H be a subgraph of G, and let $\mathcal{X} \subseteq V(H)$ be such that there is no arc from $V(G) \setminus V(H)$ to $V(H) \setminus \mathcal{X}$ or vice versa. Let $G - H$ denote the subgraph obtained from G by removing the *arcs* of H. Let $\mathcal{P} = \{P_1, \ldots, P_s\}$ be a partition of $\mathcal{X}$. Let $\mathcal{F} \subseteq \mathcal{X}$, and let $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{X}$ be a set of ordered pairs of distinct elements of $\mathcal{X}$ that does not induce a cycle in $\mathcal{X}$. One can build a DAG H' from H by adding $2s$ vertices $a_1, \ldots, a_s, b_1, \ldots, b_s$, and adding the arcs

$$\{(a_i, u) : u \in P_i, i = 1 \ldots s\} \cup \{(u, b_i) : u \in P_i \cap \mathcal{F}, i = 1 \ldots s\} \cup \{(u, v) : (u, v) \in \mathcal{D}\}$$

with weight ∞. The subproblem defined by $(H, \mathcal{X}, \mathcal{P}, \mathcal{F}, \mathcal{D})$ is the DAG Partitioning problem on H'. Informally speaking, this is the DAG Partitioning problem, confined to H, with the following extra restrictions:

- Vertices in $\mathcal{F}$ should not have their sink in H.
- Vertices in the same element of $\mathcal{P}$ should have the same sink, and vertices in different elements should have distinct sinks.

and the following assumptions about $G - H$:

- Vertex $v \in \mathcal{X}$ is in $\mathcal{F}$ if and only if v has a path in $G - H$ to a sink, and this sink is out of H. Note that by definition of the problem, in any solution there is a many-to-one mapping from vertices to sinks. Therefore, we refer to such a sink as the sink of v.

– For any pair $\{u, v\}$ of vertices in $\mathcal{X} \cap \mathcal{F}$, if u and v are in the same element of $\mathcal{P}$ then their sink (which is in $V(G) \setminus V(H)$) is the same, otherwise their sinks are distinct.
– For every pair $(u, v) \in \mathcal{X} \times \mathcal{X}$, we have $(u, v) \in \mathcal{D}$ if and only if there is a (u, v)-path in $G - H$.
– For every pair $(u, v) \in \mathcal{D}$, u and v are in the same element of $\mathcal{P}$.

In general, there can be exponentially many subproblems. However, using the path decomposition, one can choose a polynomial number of them, solving which obtains the optimal solution for the main problem. Let t be a node of the path decomposition, whose corresponding bag, W_t, does not have an arc. Let $\mathcal{X}_t = V(W_t)$, let H_t be the subgraph of G formed by taking the union of the bag W_t with all bags whose nodes are the descendants of t. Let $\mathcal{P} = \{P_1, \ldots, P_s\}$ be a partition of $\mathcal{X}_t$. Let $\mathcal{F} \subseteq \mathcal{X}_t$, and let $\mathcal{D} \subseteq \mathcal{X}_t \times \mathcal{X}_t$ be a set of ordered pairs of distinct elements of $\mathcal{X}_t$ that does not induce a cycle in $\mathcal{X}_t$. The subproblem defined by $(t, \mathcal{P}, \mathcal{F}, \mathcal{D})$ is equivalent to the subproblem defined as $(H_t, \mathcal{X}_t, \mathcal{P}, \mathcal{F}, \mathcal{D})$ above, and we denote its optimal values by $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D})$. Next we illustrate an algorithm for calculating $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D})$ based on the optimal values found for the descendants of t in the decomposition. But before doing so, let us define some notation.

Let $\mathcal{P} = \{P_1, \ldots, P_s\}$ be a partition of a set $\mathcal{X}$. For an element $v \notin \mathcal{X}$, $\mathcal{P} * v$ is the following family of partitions of $\mathcal{X} \cup \{v\}$:

$$\mathcal{P} * v = \{\{\{v\}, P_1, P_2, \ldots, P_s\}, \{P_1 \cup \{v\}, P_2, \ldots, P_s\}, \ldots, \{P_1, P_2, \ldots, P_s \cup \{v\}\}\}.$$

For an element $v \in X$, $\mathcal{P}/v$ is the following partition of $\mathcal{X} \setminus \{v\}$:

$$\mathcal{P}/v = \{P_1 \setminus \{v\}, \ldots, P_s \setminus \{v\}\}.$$

For a set $\mathcal{D}$ of pair of vertices of G and a vertex v, $\mathcal{D}/v$ is obtained from $\mathcal{D}$ by deleting all pairs in which v appears.

Theorem 2. *Algorithm 1 calculates* $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D})$ *correctly in time* $O(k^3)$*; therefore, the DAG Partitioning problem on* G *can be solved in time* $2^{O(k^2)}n$*.*

Proof. We prove correctness by induction. For any node t of the path decomposition and any valid tuple $(t, \mathcal{P}, \mathcal{F}, \mathcal{D})$, we show that the value of $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D})$ is calculated correctly, assuming this has been the case for the child of t, if any. The induction base is true since the bag corresponding to the unique leaf of the path decomposition is an empty subgraph so its optimal value is 0 [lines 1-2].

Now, assume that t is not a leaf, and t' is its unique child. Recall that H_t is the subgraph of G formed by taking the union of the bag W_t with all bags whose nodes are the descendants of t, and $H_{t'}$ is defined similarly. Let $H = H_t$, $\mathcal{X} = V(W_t)$, $H' = H_{t'}$ and $\mathcal{X}' = V(W_{t'})$. Recall that if $\mathcal{P}$ is a partition of $\mathcal{X}$,

Algorithm 1. Calculate $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D})$

```
 1: if $t$ has no child then
 2:     $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = 0$
 3: else
 4:     $t' \leftarrow$ the child of $t$
 5:     if $V(W_{t'})$ is $V(W_t) \dot\cup \{v\}$ for some vertex $v$ then
 6:         $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \min\{\mathrm{OPT}(t', \mathcal{Q}, \mathcal{F}, \mathcal{D}) : Q \in \mathcal{P} * v\}$
 7:     else if $V(W_{t'})$ is $V(W_t) \setminus \{v\}$ for some vertex $v \in V(W_t)$ then
 8:         $P \leftarrow$ the element of $\mathcal{P}$ containing $v$
 9:         if $P = \{v\}$ then
10:             $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \mathrm{OPT}(t', \mathcal{P} \setminus \{\{v\}\}, \mathcal{F} \setminus \{v\}, \mathcal{D}/v)$
11:         else if $v \in \mathcal{F}$ then
12:             if there is a $u \in P \setminus \{v\}$ with $u \in \mathcal{F}$ then
13:                 $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \mathrm{OPT}(t', \mathcal{P}/v, \mathcal{F} \setminus \{v\}, \mathcal{D}/v)$
14:             else
15:                 $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \infty$
16:             end if
17:         else if there is a $u \in P \setminus \{v\}$ with $(v, u) \in \mathcal{D}$ then
18:             $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \mathrm{OPT}(t', \mathcal{P}/v, \mathcal{F} \setminus \{v\}, \mathcal{D}/v)$
19:         else
20:             if for all $u \in P \setminus \{v\}$ we have $u \notin \mathcal{F}$
                and for some $u \in P \setminus \{v\}$ we have $(u, v) \in \mathcal{D}$ then
21:                 $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \mathrm{OPT}(t', \mathcal{P}/v, \mathcal{F} \cup \{u \in P \setminus \{v\} : (u, v) \in \mathcal{D}\}, \mathcal{D}/v)$
22:             else
23:                 $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \infty$
24:             end if
25:         end if
26:     else if $W_{t'}$ is $W_t \cup \{(u, v)\}$ for some arc $(u, v)$ then
27:         $t'' \leftarrow$ the child of $t'$
28:         if $u$ and $v$ are in different elements of $\mathcal{P}$ then
29:             $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = w(u, v) + \mathrm{OPT}(t'', \mathcal{P}, \mathcal{F}, \mathcal{D})$
30:         else
31:             $S \leftarrow \{x \in V(W_t) : (x, u) \in \mathcal{D}\} \cup \{u\}$
32:             $T \leftarrow \{y \in V(W_t) : (v, y) \in \mathcal{D}\} \cup \{v\}$
33:             $\mathcal{D}' \leftarrow \mathcal{D} \cup \{(x, y) : x \in S, y \in T\}$
34:             if $v \in \mathcal{F}$ then
35:                 $\mathcal{F}' \leftarrow \mathcal{F} \cup S$
36:             else
37:                 $\mathcal{F}' \leftarrow \mathcal{F}$
38:             end if
39:             $\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \min\{w(u, v) + \mathrm{OPT}(t'', \mathcal{P}, \mathcal{F}, \mathcal{D}), \mathrm{OPT}(t'', \mathcal{P}, \mathcal{F}', \mathcal{D}')\}$
40:         end if
41:     end if
42: end if
```

$\mathcal{F} \subseteq \mathcal{X}$ and $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{X}$, then $(t, \mathcal{P}, \mathcal{F}, \mathcal{D})$ is the DAG Partitioning problem confined to H with the following extra restrictions:

– Vertices in $\mathcal{F}$ should not have their sink in H;
– Vertices in the same element of $\mathcal{P}$ should have the same sink, and vertices in different elements should have distinct sinks;

and the following assumptions about $G - H$:

– Vertex $v \in \mathcal{X}$ is in $\mathcal{F}$ if and only if v has a path in $G - H$ to its sink, and the sink of v is out of H.
– For any pair $\{u, v\}$ of vertices in $\mathcal{X} \cap \mathcal{F}$, if u and v are in the same element of $\mathcal{P}$ then their sink (which is in $V(G) \setminus V(H)$) is the same, otherwise their sinks are distinct.
– For every pair $(u, v) \in \mathcal{X} \times \mathcal{X}$, we have $(u, v) \in \mathcal{D}$ if and only if there is a (u, v)-path in $G - H$.
– For every pair $(u, v) \in \mathcal{D}$, u and v are in the same element of $\mathcal{P}$.

Because of the augmentation of the path decomposition, the relation between W_t and $W_{t'}$ is of one of the following three types.

Type 1: $\mathcal{X}' = \mathcal{X} \uplus \{\mathbf{v}\}$ (LINES 5-6) In this case H and H' are the same. Any solution for $(t', \mathcal{P}', \mathcal{F}, \mathcal{D})$ gives a solution with the same value for $(t, \mathcal{P}, \mathcal{F}, \mathcal{D})$, as long as the partitions $\mathcal{P}$ and $\mathcal{P}'$ are consistent. That is, $\mathcal{P}'$ should keep the partitioning of $\mathcal{X}$, and then either throw the new vertex v into one of the elements of the partition, or put it in a new singleton element. Moreover, all solutions for $(t, \mathcal{P}, \mathcal{F}, \mathcal{D})$ are obtained this way. Hence [see lines 5-6]

$$\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \min\{\mathrm{OPT}(t', \mathcal{Q}, \mathcal{F}, \mathcal{D}) : Q \in \mathcal{P} * v\}.$$

Type 2: $\mathcal{X} = \mathcal{X}' \uplus \{\mathbf{v}\}$ (LINES 7-25) In this case H has an isolated vertex v which H' does not have. Let P be the element of the partition containing v. Four cases may happen:

(a) P is a singleton (LINES 9-10). First, assume that $v \in \mathcal{F}$. So v has a path in $G - H$ to its sink, and its sink is in $V(G) \setminus V(H)$. Also v is in a different element of partition from any other vertex $u \in \mathcal{X} \setminus \{v\}$, hence the sinks of u and v are different. So to solve this subproblem, one just needs to remove v and solve the resulting subproblem [see lines 9-10]:

$$\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \mathrm{OPT}(t', \mathcal{P} \setminus \{\{v\}\}, \mathcal{F} \setminus \{v\}, \mathcal{D}/v).$$

Now, assume that $v \notin \mathcal{F}$. Therefore v does not have a path in $G - H$ to its sink. Note that the sink of v can not be in H' since P is a singleton and $v \notin \mathcal{X}'$. Thus v is a sink itself, so for any other vertex $u \in \mathcal{X} \setminus \{v\}$, the sinks of u and v are different. So to solve this subproblem, again one just needs to remove v and solve the resulting subproblem [see lines 9-10]:

$$\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \mathrm{OPT}(t', \mathcal{P} \setminus \{\{v\}\}, \mathcal{F} \setminus \{v\}, \mathcal{D}/v).$$

(b) P is not a singleton, and $v \in \mathcal{F}$ (LINES 11-16). In this case v has a path in $G - H$ to its sink, which is in $V(G) \setminus V(H)$. First, assume that there is no $u \in P \setminus \{v\}$ with $u \in \mathcal{F}$. Then the sink of v is in $V(G) \setminus V(H)$ while each vertex $u \in P \setminus \{v\}$ has either its sink in $V(H)$, or has no path to its sink in $G - H$. Thus the subproblem is infeasible [see lines 14-15].

Now, assume that there is some $u \in P \setminus \{v\}$ with $u \in \mathcal{F}$. Then we know that the sink of u and v is the same. Hence in any solution for $(t', \mathcal{P}/v, \mathcal{F} \setminus \{v\}, \mathcal{D}/v)$, a vertex $w \in \mathcal{X}$ has the same sink as v, if and only if it has the same sink as u, if and only if it is in P. Hence we have [see lines 12-13]

$$\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \mathrm{OPT}(t', \mathcal{P}/v, \mathcal{F} \setminus \{v\}, \mathcal{D}/v).$$

(c) P is not a singleton, $v \notin \mathcal{F}$, and there is a $u \in P \setminus \{v\}$ with $(v, u) \in \mathcal{D}$ (LINES 17-18). In any solution for $(t', \mathcal{P}/v, \mathcal{F} \setminus \{v\}, \mathcal{D}/v)$, the sink of v is the same as the sink of u; thus a vertex $w \in \mathcal{X}$ has the same sink as v, if and only if it has the same sink as u, if and only if it is in P. Hence we have [see lines 17-18]

$$\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \mathrm{OPT}(t', \mathcal{P}/v, \mathcal{F} \setminus \{v\}, \mathcal{D}/v).$$

(d) P is not a singleton, $v \notin \mathcal{F}$, and there is no $u \in P \setminus \{v\}$ with $(v, u) \in \mathcal{D}$ (LINES 19-25). Since v has no outgoing arc in H or $G - H$, it is a sink in any solution for $(t, \mathcal{P}, \mathcal{F}, \mathcal{D})$. However, if there is a $u \in P \setminus \{v\}$ with $u \in \mathcal{F}$, then the sink of u is in $V(G) \setminus V(H)$, so the sink of u and the sink of v are distinct, and the subproblem is infeasible. Otherwise, if there is no $u \in P \setminus \{v\}$ with $(u, v) \in \mathcal{D}$, then for any $u \in P \setminus \{v\}$, the sink of u will be in $V(H) \setminus \{v\}$ while the sink of v is v, and again the subproblem would be infeasible [see lines 20-24].

So, assume that for all $u \in P \setminus \{v\}$ we have $u \notin \mathcal{F}$, and for some $u \in P \setminus \{v\}$ we have $(u, v) \in \mathcal{D}$. In this case one can remove v and solve the remaining subproblem. but it should be taken into account that vertices $u \in P \setminus \{v\}$ with $(u, v) \in \mathcal{D}$ have a path in $G - H'$ to their sink (which is $v \in V(G) \setminus V(H')$). Consequently [see lines 20-21],

$$\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \mathrm{OPT}(t', \mathcal{P}/v, \mathcal{F} \cup \{u \in P \setminus \{v\} : (u, v) \in \mathcal{D}\}, \mathcal{D}/v).$$

Type 3: $\mathcal{X} = \mathcal{X}'$ and $W_{t'} = W_t \cup \{(u, v)\}$ (LINES 26-42) Let t'' be the child of t', and let $H'' = H_{t''}$. Here H is the same as $H'' \cup (u, v)$. If u and v are in distinct elements of $\mathcal{P}$, then clearly the arc (u, v) should be in solution set (the set of arcs that are cut), that is [see lines 28-29],

$$\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = w(u, v) + \mathrm{OPT}(t'', \mathcal{P}, \mathcal{F}, \mathcal{D}).$$

Otherwise, one has the choice of putting the arc (u, v) in the solution set or not. If not, then when reducing the subproblem to a smaller one corresponding to H'', one should take into account the arc (u, v) which is not in H''. Let

$$S = \{x \in \mathcal{X} : (x, u) \in \mathcal{D}\} \cup \{u\}, \quad T = \{y \in \mathcal{X} : (v, y) \in \mathcal{D}\} \cup \{v\}.$$

Then for any $(x, y) \in S \times T$, the arc (u, v) creates an (x, y)-path in $G - H''$. Thus when solving the subproblem corresponding to H'', the set

$$\mathcal{D}' = \mathcal{D} \cup \{(x, y) : x \in S, y \in T\}$$

is precisely the set of pairs (x, y) such that there is an (x, y)-path in $G - H''$ [lines 31-33]. Moreover, if $v \in \mathcal{F}$, then any vertex in S has its sink in $V(G) \setminus V(H) = V(G) \setminus V(H'')$, and a path to its sink in $G - H''$. Thus, letting

$$\mathcal{F}' = \begin{cases} \mathcal{F} \cup S & v \in \mathcal{F} \\ \mathcal{F} & v \notin \mathcal{F}, \end{cases}$$

we have [see lines 34-39]

$$\mathrm{OPT}(t, \mathcal{P}, \mathcal{F}, \mathcal{D}) = \min\{w(u, v) + \mathrm{OPT}(t'', \mathcal{P}, \mathcal{F}, \mathcal{D}), \mathrm{OPT}(t'', \mathcal{P}, \mathcal{F}', \mathcal{D}')\},$$

where the first term in the minimum corresponds to putting the arc (u, v) in the solution, and the second term corresponds to not doing so.

Finally, we analyze the running time. First, observe that Algorithm 1 has running time $O(k^3)$ given that the size of the bags is at most $k + 1$. Let x be the root of the path decomposition. Note that the vertex set of W_x is empty, thus the optimal value of the DAG Partitioning problem on G is simply $\mathrm{OPT}(x, \emptyset, \emptyset, \emptyset)$. The total number of subproblems is $2^{O(k^2)}n$, and each of them can be solved in time $O(k^3)$ (provided the solutions to smaller subproblems have been calculated), which gives a total running time of $2^{O(k^2)}n$. □

4 Concluding Remarks

We showed that even the simplest instances of the DAG Partitioning problem are hard to approximate. Our result implies that the hardness arises from the global structure of the graph, and not from the weight of the arcs, the number of sinks, or the "local complexity" arising from a vertex with large out-degree. Thus, when encountered with this problem, one should naturally try to use heuristics that behave well in practice. This was the approach taken by the authors of [9].

We proved the problem is fixed parameter tractable, when the pathwidth of the graph is chosen as the parameter. A natural question is, what is the complexity if treewidth is chosen instead? It is known (see [7]) that a graph with treewidth k, has pathwidth $O(k \log n)$, so our algorithm has running time $n^{O(k^2 \log n)}$ on such a graph. Unfortunately our dynamic programming does not work correctly on tree decompositions, but it might be possible to alter it and come up with an algorithm with running time linear in n. Courcelle [5] proved that any property of graphs definable in monadic second-order logic (MSO2) can be decided in linear time on any class of graphs with bounded treewidth. So, another approach may be to formulate this problem using monadic second-order logic.

References

1. Adar, E., Adamic, L.A.: Tracking information epidemics in blogspace. In: Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2005, pp. 207–214. IEEE Computer Society, Washington, DC (2005), http://dx.doi.org/10.1109/WI.2005.151
2. Adar, E., Zhang, L., Adamic, L.A., Lukose, R.M.: Implicit Structure and the Dynamics of Blogspace. In: WWW 2004 Workshop on the Weblogging Ecosystem: Aggregation, Analysis and Dynamics. ACM Press, New York (2004)
3. Berger, B., Shor, P.W.: Approximation alogorithms for the maximum acyclic subgraph problem. In: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1990, pp. 236–243. Society for Industrial and Applied Mathematics, Philadelphia (1990), http://dl.acm.org/citation.cfm?id=320176.320203
4. Blei, D.M., Lafferty, J.D.: Dynamic topic models. In: Proceedings of the 23rd International Conference on Machine Learning, ICML 2006, pp. 113–120. ACM, New York (2006), http://doi.acm.org/10.1145/1143844.1143859
5. Courcelle, B.: The monadic second-order logic of graphs. i. recognizable sets of finite graphs. Inf. Comput. 85, 12–75 (1990), http://dl.acm.org/citation.cfm?id=81253.81255
6. Călinescu, G., Karloff, H., Rabani, Y.: An improved approximation algorithm for multiway cut. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC 1998, pp. 48–52. ACM, New York (1998), http://doi.acm.org/10.1145/276698.276711
7. Korach, E., Solel, N.: Tree-width, path-width, and cutwidth. Discrete Appl. Math. 43, 97–101 (1993), http://dl.acm.org/citation.cfm?id=153610.153618
8. Kwon, Y.S., Kim, S.W., Park, S., Lim, S.H., Lee, J.B.: The information diffusion model in the blog world. In: Proceedings of the 3rd Workshop on Social Network Mining and Analysis, SNA-KDD 2009, pp. 4:1–4:9. ACM, New York (2009), http://doi.acm.org/10.1145/1731011.1731015
9. Leskovec, J., Backstrom, L., Kleinberg, J.: Meme-tracking and the dynamics of the news cycle. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2009, pp. 497–506. ACM, New York (2009), http://doi.acm.org/10.1145/1557019.1557077
10. Leskovec, J., McGlohon, M., Faloutsos, C., Glance, N., Hurst, M.: Cascading behavior in large blog graphs: Patterns and a model. In: Society of Applied and Industrial Mathematics: Data Mining, SDM 2007 (2007)