

Efficient Algorithms for Measuring the Funnel-likeness of DAGs

Marcelo Garlet Millani^{*1}, Hendrik Molter¹, Rolf Niedermeier¹, and
Manuel Sorge^{†2}

¹Institut für Softwaretechnik und Theoretische Informatik, TU Berlin,
Germany,

{m.garletmillani, h.molter, rolf.niedermeier}@tu-berlin.de

²Dept. Industrial Engineering and Management, Ben-Gurion University of the
Negev, Beer Sheva, Israel,
sorge@post.bgu.ac.il

February 1, 2018

Abstract

Funnels are a new natural subclass of DAGs. Intuitively, a DAG is a funnel if every source-sink path can be uniquely identified by one of its arcs. Funnels are an analog to trees for directed graphs that is more restrictive than DAGs but more expressive than in-/out-trees. Computational problems such as finding vertex-disjoint paths or tracking the origin of memes remain NP-hard on DAGs while on funnels they become solvable in polynomial time. Our main focus is the algorithmic complexity of finding out how funnel-like a given DAG is. To this end, we study the NP-hard problem of computing the arc-deletion distance to a funnel of a given DAG. We develop efficient exact and approximation algorithms for the problem and test them on synthetic random graphs and real-world graphs.

1 Introduction

Directed acyclic graphs (DAGs) are finite directed graphs (digraphs) without directed cycles and appear in many applications, including the representation of precedence constraints in scheduling, data processing networks, causal structures, or

^{*}Partially supported by DFG project “FPTinP” NI 369/16-1.

[†]Supported by the People Programme (Marie Curie Actions) of the European Union’s Seventh Framework Programme (FP7/2007-2013) under REA grant agreement number 631163.11 and Israel Science Foundation (grant no. 551145/14).

inference in proofs. From a more graph-theoretic point of view, DAGs can be seen as a directed analog of trees; however, their combinatorial structure is much richer. Thus a number of directed graph problems remain NP-hard even when restricted to DAGs. This motivates the study of subclasses of DAGs. We study *funnels* which are DAGs where each source-sink path has at least one private arc, that is, no other source-sink path contains this arc. In independent work, Lehmann [Lehmann, 2017] studied essentially the same graph class.

Funnels are both of combinatorial and graph-theoretic as well as of practical interest: First, funnels are a natural compromise between DAGs and trees as, similarly to in- or out-trees, the private-arc property guarantees that the overall number of source-sink paths is upper-bounded linearly by its number of arcs, yet multiple paths connecting two vertices are possible. Second, in Section 2 we show that funnels, in a divide & conquer spirit, allow for a vertex partition into a set of *forking* vertices with indegree one and possibly large outdegree and a set of *merging* vertices with outdegree one and possibly large indegree. This partitioning helps in designing our algorithms. Third, in terms of applications, due to the simpler structure of funnels, problems such as DAG PARTITIONING [Leskovec et al., 2009, van Bevern et al., 2017] or VERTEX DISJOINT PATHS, (also known as k -LINKAGE) [Bang-Jensen and Gutin, 2008, Fortune et al., 1980] become tractable on funnels while they are NP-hard on DAGs. Lehmann [Lehmann, 2017] showed that a variation of the problem NETWORK INHIBITION, which is NP-hard on DAGs, can be solved in polynomial time on funnels. Altogether, we feel that funnels are one of so far few natural subclasses of DAGs.

The focus of this paper is on investigating the complexity of turning a given DAG into a funnel by a minimum number of arc deletions. The motivation for this is twofold. First, due to the noisy nature of real-world data, we expect that graphs from practice are not pure funnels, even though they may adhere to some form of funnel-like structure. To test this hypothesis we need efficient algorithms to determine funnel-likeness. Second, as mentioned above, natural computational problems become tractable on funnels (e.g., k -LINKAGE [Millani, 2017a]). Thus it is promising to try and develop fixed-parameter algorithms for such NP-hard DAG problems with respect to distance parameters to funnels. This approach is known as exploiting the “distance from triviality” [Cai, 2003, Guo et al., 2004, Niedermeier, 2010]. A natural way to measure the distance of a given DAG D to a funnel is the *arc-deletion distance to a funnel*, the minimum number of arcs that need to be deleted from D to obtain a funnel. The problem of computing this distance parallels the well-studied NP-hard FEEDBACK ARC-SET problem where the task is to turn a given digraph into a DAG by a minimum number of arc deletions. Even FEEDBACK ARC-SET on tournaments is NP-hard and it received considerable interest over the last years [Ailon and Alon, 2007, Bessy et al., 2011, Charbit et al., 2007, Kenyon-Mathieu and Schudy, 2007].

Formally, we study the ARC-DELETION DISTANCE TO A FUNNEL (ADDF) problem, where, given a DAG D , we want to find its arc-deletion distance d to a funnel.

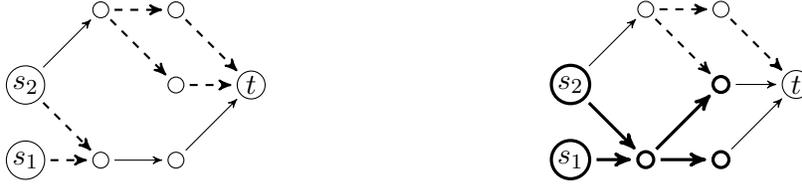


Figure 1: Example of a funnel (left) and a DAG which is not a funnel (right). Private arcs are marked as dashed lines. The DAG on the right is not a funnel because all arcs in an (s_1, t) -path are shared. Removing one arc from it turns it into a funnel. A forbidden subgraph for funnels is marked in bold.

We show that ADDF is NP-hard and that it admits a linear-time factor-two approximation algorithm and a fixed-parameter algorithm with linear running time for constant d .¹ In experiments we demonstrate that our algorithms are useful in practice.

2 Funnels: Definition and Properties

In this section we formally define funnels. We provide several equivalent characterizations, summarized in [Theorem 1](#), and analyze some basic properties of funnels. We use standard terminology from graph theory.

To define funnels as a proper subclass of DAGs, we limit the number of paths that may exist between two vertices (which can be exponential in DAGs but is one in trees). Requiring every path between two vertices to be unique would possibly be too restrictive, and in the case of a single source such DAGs would simply be so-called out-trees. Instead, we require each path going from a source to a sink to be uniquely identified by one of its *private* arcs. We say that an arc is private if there is only one source-sink path which goes through that arc. An example of a funnel can be seen in [Figure 1](#).

Definition 1 (Funnel). *A DAG D is a funnel if every source-sink path has at least one private arc.*

From this definition it is clear that the number of source-sink paths in a funnel is linearly upper-bounded in its number of arcs.

Different characterizations of funnels reveal certain interesting properties which these digraphs have, and are used in subsequent proofs and algorithms. We summarize these characterizations in the theorem below. In the following, $\text{out}^*(v)$ denotes the set of vertices that can be reached from v in a given DAG, $\text{out}(v)$ denotes the set of neighbors of v and $\text{outdeg}(v)$ denotes v 's outdegree; $\text{in}^*(v)$, $\text{in}(v)$ and $\text{indeg}(v)$ are defined analogously.

¹There is also a simple $\mathcal{O}(5^d \cdot |V| \cdot |A|)$ -time algorithm for general digraphs [[Millani, 2017a](#)].

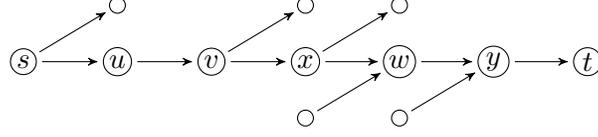


Figure 2: Illustration of the vertices used in the proof of statement (2) [Theorem 1](#). We argue that the arc (x, w) is private.

Theorem 1. *Let D be a DAG. The following statements are equivalent:*

1. D is a funnel.
2. For each vertex $v \in V : \text{indeg}(v) > 1 \Rightarrow \forall u \in \text{out}^*(v) : \text{outdeg}(u) \leq 1$.
3. No subgraph of D is contained in $\mathcal{F} = \{D_i\}_{i=0}^\infty$, where
 - $D_k = (V_k, A_k)$,
 - $V_k = \{u_1, u_2, v_0, w_1, w_2\} \cup \{v_i\}_{i=1}^k$, and
 - $A_k = \{(u_1, v_0), (u_2, v_0), (v_k, w_1), (v_k, w_2)\} \cup \{(v_i, v_{i+1})\}_{i=1}^{k-1}$.
4. D does not contain D_0 or D_1 (defined above) as a topological minor.²

Proof. We first prove that (1) \Leftrightarrow (2), that is, we show that a DAG $D = (V, A)$ is a funnel if and only if

$$\forall v \in V : \text{indeg}(v) > 1 \Rightarrow \forall u \in \text{out}^*(v) : \text{outdeg}(u) \leq 1. \quad (\text{i})$$

The idea is to identify the private arcs and to argue that each path must contain at least one of those arcs. Refer to [Figure 2](#) while reading the proof. We start by showing that a DAG satisfying (i) is a funnel.

Let $D = (V, A)$ be a DAG which satisfies (i), let $s \in V$ be a source and $t \in V$ be a sink such that some (s, t) -path exists, and let v be the first vertex in $\text{out}^*(s)$ with $\text{outdeg}(v) > 1$. If no such vertex v exists, then there is only one (s, t) -path in D and all outgoing arcs from s are private. Otherwise, due to (i) we know that $\forall u \in \text{in}^*(v) \setminus \{s\} : \text{indeg}(u) = 1$. This means that there is exactly one (s, v) -path. Let P be some (s, t) -path that goes through v and let w be the first vertex in this path with $\text{indeg}(w) > 1$. If no such w exists, then there is only one (v, t) -path and all arcs after v are private, as required. Otherwise, we consider a vertex x such that the arc (x, w) is in P . We know $\text{indeg}(x) = 1$, which implies that there is only one (v, x) -path. Since the (v, x) -path as well as the (s, v) -path are unique and $\forall y \in \text{out}^*(w) : \text{outdeg}(y) = 1$, the arc (x, w) is private for the (s, t) -path that contains it. Thus, D is a funnel.

We next show that every funnel satisfies (i). We do this by contraposition, showing that every arc of some (s, t) -path is present in at least one other source-sink path if (i) does not hold.

Let $D = (V, A)$ be DAG where (i) is not true. This means that there is some vertex $u \in V$ with $\text{indeg}(u) > 1$ and that there is some other vertex $w \in \text{out}^*(u)$ with

²A graph H is called a *topological minor* of a graph G if a subgraph of G can be obtained from H by subdividing edges (that is, replacing arcs by directed paths).

$\text{outdeg}(w) > 1$. Let w be the first such vertex. Then there are at least $\text{indeg}(u)$ many paths from some source to u , and $\text{outdeg}(w)$ many from w to some sink. Since there is at least one (u, w) -path (possibly without arcs), this implies that every arc in the induced subgraph $\text{in}^*[u]$ is shared by $\text{outdeg}(w)$ many paths, every arc in $\text{out}^*[w]$ is shared by $\text{indeg}(u)$ many paths, and all arcs in a (u, w) -path are shared by $\text{indeg}(u) \cdot \text{outdeg}(w)$ many paths. Hence, all arcs in a source-sink path which goes through u and w are shared, implying that D is not a funnel.

Next, we prove that (2) \Leftrightarrow (3) by contraposition. That is, we show that $\exists C \subseteq D : C \in \mathcal{F}$ if and only if D does not satisfy (i).

Let $C \subseteq D$ be a subgraph of D such that $C \in \mathcal{F}$. By definition of C it contains some vertex v_0 with $\text{indeg}_C(v_0) = 2$ and another vertex $v_k \in \text{out}_C^*(v_0)$ with $\text{outdeg}_C(v_k) = 2$. This implies $\text{indeg}_D(v_0) > 1$ and $\text{outdeg}_D(v_k) > 1$, violating (i).

Now assume D does not satisfy (i). That is, there is some vertex v with $\text{indeg}_D(v) > 1$ and another vertex $u \in \text{out}_D^*(v)$ with $\text{outdeg}_D(u) > 1$. Let $u_1, u_2 \in \text{in}_D(v)$ and $w_1, w_2 \in \text{out}_D(u)$ be four distinct vertices. Let $v, v_1, v_2, \dots, v_{k-1}, u$ be a (v, u) -path. We set $v_0 := v$ and $v_k := u$, obtaining the forbidden subgraph D_k if $u \neq v$, and D_0 otherwise. Hence, D contains a subgraph from \mathcal{F} .

Finally, we show that (3) \Leftrightarrow (4). It is enough to show that any $D_i \in \mathcal{F}$ can be obtained by subdividing D_0 or D_1 multiple times, and that any subdivision of D_0 and D_1 contains some digraph of \mathcal{F} as a subgraph.

We first show that we can generate \mathcal{F} by subdividing D_0 and D_1 . Let $D_i \in \mathcal{F}$. If $i \leq 1$, then D_i obviously contains itself as a topological minor. If $i > 1$, then by subdividing the arc (v_0, v_1) from D_1 a total of $i - 1$ times, we obtain D_i . Hence all digraphs in \mathcal{F} can be generated by D_0 and D_1 through subdivisions.

Now we show that any subdivision of D_0 and D_1 contains some digraph from \mathcal{F} . Since subdividing arcs does not change the degrees of the affected vertices, the degree of v_0 remains the same. Hence, any subdivision of D_0 contains $D_0 \in \mathcal{F}$ as a subgraph.

For any subdivision D'_1 of D_1 we know that $\text{indeg}_{D'_1}(v_0) = 2$, $\text{outdeg}_{D'_1}(v_1) = 2$ and $v_1 \in \text{out}_{D'_1}^*(v_0)$. If we subdivide incoming arcs of v_0 or outgoing arcs of v_1 , the resulting DAG will contain D_1 as a subgraph. If we subdivide k times the arc (v_0, v_1) , we obtain $D_{k+1} \in \mathcal{F}$ as a subgraph.

We showed that (1) \Leftrightarrow (2) \Leftrightarrow (3) \Leftrightarrow (4), thus proving that all four statements are equivalent. \square

Definition 1 does not give us a very efficient way of checking whether a given DAG is a funnel or not. A simple algorithm which counts how many paths go through each arc would take $\mathcal{O}(|A|^2)$ time. Using the characterization in **Theorem 1(2)** we can follow some topological ordering of the vertices of a DAG and check in linear time whether it is a funnel.

The *degree characterization* in **Theorem 1(2)** provides some additional insight about the structure of a funnel. We can see that a funnel can be partitioned into two induced subgraphs: One is an out-forest and the other is an in-forest. Note that this

partition is not necessarily unique. For use below, a *FM-labeling* for given a DAG with vertex set V is a function $L : V \rightarrow \{\text{FORK}, \text{MERGE}\}$ which gives a *label* to each vertex. An FM-labeling for a funnel is called *funnel labeling* if the vertices in the out-forest of the funnel are assigned the label FORK and vertices in the in-forest are assigned the label MERGE. The following holds.

Observation 1. *Let $D = (V, A)$ be a funnel and L be a funnel labeling for D . Then there is no $(v, u) \in A$ with $L(v) = \text{MERGE}$ and $L(u) = \text{FORK}$.*

With a simple counting argument it is also possible to give an upper bound on the number of arcs in a funnel. This bound is sharp.

Observation 2. *Let $D = (V, A)$ be a funnel. Then $|A| \leq |V|^2/4 + |V| - 2$.*

Proof. Let L be a funnel labeling for D . Let $V = X \uplus Y$ where $\forall v \in X : L(v) = \text{FORK}$ and $\forall v \in Y : L(v) = \text{MERGE}$. Clearly, the vertices in X form an out-forest, while those in Y form an in-forest. This gives us at most $|X| - 1$ arcs between vertices in X , and at most $|Y| - 1$ arcs between vertices in Y . Furthermore, there are at most $|X| \cdot |Y|$ arcs from vertices in X to vertices in Y and we know from the construction that there are no arcs from Y to X . Hence, $|A| \leq |X| \cdot |Y| + |X| + |Y| - 2$. This value is maximized when $|X| = |Y| = |V|/2$, which gives us the bound $|A| \leq |V|^2/4 + |V| - 2$. \square

Considering that a DAG has at most $|V|(|V| - 1)/2$ arcs, [Observation 2](#) implies that a funnel can have roughly half as many arcs as a DAG. This means that funnels are not necessarily sparse (unlike forests).

While the degree characterization is useful for algorithms, the characterizations by forbidden subgraphs and minors ([Theorem 1\(3 and 4\)](#)) help us to understand the local structure of a funnel and of graphs that are not funnels. These characterizations also imply that being a funnel is a *hereditary* graph property, that is, deleting vertices does not destroy the funnel property.

3 Computing the Arc-Deletion Distance to a Funnel

In this section we show ADDF is NP-hard, and present a linear-time factor-2 approximation algorithm and an exact fixed-parameter algorithm. Our algorithms also compute the set of arcs to be deleted. We remark that the corresponding vertex-deletion distance minimization problem is also NP-hard and that it can be solved in $\mathcal{O}(6^d |V| \cdot |A|)$ time, where d is the number of vertices to delete [[Millani, 2017a](#)]. The following result can be shown by a reduction from 3-SAT.

Theorem 2. *ADDF is NP-hard.*

Proof. We present a reduction from 3-SAT. Recall that in 3-SAT we are asked to decide the satisfiability of given a Boolean formula ϕ in conjunctive normal form where every clause has exactly three distinct literals. Given a 3-SAT formula ϕ with

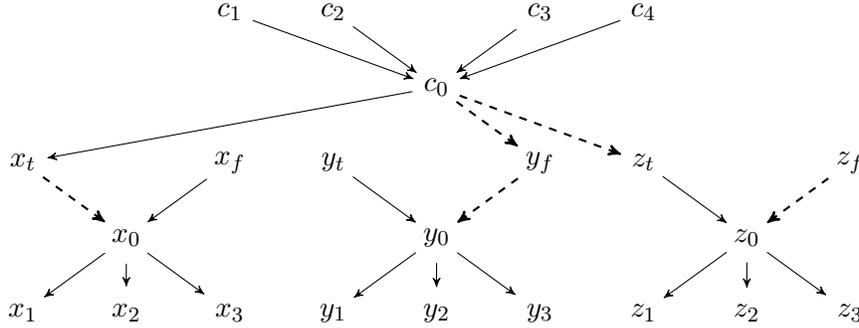


Figure 3: Example of the reduction for the formula $(x \vee \neg y \vee z)$. Dashed arcs correspond to a solution for ADDF on the reduced instance.

n variables and m clauses, we create a DAG D consisting of the following variable gadgets and clause gadgets. Figure 3 illustrates the construction. For each variable x we create the following *variable gadget* introducing the vertex set V_x and edge set A_x :

- $V_x = \{x_0, x_t, x_f, x_1, x_2, x_3\}$,
- $A_x = \{(x_t, x_0), (x_f, x_0)\} \cup \{(x_0, x_i) \mid 1 \leq i \leq 3\}$.

We call x_0 the *center* of the variable gadget for x . For each clause c , we create the following *clause gadget*, introducing the vertex set V_c and edge set A_c :

- $V_c = \{c_0, c_1, \dots, c_4\}$,
- $A_c = \{(c_i, c_0) \mid 1 \leq i \leq 4\}$.

We call c_0 the *center* of the clause gadget for c . Furthermore, if variable x appears non-negated in clause c , then we add the arc (c_0, x_t) , and if variable x appears negated in clause c , then we add the arc (c_0, x_f) . This completes the construction. It is easy to see that the DAG D can be constructed in polynomial time. We claim that D has an arc-deletion distance to funnel of $k = 2m + n$ if and only if ϕ is satisfiable.

(\Leftarrow): Assume ϕ has a satisfying assignment. Then we construct an arc-deletion set of size $k = 2m + n$ as follows: If a variable x is set to true, we delete the arc (x_t, x_0) , otherwise we delete the arc (x_f, x_0) . For each clause c we delete two of the three outgoing arcs of c_0 , where we choose the remaining arc to be one that points to a literal that causes the clause to be satisfied by the assignment. This arc deletion set clearly has the correct size, it remains to show that it destroys all forbidden subgraphs of funnels in the constructed DAG. Note that after the arcs are deleted, there are only two types of vertices with indegree greater than one: The centers of clause gadgets and potentially vertices x_t or x_f from variable gadgets. The only vertices with outdegree greater than one remaining are the centers of variable gadgets. Because the outgoing arcs of clause gadgets point to literals that cause the clause to be satisfied, we have that all paths from clause gadget centers to vertex gadget centers are destroyed. By the same argument, there are no paths between vertices x_t or x_f from variable gadgets that have indegree greater than one and centers of variable gadgets. Hence, there is no path from a vertex with indegree greater than

one to a vertex with outdegree greater than one.

(\Rightarrow): First, note that all variable and clause gadgets are pair-wise arc-disjoint. It is easy to check that for each variable gadget at least one arc needs to be deleted and for each clause gadget at least two arcs need to be deleted. Since the number of arc deletions has to be at most $k = 2m + n$, the arc deletion set contains exactly one arc from each variable gadget and exactly two arcs from each clause gadget. This implies that for clause gadgets, the two of the outgoing arcs of the center need to be deleted and for variable gadgets, one of the incoming arcs of the center needs to be deleted. We claim that the arcs deleted from the variable gadgets induce a satisfying assignment in a straightforward manner: if the arc (x_t, x_0) is deleted, set variable x to true, otherwise to false. Take any clause c of ϕ , one of the outgoing arcs from the center of the clause gadget of c remains, and this arc has to point to a vertex with outdegree zero, otherwise there is a path from a center of a clause gadget to a center of a variable gadget and hence a forbidden subgraph. This means that clause c is satisfied. This completes the proof. \square

A Factor-2 Approximation Algorithm.

We now give a linear-time factor-2 approximation algorithm for ADDF. We mention in passing that on tournament DAGs the algorithm always finds an optimal solution and on real-world DAGs, the approximation factor is typically close to one (see Section 4). The approximation algorithm works in three phases and makes extensive use of FM-labelings (defined in Section 2). First, we greedily compute an FM-labeling which we call L_a for the input graph (assigning each vertex v a FORK or a MERGE label). The labeling will be a funnel labeling of the output funnel indicating for each vertex whether it can have indegree or outdegree greater than one. To construct L_a , we try to minimize the number of arcs to be removed when only considering v . This strategy guarantees that, if the approximation algorithm assigns the wrong label to v , in the optimal solution many arcs incident to v need to be removed. This allows us to derive the approximation factor. Formally, we assign a label to a vertex v using the following rule.

$$L_a(v) := \begin{cases} \text{FORK}, & \text{if } \text{outdeg}_D(v) > \text{indeg}_D(v), \\ \text{FORK}, & \text{if } \text{outdeg}_D(v) = \text{indeg}_D(v) \wedge \\ & \exists u \in \text{in}(v) : L_a(u) = \text{FORK}, \\ \text{MERGE}, & \text{otherwise.} \end{cases}$$

Since we can assign a label whenever we know the labels of all incoming neighbors, the label of each vertex can be computed, in linear time, by following a topological ordering of the DAG.

In the second phase, after assigning labels to all vertices, we *satisfy* the labels by removing arcs. That is, for each FORK vertex v , we choose an arbitrary inneighbor u with $L(u) = \text{FORK}$ (if it exists) and remove all arcs incoming to v from vertices other

Algorithm 1 Satisfying an FM-labeling.

```

1: function ArcDeletionSet(DAG  $D = (V, A)$ ,  $L : V \rightarrow \{\text{FORK, MERGE}\}$ )
2:    $B := \emptyset$ 
3:   for all  $v \in V$  do
4:     if  $L(v) = \text{MERGE}$  then
5:       Choose an arbitrary  $u \in \text{out}(v)$  with  $L(u) = \text{MERGE}$  (if it exists)
6:        $B := B \cup \{(v, w) \mid w \neq u \wedge w \in \text{out}(v)\}$ 
7:     else if  $L(v) = \text{FORK}$  then
8:       Choose an arbitrary  $u \in \text{in}(v)$  with  $L(u) = \text{FORK}$  (if it exists)
9:        $B := B \cup \{(w, v) \mid w \neq u \wedge w \in \text{in}(v)\}$ 
10:  return  $B$ 

```

than u . Similarly, for each MERGE vertex v we choose an arbitrary outneighbor u with $L(u) = \text{MERGE}$ (if it exists) and remove all arcs outgoing from v to vertices other than u . See Algorithm 1 for the pseudocode of the second phase. For use below we call the second-phase algorithm `ArcDeletionSet`.

In the third phase, we *greedily relabel* vertices, that is, we iterate over each vertex v (in an arbitrary order), changing v 's label if the change immediately leads to an improvement in the solution size. To check if there is an improvement, we only need to consider the incident arcs of v and the labels of its endpoints. This completes the description of our approximation algorithm.

To argue about optimal solutions and for use in a search-tree algorithm below, we now show that if the input FM-labeling L corresponds to an optimal solution, then `ArcDeletionSet` outputs an optimal arc set: Say that an FM-labeling L of a DAG D is *optimal* if it is a funnel labeling for some funnel $D - A'$, $A' \subseteq A$, such that A' has minimum size among all arc sets whose deletion makes D a funnel.

Proposition 1. *Let $D = (V, A)$ be a DAG, let $A' \subseteq A$ be a minimum arc set such that $D' = D - A'$ is a funnel, and let L^* be an optimal labeling for D' . Then $|\text{ArcDeletionSet}(D, L^*)| = |A'|$.*

Proof. Let $(v, u) \in A$. We distinguish the possible cases of the labeling of u and v . First, we treat two simple cases in which we can argue that A' and `ArcDeletionSet` either both contain (v, u) or both do not contain (v, u) .

The first case is when $L^*(v) = \text{MERGE}$ and $L^*(u) = \text{FORK}$. Then (v, u) has to be both in A' as well as in the solution given by `ArcDeletionSet`, which we call from now on B . It is clearly in B since it was added to the solution on Line 6 of Algorithm 1. Due to Observation 1, we know that (v, u) is also in A' .

The second case is when $L^*(v) = \text{FORK}$ and $L^*(u) = \text{MERGE}$. In this case, clearly, removing (v, u) will not destroy any forbidden subgraph, since $\forall w \in \text{in}_{D'}^*(v) : L^*(w) = \text{FORK}$. Since `ArcDeletionSet` does not remove the arc, it is neither present in B nor in A' .

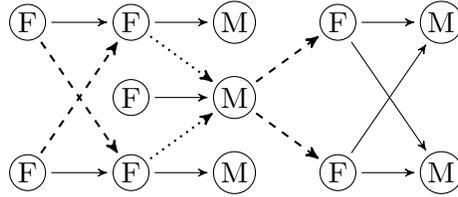


Figure 4: Example of the execution of `ArcDeletionSet`. Vertices with an F received the label FORK, and those with an M received the label MERGE. The approximation algorithm returns the four dashed arcs, while there is an optimal solution (dotted arcs) of size two. Note that changing any single label will not improve the approximate solution.

For the remaining cases we cannot guarantee that exactly the same decision was taken with respect to (v, u) . We instead argue about the total number of arcs removed between vertices with the same label. From [Theorem 1\(2\)](#) we know that FORK vertices form an induced outforest in D , while MERGE vertices form an induced inforest. The number of arcs in an in- or outforest is given by the number of vertices minus the number of roots (i.e. sources or sinks). All incoming arcs of a FORK vertex v are removed by `ArcDeletionSet` only if v has no inneighbors labeled with FORK. Hence, v is a source in $D - B$ if and only if it is a source in $D - A'$. This implies that the number of arcs in the outforest composed of FORK vertices is the same in $D - B$ as in $D - A'$. An analogous argument holds for the inforest induced by MERGE vertices. Hence, the total number of arcs between equally labeled vertices is the same in B and A' . Since these were all cases and in all of them `ArcDeletionSet` deletes as many arcs as the optimal solution, we conclude that $|B| = |A'|$. \square

We now give a guarantee of the approximation factor. A DAG where the approximation algorithm removes twice as many arcs as an optimal solution is given in [Figure 4](#).

Theorem 3. *There is a linear-time factor-two approximation for ADDF.*

Proof. After computing $B = \text{ArcDeletionSet}(D, L_a)$, the approximation algorithm iterates over $D - B$, flipping labels whenever the flip leads to an improvement in the solution. This implies that, if we remove all incoming arcs of a vertex v with $L_a(v) = \text{MERGE}$, then we set the label of v to FORK instead. Analogously, we flip the label of v if all of its outgoing arcs have been removed and $L_a(v) = \text{FORK}$.

Let $B = \text{ArcDeletionSet}(D, L_a)$ and let A' be a minimum arc set such that $D - A'$ is a funnel. Let L^* be an optimal FM-labeling for the input DAG $D = (V, A)$ such that $A' = \text{ArcDeletionSet}(D, L^*)$. We define two functions $b : V \rightarrow \mathcal{P}(B)$ and $a : V \rightarrow \mathcal{P}(A')$ such that $\bigsqcup_{v \in V} b(v) = B$ and $\bigsqcup_{v \in V} a(v) = A'$, where \bigsqcup is a disjoint union and $\mathcal{P}(X)$ denotes the family of all subsets of a set X . Our goal is to assign each arc in A' and B to one of its endpoints via a and b , respectively, such

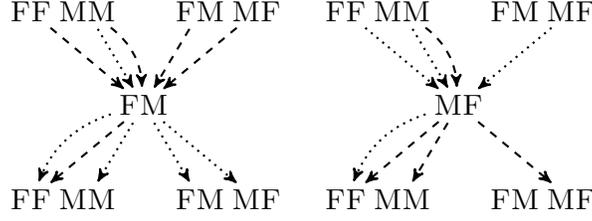


Figure 5: Graphical representation of a and b. Vertices are identified with their type. Arcs are assigned to the vertex in the middle. Dashed arcs correspond to arcs counted in b, while dotted arcs are counted in a.

that $|b(v)| \leq 2|a(v)|$ for every $v \in V$. We say that a vertex v has type $T(v) = \text{FM}$ if $L_a(v) = \text{FORK}$ and $L^*(v) = \text{MERGE}$. The types FF, MM and MF are defined analogously. A vertex v is correctly labeled if $L_a(v) = L^*(v)$.

We define a and b in such a way that $|b(v)| = |a(v)|$ if v is correctly labeled. To this end, we only assign a removed arc to a correctly labeled vertex v if both endpoints are correctly labeled. For an incorrectly labeled vertex, we assign the arcs which are potentially removed by `ArcDeletionSet` when considering v , together with those of correctly labeled vertices. We additionally need to define a and b in such a way that no arc is assigned to both endpoints. Refer to Figure 5 for a graphical representation of a and b .

$$b(v) := \begin{cases} B \cap \{(u, v) \mid T(u) = \text{FF}\}, & T(v) = \text{FF}, \\ B \cap \{(v, u) \mid T(u) = \text{FF} \vee T(u) = \text{MM}\}, & T(v) = \text{MM}, \\ B \cap (\{(u, v) \mid u \in \text{in}(v)\} \cup \{(v, u) \mid T(u) = \text{FF}\}), & T(v) = \text{FM}, \\ B \cap (\{(u, v) \mid T(u) = \text{MM}\} \cup \{(v, u) \mid T(u) \neq \text{FM}\}), & T(v) = \text{MF}. \end{cases}$$

$$a(v) := \begin{cases} A' \cap \{(u, v) \mid T(u) = \text{FF}\}, & T(v) = \text{FF}, \\ A' \cap \{(v, u) \mid T(u) = \text{FF} \vee T(u) = \text{MM}\}, & T(v) = \text{MM}, \\ A' \cap (\{(v, u) \mid u \in \text{out}(v)\} \cup \{(u, v) \mid T(u) = \text{MM}\}), & T(v) = \text{FM}, \\ A' \cap (\{(v, u) \mid T(u) = \text{FF}\} \cup \{(u, v) \mid T(u) \neq \text{FM}\}), & T(v) = \text{MF}. \end{cases}$$

We now consider each vertex type t and argue that $|b(v)| \leq 2|a(v)|$ for every vertex v with $T(v) = t$. By construction of a and b , this is easy to prove for correctly labeled vertices. Further, induced paths of any length behave just like an induced with three vertices, and so we only need to consider the latter case.

Lemma 1. *If $L_a(v) = L^*(v)$ or $\text{indeg}(v) = \text{outdeg}(v) = 1$, then $|b(v)| = |a(v)|$.*

Proof. When deciding which incident arc of a vertex v is kept, `ArcDeletionSet` makes an arbitrary choice among the valid possibilities. However, all choices lead to a solution of the same size. Hence, we can assume, without loss of generality, that if `ArcDeletionSet` can keep an arc from a correctly labeled neighbor, then it does

so. This allows us to assume, for the sake of this analysis, that, if an arc between two correctly labeled vertices is removed by the approximation algorithm, then it is also removed in an optimal solution. Formally, we can assume the following for any $v \in V$ which is correctly labeled. If $L^*(v) = \text{FORK}$ and there is some correctly labeled $u \in \text{in}(v)$ with $L^*(u) = \text{FORK}$, then $(w, v) \in A' \cap B$ for all incorrectly labeled $w \in \text{in}(v)$. If $L^*(v) = \text{MERGE}$ and there is some correctly labeled $u \in \text{out}(v)$ with $L^*(u) = \text{MERGE}$, then $(v, w) \in A' \cap B$ for all incorrectly labeled $w \in \text{out}(v)$.

We now show for every correctly labeled v that $|b(v)| = |a(v)|$. We first define variables which count how many neighbors of each type v has. Let i_{FM} be the number of inneighbors of v with type FM. The variables i_{FF} , i_{MM} and i_{MF} are defined analogously, and o_{FM} , o_{FF} , o_{MM} and o_{MF} are defined analogously for the outneighbors of v .

Let $L^*(v) = \text{MERGE}$, then $o_{\text{FF}} \leq |b(v)| \leq o_{\text{MM}} + o_{\text{FF}}$ and $o_{\text{FF}} \leq |a(v)| \leq o_{\text{MM}} + o_{\text{FF}}$. If $o_{\text{MM}} = 0$, then $|b(v)| = o_{\text{FF}} = |a(v)|$. Otherwise, due to the initial assumption, $|b(v)| = o_{\text{FF}} + o_{\text{MM}} - 1 = |a(v)|$. Hence, $|b(v)| = |a(v)|$. The case where $L^*(v) = \text{FORK}$ follows analogously.

For any path $v_1, v_2 \dots v_k$ where all vertices have in- and outdegree one, the approximation assigns the same label to all vertices. Furthermore, it removes at most two arcs in such a path. The decision of whether to remove an arc or not depends only on the label of the predecessor of v_1 and of the successor of v_k , and not on the length of the path. Hence, we can treat this case by contracting the path into a single vertex v . Let $u \in \text{indeg}(v)$ and $w \in \text{outdeg}(v)$ be the unique neighbors of v . Note that, by definition, $L_a(v) = L_a(u)$.

If both incident arcs of v are in B , we flip the label of v in the greedy relabeling phase. Either before or after the flip v is correctly labeled. Since flipping the label of v does not worsen the solution, it follows from the previous case that $|b(v)| = |a(v)|$.

If only one arc of v was removed, the path uvw behaves as a single arc (u, w) , and the removed arc can be assigned to a vertex by considering the types of u and w , taking the same decision as if we were assigning the arc (u, w) to a vertex. Hence, $|b(v)| = 0 = |a(v)|$ in this case. \square

We are now ready to prove an approximation factor of two.

Lemma 2. $|B| \leq 2 \cdot |A'|$

Proof. We first define variables which count the number of neighbors of v for each type. Let i_{FM} be the number of inneighbors of v with type FM. The variables i_{FF} , i_{MM} and i_{MF} are defined analogously, and o_{FM} , o_{FF} , o_{MM} and o_{MF} are defined analogously for the outneighbors of v . Next, we show for every incorrectly labeled vertex v (with in- or outdegree greater than one) that $|b(v)| + o_{\text{MF}} \leq 2|a(v)|$ (if $T(v) = \text{FM}$) and $|b(v)| - i_{\text{FM}} \leq 2|a(v)|$ (if $T(v) = \text{MF}$). Note that the sum of all o_{MF} equals the sum of all i_{FM} . Hence, we also show that $|\bigoplus_{v \in X} b(v)| \leq 2|\bigoplus_{v \in X} a(v)|$, where X is the set of all incorrectly labeled vertices.

Case 1. $T(v) = \text{FM}$. By definition, $|b(v)| \leq c + o_{\text{FF}}$, where $c \leq \text{indeg}(v)$ is the number of incoming arcs removed from v by B . Since $L^*(v) = \text{MERGE}$, any arc (v, u) with $L^*(u) = \text{FORK}$ must be in A' . Furthermore, we need to remove at least $\text{outdeg}(v) - 1$ many arcs from v in order to satisfy its label. Hence, $|a(v)| \geq d + o_{\text{MF}} + o_{\text{FF}} \geq \text{outdeg}(v) - 1$ for some $0 \leq d \leq \text{outdeg}(v)$. Thus, $|b(v)| + o_{\text{MF}} \leq 2|a(v)| \Leftarrow c + o_{\text{FF}} + o_{\text{MF}} \leq 2(d + o_{\text{FF}} + o_{\text{MF}}) \Leftarrow c \leq d + \text{outdeg}(v) - 1$.

If $\text{indeg}(v) = \text{outdeg}(v)$, we know (from the definition of L_a) that some inneighbor of v is labeled FORK by L_a . In this case, $c \leq \text{indeg}(v) - 1 = \text{outdeg}(v) - 1$. If $\text{indeg}(v) < \text{outdeg}(v)$, then $c \leq \text{indeg}(v) \leq \text{outdeg}(v) - 1$. In both cases, $c \leq d + \text{outdeg}(v) - 1$ and so $|b(v)| + o_{\text{MF}} \leq 2|a(v)|$.

Case 2. $T(v) = \text{MF}$. By definition, $|b(v)| \leq c + i_{\text{MM}}$, where $c \leq \text{outdeg}(v)$ is the number of outneighbors of v of type different from FM contained in B . Since $L^*(v) = \text{FORK}$, all incoming arcs (u, v) with $T(u) = \text{MM}$ must be contained in A' . If $T(u) = \text{FM}$, then the arc (v, u) is assigned by a to u and not to v (if it is in A'). Furthermore, we need to remove at least $\text{indeg}(v) - 1$ arcs, whereas arcs from inneighbors with type FM are not counted in $a(v)$. Hence, $|a(v)| \geq d + i_{\text{MM}} \geq \text{indeg}(v) - 1 - i_{\text{FM}}$, where $0 \leq d \leq \text{indeg}(v)$. It suffices to show that $|b(v)| - i_{\text{FM}} \leq 2|a(v)| \Leftarrow c + i_{\text{MM}} - i_{\text{FM}} \leq 2d + 2i_{\text{MM}} \Leftarrow c \leq d + \text{indeg}(v) - 1$.

If $\text{indeg}(v) = \text{outdeg}(v) \geq 2$, then $i_{\text{FM}} = 0$ since $L_a(v) = \text{MERGE}$. This implies that $|a(v)| = d + i_{\text{MM}} \geq \text{indeg}(v) - 1 = \text{outdeg}(v) - 1$. If $i_{\text{MM}} = 0$, then $|b(v)| \leq c \leq \text{outdeg}(v) \leq 2(\text{outdeg}(v) - 1) = 2(\text{indeg}(v) - 1) \leq 2|a(v)|$. If $i_{\text{MM}} > 0$, then we argue that at least one incoming arc of v was not removed, since otherwise the label of v would be changed in the greedy relabeling phase of the approximation. Hence, $|b(v)| \leq 2|a(v)| \Leftarrow c + i_{\text{MM}} - 1 \leq 2d + 2i_{\text{MM}} \Leftarrow c \leq d + \text{indeg}(v) = d + \text{outdeg}(v)$. If $\text{indeg}(v) > \text{outdeg}(v)$, then $c \leq \text{outdeg}(v) \leq \text{indeg}(v) - 1$. In both cases, we have $|b(v)| - i_{\text{FM}} \leq 2|a(v)|$.

Thus, $|b(v)| \leq 2|a(v)|$ for any incorrectly labeled vertex v . The same holds for correctly labeled vertices by [Lemma 1](#). By definition we know that $\biguplus_{v \in V} b(v) = B$ and $\biguplus_{v \in V} a(v) = A'$. Hence, $|\biguplus_{v \in V} b(v)| = |B| \leq 2|A'| = 2|\biguplus_{v \in V} a(v)|$. \square

[Algorithm 1](#) clearly runs in linear time, as computing the topological ordering of a DAG can be done in linear time. The third phase of the algorithm, where labels are changed, can also be executed in linear time by following any ordering of the vertices. We only change the label of a vertex if this leads to a better solution. To check if we have a better solution we only need to consider all incident arcs of a vertex and the labels of their endpoints. Since [Lemmas 1](#) and [2](#) consider all cases for all vertices $v \in V$, we conclude that ADDF can be approximated in linear time within a factor of two. \square

A Fixed-Parameter Algorithm.

Using the forbidden subgraph characterization ([Theorem 1\(3\)](#)), we can compute a digraph's arc-deletion distance d to a funnel in $\mathcal{O}(5^d \cdot (|V|^2 + |V| \cdot |A|))$ time: After contracting the arcs on each vertex with in- and outdegree one into a single arc, it is enough to destroy all subgraphs D_0 or D_1 as in [Theorem 1\(3\)](#). The optimal arc-deletion set to destroy all these subgraphs can be found by branching into the at most five possibilities for each subgraph D_0 or D_1 .

In this section, we show that, if the input is a DAG, we can solve ADDF in $\mathcal{O}(3^d \cdot (|V| + |A|))$ time instead; thus, in particular, we have linear running time if $d \in \mathcal{O}(1)$. Moreover, the resulting algorithm has also better running time in practice. As in the approximation algorithm, we again label the vertices. [Proposition 1](#) shows that, after the vertices are correctly labeled with either MERGE or FORK, solving ADDF can be done in linear time on DAGs. Hence, the complicated part of the problem lies in finding such a labeling.

In the following, we describe a search-tree algorithm that receives a DAG $D = (V, A)$ and an upper bound $d \in \mathbb{N}$ on the size of the solution as input, and it maintains a partial labeling $L: V \rightarrow \{\text{FORK}, \text{MERGE}\}$ of the vertices and a partial arc-deletion set A' that will constitute the solution in the end. Initially, $A' = \emptyset$ and $L(v)$ is undefined for each $v \in V$, denoted by $L(v) = \perp$. The algorithm exhaustively and alternately applies the data reduction and branching rules described below and aborts if $|A'| > d$. The rules either determine a label of a vertex (based on preexisting labels and on the degree of the vertex) or put some arcs into the solution A' . Herein, when we say that an arc is put into the solution, we mean that it is deleted from D and put into A' . To show that the algorithm finds a size- d arc deletion set to a funnel if there is one, we ensure that the rules are *correct*, meaning that, if there is a solution of size d that respects the labeling L and contains A' before applying a data reduction rule or branching rule, then there is also such a solution in at least one of the resulting instances.

[Reduction Rule 1](#) labels vertices of indegree (outdegree) at most one in a greedy fashion, based on the label of the single predecessor (successor) if it exists.

Reduction Rule 1 (Set Label). Let $v \in V$ be an unlabeled vertex.

Set $L(v) := \text{FORK}$ if at least one of the following is true: I) $\text{indeg}(v) = 0$; II) $\text{indeg}(v) = 1$ and $\exists u \in \text{in}(v) : L(u) = \text{FORK}$; III) $\text{outdeg}(v) > 1$, $\text{indeg}(v) = 1$ and $\forall u \in \text{out}(v) : L(u) \neq \perp$.

Set $L(v) := \text{MERGE}$ if at least one of the following is true: I) $\text{outdeg}(v) = 0$; II) $\text{outdeg}(v) = 1$ and $\exists u \in \text{out}(v) : L(u) = \text{MERGE}$; III) $\text{outdeg}(v) = 1$, $\text{indeg}(v) > 1$ and $\forall u \in \text{in}(v) : L(u) \neq \perp$.

Correctness of [Reduction Rule 1](#). Clearly, in a funnel the function label attributes every source a FORK label and every sink a MERGE label. Since destroying sinks and sources is not possible, [Reduction Rule 1](#) labels these vertices optimally.

Let v be a vertex with $\text{indeg}(v) = 1$, let $u \in \text{in}(v)$ be its only predecessor and assume $L(u) = \text{FORK}$. If we set $L(v) := \text{FORK}$, then `ArcDeletionSet` will not remove any arc when considering v . If some outgoing arc (v, w) is removed, then necessarily $L(w) = \text{FORK}$. Hence, if we instead set $L(v) := \text{MERGE}$ we also need to remove this arc, and potentially more. This implies that it is never worse to set $L(v) := \text{FORK}$ in this case. An analogous argument holds for the case where $\text{outdeg}(v) = 1$ and $L(u) = \text{MERGE}$ for the only successor u of v .

Finally, let v be a vertex where $\text{outdeg}(v) = 1$, $\text{indeg}(v) > 1$, and $\forall u \in \text{in}(v) : L(u) \neq \perp$. Since, by assumption, all outneighbors of v already have their labels set and satisfied, we only need to consider the label of v and of its only predecessor u . If $L(u) = \text{FORK}$ in an optimal solution, then we know by the previous case that it is optimal to set $L(v) := \text{FORK}$. If $L(u) = \text{MERGE}$ in an optimal solution, then we need to remove the arc (u, v) or some outgoing arc of v . That is, we need to remove at least one arc of v . By setting $L(v) := \text{FORK}$, we know that we need to remove exactly one arc of v . Hence, doing so is optimal. An analogous argument also holds for the last case where we set $L(v) := \text{MERGE}$. \square

Having labeled some vertices—whose labels will be as in an optimal labeling in some branch of the search tree—we simulate in `Satisfy Label` the behavior of `ArcDeletionSet` and remove arcs from labeled vertices.

Reduction Rule 2 (`Satisfy Label`). Let v be some vertex where $L(v) = \text{FORK}$ and $\text{indeg}(v) > 1$. If $\exists u \in \text{in}(v) : L(u) = \text{FORK}$, then put the arcs $\{(x, v) \mid x \in \text{in}(v) \wedge x \neq u\}$ into the solution. Otherwise, put $\{(x, v) \mid x \in \text{in}(v) \wedge L(x) = \text{MERGE}\}$ into the solution.

Let v be some vertex where $L(v) = \text{MERGE}$ and $\text{outdeg}(v) > 1$. If $\exists u \in \text{out}(v) : L(u) = \text{MERGE}$, then put the arcs $\{(v, x) \mid x \in \text{out}(v) \wedge x \neq u\}$ into the solution. Otherwise, put $\{(v, x) \mid x \in \text{out}(v) \wedge L(x) = \text{FORK}\}$ into the solution.

Correctness of Reduction Rule 1. The arcs removed by `Satisfy Label` would also be removed by `ArcDeletionSet` if all vertices had a label. Hence, if the labels are correct, by [Proposition 1](#), `Satisfy Label` only removes arcs that are present in some optimal arc-deletion set. \square

To assign a label to each remaining vertex, we branch into assigning one of the two possible labels. Key to an efficient running time is the observation that there is always a vertex which, regardless of the label set, has some incident arc which then has to be in the solution. This observation is exploited in [Branching Rule 1](#).

Branching Rule 1 (`Label Branch`). If there is some vertex v such that $\forall w \in \text{in}(v) : L(w) \neq \perp$ or $\exists w \in \text{in}(v) : L(w) = \text{FORK}$, then branch into two possibilities: Set $L(v) := \text{FORK}$; Set $L(v) := \text{MERGE}$.

If there is some vertex v such that $\forall w \in \text{out}(v) : L(w) \neq \perp$ or $\exists w \in \text{out}(v) : L(w) = \text{MERGE}$, then branch into two possibilities: Set $L(v) := \text{FORK}$; Set $L(v) := \text{MERGE}$.

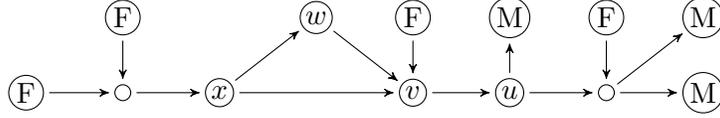


Figure 6: A DAG where [Satisfy Label](#) and [Set Label](#) are not applicable. The letter F stands for a FORK label and M stands for MERGE. [Label Branch](#) cannot be applied to v since u does not have a label, yet it can be applied to $x \in \text{in}^*(w)$.

The final [Branching Rule 2](#) tries all possibilities of satisfying a label of a vertex.

Branching Rule 2 (Arc Branch). If there is a vertex v with $L(v) = \text{FORK}$ and $\text{indeg}(v) > 1$, then branch into all possibilities of removing all but one incoming arc of v . If there is a vertex v with $L(v) = \text{MERGE}$ and $\text{outdeg}(v) > 1$, then branch into all possibilities of removing all but one outgoing arc of v .

The correctness of [Arc Branch](#) follows from [Proposition 1](#). To show the algorithm's correctness, it remains to show the following central lemma.

Lemma 3. *Let D be a DAG. If [Label Branch](#), [Arc Branch](#), [Set Label](#) and [Satisfy Label](#) are not applicable, then D is a funnel and all vertices have a label.*

Proof. First, note that if the label of a vertex has been set, it will be satisfied by either applying [Satisfy Label](#) or by branching with [Arc Branch](#). Since satisfying all labels turns D into a funnel ([Theorem 1\(2\)](#)), it is enough to show that all vertices have a label if [Label Branch](#), [Set Label](#), and [Satisfy Label](#) are not applicable.

We first show that if there is some forbidden subgraph $D' = (V', A') \subseteq D$, that is, D' is isomorphic to some D_i from [Theorem 1\(3\)](#), and if additionally [Set Label](#) and [Satisfy Label](#) are not applicable, then [Label Branch](#) is applicable. Let D' be the forbidden subgraph in D with the smallest number of vertices. Let $v, u \in V'$ be two (not necessarily distinct) vertices in D' such that $\text{indeg}_{D'}(v) > 1$, $\text{outdeg}_{D'}(u) > 1$. Observe that all vertices between v and u in D' (if any) have in- and outdegree one in D , because D' has the smallest number of vertices. We distinguish two cases.

Case 1: $\forall w \in \text{in}_D(v) : L(w) \neq \perp$. Then either $\text{outdeg}_D(v) > 1$, meaning that we can apply [Label Branch](#) (as required), or $L(v) = \text{MERGE}$ due to [Set Label](#). Since all vertices between v and u have in- and outdegree one, we also know from the latter case that there is some arc (x, y) in the (uniquely defined) (v, u) -path such that $L(x) = \text{MERGE}$ and $L(y) = \perp$. Note that it cannot happen that $L(y) = \text{FORK}$ since [Satisfy Label](#) is not applicable. We also know that $\text{outdeg}_D(y) > 1$ since [Set Label](#) is not applicable. This implies [Label Branch](#) is applicable on y .

Case 2: $\exists w \in \text{in}_D(v) : L(w) = \perp$. This case is illustrated in [Figure 6](#). We show that we can find some vertex in $\text{in}^*(w)$ to which we can apply [Label Branch](#). Consider the longest (x, w) -path that only contains vertices in $\text{in}^*(w)$ which do not have a label. Clearly, $\forall y \in \text{in}(x) : L(y) \neq \perp$ and $\text{indeg}(x) > 0$ since all sources have a label. Thus, we can apply [Label Branch](#) on x .

Since only these two cases are possible, and in both we can apply [Label Branch](#), it follows, by contraposition, that D is a funnel and all vertices have a label if [Label Branch](#), [Set Label](#), and [Satisfy Label](#) are not applicable. \square

By combining the previous data reduction and branching rules, we obtain a search-tree algorithm for ADDF on DAGs:

Theorem 4. *ADDF can be solved in time $\mathcal{O}(3^d \cdot (|V| + |A|))$, where d is the arc-deletion distance to a funnel of a given DAG $D = (V, A)$.*

Proof. The algorithm is as follows. On input of a DAG D , budget $d \in \mathbb{N}$, partial labeling L , and partial solution A' (initially, L does not label any vertex and $A' = \emptyset$), apply [Set Label](#) and [Satisfy Label](#) until they do not apply anymore. If $|A'| > d$, then abort. Otherwise, apply [Label Branch](#), if possible. In each of the two resulting instances, apply [Satisfy Label](#) until it does not apply anymore, and then apply [Arc Branch](#), if possible. Make a recursive call for each of the resulting instances. If no branching rule applies and $|A'| \leq d$, return A' as a solution.

By the correctness of the individual rules, the algorithm finds a solution if there is one (and otherwise does not return anything): From [Lemma 3](#) we know that the algorithm turns the input into a funnel. It remains to prove the running time bound. With some simple bookkeeping and auxiliary tables, we can apply [Set Label](#) and [Satisfy Label](#) to all vertices of D in total running time of $\mathcal{O}(|V| + |A|)$. In the same running time we can find out whether [Label Branch](#) and [Arc Branch](#) is applicable. Hence, we need at most $\mathcal{O}(|V| + |A|)$ running time per recursive call.

It remains to bound the size of the search tree, that is, the outtree \mathcal{T} whose vertices are the calls of the algorithm and whose edges represent recursive calling relation. To ease the analysis, we instead bound the modified tree \mathcal{T}' in which we replace the outneighbors of a vertex corresponding to the recursive calls resulting from [Arc Branch](#) by a binary tree as follows. If [Arc Branch](#) branches into all possibilities of putting into the solution a subset of size $d' - 1$ from an arc set B of size d' , we instead recursively choose two arcs and introduce two recursive calls in which one of the two arcs is put into the solution until B has size 1. Clearly, the size of \mathcal{T} is upper bounded by the size of \mathcal{T}' .

We claim that \mathcal{T}' has maximum outdegree three. Consider the instances resulting from [Label Branch](#). Without loss of generality assume that the first portion of [Label Branch](#) was applied. The proof for the second portion is analogous. If $L(v)$ was set to FORK, then after [Satisfy Label](#) has been applied exhaustively, [Label Branch](#) is not applicable. Otherwise, if $L(v)$ was set to MERGE, the modified [Label Branch](#) with two branches is applied. Hence, indeed, there are at most three recursive calls.

To bound the size of \mathcal{T}' , consider a path P from the root to a leaf. Whenever [Label Branch](#) is applied, in each of the following recursive calls, at least one arc is put into the solution: Without loss of generality, assume that the first portion of [Label Branch](#) was applied. The proof for the second portion is analogous. If $L(v)$ was set to FORK, then [Satisfy Label](#) will put at least one arc into the solution (note that,

since `Set Label` is not applicable, v has indegree at least two). Otherwise, if $L(v)$ was set to `MERGE`, then, since `Set Label` is not applicable, `Arc Branch` is applicable and will put at least one outarc of v into the solution. Hence, P has length at most d , since no further recursive calls are made if $|A'| > d$. Combining this with the fact that \mathcal{T}' has outdegree at most three, it follows that \mathcal{T}' has size $O(3^d)$.

Hence, the running time of the search-tree algorithm is $\mathcal{O}(3^d \cdot (|V| + |A|))$. \square

To improve the running time of the search-tree algorithm in practice, we compute a lower bound of the arc-deletion distance to a funnel of the input and we stop expanding a branch of the search tree when the lower bound exceeds the available budget. A simple method for computing a lower bound is to find arc-disjoint forbidden subgraphs. Clearly, the sum of the arc-deletion distances to a funnel of the subgraphs found is not larger than the distance of the input DAG. To find such subgraphs, we first look for vertices with both in- and outdegree greater than one, which are not allowed in funnels. Then we search for paths v_1, v_2, \dots, v_k such that $\text{indeg}(v_1) > 1$ and $\text{outdeg}(v_k) > 1$. With some bookkeeping we can find a maximal set of arc-disjoint forbidden subgraphs in linear time.

4 Empirical Evaluation of the Developed Algorithms

In this section, we empirically evaluate the approximation algorithm and the fixed-parameter algorithm for ADDF described in Section 3. We used artificial data sets and data based on publicly available real-world graphs. Our experiments show that both our algorithms are efficient in practice.

We implemented the algorithms in Haskell 2010. All experiments were run on an Intel[®] Xeon[®] E5-1620 3.6 GHz processor with 64 GB of RAM. The operating system was GNU/Linux, with kernel version 4.4.0-67. For compiling the code, we used GHC version 7.10.3. The code is released as free software [Millani, 2017b].

Experiments on Synthetic Funnel-like DAGs. We generated random funnel-like DAGs through the following steps. (1) Choose the number of vertices, arc density $p \in [0, 1]$, and some $s \in \mathbb{N}$. (2) Fix a topological ordering of the vertices. (3) Uniformly at random assign a label `FORK` or `MERGE` to each vertex. (4) Create an out-forest with `FORK` vertices, and an in-forest with `MERGE` vertices. (5) Add random arcs from `FORK` to `MERGE` vertices until a density of p (relative to the maximum number of arcs allowed by the labeling) is achieved. (6) Add s random arcs which respect the topological ordering. Steps (1) through (5) result in a funnel which we call *planted* funnel below.

For a fixed labeling, the algorithm above generates funnels uniformly at random from the input parameters. The labeling, however, is drawn uniformly at random from all $2^{|V|}$ possible labelings, without considering how many different funnels exist with a given labeling. Hence, funnels with fewer arcs have a larger chance of being

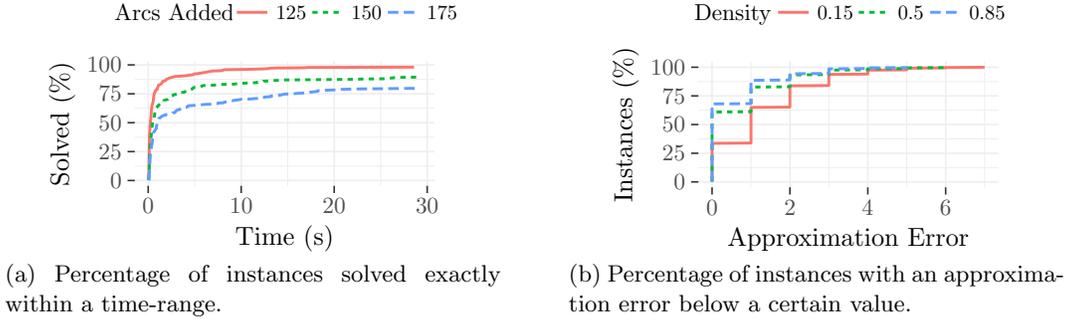


Figure 7: Running time and approximation error.

generated than funnels with many arcs (when compared to the chances in a uniform distribution). We consider this bias to be harmless for the experiments since, for the exact algorithm, the number of arcs is not decisive for the running time, and for the approximation algorithm the number of arcs should not have a big impact on the solution quality.

For $n \in \{250, 300, 500, 1000\}$, $p \in \{0.15, 0.5, 0.85\}$ and $s \in \{125, 150, 175\}$ we generated 30 funnels with n vertices and density p , and then added s random arcs as described above. This gives us a total of 1080 DAGs.

Our fixed-parameter algorithm was able to compute the arc-deletion distance to funnel of 1059 instances (98%) within 10 minutes. The approximation algorithm finished on average in less than 72 ms. A cumulative curve with the percentage of instances solved within a certain time range is depicted in Figure 7a. Most instances were solved fairly quickly: Within 15 seconds 932 (86%) instances were solved optimally. We can also observe that there were essentially two types of instances: Easy ones which were solved within few seconds, and harder ones which often were not solved within 10 minutes. That is, if we limit the running time to five seconds, then we can solve 856 (79%) instances, and if we increase it to sixty seconds, we can solve only 141 additional instances.

Figure 7b shows the relation between the error of the approximation algorithm with the density of the planted funnel. The approximation algorithm found an optimal solution in 574 (54%) instances, and in 260 (25%) it removed only one more arc than necessary. As the arc-deletion distance to a funnel of most instances was greater than 100, this means that the approximation ratio is very close to one. Since the DAGs used here are already close to funnels, most decisions of the approximation algorithm are correct. Intuitively, having correct local information helps the approximation make a globally optimal decision, and so it is unsurprising that the approximation factor in funnel-like DAGs is much better than the theoretical bound. This is supported also by the fact that the approximation performed worse on sparse planted funnels than on dense ones, since the proportion of “wrong” information regarding

the arcs is larger on sparse funnels (when adding the same number of random arcs).

Experiments on DAGs Based on Real-World Data Sets. We obtained ten digraphs from the Konect database [Kunegis, 2013], containing food-chains, interactions between animals, and source-code dependencies. We also downloaded the dependency network of all packages in Arch Linux.³ Since most of the gathered digraphs contain cycles, we performed a pre-processing step turning them into DAGs: we merged cycles into a single vertex, and then removed self-loops. For each of the eleven DAGs we computed a lower bound and an approximation of its arc-deletion distance to funnel. We also attempted to compute the real distance, stopping the algorithm if no solution was found within four hours.

The dataset was divided into six small DAGs (≤ 156 vertices and ≤ 1197 arcs) and five larger ones (≥ 5730 vertices and ≥ 26218 arcs). In the small ones, our fixed-parameter algorithm solved ADDF within one second, and our approximation algorithm found the correct distance in ≤ 2 ms. In two of the six small DAGs the distance was 60 and 129, which means that the exact algorithm is in practice much faster than what the worst-case upper bound predicts.

On the larger DAGs the fixed-parameter algorithm could not solve ADDF within four hours. By computing a lower bound for the distance, we managed to give an upper bound for the approximation factor, which was at most 1.16. This means that the approximation algorithm is practical since it is fast (≤ 228 ms on average) and yields a near-optimal solution. Relative to the number of arcs, the arc-deletion distance to a funnel parameter was small (9% on average).

5 Conclusion

We believe that our results add to the relatively small list of fixed-parameter tractability results for directed graphs and introduce a novel interesting structural parameter for directed (acyclic) graphs. In particular, our approximation and fixed-parameter algorithms could help to establish the arc-deletion distance to a funnel as a useful “distance-to-triviality measure” [Cai, 2003, Guo et al., 2004, Niedermeier, 2010] for designing fixed-parameter algorithms for NP-hard problems on DAGs. We leave open whether computing the arc-deletion distance to funnel of a DAG is APX-hard. Finally, funnels might provide a basis for defining some useful digraph width or depth measures [Ganian et al., 2014, 2016, Millani, 2017a].

References

N. Ailon and N. Alon. Hardness of fully dense problems. *Information and Computation*, 205(8):1117–1129, 2007.

³Listed at <https://www.archlinux.org/packages/> and obtained using `pacman`.

- J. Bang-Jensen and G. Z. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer, 2008. ISBN 1848009976, 9781848009974.
- S. Bessy, F. V. Fomin, S. Gaspers, C. Paul, A. Perez, S. Saurabh, and S. Thomassé. Kernels for feedback arc set in tournaments. *Journal of Computer and System Sciences*, 77(6):1071–1078, 2011.
- L. Cai. Parameterized complexity of vertex colouring. *Discrete Appl. Math.*, 127(3):415–429, 2003.
- P. Charbit, S. Thomassé, and A. Yeo. The minimum feedback arc set problem is np-hard for tournaments. *Combinatorics, Probability and Computing*, 16(1):1–4, 2007.
- S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10(2):111–121, 1980. ISSN 0304-3975. doi: [http://dx.doi.org/10.1016/0304-3975\(80\)90009-2](http://dx.doi.org/10.1016/0304-3975(80)90009-2).
- R. Ganian, P. Hlinený, J. Kneis, A. Langer, J. Obdržálek, and P. Rossmanith. Digraph width measures in parameterized algorithmics. *Discrete Appl. Math.*, 168:88–107, 2014.
- R. Ganian, P. Hlinený, J. Kneis, D. Meister, J. Obdržálek, P. Rossmanith, and S. Sikdar. Are there any good digraph width measures? *J. Comb. Theory, Ser. B*, 116:250–286, 2016.
- J. Guo, F. Hüffner, and R. Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proc. 1st IWPEC*, pages 162–173. Springer, 2004.
- C. Kenyon-Mathieu and W. Schudy. How to rank with few errors. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 95–103. ACM, 2007.
- J. Kunegis. KONECT – The Koblenz Network Collection. In *Proc. 22nd WWW*, pages 1343–1350. ACM, 2013.
- J. Lehmann. The computational complexity of worst case flows in unreliable flow networks. Bachelor thesis, Institut für Theoretische Informatik, Universität zu Lübeck, Oct 2017.
- J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *Proc. 15th ACM SIGKDD*, pages 497–506. ACM, 2009. ISBN 978-1-60558-495-9. doi: 10.1145/1557019.1557077.
- M. G. Millani. Funnels—algorithmic complexity of problems on special directed acyclic graphs. Master thesis, Department of Electrical Engineering and Computer Science, TU Berlin, Aug 2017a. URL <http://fpt.akt.tu-berlin.de/publications/theses/MA-marcelo-millani.pdf>.

- M. G. Millani. Parfunn – Parameters for Funnels, Aug 2017b. URL <https://gitlab.tubit.tu-berlin.de/mgmillani1/parfunn>.
- R. Niedermeier. Reflections on multivariate algorithmics and problem parameterization. In *Proc. 27th STACS*, pages 17–32. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010.
- R. van Bevern, R. Bredereck, M. Chopin, S. Hartung, F. Hüffner, A. Nichterlein, and O. Suchý. Fixed-parameter algorithms for DAG partitioning. *Discrete Appl. Math.*, 220:134–160, 2017. ISSN 0166-218X. doi: <https://doi.org/10.1016/j.dam.2016.12.002>.