When Recursion is Better than Iteration: A Linear-Time Algorithm for Acyclicity with Few Error Vertices*

Daniel Lokshtanov† M. S. Ramanujan‡ Saket Saurabh§†

Abstract

Planarity, bipartiteness and (directed) acyclicity are basic graph properties with classic linear time recognition algorithms. However, the problems of testing whether a given (di)graph has k vertices whose deletion makes it planar, bipartite or a directed acyclic graph (DAG) are all fundamental NP-complete problems when k is part of the input. As a result, a significant amount of research has been devoted to understanding whether, for every *fixed* k, these problems admit a polynomial time algorithm (where the exponent in the polynomial is independent of k) and in particular, whether they admit linear time algorithms.

While we now know that for any fixed k, we can test in linear time whether a graph is k vertices away from being planar [FOCS 2009, SODA 2014] or bipartite [SODA 2014, SICOMP 2016], the best known algorithms in the case of directed acyclicity are the algorithm of Garey and Tarjan [IPL 78] which runs in time $\mathcal{O}(n^{k-1}m)$ and the algorithm of Chen, Liu, Lu, O'Sullivan and Razgon [JACM 2008] which runs in time $\mathcal{O}(k!4^k k^4 nm)$. In other words, it has remained open whether it is possible to recognize in linear time, a graph which is *2 vertices* away from being acyclic!

In this paper, we settle this question by giving an algorithm that decides whether a given graph is k vertices away from being acyclic, in time $\mathcal{O}(k!4^k k^5(n+m))$. That is, for every fixed k, our algorithm runs in time $\mathcal{O}(m+n)$, thus mirroring the case for planarity and bipartiteness.

Our algorithm is designed via a general methodology that shaves off a factor of n from some algorithms that use the powerful technique of iterative compression. The two main features of our methodology are: (i) This is the first generic technique for designing linear time algorithms for *directed cut-problems* and (ii) it can be used in combination with future improvements in algorithms for the *compression* version of other well-studied cut-problems such as Multicut and Directed Subset Feedback Vertex Set.

1 Introduction

The classes of planar graphs, bipartite graphs and acyclic graphs are among the fundamental graph classes with seminal linear time recognition algorithms. However, the decision problem, that is, deciding whether there is a vertex set of size k (an outlier set) whose *removal* places the input graph in a specific graph class is NP-complete for numerous basic graph classes, including the aforementioned three. As a result, a significant amount of research has been devoted to understanding whether, for every *fixed* k, these problems admit an algorithm with running time $\mathcal{O}(n^c)$ where c is independent of k (through the paradigm of fixed-parameter tractability and parameterized complexity) and if so, what the best possible value of c is.

In fact, this area of research actually predates the area of parameterized complexity. The genesis of parameterized complexity is in the theory of graph minors, developed by Robertson and Seymour [46, 47, 48]. Some of the important algorithmic consequences of this theory include $\mathcal{O}(n^3)$ algorithms for Disjoint Paths and $\mathcal{F}$-Deletion for every fixed value of k. Another early work on obtaining algorithms with improved dependence on the input size was the seminal work of Bodlaender giving a linear time algorithm for Treewidth [2, 3].

However, the advent of parameterized complexity started to shift the focus away from the running time dependence on input size to the dependence *only* on the parameter. That is, the goal became designing parameterized algorithms with running time upper bounded by $f(k)n^{\mathcal{O}(1)}$, where the function f grows as slowly as possible, without worrying about the polynomial dependence on n at all. But the last decade has witnessed several efforts aimed at obtaining linear time parameterized algorithms (or algorithms having the best possible dependence on the input size) that compromise as little as possible on the dependence of the running time on the parameter k. The gold standard for these results are algorithms with linear dependence on input size as well as provably optimal dependence on

*Research supported by *Pareto-Optimal Parameterized Algorithms*, ERC Starting Grant 715744 and *Parameterized Approximation*, ERC Starting Grant 306992 and *X-Tract*, Austrian Science Fund (FWF, project P26696).

†University of Bergen, Norway, danielo@ii.uib.no

‡University of Warwick, UK, R.Maadapuzhi-sridharan@warwick.ac.uk

§The Institute of Mathematical Sciences, HBNI, India and UMI ReLax, saket@imsc.res.in.

the parameter under a complexity hypothesis such as the Exponential Time Hypothesis (ETH).

It was only relatively recently that the first linear time algorithms were obtained for testing whether a graph is k vertices away from being planar [33, 25] or bipartite [31, 43]. Some of the other important results in this line of research include the linear time algorithms for Subgraph Isomorphism [12], Subset Feedback Vertex Set [37], Planar $\mathcal{F}$-Deletion [2, 3, 16, 19, 18], Crossing Number [22, 23, 28], Interval Vertex Deletion [6], as well as a single-exponential and linear time parameterized constant factor approximation algorithm for Treewidth [4]. In addition, there are several recent results which provide parameterized algorithms with improved (but not linear) dependence on input size for a host of problems [24, 26, 27, 34, 29, 30].

However, in spite of this progress, a linear time algorithm for testing whether a graph is k vertices away from being acyclic (for every fixed k), has still proved elusive. In fact, even the existence of a $\mathcal{O}(n^c)$ algorithm for every fixed k was widely posed as the most important open problem in parameterized complexity for well over a decade starting from the first few papers on fixed-parameter tractability (FPT) [13, 14]. In a break-through paper, Chen, Liu, Lu, O'Sullivan and Razgon [7] answered this question in the affirmative by proving that this problem, formally called Directed Feedback Vertex Set (DFVS) and defined below, is *fixed-parameter tractable* (FPT). That is, it has an algorithm running in time $f(k)n^c$ for some computable function f and a constant c independent of k.

Directed Feedback Vertex Set (DFVS)

Input:	A digraph D on n vertices and m edges and a positive integer k.
Parameter:	k
Problem:	Does there exist a vertex subset of size at most k that intersects every cycle in D?

The algorithm of Chen et al. runs in time $\mathcal{O}(4^k k! k^4 n^4)$ where n is the number of vertices in the input digraph. Subsequently, it was observed that, in fact, the running time of this algorithm is $\mathcal{O}(4^k k! k^4 nm)$ (see for example, [10]). That is, it runs in time $\mathcal{O}(mn)$ for every fixed k. On the other hand, Garey and Tarjan [21] gave an elegant algorithm for DFVS running in time $\mathcal{O}(n^{k-1}m)$ (as opposed to the trivial $\mathcal{O}(n^k)$ algorithm). This algorithm clearly outperforms the algorithm of Chen et al. for $k = 1$ and runs in linear time. However, although the techniques used by Chen et al. have found numerous applications subsequently, it remained open whether one could detect in linear time, even a vertex subset of size 2 that intersects every cycle in a given digraph!

In this paper we resolve this question (for every fixed k) and obtain the first linear-time FPT algorithm for DFVS. In particular we prove the following theorem.

Theorem 1.1. *There is an algorithm for* DFVS *running in time* $\mathcal{O}(k!4^k k^5 \cdot (n + m))$.

Our algorithm achieves the best possible dependence on the input size while matching the current best-known parameter-dependence – that of the algorithm of Chen et al. [7], up to a $O(k)$ factor. Since it is well known that DFVS cannot be solved in time $2^{o(k)}n^c$ for any constant c under the Exponential Time Hypothesis (ETH) [10, 11], our algorithm is in fact *nearly-optimal*. Finally, our algorithm only relies on basic algorithmic and combinatorial tools.

Methodology. At the heart of numerous FPT algorithms lies the fact that, if one could efficiently compute a sufficiently good approximate solution, it is then sufficient to design an FPT algorithm for the "compression version" of a problem in order to obtain an FPT algorithm for the general version. In the compression version of a problem, the input also includes an approximate solution whose size depends only on the parameter. Since a given approximate solution may be used to infer significant structural information about the input, it is usually much easier to design FPT algorithms for the compression version than for the original problem. The efficiency of this approach clearly depends on two factors – (a) the time required to compute an approximate solution and (b) the time required to solve the compression version of the problem when the approximate solution is provided as input.

This approach has been used mainly in the following two settings. In the first setting, the objective is the design of linear-time FPT algorithms. In this setting, for certain problems, it can be shown that if the treewidth of the input graph is bounded by a function of the parameter then the problem can be solved by a linear-time FPT algorithm (either designed explicitly or obtained by invocation of an appropriate algorithmic meta-theorem). On the other hand, if the treewidth of the input graph exceeds a certain bound, then there is a sufficiently large (induced) matching which one can contract and obtain an instance whose *size* is a constant fraction of that of the original input. Now, the algorithm is recursively invoked on the reduced instance and certain problem-specific steps are used to convert the recursively computed solution into an approximate solution for the given instance. Then, a linear-time FPT algorithm for the compression version is executed to solve the general problem on this instance. Some of the results that fall under this paradigm are Bodlaender's linear FPT algorithm for TREEWIDTH [3], the FPT-approximation algorithms for TREEWIDTH [4, 44], as well as algorithms for VERTEX PLANARIZATION [25, 33]. Let us call this the method of *recursive compression*. This is one of the most commonly used techniques in designing linear-time FPT-algorithms on *undirected graphs*. However, this approach of recursion combined with a win/win approach based on the treewidth of the graph, fails when one attempts to extend it to directed graphs.

On the other hand, when designing FPT algorithms where the dependence on the input is *not* required to be linear, one can use the *iterative compression* technique, introduced by Reed, Smith and Vetta [45]. Here the input instance is gradually built up by simple operations, such as vertex additions. After each operation, an optimal solution is re-computed, starting from an optimal solution to the smaller instance. Though it is very helpful for problems on directed graphs, by its very definition, the iterative compression technique does not lend itself to the design of linear-time FPT algorithms. Hence, it appears that one has to look for alternative ways when aiming for linear-time FPT algorithms. In recent years, some of the problems which were initially solved using the iterative compression technique, have seen the development of entirely new algorithms. Examples include the first linear-time FPT algorithms for the ODD CYCLE TRANSVERSAL, ALMOST 2-SAT, EDGE UNIQUE LABEL COVER and NODE UNIQUE LABEL COVER problems [31, 43, 32, 38]. All of these algorithms are based on branching and linear programming techniques.

Another general approach to the design of linear-time FPT algorithms has been introduced by Marx et al. [39]. These algorithms are based on the "Treewidth Reduction Theorem"' which states that in undirected graphs, for any pair of vertices s and t, all minimal s-t separators of bounded size are contained in a part of the graph that has bounded treewidth.

However, this technique is also specifically designed for undirected graphs and hence fails when addressing problems on directed graphs. Our main contribution is a novel approach for 'lifting' linear-time FPT algorithms for the compression version of feedback-set problems on *digraphs* to linear-time FPT algorithms for the general version of the problem. Although our approach follows the recursive compression paradigm pioneered by Bodlaender [3] in his celebrated linear time FPT algorithm for TREEWIDTH, we need to identify highly non-trivial structure in the given digraph to be even able to compute the parts of the input digraph which we want to 'recursively compress'.

Given a digraph D, we say that S is a *directed feedback vertex set (dfvs)* if deleting S from D results in a DAG. At the core of our algorithm lies the following new structural lemma regarding digraphs with a small dfvs.

LEMMA 1.1. *Let D be a strongly connected digraph and $p \in \mathbb{N}$. There is an algorithm that, given D and p, runs in time $\mathcal{O}(p^2 m)$ (where m is the number of arcs in D) and either correctly concludes that D has no dfvs of size at most p or returns a set S with at most $2p+2$ vertices such that one of the following holds.*

- *S is a dfvs for D.*
- *$D - S$ has at least 2 non-trivial strongly connected components (strongly connected components with at least 2 vertices).*
- *The number of arcs of D whose head and tail occur in the same non-trivial strongly connected component of $D - S$ (arcs participating in a cycle of $D - S$) is at most $\frac{m}{2}$.*
- *If D has a dfvs of size at most p then $D - S$ has a dfvs of size at most $p - 1$.*

Our linear-time FPT algorithm for DFVS is obtained by a careful interleaving of the algorithm of Lemma 1.1 with an algorithm solving the compression version of DFVS (in this case, the compression routine of Chen et al. [7]). The proof of Lemma 1.1 itself is based on extending the notion of important sequences [36] to digraphs, and then analyzing a single such sequence. Furthermore, the proof of Lemma 1.1 only relies on properties of DFVS that are shared by several other feedback set and graph separation problems. Hence, we also prove a more general version of this lemma and show how it can be used as a black box to shave off a factor of n from existing iterative compression based algorithms for other problems which satisfy certain conditions. This results in speeding up by a factor of n, the current best FPT algorithms for MULTICUT [40, 41, 5] and DIRECTED SUBSET FEEDBACK VERTEX SET [8, 9].

2 Preliminaries

Parameterized Complexity. Formally, a *parameterization* of a problem is the assignment of an integer k to each input instance and we say that a parameterized problem is *fixed-parameter tractable* (FPT) if there is an algorithm that solves the problem in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where $|I|$ is the size of the input instance and f is an arbitrary computable function depending only on the parameter k. For more background, the reader is referred to the monographs [15, 17, 42, 10].

Digraphs. For a digraph D and vertex set $X \subseteq V(D)$, we say that X is a *dfvs* of D if X intersects every cycle in D. We say that X is a minimal dfvs of D if no proper subset of X is also a dfvs of D. We call X a minimum dfvs of D if there is no smaller dfvs of D. For an arc $(u,v) \in A(D)$, we refer to u as the *tail* of the arc and v as the *head*. D is a bidirectional digraph if for every $(u,v) \in A(D)$, there is an arc $(v,u) \in A(D)$. For a subset X of vertices, we use $N^+(X)$ to denote the set of out-neighbors of X and $N^-(X)$ to denote the set of in-neighbors of X. We use $N^i[X]$ to denote the set $X \cup N^i(X)$ where $i \in \{+,-\}$. We denote by $A[X]$ the subset of $A(D)$ with both endpoints in X. A strongly connected component of D is a maximal subgraph in which every vertex has a directed path to every other vertex. We say that a strongly connected component is *non-trivial* if it has at least 2 vertices and *trivial* otherwise. For disjoint vertex sets X and Y, Y is said to be reachable from X if for *every* vertex $y \in Y$, there is a vertex $x \in X$ such that the digraph contains a directed path from x to y.

Structures. For $\eta \in \mathbb{N}$, an η-structure is a tuple where the first element of the tuple is a digraph D with the remaining elements of the tuple being relations of arity at most η over $V(D)$. Formally, an η-structure is a tuple $(D, R_1, \ldots, R_\ell)$ where D is a digraph and for every $i \in [\ell]$, $R_i \subseteq V(D)^p$ for some $p \in [\eta]$.

Two η-structures Q_1 and Q_2 are said to have the same *type* if they both have the same number of elements and the corresponding relations have the same arity when non-empty. Formally, we say that Q_1 and Q_2 have the same *type* if $Q_1 = (D_1, R_1, \ldots, R_\ell)$, $Q_2 = (D', R'_1, \ldots, R'_\ell)$ and for each $i \in [\ell]$, there exists $p \in [\eta]$ such that, if $R_i, R'_i \neq \emptyset$ then $R_i, R'_i \subseteq V(D)^p$ and $R_i, R'_i \not\subseteq V(D)^{p-1}$.

The *size* of an η-structure $Q = (D, R_1, \ldots, R_\ell)$ is denoted as $|Q|$ and is defined as $m + n + \eta \cdot \Sigma_{i=1}^{\ell} |R_i|$, where m and n are the number of arcs and vertices in D respectively and $|R_i|$ is the number of tuples in R_i. In this paper, whenever we talk about a family $\mathcal{Q}$ of η-structures, it is to be understood that $\mathcal{Q}$ only contains η-structures which are pairwise of the same type and this type is also called the *type of* $\mathcal{Q}$.

Definition 1. *Let $Q = (D, R_1, \ldots, R_\ell)$ be an η-structure. For a set $X \subseteq V(D)$, we define the induced substructure $Q[X] = (D[X], R_1|_X, \ldots, R_\ell|_X)$ where $R_i|_X$ is the restriction of the relation R_i to the set X, that is, $R_i|_X = R_i \cap (\bigcup_{p\in[\eta]} X^p)$. For any $X \subseteq V(D)$, we denote by $Q - X$ the substructure $Q[V(D) \setminus X]$.*

Definition 2. *Let $\mathcal{Q}$ be a family of η-structures. We say that $\mathcal{Q}$ is* hereditary *if for every $Q \in \mathcal{Q}$, every induced substructure of Q is also in $\mathcal{Q}$. We say that a family $\mathcal{Q}$ of η-structures is* linear-time recognizable *if there is an algorithm that, given an η-structure Q, runs in time $\mathcal{O}(|Q|)$ and correctly decides whether $Q \in \mathcal{Q}$. Finally, we say that $\mathcal{Q}$ is* rigid *if the following two properties hold:*

- *For every η-structure $Q = (D, R_1, \ldots, R_\ell)$, if D has no arcs then $Q \in \mathcal{Q}$ and*
- *$Q = (D, R_1, \ldots, R_\ell) \in \mathcal{Q}$ if and only if for every strongly connected component C in the digraph D, the induced substructure $Q[C] \in \mathcal{Q}$.*

The $\mathcal{Q}$-Deletion(η) problem is formally defined as follows.

$\mathcal{Q}$-Deletion(η)

Input:	An η-structure $Q = (D, R_1, \ldots, R_\ell)$ and a positive integer k.
Parameter:	k
Problem:	Does there exist a set $X \subseteq V(D)$ of size at most k such that $Q - X \in \mathcal{Q}$?

Our main contribution is a theorem (Theorem 3.1) that, under certain conditions which are fulfilled by several well-studied special cases of $\mathcal{Q}$-Deletion(η), guarantees an FPT algorithm for $\mathcal{Q}$-Deletion(η) whose running time has a specific form.

A set $X \subseteq V(D)$ such that $Q - X \in \mathcal{Q}$ is called a *deletion set of Q into $\mathcal{Q}$*. In the $\mathcal{Q}$-Deletion(η) Compression problem, the input is a triple $(Q, k, \hat{W})$ where (Q, k) is an instance of $\mathcal{Q}$-Deletion(η) and $\hat{W}$ is a vertex set such that $Q - \hat{W} \in \mathcal{Q}$. The question remains the same as for $\mathcal{Q}$-Deletion(η). However, the parameter for this problem is $k + |\hat{W}|$ and for the input to be interesting, $|\hat{W}| > k$ (otherwise the instance is trivially a Yes instance). We say that an algorithm $\mathbf{A}$ is an algorithm for the $\mathcal{Q}$-Deletion(η) Compression problem if, on input $Q, k, \hat{W}$ the algorithm either correctly concludes that (Q, k) is a No instance of $\mathcal{Q}$-Deletion(η) or *computes* a *smallest* set X of size at most k such that $Q - X \in \mathcal{Q}$. Note that the requirement that the set X is a *smallest* such set, does not affect generality for the following reason. Let $\mathbf{A}'$ be an algorithm that, on input $Q, k, \hat{W}$ either correctly concludes that (Q, k) is a No instance of $\mathcal{Q}$-Deletion(η) or computes a (not necessarily smallest) set X of size at most k such that $Q - X \in \mathcal{Q}$. We can then set $\hat{W}' = X$ and run $\mathbf{A}'$ with input $Q, k-1, \hat{W}'$ to compute a smaller solution if one exists and repeat this procedure (at most k times) until we find a smallest such set. In this case, the running time of $\mathbf{A}$ is bounded by k times that of $\mathbf{A}'$. However, for all specific problems we address in this paper, we will not be required to take this route because the existing compression algorithms for DFVS, Multicut, and Directed Subset Feedback Vertex Set which we invoke can already be seen to output a *smallest* set of size at most k which is a solution.

3 The FPT algorithm for $\mathcal{Q}$-deletion(η)

In this section, we formally state our main theorem and demonstrate how a direct application of this theorem speeds up by a factor of n, existing FPT algorithms for certain well-studied feedback set and graph separation problems. We then prove this theorem assuming a generalization of Lemma 1.1 as a black box.

Theorem 3.1. *Let $\eta \in \mathbb{N}$ and let $\mathcal{Q}$ be a linear-time recognizable, hereditary and rigid family of η-structures. Let $\gamma \in \mathbb{N}, d \in \mathbb{R}_{>1}$ and $f : \mathbb{N} \to \mathbb{N}$ such that $f(t) \geq t$ and $f(t-1) \leq \frac{f(t)}{d}$ for every $t \in \mathbb{N}$.*

- *Let $\mathbf{A}$ be an algorithm for $\mathcal{Q}$-Deletion(η) Compression that, on input $Q = (D, R_1, \ldots, R_\ell)$, k and $\hat{W}$, runs in time $\mathcal{O}(f(k) \cdot |Q|^\gamma \cdot |\hat{W}|)$, where $\hat{W}$ is a deletion set of Q into $\mathcal{Q}$,*
- *Let $\mathbf{B}$ be an algorithm that, on input $Q = (D, R_1, \ldots, R_\ell) \notin \mathcal{Q}$, runs in time $\mathcal{O}(|Q|)$ and returns a pair of vertices u, v such that every deletion set of Q into $\mathcal{Q}$ which is disjoint from u and v is a u-v separator in D.*

Then, there is an algorithm that, given an instance $(Q = (D, R_1, \ldots, R_\ell), k)$ of $\mathcal{Q}$-Deletion(η) and the algorithms $\mathbf{A}$ and $\mathbf{B}$, runs in time $\mathcal{O}(f(k) \cdot k \cdot |Q|^\gamma)$ and either computes a set X of size at most k such that $Q - X \in \mathcal{Q}$ or correctly concludes that no such set exists.

Before we proceed, we make a few remarks regarding the conditions in the premise of the theorem. Note that we require the running time of Algorithm $\mathbf{A}$ to be of the form $\mathcal{O}(f(k) \cdot |Q|^\gamma \cdot |\hat{W}|)$ in spite of the $\mathcal{Q}$-deletion(η) compression problem being formally parameterized by $|\hat{W}| + k$. At first glance, it may appear that this is a requirement that is much stronger than simply asking for an FPT algorithm for $\mathcal{Q}$-deletion(η) compression. However, we point out that as long as $\mathcal{Q}$ is hereditary, this requirement is in fact no stronger than simply asking for an FPT algorithm for $\mathcal{Q}$-deletion(η) compression. Precisely, if there is an FPT algorithm for $\mathcal{Q}$-deletion(η) compression, that is an algorithm that runs in time $\mathcal{O}(g(k + |\hat{W}|) \cdot |Q|^\delta)$ for some function g and constant δ, then we can obtain an algorithm for $\mathcal{Q}$-deletion(η) compression that runs in time $\mathcal{O}(g(2k+1) \cdot |Q|^\delta \cdot |\hat{W}|)$ by using the folklore trick of running the compression step for the special case of $|\hat{W}| = k+1$, $|\hat{W}|$ times. This clearly suffices. We now illustrate the power of our theorem by applying it to a few well-studied problems.

3.1 Applications

We describe how Theorem 3.1 can be invoked to shave off a factor of n from existing iterative compression based algorithms for DFVS, Directed Feedback Arc Set (DFAS), Directed Subset Feedback Vertex Set and Multicut. Here, DFAS is the *arc* deletion version of DFVS where the objective is to delete at most k arcs from the given digraph to make it acyclic.

1. Application to DFVS. We set $\eta = 1$ and define $\mathcal{Q}$ to be the set of all directed acyclic graphs. That is, $\mathcal{Q} = \{(D, \emptyset) \mid D \text{ is acyclic}\}$. Clearly, $\mathcal{Q}$ is linear-time recognizable, hereditary and rigid. The algorithm $\mathbf{B}$ is defined to be an algorithm that, given as input a digraph D which is not acyclic, simply picks an arc (a, b) which is part of a directed cycle in D and returns u, v where $u = b$ and $v = a$. The algorithm $\mathbf{A}$ can be chosen to be any compression routine for DFVS. In particular, we choose the compression routine of Chen et al. [7] which runs in time $\mathcal{O}(f(k)(n+m) \cdot |W|)$ where $f(k) = 4^k k! k^4$. Invoking Theorem 3.1 for $\mathcal{Q}$-Deletion (1), we obtain our linear-time algorithm for DFVS.

Theorem 3.2. *There is an algorithm for* DFVS *running in time $\mathcal{O}(k! 4^k k^5 \cdot (n+m))$.*

It is easy to see that DFAS can be reduced to DFVS in the following way. For an instance (D, k) of DFAS, subdivide each arc, and make $k+1$ copies of the original vertices to obtain a graph D'. It is straightforward to see that (D, k) is a Yes instance of DFVS if and only if (D', k) is a Yes instance of DFAS. Since $|D'| \leq 2(k+1)|D|$, we also obtain a linear-time FPT algorithm for DFAS.

Corollary 3.1. *There is an algorithm for* DFAS *running in time $\mathcal{O}(k! 4^k k^6 \cdot (n+m))$.*

2. Application to Multicut. In the Multicut problem, the input is an undirected graph G, integer k and pairs of vertices $(s_1, t_1), \ldots, (s_r, t_r)$ and the objective is to check whether there is a set X of at most k vertices such that for every $i \in [r]$, s_i and t_i are in different connected components of $G - X$. The parameterized complexity of this problem was open for a long time until Marx and Razgon [41] and Bousquet, Daligault and Thomassé [5] showed it to be FPT. Marx and Razgon obtained their FPT algorithm via the iterative compression technique. They gave an algorithm for the compression version of Multicut that, on input $D, (s_1, t_1), \ldots, (s_r, t_r), k$ and $\hat{W}$, runs in time $2^{\mathcal{O}(k^3)} \cdot n^{\gamma} \cdot |\hat{W}|$ for some γ. As a result, they were able to obtain an algorithm for Multicut that runs in time $2^{\mathcal{O}(k^3)} \cdot n^{\gamma+1}$. Since the objective of Marx and Razgon in their paper was to show the fixed-parameter tractability of Multicut, they did not try to optimize γ. However, going through the algorithm of Marx and Razgon and making careful (but standard) modifications of the derandomization step in their algorithm using Theorem 5.16 [10] (see also [1]) as well as the more recent linear time FPT algorithms for the Almost 2-SAT problem [43, 31] instead of the algorithm in [35], it is possible to bound the running time of their compression routine by $2^{O(k^3)}mn \log n$ and hence that of their algorithm by $2^{O(k^3)}mn^2 \log n$. We now show by an application of Theorem 3.1 that we can improve this running time by a factor of n.

Set $\eta = 2$ and define $\mathcal{Q}$ to be the set of all pairs (D, S) where D is a bidirectional digraph and S is the relation capturing the pairs to be separated. Formally, $\mathcal{Q} = \{(D, S) \mid D$ is bidirectional, $S \subseteq V(D)^2$, if $S \neq \emptyset$ then $\forall (u, v) \in S$, u and v are in distinct strongly connected components of $D\}$. Clearly, $\mathcal{Q}$ is linear-time recognizable, hereditary and rigid. We define **A** to be the compression routine of Marx and Razgon [41] and **B** to be an algorithm that computes the strongly connected components of D and simply returns a pair $(u, v) \in S$ (if it exists) such that u and v are in the same strongly connected component of D. By invoking Theorem 3.1 for $\mathcal{Q}$-Deletion(2) with these parameters, we obtain the following corollary.

Corollary 3.2. *There is an algorithm for* Multicut *running in time $2^{O(k^3)}mn \log n$.*

3. Application to Directed Subset Feedback Vertex Set.

In the Directed Subset Feedback Vertex Set (DSFVS) problem, the input is a digraph D, a set S of vertices in D and the objective is to check whether D contains a vertex set X of size at most k such that $D - X$ has no cycles passing through S, also called S-cycles. This problem is a clear generalization of DFVS and was shown to be FPT by Chitnis et al. [9] via the iterative compression technique.

They also observed that this problem is equivalent to the Arc Directed Subset Feedback Vertex Set (ADSFVS) where the input is a digraph D and a set S of *arcs* in D and the objective is to check whether D contains a vertex set X of size at most k such that $D - X$ has no cycles passing through S. Chitnis et al. gave an algorithm for the compression version of ADSFVS that, on input D, S, k and $\hat{W}$, runs in time $2^{\mathcal{O}(k^3)} \cdot n^{\gamma} \cdot |\hat{W}|$ for some γ. As a result, they were able to obtain an algorithm for ADSFVS that runs in time $2^{\mathcal{O}(k^3)} \cdot n^{\gamma+1}$. We show by an application of Theorem 3.1 that we can directly shave off a factor of n from this running time.

We first argue that ADSFVS is a special case of $\mathcal{Q}$-Deletion (2). We define by $\mathcal{Q}$ the set of all pairs (D, S) where $S \subseteq A(D)$ and D has no cycle passing through an arc in S. Clearly, $\mathcal{Q}$ is linear-time recognizable, hereditary and rigid. We define **A** to be the compression routine of Chitnis et al. [9] and **B** to be an algorithm that, given as input the pair (D, S), computes the strongly connected components of D and simply returns an arc in S which is contained in a strongly connected component of D. By invoking Theorem 3.1 for $\mathcal{Q}$-Deletion (2) with these parameters, we obtain the following corollary.

Corollary 3.3. *There is an algorithm for* Arc Directed Subset Feedback Vertex Set *running in time $2^{\mathcal{O}(k^3)} \cdot n^{\gamma}$.*

Due to the aforementioned observation of Chitnis et al., we also get an algorithm with the same running time for Directed Subset Feedback Vertex Set. Having described the main applications of our theorem, we now proceed to its proof.

3.2 Proof of Theorem 3.1

The main technical component of the proof of this theorem is a generalization of Lemma 1.1. The proof of this lemma (Lemma 3.1), is fairly technical and requires the introduction of more notation. For readers who are interested in a quick look at the central ideas behind the proof of Lemma 3.1 without having to deal with the technical complications brought about by dealing with structures, we direct them to Section 4 (for the relevant structural lemmas) and Section 5 for a separate proof of Lemma 1.1. We only state Lemma 3.1 here and omit the proof due to space constraints.

Lemma 3.1. *Let $\eta \in \mathbb{N}$ and let $\mathcal{Q}$ be a linear-time recognizable, hereditary and rigid family of η-structures. There is an algorithm that, given an η-structure $Q = (D, R_1, \ldots, R_\ell) \notin \mathcal{Q}$ where D is strongly connected, vertices $u, v \in V(D)$, and $p \in \mathbb{N}$, runs in time $\mathcal{O}(p^2|Q|)$ and either correctly concludes that D has no*

u-v separator of size at most p or returns a set S with at most $2p+2$ vertices such that one of the following holds.

- *$Q - S \in \mathcal{Q}$.*
- *$D - S$ has at least 2 strongly connected components each of which induces a substructure of Q not in $\mathcal{Q}$.*
- *The strongly connected components of $D - S$ can be partitioned into 2 sets inducing substructures of Q, say Q_1 and Q_2 such that $Q_1 \notin \mathcal{Q}$, $Q_2 \in \mathcal{Q}$ and $|Q_1| \leq \frac{1}{2}|Q|$.*
- *If Q has a deletion set of size at most p into $\mathcal{Q}$ then $Q - S$ has a deletion set of size at most $p-1$ into $\mathcal{Q}$.*

We now return to Theorem 3.1 and proceed to prove it assuming this lemma as a black-box. We describe our algorithm for $\mathcal{Q}$-DELETION(η) using the algorithms **A**, **B** and the algorithm of Lemma 3.1 as subroutines. The input to the algorithm in Theorem 3.1 is an instance $(Q = (D, R_1, \ldots, R_\ell), k)$ of $\mathcal{Q}$-DELETION(η) and the output is NO if Q has no deletion set into $\mathcal{Q}$ of size at most k and otherwise, the output is a set X which is a *minimum size* deletion set of Q into $\mathcal{Q}$ of size at most k.

Description of the Algorithm of Theorem 3.1 and Correctness. We now give a formal description of the algorithm. The algorithm is recursive, each call takes as input an η-structure $Q = (D, R_1, \ldots, R_\ell)$ and integer k. In the course of describing the algorithm we will also prove by induction on $k + |Q|$ that the algorithm either correctly concludes that Q has no deletion set into $\mathcal{Q}$ of size at most k, or finds a minimum size deletion set of Q into $\mathcal{Q}$, say X of size at most k. The algorithm proceeds as follows.

In time linear in the size of the digraph D, the algorithm computes the decomposition of D into strongly connected components. Let D' be the digraph obtained from D by removing from D all strongly connected components which induce a substructure of Q that is already in $\mathcal{Q}$. This operation is safe because the class $\mathcal{Q}$ is rigid and hereditary. That is, if $Q' = (D', R'_1, \ldots, R'_\ell)$ is the substructure of Q induced on $V(D')$ then any deletion set of Q into $\mathcal{Q}$ is a deletion set of Q' into $\mathcal{Q}$ and vice versa. So the algorithm proceeds by working on Q' instead. For ease of description, we now revert back to the input η-structure $Q = (D, R_1, \ldots, R_\ell)$ and assume without loss of generality that D does not contain any trivial strongly connected components.

If D is the empty graph or more generally, if $Q \in \mathcal{Q}$, then the algorithm correctly returns the empty set as a minimum size deletion set of Q into $\mathcal{Q}$. From now on we assume that D is non-empty. Since D does not contain any trivial strongly connected components this implies that $m \geq n \geq 2$ and hence $|Q| \geq 2$.

If $k = 0$ the algorithm correctly returns NO, since $Q \notin \mathcal{Q}$. From now on we assume that $k \geq 1$. For $k \geq 1$, we determine from the computed decomposition of D into strongly connected components whether D is strongly connected. If it is not, then let C be the vertex set of an arbitrarily chosen strongly connected component of D. The algorithm calls itself recursively on the instances $(Q[C], k-1)$ and $(Q - C, k-1)$. If either of the recursive calls return NO the algorithm returns NO as well since, both $Q[C]$ and $Q - C$ need to contain at least one vertex from any deletion set of Q into $\mathcal{Q}$. Otherwise the recursive calls return sets X_1 and X_2 such that X_1 is a deletion set of $Q[C]$ into $\mathcal{Q}$, X_2 is a deletion set of $Q - C$ into $\mathcal{Q}$ and both X_1 and X_2 have size at most $k-1$ each. The algorithm executes Algorithm **A** on (Q, k) with $\hat{W} = X_1 \cup X_2$, and returns the same answer as the Algorithm **A**. From now on we assume that D is strongly connected.

For $k \geq 1$ and strongly connected graph D the algorithm proceeds as follows. It starts by running the algorithm **B** on Q to compute in time $\mathcal{O}(|Q|)$ a pair of vertices $u, v \in V(D)$ such that *every* deletion set of Q into $\mathcal{Q}$ which is disjoint from u and v hits all u-v paths in D. Clearly, Q, u, v satisfy the premise of Lemma 3.1. Hence we execute the subroutine described in Lemma 3.1 on Q, u, v with $p = k$. Recall that the execution of this subroutine will have one of two possible outcomes. In the first case, the subroutine returns a set $S \subseteq V(D)$ of size at most $2k + 2 \leq 3k$ satisfying one of the properties in the statement of Lemma 3.1. In the second case, the subroutine concludes that D has no u-v separator of size at most p. But in this case, we infer that Q has no deletion set into $\mathcal{Q}$ of size at most k *disjoint from* $\{u, v\}$ and hence we define S to be the set $\{u, v\}$. Now, observe that this set S trivially satisfies the last property in the statement of Lemma 3.1. Hence, irrespective of the outcome of the subroutine, we will have computed a set S of size at most $3k$ which satisfies one of the four properties in the statement of Lemma 3.1.

Observe that it is straightforward to check in linear time whether S satisfies any of the first 3 properties. Therefore, if none of these properties are satisfied, then we assume that S satisfies the last property. Furthermore, we work with the earliest property that S satisfies. That is, if S satisfies Property i and Property j where $1 \leq i < j \leq 4$ then we execute the steps corresponding to Case i. Subsequent steps of our algorithm will depend on the output of this check on S.

Case 1: $Q - S \in \mathcal{Q}$. In this case, we execute Algorithm **A** on Q, k, with $\hat{W} = S$ to either conclude that Q has no deletion set into $\mathcal{Q}$ of size at most k, in which case we return No, or obtain a minimum size set X which has size at most k and is a deletion set of Q into $\mathcal{Q}$. In this case we return X.

Case 2: *$D - S$ has at least 2 non-trivial strongly connected components each of which induces a substructure of Q not in $\mathcal{Q}$.* Let C be one such non-trivial strongly connected component of $D - S$. We know that any deletion set of Q into $\mathcal{Q}$ must contain at least one vertex in C and at least one vertex in $D - (S \cup C)$. Hence any deletion set of Q into $\mathcal{Q}$ of size at most k must contain at most $k - 1$ vertices in C and at most $k - 1$ vertices in $D - (S \cup C)$. Thus, the algorithm solves recursively the instances $(Q[C], k - 1)$ and $(Q - (C \cup S), k - 1)$. If either of the the recursive calls return No the algorithm returns No as well. Otherwise the recursive calls return vertex sets X_1 and X_2 such that X_1 is a deletion set of $Q[C]$ into $\mathcal{Q}$, X_2 is a deletion set of $Q - (C \cup S)$ into $\mathcal{Q}$, and both X_1 and X_2 have size at most $k - 1$ each. The algorithm then calls the Algorithm **A** on Q, k with $\hat{W} = X_1 \cup X_2 \cup S$, and returns the same answer as the Algorithm **A**.

Case 3: *The strongly connected components of $D - S$ can be partitioned into 2 sets inducing substructures of Q, say Q_1 and Q_2 such that $Q_1 \notin \mathcal{Q}$, $Q_2 \in \mathcal{Q}$ and* $|Q_1| \leq \frac{1}{2}|Q|$. Observe that since S did not fall into the earlier cases, we may assume that S is *not* a deletion set of Q into $\mathcal{Q}$ and $D - S$ has at most 1 non-trivial strongly connected component. Thus $D - S$ has exactly one non-trivial strongly connected component C which induces a structure not in $\mathcal{Q}$, and this component induces a structure of size at most $\frac{1}{2}|Q|$. We recursively invoke the algorithm on input $(Q[C], k)$. If the recursive invocation returned No, then it follows that Q does not have a deletion set into $\mathcal{Q}$ of size at most k, so we can return No as well. On the other hand, if the recursive call returned a set X which is a deletion set of $Q[C]$ into $\mathcal{Q}$ of size at most k then $S \cup X$ is a deletion set of Q into $\mathcal{Q}$ of size at most $4k$. Now, we execute Algorithm **A** on Q, k with $\hat{W} = S \cup X$ and return the same answer as the output of this algorithm

Case 4: *If Q has a deletion set into $\mathcal{Q}$ of size at most k then $Q - S$ has a deletion set into $\mathcal{Q}$ of size at most $k - 1$.* Recall that we arrive at this case only if the other cases do not occur. We recursively invoke the algorithm on the instance $(Q - S, k - 1)$. If the recursion concluded that $Q - S$ does not have a deletion set into $\mathcal{Q}$ of size at most $k - 1$, then we return that Q has no deletion set into $\mathcal{Q}$ of size at most k. Otherwise, suppose that the recursive call returns a set X which is a deletion set of $Q - S$ into $\mathcal{Q}$ of size at most $k - 1$. Now, $S \cup X$ is a deletion set of Q into $\mathcal{Q}$ of size at most $4k$. Hence, we execute Algorithm **A** on Q, k with $\hat{W} = S \cup X$ and return the same answer the output of this algorithm.

Whenever the algorithm makes a recursive call, either the parameter k is reduced to $k - 1$ or the size of the substructure the algorithm is called on is smaller than Q. Thus the correctness of the algorithm and the fact that the algorithm terminates follows from induction on $k + |Q|$.

Running Time analysis. We now analyse the running time of the above algorithm when run on an instance (D, k) in terms of the parameters k, n and m. Before proceeding with the analysis, let us fix some notation. In the remainder of this section, we set

- α to be a constant such that Algorithm **A** on input $Q, k, \hat{W}$ runs in time $\alpha f(k) \cdot |Q|^{\gamma} \cdot |\hat{W}|$,
- β be a constant so that computing the decomposition of D into strongly connected components, removing all trivial strongly connected components, running the algorithm of Lemma 3.1, then determining which of the four cases apply, and then outputting the substructure induced by a strongly connected component of $D - S$ such that this substructure is not in $\mathcal{Q}$, takes time $\beta \cdot k^2 \cdot |Q|$.

Based on α and β we pick a constant μ such that $\mu \geq \max\left\{20\beta, \frac{2\beta d}{d-1}\right\}$ and such that $\mu \geq \max\left\{20\alpha, \frac{10\alpha d}{d-1}\right\}$. Let $T(|Q|, k)$ be the maximum running time of the algorithm on an instance with size $|Q|$ and parameter k. To complete the running time analysis we will prove the following claim.

Claim 1. $T(|Q|, k) \leq \mu \cdot f(k) \cdot k \cdot |Q|^{\gamma}$.

Proof. We prove the claim by induction on $|Q| + k$. We will regularly make use of the facts that $f(k - 1) \leq \frac{f(k)}{d}$ and that $f(k) \geq k$. We consider the execution of the algorithm on an instance $(Q = (D, R_1, \ldots, R_\ell), k)$. We need to prove that the running time of the algorithm is upper bounded by $\mu \cdot f(k) \cdot k \cdot |Q|^{\gamma}$. For the base cases if every strongly connected component in D induces a substructure of Q that is already in $\mathcal{Q}$ or $k = 0$, then the statement of the claim is satisfied by the choice of μ. We now proceed to prove the inductive step. We will assume throughout the argument that $k \geq 1$ and that $Q \notin \mathcal{Q}$.

If D is not strongly connected then the algorithm makes two recursive calls; one to $Q_1 = (Q[C], k - 1)$ and one to $Q_2 = (Q - C, k - 1)$. Observe that $|Q_1| + |Q_2| \leq |Q|$. In this case the total time of the algorithm can be upper bounded by

$$\beta k^2|Q| + T(|Q_1|, k-1) + T(|Q_2|, k-1) + \alpha f(k)|Q|^\gamma \cdot 2k \le \mu \cdot f(k) \cdot k \cdot |Q|^\gamma$$

We will now assume in the rest of the argument that D is strongly connected. For $k \ge 1$ and strongly connected D the algorithm invokes Lemma 3.1. Following the execution of the algorithm of Lemma 3.1, we execute the steps corresponding to exactly *one* of the 4 cases. We show that in each of the four cases, the algorithm runs within the claimed time bound. Let S be the set output by the algorithm of Lemma 3.1. We now proceed with the case analysis.

Case 1: In this case the algorithm terminates after one execution of Algorithm **A** with a set $\hat{W}$ of size at most $3k$. Thus the total running time of the algorithm is upper bounded by $\beta k^2|Q| + \alpha f(k)|Q|^\gamma \cdot 3k$, which is

$$\begin{aligned}&\le \frac{1}{20}\mu \cdot f(k) \cdot k \cdot |Q|^\gamma + \frac{3}{20}\mu \cdot f(k) \cdot k \cdot |Q|^\gamma \\ &\le \mu \cdot f(k) \cdot k \cdot |Q|^\gamma\end{aligned}$$

Case 2: In this case the algorithm makes two recursive calls, one to $(Q[C], k-1)$ and one to $(Q - C, k-1)$. After this, the algorithm executes Algorithm **A** with a set $\hat{W}$ of size at most $5k$ and terminates. Let $Q_1 = Q[C]$ and $Q_2 = Q - C$. In this case the total time of the algorithm is upper bounded as follows.

$$\begin{aligned}&\beta k^2|Q| + T(|Q_1|, k-1) + T(|Q_2|, k-1) \\ &\quad + \alpha f(k)|Q|^\gamma \cdot 5k \\ &\quad \le \frac{d-1}{2d} \cdot \mu \cdot f(k) \cdot k \cdot |Q|^\gamma \\ &\quad + \mu f(k-1) \cdot (k-1) \cdot |Q|^\gamma \\ &\quad + \frac{d-1}{2d} \cdot \mu \cdot f(k) \cdot k \cdot |Q|^\gamma \\ &\quad = \mu \cdot f(k) \cdot k \cdot |Q|^\gamma\end{aligned}$$

Case 3: In this case the algorithm makes a single recursive call on the instance $(Q[C], k)$, where $Q[C]$ has size at most $\frac{1}{2}|Q|$. After the recursive call the algorithm executes Algorithm **A** with a set $\hat{W}$ of size at most $4k$ and terminates. Hence, in this case the total time of the algorithm is upper bounded as follows.

$$\begin{aligned}&\beta k^2(|Q|) + T\left(\frac{1}{2}|Q|, k\right) + \alpha f(k)|Q|^\gamma \cdot 4k \\ &\quad \le \frac{1}{20} \cdot \mu \cdot f(k) \cdot k \cdot |Q|^\gamma \\ &\quad + \frac{1}{2} \cdot \mu f(k) \cdot k \cdot |Q|^\gamma \\ &\quad + \frac{4}{20}\mu \cdot f(k) \cdot k \cdot |Q|^\gamma \\ &\quad \le \mu \cdot f(k) \cdot k \cdot |Q|^\gamma\end{aligned}$$

Case 4: Here the algorithm makes a single recursive call on $(Q - S, k-1)$. Following the recursive call, there is a single call to Algorithm **A** with a set $\hat{W}$ of size at most $4k$. This yields the following bound on the running time in this case.

$$\begin{aligned}&\beta k^2|Q| + T(|Q|, k-1) + \alpha f(k)|Q|^\gamma \cdot 4k \\ &\quad \le \frac{d-1}{2d} \cdot \mu \cdot f(k) \cdot k \cdot |Q|^\gamma \\ &\quad + \mu f(k-1) \cdot (k-1) \cdot |Q|^\gamma \\ &\quad + \frac{d-1}{2d} \cdot \mu \cdot f(k) \cdot k \cdot |Q|^\gamma \\ &\quad \le \mu \cdot f(k) \cdot k \cdot |Q|^\gamma\end{aligned}$$

In each of the four cases the running time of the algorithm, and hence $T(|Q|, k)$ is upper bounded by $\mu \cdot f(k) \cdot k \cdot |Q|^\gamma$. This completes the proof of the claim. □

The algorithm and its correctness proof, together with Claim 1 completes the proof of Theorem 3.1.

4 Setting up common machinery

Before we proceed to the proof of Lemma 1.1 in Section 5, we need to set up some notation and recall known results on separators in digraphs. We use this section to describe the notations and lemmas common to the special case of DFVS as well as the general $\mathcal{Q}$-DELETION(η) problem.

DEFINITION 3. *Let D be a digraph and X and Y be disjoint vertex sets. A vertex set S disjoint from $X \cup Y$ is called an X-Y* separator *if there is no X-Y path in $D - S$. We denote by $R(X, S)$ the set of vertices of $D - S$ reachable from vertices of X via directed paths and by $NR(X, S)$ the set of vertices of $D - S$* not *reachable from vertices of X. We denote by $\lambda_D(X, Y)$ the size of a smallest X-Y separator in D with the subscript ignored if the digraph is clear from the context.*

We remark that it is not necessary that Y and $N^+[X]$ be disjoint in the above definition. If these sets do intersect, then there is no X-Y separator in the digraph and we define $\lambda(X, Y)$ to be ∞.

DEFINITION 4. *Let D be a digraph and X and Y be disjoint vertex sets. Let S_1 and S_2 be X-Y separators. We say that S_2* covers *S_1 if $R(X, S_2) \supseteq R(X, S_1)$.*

Note that for a set $S \subseteq V(D)$ which is an X-Y separator in D for some $X, Y \subseteq V(D)$ the sets $R(X, S)$, $NR(X, S)$ and S form a partition of the vertex set of D.

4.1 Finding useful separators We begin with a lemma which gives a polynomial time procedure to compute, for every pair of vertices s and t in a digraph, a sequence of vertex sets each containing s and excluding t such that every minimum s-t separator is contained in the union of the out-neighborhoods of these sets. Moreover, for each set, the out-neighborhood is in fact a minimum s-t separator. The statement of this lemma is almost identical to the statements of Lemma 2.4 in [39] and Lemma 3.2 in [43]. However, the statement of Lemma 2.4 in [39] deals with undirected graphs while that of Lemma 3.2 in [43] deals with arc-separators instead of vertex separators. Furthermore, the second property in the statement of the following lemma is not part of the latter, although a closer inspection of the proof shows that this property is indeed guaranteed. Note that this proof closely follows that in [39]. We give a full proof here for the sake of completeness.

LEMMA 4.1. *Let s, t be two vertices in a digraph D such that the minimum size of an s-t separator is $\ell > 0$. Then, there is an ordered collection $\mathcal{X} = \{X_1, \ldots, X_q\}$ of vertex sets where $\{s\} \subseteq X_i \subseteq V(D) \setminus (\{t\} \cup N^-(t))$ such that*

1. *$X_1 \subset X_2 \subset \cdots \subset X_q$,*
2. *X_i is reachable from s in $D[X_i]$ and every vertex in $N^+(X_i)$ can reach t in $D - X_i$,*
3. *$|N^+(X_i)| = \ell$ for every $1 \leq i \leq q$ and*
4. *every s-t separator of size ℓ is fully contained in $\bigcup_{i=1}^q N^+(X_i)$.*

Furthermore, there is an algorithm that, given $k \in \mathbb{N}$, runs in time $\mathcal{O}(k(|V(D)| + |A(D)|))$ and either correctly concludes that $\ell > k$ or produces the sets $X_1, X_2 \setminus X_1, \ldots, X_q \setminus X_{q-1}$ corresponding to such a collection $\mathcal{X}$.

Proof. We denote by D' the directed network obtained from D by performing the following operation. Let $v \in V(D) \setminus \{s, t\}$. We remove v and add 2 vertices v^+ and v^-. For every $u \in N^-(v)$, we add an arc (u, v^-) of infinite capacity and for every $u \in N^+(v)$, we add an arc (v^+, u) of infinite capacity and finally we add the arc (v^-, v^+) with capacity 1. We now make an observation relating s-t arc-separators in D' to s-t separators in D. But before we do so, we need to formally define arc-separators.

DEFINITION 5. *Let D be a digraph and s and t be distinct vertices. An arc-set S is called an s-t* arc-separator *if there is no s-t path in $D - S$. We denote by $R(s, S)$ the set of vertices of $D - S$ reachable from s via directed paths and by $NR(s, S)$ the set of vertices of $D - S$* not *reachable from s.*

The following observation is a consequence of the definition of arc-separators and the construction of D'.

OBSERVATION 1. *If $S \subseteq \{(v^-, v^+) | v \in V(D) \setminus \{s, t\}\}$ is an s-t arc-separator in D', then the set $S^{-1} = \{v | (v^-, v^+) \in S\}$ is an s-t separator in D. Conversely for every s-t separator X in D, the set $\{(v^-, v^+) | v \in X\}$ is an s-t arc-separator in D'.*

We now proceed to the proof of the lemma statement. We first run $min\{k + 1, \ell\}$ iterations of the Ford-Fulkerson algorithm [20] on the network D'. Since we do not know ℓ to begin with, we simply try to execute $k+1$ iterations. If we are able to execute $k+1$ iterations, then it must be the case that $\ell > k$ and hence we return that $\ell > k$. Otherwise, we stop after at most $\ell \leq k$ iterations with a maximum s-t flow. Let D_1 be the residual graph. Let $C_1, \ldots, C_q$ be a topological ordering of the strongly connected components of D_1 such that $i < j$ if there is a path from C_i to C_j. Recall that there is a t-s path in D_1. Let C_x and C_y be the strongly connected components of D_1 containing t and s respectively. Since there is a path from t to s in D_1, it must be the case that $x < y$. For each $x < i \leq y$, let $Y_i = \bigcup_{j=i}^q C_j$ (see Figure 1). We first show that $|\delta^+_{D'}(Y_i)| = \ell$ for every $x < i \leq y$. Since no arcs leave Y_i in the graph D_1, no flow enters Y_i and every arc in $\delta^+_{D'}(Y_i)$ is saturated by the maximum flow. Therefore, $|\delta^+_{D'}(Y_i)| = \ell$.

We now show that every arc which is part of a minimum s-t arc-separator is contained in $\bigcup_{i=1}^q \delta^+_{D'}(Y_i)$. Consider a minimum s-t arc-separator S and an arc $(a, b) \in S$. Let Y be the set of vertices reachable from s in $D' - S$. Since F is a minimum s-t arc-separator, it must be the case that $\delta^+_{D'}(Y) = F$ and therefore, $\delta^+_{D'}(Y)$ is saturated by the maximum flow. Therefore, we have that (b, a) is an arc in D_1. Since no flow enters the set Y, there is no cycle in D_1 containing the arc (b, a) and therefore, if the strongly connected component containing b is C_{i_b} and that containing a is C_{i_a}, then

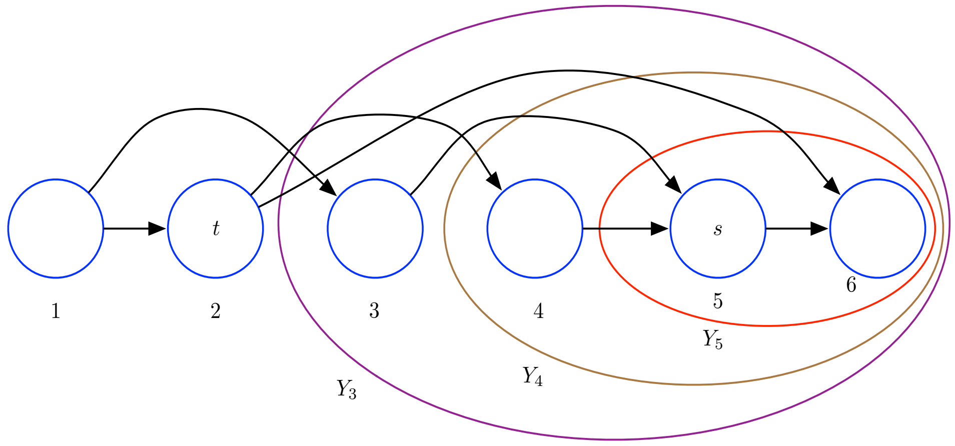

Figure 1: An illustration of the sets in the proof of Lemma 4.1. The chain of circles in the middle are the strongly connected components of D_1 and $\alpha(s) = 5$ and $\alpha(t) = 2$.

$i_b < i_a$. Furthermore, since there is flow from s to a from b to t, it must be the case that $x < i_b < i_a < y$ and hence the arc (a, b) appears in the set $\delta^+_{D'}(Y_{i_a})$.

Finally, we define the set $R(Y_i)$ to be the set of vertices of Y_i which are reachable from s in the graph $D'[Y_i]$. For each set $R(Y_i)$ we define the set $R^{-1}(Y_i)$ as $\{v|\{v^+, v^-\} \subseteq R(Y_i)\}$. Due to the correspondence between s-t separators in D and s-t arc-separators in D' (Observation 1), the sets $R^{-1}(Y_y) \subset R^{-1}(Y_{y-1}) \subset \cdots \subset R^{-1}(Y_{x+1})$ indeed form a collection of the kind described in the statement of the lemma. It remains to describe the computation of these sets.

In order to compute these sets, we first need to run the Ford-Fulkerson algorithm for ℓ iterations and perform a topological sort of the strongly connected components of D_1. This takes time $\mathcal{O}(\ell(|V(D)| + |A(D)|))$. During this procedure, we also assign indices to the strongly connected components in the manner described above, that is, $i < j$ if C_i occurs before C_j in the topological ordering.

In $\mathcal{O}(\ell(|V(D)|+|A(D)|))$ time, we can assign indices to vertices such that the index of a vertex v (denoted by $\alpha(v)$) is the index of the strongly connected component containing v. We then perform a modified (directed) breadth first search (BFS) starting from s by using only *out-going* arcs. The only difference between our BFS and the standard BFS algorithm is that we need to visit vertices in the order dictated by the function α. The details are straightforward and we omit them. □

We also require the following well known property of minimum separators. This is a simple consequence of Property 4 in Lemma 4.1.

Lemma 4.2. *Let D be a digraph and s, t be two vertices. Let $\mathcal{X} = \{X_1, \ldots, X_q\}$ be the collection given by Lemma 4.1 and $\ell = |N^+(X_i)|$ for each $i \in [q]$. Define $X_0 = \emptyset$ and $X_{q+1} = V(D)$. Let Z_i denote the set $X_{i+1} \setminus N^+[X_i]$ for each $0 \le i \le q$. Then, any minimal s-t separator in D that intersects Z_i for any $0 \le i \le q$ has size at least $\ell + 1$.*

Proof. Let $Q = \bigcup_{j=1}^{q} N^+(X_j)$. We claim that for any $0 \le i \le q$, the set Z_i is disjoint from Q. Fix an index i and consider a vertex $u \in Z_i$. By definition, $u \in X_{i+1}$ and $u \notin N^+[X_i]$. Since $u \in X_{i+1}$, it must be the case that $u \in X_r$ and hence *not in* $N^+[X_r]$ for every $r > i$ (by Property 1 in Lemma 4.1). Similarly, since $u \notin N^+[X_i]$, it must be the case that $u \notin N^+[X_r]$ for any $r \le i$. Therefore, $u \notin Q$ and we conclude that Z_i is disjoint from Q.

The lemma now follows from the fact that Z_i is disjoint from Q and Property 4 in Lemma 4.1 which guarantees that every s-t separator of size ℓ is contained in Q. This completes the proof of the lemma. □

We now recall the notion of a tight separator sequence. This was first defined in [36] for undirected graphs. Here we define a similar notion for directed graphs.

Definition 6. *Let s,t be two vertices in a digraph D and let $k \in \mathbb{N}$. A* tight s-t separator sequence *of order k is an ordered collection $\mathcal{H} = \{H_1, \ldots, H_q\}$ of sets in*

$V(D)$ where $\{s\} \subseteq H_i \subseteq V(D) \setminus (\{t\} \cup N^-(t))$ for any $1 \leq i \leq q$ such that,

- *$H_1 \subset H_2 \subset \cdots \subset H_q$,*
- *H_i is reachable from s in $D[H_i]$ and every vertex in $N^+(H_i)$ can reach t in $D - H_i$ (implying that $N^+(H_i)$ is a minimal s-t separator in D)*
- *$|N^+(H_i)| \leq k$ for every $1 \leq i \leq q$,*
- *for any $1 \leq i \leq q-1$, there is no s-t separator S of size at most k where $S \subseteq H_{i+1} \setminus N^+[H_i]$ or $S \cap N^+[H_q] = \emptyset$.*

We have the following obvious but useful consequence of the definition of tight separator sequences.

Lemma 4.3. *Let s,t be two vertices in a digraph D and let $k \in \mathbb{N}$. Let $u \in V(D)$ be a vertex which is part of* every *minimal s-t separator of size at most k. Then, $\mathcal{H}$ is a tight s-t separator sequence of order k in D if and only if it is a tight s-t separator sequence of order $k-1$ in $D - \{u\}$. Furthermore, $u \in N^+(H)$ for every $H \in \mathcal{H}$.*

The following lemma gives a linear-time FPT algorithm to compute a tight separator sequence for a given parameter k. In fact, it is a *polynomial* time algorithm which depends linearly on the input size while the dependence on the parameter is a polynomial. This subroutine plays a major role in the proofs of Lemma 1.1 and Lemma 3.1.

Lemma 4.4. *There is an algorithm that, given a digraph D with no isolated vertices, vertices $s,t \in V(D)$ and $k \in \mathbb{N}$, runs in time $\mathcal{O}(k^2 m)$ and either correctly concludes that there is no s-t separator of size at most k in D or returns the sets $H_1, H_2 \setminus H_1, \ldots, H_q \setminus H_{q-1}$ corresponding to a tight s-t separator sequence $\mathcal{H} = \{H_1, \ldots, H_q\}$ of order k.*

Proof. The algorithm we present executes the algorithm of Lemma 4.1 on various carefully chosen subdigraphs of the given graph and Lemma 4.2 allows us to prove a bound on the number of times any single arc of D participates in these computations.

Suppose that $\lambda(s,t) = \ell < k$ and consider the output of the algorithm of Lemma 4.1 on input D, s and t. By definition, this invocation returns the sets $X_1, X_2 \setminus X_1, X_q \setminus X_{q-1}$ corresponding to the collection $\mathcal{X} = \{X_1, \ldots, X_q\}$. We define X_{q+1} to be the set $R(s, \emptyset) \setminus \{t\}$. We set $X_0 = \emptyset$ and for each $1 \leq i \leq q+1$, we define the following sets (see Figure 2) :

- $Y_i = X_i \setminus X_{i-1}$
- $P_i = Y_i \cap N^+(X_{i-1})$
- $Q_i = N^+(X_i) \setminus N^+(X_{i-1})$
- $W_i = N^+(X_i) \setminus Q_i$

with $P_1 = \{s\}$. That is, P_i is defined to be those vertices in Y_i (which is non-empty due to Property 1 in Lemma 4.1) which are out-neighbors of vertices in X_{i-1}, Q_i is the set of those vertices in the out-neighborhood of X_i which are *not* in the out-neighborhood of X_{i-1} and W_i is the set of vertices in the out-neighborhood of X_i which are not already in Q_i. Observe that Q_i can also be written as $Q_i = (V(D) \setminus X_i) \cap (N^+(Y_i) \setminus N^+(X_{i-1}))$. Also note that P_i and Q_i are by definition disjoint. Furthermore, it is important to note that P_i and Q_i are non-empty. The set P_i is non-empty because Property 1 of Lemma 4.1 guarantees that the set Y_i is non-empty and Property 2 of Lemma 4.1 ensures that every vertex in X_i (and hence in Y_i) is reachable from s in $D[X_i]$ implying that there is at least one vertex in Y_i which has a vertex in X_{i-1} as an in-neighbor. On the other hand, if Q_i is empty then $N^+(X_i) = W_i$ and $N^+(X_{i-1}) \supset W_i$ (strict superset since P_i is non-empty). This contradicts Property 3 of Lemma 4.1. Finally, note that $P_1 = \{s\}$, $Q_{q+1} = \{t\}$, $W_1 = W_{q+1} = \emptyset$ and $P_{q+1} = N^+(X_q)$. For each $1 \leq i \leq q+1$ we define the digraph D_i as follows:

$$V(D_i) = (Y_i \setminus P_i) \cup \{s_i, t_i\} \cup W_i$$

$$A(D_i) = A(D)[Y_i \setminus P_i]$$

$$\bigcup \{(s_i, p) | p \in (N^+(P_i) \cap (Y_i \setminus P_i)) \cup W_i\}$$

$$\bigcup \{(p, t_i) | p \in N^-(Q_i) \cup W_i\}$$

Finally, if $Q_i \cap N^+(P_i) \neq \emptyset$, then we add an arc (s_i, t_i). That is, the digraph D_i is defined as the digraph obtained from $D[Y_i \cup Q_i]$ by adding the vertices in W_i, identifying the vertices of P_i into a single vertex called s_i (removing self-loops and parallel arcs), identifying the vertices of Q_i into a single vertex called t_i and adding arcs from s_i to all vertices in W_i and from all vertices in W_i to t_i. Since P_i and Q_i are disjoint and non-empty, this digraph is well-defined. Also note that there is no isolated vertex in D_i. This is because every vertex in D_i is reachable from s_i by definition. We now make the following claim regarding the connectivity from s_i to t_i in the digraph D_i.

Claim 2. *For each $1 \leq i \leq q+1$, $\lambda_{D_i}(s_i, t_i) > \ell$.*

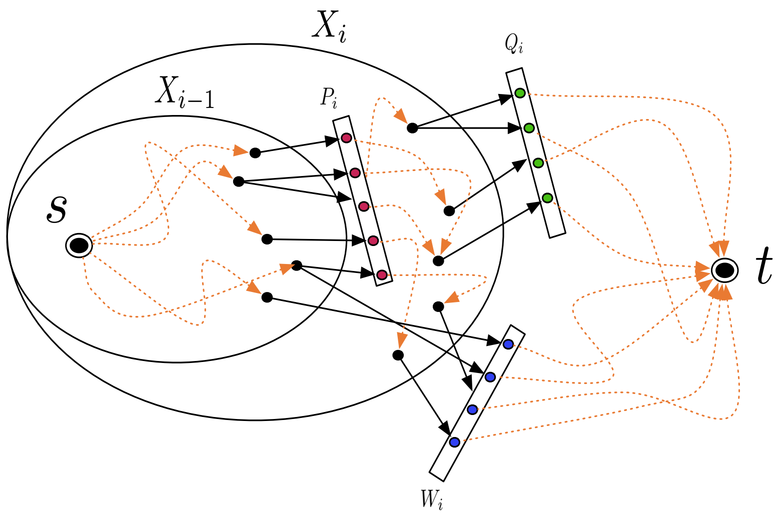

Figure 2: An illustration of the various sets defined in the proof of Lemma 4.4. The dotted arrows denote directed paths while the solid ones denote arcs.

The claim above allows us to recursively apply our algorithm to compute tight separator sequences on each graph D_i while Claim 2 guarantees a bound on the depth of this recursion. The next claim shows that once we recursively compute a tight separator sequence in each of these digraphs, there is a linear time procedure to combine these sequences to obtain a tight separator sequence in the original graph.

Claim 3. *For each $1 \leq i \leq q+1$, let $\mathcal{L}^i$ denote a tight s_i-t_i separator sequence $\{L_1^i, L_2^i, \ldots, L_{r_i}^i\}$ of order k in the digraph D_i. For each $1 \leq i \leq q+1$ and $1 \leq j \leq r_i$, let H_j^i denote the set $(L_j^i \setminus \{s_i\}) \cup P_i$. Then, the ordered collection $\mathcal{H}$ defined as $X_0 \cup H_1^1, \ldots, X_0 \cup H_{r_1}^1, X_1, X_1 \cup H_1^2, \ldots, X_1 \cup H_{r_2}^2, \ldots, X_q, X_q \cup H_1^{q+1}, \ldots, X_q \cup H_{r_{q+1}}^{q+1}$ is a tight s-t separator sequence of order k in D.*

We now use the claims above to complete the proof of the lemma.

Description of the algorithm. We begin by running the algorithm of Lemma 4.1 on the graph D with s and t the same as those in the premise of the lemma. If this subroutine concludes that there is no s-t separator of size at most k in D then we return the same. Otherwise, the subroutine returns the sets $X_1, X_2 \setminus X_1, \ldots, X_q \setminus X_{q-1}$ corresponding to the collection $\mathcal{X} = \{X_1, \ldots, X_q\}$. We define X_{q+1} to be the set $R(s, \emptyset) \setminus \{t\}$.

Having computed the sets $X_1, \ldots, X_{q+1}$, for each $1 \leq i \leq q+1$ we compute the graph D_i, and recursively compute the sets $L_1^i, L_2^i \setminus L_1^i, \ldots, L_{r_i}^i \setminus L_{r_i-1}^i$ corresponding to a tight s_i-t_i separator sequence $\mathcal{L}^i = \{L_1^i, L_2^i, \ldots, L_{r_i}^i\}$ of order k in the graph D_i. At this point, we note a subtle computational simplification we use. In order to compute $\mathcal{L}^i$, for those D^is where $W_i \neq \emptyset$, we can invoke Lemma 4.3 and compute a tight s_i-t_i separator sequence of order $k - |W_i|$ in the graph $D_i - W_i$. As a result, we never actually need to construct the entire graph D_i as defined earlier. Instead it suffices to construct $D_i - W_i$. The reason behind this is that we can now consider the arcs in the graphs $D_1, \ldots, D_{q+1}$ to be a partition of a subset of the arcs in D.

For each $1 \leq i \leq q+1$ and $1 \leq j \leq r_i$, let H_j^i denote the set $(L_j^i \setminus \{s_i\}) \cup P_i$. We output the sets $H_1^1, H_2^1 \setminus H_1^1, \ldots, H_{r_1}^1 \setminus H_{r_1-1}^1$, $X_1 \setminus H_{r_1-1}^1$, $H_1^2, H_2^2 \setminus H_1^2, \ldots, H_{r_2}^2 \setminus H_{r_2-1}^2$, $X_2 \setminus H_{r_2-1}^2, \ldots$ which correspond (by Claim 3) to a tight s-t separator sequence $\mathcal{H} = H_1^1, \ldots, H_{r_1}^1, X_1, X_1 \cup H_1^2, \ldots, X_1 \cup H_{r_2}^2, \ldots, X_q, X_q \cup H_1^{q+1}, \ldots, X_q \cup H_{r_{q+1}}^{q+1}$ or order k. Since the correctness is a direct consequence of Claim 3, we now proceed to the running time analysis.

Running time. We analyse the running time of this algorithm in terms of k, m and $\lambda_D(s,t)$. We let $T(k, \lambda, m)$ denote the running time of the algorithm when $\lambda = \lambda_D(s,t)$. If $\lambda > k$, then $T(k, \lambda, m) = \mathcal{O}(km)$. This is because in this case, we only require a single execution of the algorithm of Lemma 4.1 to conclude that $k < \lambda$. Otherwise, the description of the algorithm clearly implies the following recurrence.

$$T(k, \lambda, m) = \mathcal{O}(\lambda m) + \sum_{i=1}^{q+1} T(k, \lambda_i, m_i)$$

where $\lambda_i = \lambda_{D_i}(s_i, t_i)$ and m_i denotes the number of arcs in D_i. Note that $m \geq \sum_{i=1}^{q+1} m_i$. The $\mathcal{O}(\lambda m)$ term includes the time required to execute the algorithm of Lemma 4.1 as well as the time required to compute the graphs $D_1, \ldots, D_{q+1}$. Now, due to Claim 2, we have that $\lambda_i > \lambda$ for each $i \in [q+1]$. Unrolling the recurrence with $\lambda > k$ being the base case, the claimed running time follows. This completes the proof of the lemma. □

5 Proving Lemma 1.1

Definition 7. *Let D be a strongly connected digraph and let $u, v \in V(D)$. Let $S \subseteq V(D)$ be a u-v separator in D. Then, we say that S is*

- *an* l-good *u-v separator if $D[R(u, S)]$ is acyclic but $D[NR(u, S)]$ contains a cycle.*
- *an* r-good *u-v separator if $D[R(u, S)]$ contains a cycle but $D[NR(u, S)]$ is acyclic.*
- *a* dual-good *u-v separator if both $D[R(u, S)]$ and $D[NR(u, S)]$ contain cycles.*
- *a* completely-good *u-v separator if $D[R(u, S)]$ and $D[NR(u, S)]$ are both acyclic.*
- *an* l-light *u-v separator if $|A[R(u, S)]| \leq \frac{1}{2}|A(D)|$.*
- *an* r-light *u-v separator if $|A[NR(u, S)]| \leq \frac{1}{2}|A(D)|$.*

See Figure 3 for an illustration of separators of various types. The next lemma shows that a pair of separators in D with one covering the other have a certain monotonic dependency between them regarding their (l/r)-goodness and (l/r)-lightness.

Lemma 5.1. (**Monotonicity Lemma** (DFVS)) *Let D be a strongly connected digraph. Let $u, v \in V(D)$ and let S_1 and S_2 be a pair of u-v separators in D such that S_2 covers S_1. Furthermore, suppose that neither S_1 nor S_2 is* dual-*good or* completely-*good. Then the following statements hold.*

- *If S_1 is* r-*good then S_2 is also* r-*good.*
- *If S_2 is* l-*good then S_1 is also* l-*good.*
- *If S_1 is* r-*light then S_2 is also* r-*light.*
- *If S_2 is* l-*light then S_1 is also* l-*light.*

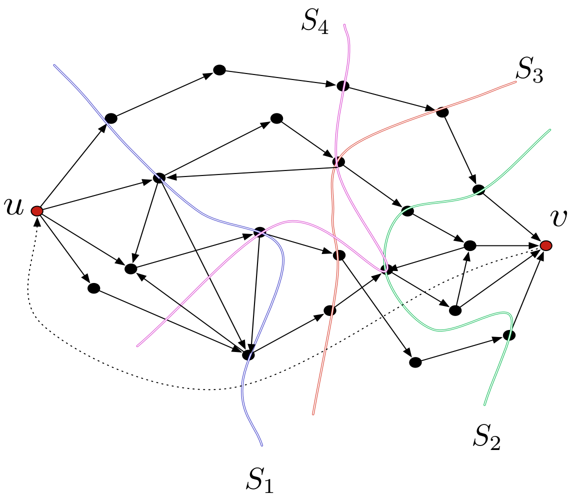

Figure 3: An illustration of the various u-v separator types. Here, S_1 is l-good, S_2 is r-good, S_3 is dual-good and S_4 is completely-good.

Proof. We begin by proving the first statement of the lemma. Suppose to the contrary that S_1 is r-good and S_2 is l-good. By definition, the graph $D_1 = D[R(u, S_1)]$ is not acyclic and $D_2 = D[R(u, S_2)]$ is acyclic. However, since S_2 covers S_1, we know that $R(u, S_2) \supseteq R(u, S_1)$. This implies that D_1 is a subgraph of D_2. However, since D_1 has a cycle, D_2 cannot be acyclic, a contradiction. This completes the proof of the first statement. The proofs of the remaining statements are all analogous. □

We now prove the following lemma which provides a linear time-testable sufficient condition for a separator to reduce the size of the solution upon deletion.

Lemma 5.2. *Let D be a strongly connected digraph. Let $u, v \in V(D)$, $k \in \mathbb{N}$ and suppose that every dfvs of D of size at most k hits all u-v paths in D. Let Z be an* r-*good* (l-*good*) *u-v separator of size at most k such that there is no u-v separator of size at most k contained entirely in the set $R(u, Z)$ (respectively $NR(u, Z)$). If D has a dfvs of size at most k disjoint from $\{u, v\}$ then $D - Z$ has a dfvs of size at most $k - 1$.*

Proof. Let X be a dfvs of D. Consider the case when Z is an r-good separator. The argument for the other case is analogous. Since Z is r-good, we know that the subgraph $D[NR(u, Z)]$ is acyclic. Therefore, any non-trivial strongly connected component in the digraph $D - Z$ lies in the set $R(u, Z)$. Also, the set $X' = X \cap R(u, Z)$ is by definition a dfvs of $D[R(u, Z)]$. Since every non-trivial strongly connected component of $D - Z$ lies in the digraph $D[R(u, Z)]$, it follows that X' is in fact a dfvs for $D - Z$. We now claim that $X' \subset X$.

Suppose to the contrary that $X' = X$. By the premise of the lemma, we have that X is a u-v separator of size at most k. Since $X' = X$, we conclude that X is a u-v separator of size at most k which is contained in the set $R(u, Z)$, a contradiction to the premise of the lemma, implying that $X' \subset X$. This completes the proof of the lemma. □

Having set up the definitions and certain properties of the separators we are interested in, we now describe our linear time subroutines that perform certain computations on separator sequences that will then be used in the linear time implementation of our algorithm.

In this lemma, we argue that given the output of Lemma 4.4, one can, in linear time find a pair of consecutive separators in the sequence where the first is l-light and the second one is not. The output of this lemma will form an 'extremal' point of interest in the algorithm of Lemma 1.1.

Lemma 5.3. *Let D be a strongly connected graph. Let $u, v \in V(D)$, $k \in \mathbb{N}$. Let $\mathcal{H} = \{H_1, \dots, H_q\}$ be a tight u-v separator sequence of order k in D with the algorithm of Lemma 4.4 returning the sets $H_1, H_2 \setminus H_1, \dots, H_q \setminus H_{q-1}$. There is an algorithm that, given D, u, v, k and these sets, runs in time $\mathcal{O}(km)$ and computes the least i for which the separator $N^+(H_i)$ is l-light and the separator $N^+(H_i)$ is not l-light (and consequently is r-light) or correctly concludes that there is no such $1 \leq i \leq q$.*

The next lemma provides a linear time subroutine that checks whether the subgraph induced by the set $H_2 \setminus N^+[H_1]$ is acyclic, for a pair H_1, H_2 of consecutive sets in the tight separator sequence computed by the algorithm of Lemma 4.4.

Lemma 5.4. *Let D be a strongly connected digraph. Let $u, v \in V(D)$ and $k \in \mathbb{N}$. Let $\mathcal{H} = \{H_1, \dots, H_q\}$ be a tight u-v separator sequence of order k in D with the algorithm of Lemma 4.4 returning the sets $H_1, H_2 \setminus H_1, \dots, H_q \setminus H_{q-1}$. There is an algorithm that, given D, u, v, k and these sets, runs in time $\mathcal{O}(km)$ and computes the least i for which the subgraph $D[H_{i+1} \setminus N^+[H_i]]$ is not acyclic or correctly concludes that there is no such $1 \leq i \leq q-1$.*

Proof. The proof of this lemma is similar to that of the previous lemma. Given the sets $H_1, H_2 \setminus H_1, \dots, H_q \setminus H_{q-1}$ we label the vertices of $V(D)$ in the following way with elements from $\{1, \dots, q\}$. We set $H_0 = \{u\}$, $H_{q+1} = V(D)$ and for each $i \in \{0, \dots, q\}$, we label the vertices of $H_{i+1} \setminus H_i$ with the label $i + 1$. For each $0 \leq i \leq q$, we do a directed bfs/dfs on the set of vertices which are labeled i but not marked as being part of the set $N^+(H_j)$ for some $j < i$. Since each arc is examined $\mathcal{O}(k)$ times, the time bound follows. □

Having set up all the required definitions as well as the subroutines tailored to Lemma 1.1, we now proceed to its proof.

Lemma 1.1. *Let D be a strongly connected digraph and $p \in \mathbb{N}$. There is an algorithm that, given D and p, runs in time $\mathcal{O}(p^2 m)$ (where m is the number of arcs in D) and either correctly concludes that D has no dfvs of size at most p or returns a set S with at most $2p + 2$ vertices such that one of the following holds.*

- *S is a dfvs for D.*
- *$D - S$ has at least 2 non-trivial strongly connected components (strongly connected components with at least 2 vertices).*
- *The number of arcs of D whose head and tail occur in the same non-trivial strongly connected component of $D - S$ (arcs participating in a cycle of $D - S$) is at most $\frac{m}{2}$.*
- *If D has a dfvs of size at most p then $D - S$ has a dfvs of size at most $p - 1$.*

Proof. We execute the algorithm of Lemma 4.4 to either conclude that there is no u-v separator of size at most p or compute a tight u-v separator sequence of order p. If this algorithm concludes that there is no u-v separator of size at most p in D, then we return the same. Hence, we may assume that the subroutine returns sets $H_1, H_2 \setminus H_1, \dots, H_q \setminus H_{q-1}$ corresponding to a tight u-v separator sequence $\mathcal{H} = \{H_1, \dots, H_q\}$ of order p.

We let Z_i denote the set $N^+(H_i)$ for each $1 \leq i \leq q$ and focus our attention on the sets Z_1 and Z_q (which are not necessarily distinct). We begin by examining the set Z_1. If Z_1 is dual-good then setting $S = Z_1$ satisfies Property 2. This is because we started with a strongly connected digraph and by the definition of dual-goodness both subgraphs $D[R(u, Z_1)]$ and $D[NR(u, Z_1)]$ contain cycles and hence $D - Z_1$ has at least 2 non-trivial strongly connected components. Similarly, if Z_1 is completely-good, then setting $S = Z_1$ satisfies Property 1. Now, suppose that Z_1 is r-good. It follows from Definition 6 that there is no u-v separator of size at most p contained entirely in the set $R(u, Z_1)$. Then, by Lemma 5.2, if D has dfvs of size at most p disjoint from $\{u, v\}$, then $D - Z_1$ has a dfvs of size at most $p - 1$ and hence we set $S = Z_1 \cup \{u, v\}$ and we satisfy Property 4. Therefore, going forward, we assume that Z_1 is l-good. That is, the subgraph $D[H_1]$ is acyclic. Note that given Z_1, this check can be performed in time $\mathcal{O}(m)$.

We have a symmetric argument for Z_q. That is, if Z_q is dual-good or completely-good then setting $S = Z_q$ satisfies Properties 2 or 1 respectively. Otherwise, if Z_q is l-good, then by Definition 6 we know that there is no u-v separator of size at most p contained entirely in the set $NR(u, Z_q)$ and by Lemma 5.2, if D has a dfvs of size at most p disjoint from $\{u, v\}$ then $D - Z_q$ has a dfvs of size at most $p - 1$ and hence we set $S = Z_q \cup \{v, u\}$ and we are done. Therefore, from this point on, we assume that Z_q is r-good. That is, the subgraph induced on $NR(u, Z_q)$ is acyclic. Again checking which one of these cases hold can be done in time $\mathcal{O}(m)$.

We now examine each of the sets $H_1, H_2 \setminus H_1, \ldots, H_q \setminus H_{q-1}$ and check if for any i, the digraph $D[H_{i+1} \setminus N^+[H_i]]$ has a cycle. This procedure can be performed in time $\mathcal{O}(km)$ due to Lemma 5.4. We now have 2 cases.

In the first case, suppose that the subroutine returned an index $1 \le i \le q-1$ such that $D[H_{i+1} \setminus N^+[H_i]]$ has a cycle. We now examine the sets Z_i and Z_{i+1}. By definition, it cannot be the case that Z_i is r-good or Z_{i+1} is l-good. Also, if either Z_i or Z_{i+1} is dual-good or completely-good (which can be checked in linear time) then we are done in a manner similar to that discussed earlier by setting $S = Z_i$ or $S = Z_{i+1}$. Hence, we may assume that Z_i is l-good and Z_{i+1} is r-good. Now, let $S = Z_i \cup Z_{i+1} \cup \{u, v\}$. Clearly, $|S| \le 2p + 2$. It remains to prove that S satisfies one of the properties in the statement of the lemma. Precisely, we will prove that if D has a dfvs of size at most p then $D - S$ has a dfvs of size at most $p - 1$, that is, S satisfies Property 4.

Let X be a dfvs for D of size at most p. If $u \in X$ or $v \in X$, then we are already done. Therefore, assume that $u, v \notin X$. We claim that $X' = X \cap (H_{i+1} \setminus N^+[H_i])$ is in fact a dfvs for $D - S$. This is because, any non-trivial strongly connected component in $D - S$ must be contained entirely within $R(u, Z_i)$ or $H_{i+1} \setminus N^+[H_i]$ or $NR(u, Z_{i+1})$. Since Z_i is l-good and Z_{i+1} is r-good, the subgraphs induced by the first and third sets are acylic. Therefore, any non-trivial strongly connected component in $D - S$ lies entirely in the set $H_{i+1} \setminus N^+[H_i]$. Since X is a dfvs for D, it follows that X' is a dfvs for $D - S$. We now claim that $X' \subset X$ and hence has size at most $|X| - 1$. Suppose that this is not the case and $X' = X$. By the premise of the lemma, we know that X is a u-v separator and hence we obtain a contradiction to our assumption that $\mathcal{H}$ is a tight-separator sequence (violates condition 4 in Definition 6). This is because X itself will be a u-v separator of size at most p which is contained in the set $H_{i+1} \setminus N^+[H_i]$. This completes the argument for the case when the subroutine returns an $1 \le i \le q-1$ for which the graph $D[H_{i+1} \setminus N^+[H_i]]$ contains a cycle. Henceforth, we will assume that for every $1 \le i \le q-1$, the subgraph $D[H_{i+1} \setminus N^+[H_i]]$, denoted by $\hat{D}_i$ is acyclic.

We now revisit the separators Z_1 and Z_q. Recall that Z_1 is l-good and Z_q is r-good. Now, suppose that Z_1 is r-light. That is, the number of arcs in the subgraph of D induced by the set $V(D) \setminus H_1$ is at most $\frac{1}{2}m$. Then, we set $S = Z_1$. Observe that since $D[H_1]$ is acyclic, every non-trivial strongly connected component of $D - S$ *must* lie in the set $V(D) \setminus H_1$ and hence setting $D_1 = D[V(D) \setminus H_1]$ and $D_2 = D[H_1]$ satisfies Property 3. A symmetric argument holds if Z_q is l-light. Therefore, we conclude that Z_1 is not r-light and Z_2 is not l-light. Therefore, Z_1 is l-light and Z_2 is r-light.

Due to the monotonicity lemma (Lemma 5.1), we know that there is an $i \ge 1$ such that Z_i is l-light, Z_{i+1} is not l-light (and so is r-light), and for all $j \le i$, Z_j is l-light and for all $j > i$, Z_j is not l-light. We examine the sets in $\mathcal{H}$ and find this index i. That is, Z_i is l-light and Z_{i+1} is r-light. This can be done in linear time due to Lemma 5.3.

If either of Z_i or Z_{i+1} is dual-good or completely-good then we are done as argued earlier. So, we assume that each of Z_i and Z_{i+1} is either l-good or r-good.

If Z_{i+1} is l-good then setting $S = Z_{i+1} \cup \{v\}$ satisfies Property 3. Similarly, if Z_i is r-good then setting $S = Z_i \cup \{v\}$ satisfies Property 3. It remains to handle the case when Z_i is l-good and Z_{i+1} is r-good. However, in this case, we claim that $Z_i \cup Z_{i+1}$ is in fact a dfvs of D. Observe that any non-trivial strongly connected component of $D - (Z_i \cup Z_{i+1})$ lies entirely in one of the sets H_i or $H_{i+1} \setminus N^+[H_i]$ or $V(D) \setminus N^+[H_{i+1}]$. The first and third sets induce acyclic subgraphs because Z_i is l-good and Z_{i+1} is r-good. The second set induces an acyclic digraph because we have already argued that for every $1 \le j \le q-1$, the graph $D[H_{j+1} \setminus N^+[H_j]]$ is acyclic.

Therefore, we conclude that $Z_i \cup Z_{i+1}$ is a dfvs of D and setting $S = Z_i \cup Z_{i+1} \cup \{u, v\}$ satisfies Property 1. This completes the proof of the lemma. □

6 Conclusions

We have presented the first linear-time FPT algorithm for the classic Directed Feedback Vertex Set problem. For this, we introduced a new separator based 'recursive compression' approach that either reduces the parameter or reduces the size of the instance by a constant fraction and showed that our approach can be extended to the directed version of the Subset

Feedback Vertex Set (Subset FVS) problem as well as to the Multicut problem.

One of the central features of our technique is that *any* linear-time FPT algorithm for the *compression* version of these problems can be converted to a linear time FPT algorithm for the general problem as well. In other words, any further improvements in the running time of the compression routine for these problems can be directly lifted to the general problem.

An interesting problem for future work in this direction is determining whether there is an algorithm that, for every fixed k, runs in linear time and decides whether a given graph is k vertices away from a Chordal graph.

Acknowledgements. The authors thank Dániel Marx for helpful discussions about the polynomial factor of the algorithm for Multicut in [41].

References

[1] N. Alon, R. Yuster, and U. Zwick, *Color-coding*, J. ACM, 42 (1995), pp. 844–856. 6

[2] H. L. Bodlaender, *A linear time algorithm for finding tree-decompositions of small treewidth*, in Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA, 1993, pp. 226–234. 1, 2

[3] ———, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317. 1, 2, 3

[4] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk, *An $O(c^k n)$ 5-approximation algorithm for treewidth*, in FOCS, 2013, pp. 499–508. 2, 3

[5] N. Bousquet, J. Daligault, and S. Thomassé, *Multicut is fpt*, in STOC, 2011, pp. 459–468. 3, 6

[6] Y. Cao, *Linear recognition of almost interval graphs*, in Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016, 2016, pp. 1096–1115. 2

[7] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon, *A fixed-parameter algorithm for the directed feedback vertex set problem*, J. ACM, 55 (2008). 2, 3, 5

[8] R. H. Chitnis, M. Cygan, M. T. Hajiaghayi, and D. Marx, *Directed subset feedback vertex set is fixed-parameter tractable*, in ICALP (1), vol. 7391 of Lecture Notes in Computer Science, 2012, pp. 230–241. 3

[9] R. H. Chitnis, M. Cygan, M. T. Hajiaghayi, and D. Marx, *Directed subset feedback vertex set is fixed-parameter tractable*, ACM Transactions on Algorithms, 11 (2015), p. 28. 3, 6

[10] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*, Springer, 2015. 2, 4, 6

[11] F. K. H. A. Dehne, M. R. Fellows, M. A. Langston, F. A. Rosamond, and K. Stevens, *An $\mathcal{O}(2^{\mathcal{O}(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem*, Theory Comput. Syst., 41 (2007), pp. 479–492. 2

[12] F. Dorn, *Planar subgraph isomorphism revisited*, in STACS, 2010, pp. 263–274. 2

[13] R. G. Downey and M. R. Fellows, *Fixed-parameter intractability*, in Proceedings of the Seventh Annual Structure in Complexity Theory Conference, Boston, Massachusetts, USA, June 22-25, 1992, 1992, pp. 36–49. 2

[14] ———, *Fixed-parameter tractability and completeness I: basic results*, SIAM J. Comput., 24 (1995), pp. 873–921. 2

[15] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer-Verlag, New York, 1999. 4

[16] M. R. Fellows and M. A. Langston, *Nonconstructive tools for proving polynomial-time decidability*, J. ACM, 35 (1988), pp. 727–739. 2

[17] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Texts in Theoretical Computer Science. An EATCS Series, Springer-Verlag, Berlin, 2006. 4

[18] F. V. Fomin, D. Lokshtanov, N. Misra, M. S. Ramanujan, and S. Saurabh, *Solving* d-*sat via backdoors to small treewidth*, in Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015, 2015, pp. 630–641. 2

[19] F. V. Fomin, D. Lokshtanov, N. Misra, and S. Saurabh, *Planar f-deletion: Approximation, kernelization and optimal FPT algorithms*, in 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012, 2012, pp. 470–479. 2

[20] L. R. Ford, Jr. and D. R. Fulkerson, *Maximal flow through a network*, 8 (1956), pp. 399–404. 10

[21] M. R. Garey and R. E. Tarjan, *A linear-time algorithm for finding all feedback vertices*, Inf. Process. Lett., 7 (1978), pp. 274–276. 2

[22] M. Grohe, *Computing crossing numbers in quadratic time*, in Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece, 2001, pp. 231–236. 2

[23] ———, *Computing crossing numbers in quadratic time*, J. Comput. Syst. Sci., 68 (2004), pp. 285–302. 2

[24] M. Grohe, K. ichi Kawarabayashi, and B. A. Reed, *A simple algorithm for the graph minor decomposition - logic meets structural graph theory*, in SODA, 2013, pp. 414–431. 2

[25] K. ichi Kawarabayashi, *Planarity allowing few error vertices in linear time*, in FOCS, 2009, pp. 639–648. 2, 3

[26] K. ichi Kawarabayashi, Y. Kobayashi, and B. A. Reed, *The disjoint paths problem in quadratic time*, J. Comb. Theory, Ser. B, 102 (2012), pp. 424–435. 2

[27] K. ichi Kawarabayashi and B. Mohar, *Graph and map isomorphism and all polyhedral embeddings in linear time*, in STOC, 2008, pp. 471–480. 2

[28] K. ichi Kawarabayashi and B. A. Reed, *Computing crossing number in linear time*, in STOC, 2007, pp. 382–390. 2

[29] ———, *A nearly linear time algorithm for the half integral parity disjoint paths packing problem*, in SODA, 2009, pp. 1183–1192. 2

[30] ———, *An (almost) linear time algorithm for odd cycles transversal*, in SODA, 2010, pp. 365–378. 2

[31] Y. Iwata, K. Oka, and Y. Yoshida, *Linear-time FPT algorithms via network flow*, in SODA, 2014, pp. 1749–1761. 2, 3, 6

[32] Y. Iwata, M. Wahlström, and Y. Yoshida, *Half-integrality, lp-branching, and FPT algorithms*, SIAM J. Comput., 45 (2016), pp. 1377–1411. 3

[33] B. M. P. Jansen, D. Lokshtanov, and S. Saurabh, *A near-optimal planarization algorithm*, in SODA, 2014, pp. 1802–1811. 2, 3

[34] K. Kawarabayashi, B. Mohar, and B. A. Reed, *A simpler linear time algorithm for embedding graphs into an arbitrary surface and the genus of graphs of bounded tree-width*, in 49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA, 2008, pp. 771–780. 2

[35] D. Lokshtanov, N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh, *Faster parameterized algorithms using linear programming*, ACM Trans. Algorithms, 11 (2014), pp. 15:1–15:31. 6

[36] D. Lokshtanov and M. S. Ramanujan, *Parameterized tractability of multiway cut with parity constraints*, in Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I, 2012, pp. 750–761. 3, 11

[37] D. Lokshtanov, M. S. Ramanujan, and S. Saurabh, *Linear time parameterized algorithms for subset feedback vertex set*, in Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I, 2015, pp. 935–946. 2

[38] ———, *A linear-time parameterized algorithm for node unique label cover*, in 25th Annual European Symposium on Algorithms (ESA 2017), vol. 87 of Leibniz International Proceedings in Informatics (LIPIcs), 2017, pp. 57:1–57:15. 3

[39] D. Marx, B. O'Sullivan, and I. Razgon, *Finding small separators in linear time via treewidth reduction*, ACM Transactions on Algorithms, 9 (2013), p. 30. 3, 10

[40] D. Marx and I. Razgon, *Fixed-parameter tractability of multicut parameterized by the size of the cutset*, in STOC, 2011, pp. 469–478. 3

[41] ———, *Fixed-parameter tractability of multicut parameterized by the size of the cutset*, SIAM J. Comput., 43 (2014), pp. 355–388. 3, 6, 17

[42] R. Niedermeier, *Invitation to Fixed-Parameter Algorithms*, vol. 31 of Oxford Lecture Series in Mathematics and its Applications, Oxford University Press, Oxford, 2006. 4

[43] M. S. Ramanujan and S. Saurabh, *Linear time parameterized algorithms via skew-symmetric multicuts*, in SODA, 2014, pp. 1739–1748. 2, 3, 6, 10

[44] B. A. Reed, *Finding approximate separators and computing tree width quickly*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada, 1992, pp. 221–228. 3

[45] B. A. Reed, K. Smith, and A. Vetta, *Finding odd cycle transversals*, Oper. Res. Lett., 32 (2004), pp. 299–301. 3

[46] N. Robertson and P. D. Seymour, *Graph minors .xiii. the disjoint paths problem*, J. Comb. Theory, Ser. B, 63 (1995), pp. 65–110. 1

[47] ———, *Graph minors. XVIII. tree-decompositions and well-quasi-ordering*, J. Comb. Theory, Ser. B, 89 (2003), pp. 77–108. 1

[48] ———, *Graph minors. XX. wagner's conjecture*, J. Comb. Theory, Ser. B, 92 (2004), pp. 325–357. 1