On the Parameterized Complexity of Deletion to $\mathcal{H}$-free Strong Components

Rian Neogi[1], M. S. Ramanujan[2], Saket Saurabh[1,3], and Roohani Sharma[1]

[1]Institute of Mathematical Sciences, HBNI, India,
{rianneogi,saket,roohani}@imsc.res.in
[2]University of Warwick, UK,
r.maadapuzhi-sridharan@warwick.ac.uk
[3]University of Bergen, Norway

Abstract

DIRECTED FEEDBACK VERTEX SET (DFVS) is a fundamental computational problem that has received extensive attention in parameterized complexity. In this paper, we initiate the study of a wide generalization, the $\mathcal{H}$-FREE SCC DELETION problem. Here, one is given a digraph D, an integer k and the objective is to decide whether there is a vertex set of size at most k whose deletion leaves a digraph where every strong component excludes graphs in the fixed finite family $\mathcal{H}$ as (not necessarily induced) subgraphs. When $\mathcal{H}$ comprises only the digraph with a single arc, then this problem is precisely DFVS.

Our main result is a proof that this problem is fixed-parameter tractable parameterized by the size of the deletion set if $\mathcal{H}$ only contains rooted graphs or if $\mathcal{H}$ contains at least one directed path. Along with generalizing the fixed-parameter tractability result for DFVS, our result also generalizes the recent results of Göke et al. [CIAC 2019] for the 1-OUT-REGULAR VERTEX DELETION and BOUNDED SIZE STRONG COMPONENT VERTEX DELETION problems. Moreover, we design algorithms for the two above mentioned problems, whose running times are better and match with the best bounds for DFVS, without using the heavy machinery of shadow removal as is done by Göke et al. [CIAC 2019].

1 Introduction

In the Directed Feedback Vertex Set (DFVS) problem, the input is a digraph D and an integer k and the objective is to decide whether there is a set $X \subseteq V(D)$ of size at most k such that $D - X$ is acyclic. DFVS is a fundamental computational problem that has received extensive attention in various subdomains of algorithmics. The parameterized complexity of this problem was a long standing open problem in the area until Chen et al. [2] gave a fixed-parameter tractable (FPT) algorithm with running time $O(k!4^k k^4 nm)$. Here, n and m denote the number of vertices and arcs in the digraph respectively. Although subsequent work [10] has improved the dependence on the input size to linear, it remains an open problem whether the $2^{O(k \log k)}$ dependence on k is asymptotically the best possible.

The result of Chen et al. and the techniques used therein also helped kick off a line of research in parameterized complexity where the goal is to understand how far the fixed-parameter tractability of DFVS can be extended to various generalizations of DFVS. Chitnis et al. [3] obtained an FPT algorithm for the Subset DFVS problem, where the goal is to delete at most k vertices that intersect all directed cycles passing through a specified subset of vertices. A general and abstract formulation of the powerful directed *shadow removal* technique first designed by Chitnis et al. [4], was developed in this work and it has found several applications in subsequent work [8, 1, 11, 6]. Lokshtanov et al. [11] studied the Directed Odd Cycle Transversal problem where the objective is to delete at most k vertices that intersect all directed *odd* cycles in the given digraph. They proved that this problem is W[1]-hard and so is unlikely to admit an FPT algorithm. Moreover, they used the shadow removal technique to obtain a fixed-parameter 2-approximation algorithm for this problem. More recently, Göke et al. [6] studied the problems of deleting at most k vertices to (i) obtain a digraph where every strong component is of bounded size and (ii) obtain a digraph where every strong component induces a graph where every vertex has out-degree exactly 1, i.e. is a 1-out-regular digraph.

In this paper, we extend this line of research by initiating the study of a wide generalization of the problems studied by Göke et al., which we call the $\mathcal{H}$-Strong Connected Component Deletion ($\mathcal{H}$-SCC Deletion) problem and define below. Here, $\mathcal{H}$ is a fixed finite family of digraphs.

$\mathcal{H}$-SCC Deletion ──────────────────────────────────

Input: A digraph D, an integer k.
Parameter: k
Problem: Does there exist a set S of at most k vertices such that no strong component
             of D-S contains a graph in $\mathcal{H}$ as a subgraph?

In all our results, n denote the number of vertices in the input graph and $h = \max_{H \in \mathcal{H}} |V(H)|$. Rooted $\mathcal{H}$-SCC Deletion (r-$\mathcal{H}$-SCC Deletion) denotes the special case of $\mathcal{H}$-SCC Deletion where every graph in $\mathcal{H}$ contains an arborescence. An arborescence is a rooted directed tree where every vertex except the root has in-degree exactly 1 and the root has in-degree 0. Notice that r-$\mathcal{H}$-SCC Deletion already generalizes several problems described above including DFVS ($\mathcal{H}$ comprises the graph with a single arc), obtaining strong components of size at most s ($\mathcal{H}$ comprises all arborescences of size $s + 1$) and obtaining strong components with out-degree at most 1 ($\mathcal{H}$ comprises the star with two leaves and both arcs oriented away from the centre). Our main result gives a unified proof of the fixed-parameter tractability of these problems.

Theorem 1. r-$\mathcal{H}$-SCC Deletion *can be solved in time* $2^{O(k^3 \log k)} \cdot n^{O(h)}$.

Theorem 1 also holds for $\mathcal{H}$-SCC Deletion in the case where, for every graph in $\mathcal{H}$ there is

a vertex that is reachable from every other vertex. One can infer this by simply reversing both the input graph and the forbidden graphs and applying the main theorem. We also remark that in general, the $n^{O(h)}$ dependence in the running time of the algorithm of Theorem 1 is very likely unavoidable. Indeed, consider the following reduction from the CLIQUE problem where the input is an undirected graph G and $\ell \in \mathbb{N}$, and the objective is to decide whether G contains a clique of size ℓ. We orient all edges in G arbitrarily, add a universal sink vertex $v^\star$ and then a universal source vertex $u^\star$ and the arc $(v^\star, u^\star)$ to obtain a strongly connected digraph, set $k = 0$, and set $\mathcal{H}$ to be the set of all tournaments on exactly $\ell + 2$ vertices. Then, an FPT algorithm for R-$\mathcal{H}$-SCC DELETION parameterized by k and ℓ would imply an FPT algorithm for CLIQUE parameterized by ℓ.

When $\mathcal{H}$ only comprises of the star with $d + 1$ leaves with all arcs oriented away from the centre, a closer inspection of the algorithm of Theorem 1 demonstrates that it can be implemented in a way that implies a fixed-parameter algorithm parameterized by both k and d for this problem. We call this problem, d-OUT-DEGREE SCC DELETION. In this problem, the objective is to decide whether k vertices can be deleted from a given digraph to ensure that the graph induced by each strong component has out-degree at most d.

Theorem 2. d-OUT-DEGREE SCC DELETION *can be solved in time* $2^{O(k^3 \log k)} \cdot n^{O(1)}$.

Our next result concerns the PATH $\mathcal{H}$-SCC DELETION (P-$\mathcal{H}$-SCC DELETION) problem, which is the special case of $\mathcal{H}$-SCC Deletion where $\mathcal{H}$ contains at least one directed path. We show that with an appropriate fixed-parameter preprocessing routine, this problem can be reduced to R-$\mathcal{H}$-SCC DELETION where $\mathcal{H}$ only comprises of the path of length $g(\mathcal{H})$ for some function g. Invoking Theorem 1 then leads us to the following result.

Theorem 3. P-$\mathcal{H}$-SCC DELETION *can be solved in time* $2^{O(k^3 \log k)} \cdot h^{O(k)} \cdot 2^{O(h^6)} \cdot n^{O(h^3)}$.

We then pay special attention to the R-$\mathcal{H}$-SCC DELETION problem when $\mathcal{H}$ contains only the out-directed 2-star, i.e., the 1-OUT-DEGREE SCC DELETION problem. Notice that a strongly connected graph with at least two vertices that excludes this graph as a subgraph must be a simple cycle, and so is 1-out-regular. A 1-out-regular digraph is a digraph where every vertex has out-degree exactly 1. Therefore, this special case of R-$\mathcal{H}$-SCC DELETION is precisely the 1-OUT-REGULAR DELETION problem where one is given a digraph D and an integer k and the objective is to decide whether there is a set of vertices of size at most k whose deletion leaves a digraph where every strong component induces a 1-out-regular subgraph. Göke et al. [6] recently gave an algorithm for this problem with running time $2^{O(k^3)} \cdot n^{O(1)}$. We give an improved algorithm for this problem with an asymptotic dependence on k that matches that of the current best algorithm for DFVS [2], which is a special case of 1-OUT-REGULAR DELETION.

Theorem 4. 1-OUT-REGULAR VERTEX DELETION *can be solved in time* $2^{O(k \log k)} \cdot n^{O(1)}$.

Finally, we also study the special case of R-$\mathcal{H}$-SCC DELETION when $\mathcal{H}$ is the set of all arborescences on exactly $s + 1$ vertices. Notice that the strongly connected graphs that exclude the graphs in $\mathcal{H}$ as subgraphs are precisely the strongly connected graphs of size at most s. This problem when $\mathcal{H}$ is the set of all arborescences on exactly $s + 1$ vertices is called the BOUNDED SIZE STRONG COMPONENT VERTEX DELETION (BSSCVD). We improve upon the result of Göke et al. [6] who gave an algorithm for BSSCVD with running time $4^k(ks + k + s)! \cdot n^{O(1)}$.

Theorem 5. BSSCVD *can be solved in time* $2^{O(k(\log k + \log s))} \cdot n^{O(1)}$.

We now give an overview of the techniques used to prove our results.
Algorithm for R-$\mathcal{H}$-SCC DELETION. We begin by using the technique of iterative compression to obtain a tuple $(D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$ such that W is a solution for the instance

$(D, k + 1)$ of R-$\mathcal{H}$-SCC DELETION, $S_1, \ldots, S_q$ partition W and moreover, if there is a solution for (D, k), then there is a solution that is disjoint from W and intersects S_i-S_j paths for $i < j$. We note that this step is standard when dealing with directed cut problems.

A commonly used technique subsequent to this step (albeit one that we *do not* employ) is the directed *shadow removal* technique introduced by Chitnis et al. [4] where one identifies a set of vertices Z such that for some hypothetical solution X, Z is disjoint from X and contains the set of vertices that are either unable to reach W or are unreachable from W in $D - X$. This set is then removed in a problem specific way while preserving all obstructions. While this can be easily achieved for certain simple obstructions, we are dealing with an arbitrary family of digraphs with the only assumption being that they are rooted. Consequently, it is not at all clear how one could implement the removal of vertices in Z and that makes our task significantly more challenging. To avoid this obstacle, we forgo the technique of shadow removal and directly design an intricate branching algorithm.

The crux of this algorithm is the observation that for a special type of solution X, for every forbidden subgraph F, either X intersects $V(F)$ or X contains an S_1-$\{r(F)\}$ separator ($r(F)$ denotes a fixed root of F) or X contains a $\{u\}$-W separator for some vertex $u \in V(F)$. We then show that there is always an efficiently computable forbidden subgraph upon which we can branch exhaustively using the above observation in such a way that we always make progress. The fact that such a subgraph can always be identified efficiently is far from obvious and proving it is one of our main technical challenges.

Algorithm for P-$\mathcal{H}$-SCC DELETION. We show that for every finite family of digraphs $\mathcal{H}$, there exists another (infinite) family $\mathcal{H}^*$, such that $\mathcal{H}$-SCC problem is equivalent to the problem of deleting at most k vertices to exclude all graphs in $\mathcal{H}^*$ as subgraphs in the remaining graph (not necessarily in a single strong component). Moreover, we show that when $\mathcal{H}$ contains a directed path, then the family $\mathcal{H}^*$ can be partitioned into two, say $\mathcal{H}_1^*$ and $\mathcal{H}_2^*$ such that $\mathcal{H}_1^*$ is finite and the problem of deleting at most k vertices to exclude all graphs in $\mathcal{H}_2^*$ as subgraphs in the remaining graph is equivalent to the R-$\mathcal{X}$-SCC DELETION where $\mathcal{X}$ only contains a directed path whose length depends on $\mathcal{H}$. This allows us to branch on all subgraphs isomorphic to graphs in $\mathcal{H}_1^*$ and then invoke Theorem 1.

Improved algorithms for 1-OUT-REGULAR VERTEX DELETION **and** BSSCV. Here, we begin in the same way as for R-$\mathcal{H}$-SCC DELETION by obtaining a tuple $(D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$ such that if there is a solution for (D, k), then there is a solution that is disjoint from W and intersects all S_i-S_j paths for $i < j$. In the case of 1-OUT-REGULAR VERTEX DELETION, the main new contribution that results in a speedup over Theorem 1 is a lemma that shows that if we consider an S_1-$W \setminus S_1$ separator C such that every vertex reachable from S_1 in $D - C$ has out-degree at most 1 in D, then there is a solution whose intersection with this set of reachable vertices is contained within an efficiently computable set of $O(k)$ vertices. This gives us a branching algorithm where we essentially compute a "furthest" S_1-$W \setminus S_1$ separator C of size at most k in time $2^{O(k)} \cdot n^{O(1)}$ and then branch on deleting a vertex of C or one of these $O(k)$ vertices in the reachable set. In the case of BSSCV, we show that if for some $x \in S_1$, D contains a subgraph of size $s + 1$ with an arborescence rooted at x, then at least one of these vertices must either be deleted or must have all its paths to W deleted when removing a solution. In the latter case, we will be able to branch on an $\{x\}$-W important separator [12] of size at most k.

Further Remarks. The results of Göke et al. [6] crucially use the technique of *covering the shadow* which adds a factor of $2^{O(k^2)} \cdot n^{O(1)}$ to the running time of any algorithm that uses it. Thus, our $2^{O(k \log k)} \cdot n^{O(1)}$ algorithm (Theorem 3) is an improvement over what is currently possible using the shadow covering technique. Moreover, although all our results are stated for the vertex deletion version of the problems, we would like to mention that these results will

apply for the corresponding arc deletion versions of the problems as well, as all our proofs go through for the later case also.

2 Preliminaries

We use standard notion regarding digraphs. Given two graphs D_1, D_2 we denote the digraph $D_1 \cup D_2$ as the digraph with vertex set $V(D_1) \cup V(D_2)$ and arc set $A(D_1) \cup A(D_2)$. Note that the vertex sets of D_1 and D_2 are not necessarily disjoint. By $D_1 \subseteq D_2$, we mean that D_1 is a *subdigraph* of D_2. We use $|D|$ as a shorthand for $|V(D)|$. A *strongly connected component (or strong component)* of a digraph D is a maximal set $S \subseteq V(D)$ such that for any pair u, v of vertices in S, there is a path from u to v and from v to u in D. It is well known that it is possible to order the strongly connected components of D such that there is no path from a component to another component behind it in the ordering. This is called the *topological ordering* of the strongly connected components of D. For any digraph D, for any $X \subseteq V(D)$, $D[X]$ denotes the subdigraph of D *induced* by X and $D - X$ denotes the induced subdigraph $D[V(D) \setminus S]$. For a vertex $v \in V(D)$, by $N_D^+(v)$ we denote $\{u \in V(D) \mid (v, u) \in A(D)\}$ and by $N_D^-(v)$ we denote $\{u \in V(D) \mid (u, v) \in A(D)\}$. For a subset $S \subseteq V(G)$, by $N_D^+(S)$ we denote the set $\bigcup_{v \in S} N_D^+(v) \setminus S$. By $N_D^+[S]$ we denote $N_D^+(S) \cup S$. Analogous definitions hold for $N_D^-(S)$ and $N_D^-[S]$. When the digraph D is clear from the context, we drop the subscript D from the notation.

For any $S, T \subseteq V(D)$, by an *S-T path* in D we mean a path from some vertex of S to some vertex of T in D. When $S = \{v\}$ (resp. $T = \{v\}$) is a singleton set, we write a *v-T* (resp. *S-v* path). For $S, T, C \subseteq V(D)$ such that $S \cap T = \emptyset$, we say that C is an *S-T separator* if there is no directed S-T path in $D - C$ and $C \cap S = C \cap T = \emptyset$. For an S-T separator C, by $R_D(S, C)$ we denote the set of vertices v such that there exists an S-$\{v\}$ path in $D - C$. By $\overline{R}_D(S, C)$ we denote the set $V(D) \setminus R_D(S, C)$, that is the set of vertices that are unreachable from S after removing C. Note that for any set $R \subseteq V(D)$ such that $S \subseteq R$, $R \cap T = \emptyset$ and $N^+(R) \cap T = \emptyset$, if R is reachable from S in $D[R]$, then the set $C = N^+(R)$ is an S-T separator such that $R_D(S, C) = R$. We say that an S-T separator C *covers* an S-T separator C' if $R_D(S, C) \supseteq R_D(S, C')$. We say that C *tightly covers* C' if C covers C' and there does not exist a C'' that covers C' and is covered by C. Two S-T separators are *incomparable* if neither covers the other. $\lambda_D(S, T)$ denotes the size of a minimum S-T separator in D. It is well known [2] that there exists a unique minimum S-T separator $C_{\min}(S, T)$ and a unique minimum S-T separator $C_{\max}(S, T)$ such that for every minimum S-T separator C, $C_{\min}(S, T)$ is covered by C and $C_{\max}(S, T)$ covers C. We call $C_{\min}(S, T)$ the *closest minimum S-T separator* and $C_{\max}(S, T)$ the *furthest minimum S-T separator*. Moreover, we define $R_{\min}(S, T) = R_D(S, C_{\min}(S, T))$ and $R_{\max}(S, T) = R_D(S, C_{\max}(S, T))$. All four of these sets can be computed in polynomial time using max-flow computations (see [12]).

An *arborescence* is a rooted directed tree where every vertex except the root has in-degree exactly 1 and the root has in-degree 0. A digraph D is said to be *rooted* at $v \in V(D)$ if D contains as a subdigraph on $V(D)$ an arborescence rooted at v. That is, there is a directed v-w path in D for every $w \in V(D)$. A digraph D is simply said to be *rooted* if it is rooted at some vertex. By $r(D)$, we denote the vertex that is the root of D. If there are multiple roots for D, we canonically fix one vertex for $r(D)$. For a digraph D and a family of digraphs $\mathcal{H}$ (potentially containing rooted digraphs), we say that a subset $S \subseteq V(D)$ is $\mathcal{H}$-*free* if $D[S]$ does not contain any graph in $\mathcal{H}$ as a subgraph. When $S = V(D)$, we say that D is $\mathcal{H}$-free. In the case when every graph in $\mathcal{H}$ is a rooted graph, we say that S is *root-$\mathcal{H}$-free* if the root of every subgraph of D that is isomorphic to a graph in $\mathcal{H}$ is not contained in S. Observe that if a set S is root-$\mathcal{H}$-free then it is also $\mathcal{H}$-free. We say that a set $X \subseteq V(D)$ is an $\mathcal{H}$-*deletion set* for D

4

if there is no subgraph isomorphic to a graph in $\mathcal{H}$ that is contained in any strong component of $D - X$. Furthermore, we say that X is a *solution* for the tuple (D, k) if X is an $\mathcal{H}$-deletion set and $|X| \leq k$. Throughout the paper, $h = \max_{H \in \mathcal{H}} |V(H)|$.

The following observation follows from the sub-modularity of separators.

Observation 1 ([5]). *Let C_1 and C_2 be two minimum S-T separators in a digraph D. Let $R_1 = R_D(S, C_1)$ and $R_2 = R_D(S, C_2)$, then $N^+(R_1 \cap R_2)$ and $N^+(R_1 \cup R_2)$ are also minimum S-T separators of D.*

Lemma 1. [10] *There is a polynomial-time algorithm that, given a digraph D and an S-T separator C in D, either correctly concludes that $C = R_{\max}(S, T)$ or outputs a minimum S-T separator that tightly covers C.*

Lemma 2. [10] *Let $C_1 = N^+(R_1)$ and $C_2 = N^+(R_2)$ be minimum S-T separators such that C_2 tightly covers C_1. Then there does not exist any minimum S-T separator C such that $C \cap (R_2 \setminus N[R_1]) \neq \emptyset$.*

Lemma 2 basically implies that there is no minimum S-T separator in D that contains a vertex in $R_D(S, C_2) \setminus N[R_D(S, C_1)]$.

Lemma 3. [9] *Let D be a digraph and let $S, T \subseteq V(D)$ such that $S \cap T = \emptyset$. Let C be the closest (resp. furthest) S-T separator in D and let v be a vertex in $R_D(S, C)$ (resp. $\overline{R}_D(S, C)$). Then every S-$(T \cup \{v\})$ (resp. $(S \cup \{v\})$-T) separator is of size strictly greater than $\lambda_D(S, T)$.*

Lemma 4. *Let D be a digraph and let S, T be disjoint subsets of $V(D)$. Let $C_1 = N^+(R_1)$ and $C_2 = N^+(R_2)$ be two minimum S-T separators such that C_2 covers C_1 and $C_1 \neq C_2$. Let $u \in C_1$ and $v \in R_2 \setminus N[R_1]$. Then every $(S \cup \{u\}) - (T \cup \{v\})$ separator is of size strictly greater than $\lambda_D(S, T)$.*

Proof. Let C' be an $(S \cup \{u\})$-$(T \cup \{v\})$ separator. Since $u \in N[R_1]$ and $v \in R_2 \setminus N[R_1]$, it must be the case that C' contains a vertex in $R_2 \setminus N[R_1]$. Thus it follows from Lemma 2 that C' is of size greater than $\lambda_D(S, T)$. $\square$

Definition 1. [2, 5, 12] *For a digraph D and disjoint subsets S, T of $V(D)$, an S-T separator C is said to be* important *if there is no S-T separator $C' \neq C$ such that $|C'| \leq |C|$ and C' covers C.*

Lemma 5. [2, 5, 12] *There are at most 4^k important separators of size at most k. Moreover, they can be enumerated in $O^*(4^k)$ time.*

The notion of important separators has been extrememly useful in the design of parameterized algorithms (see Marx' survey [13]).

Lemma 6. *Let D be a digraph and $S, T \subseteq V(D)$ such that $S \cap T = \emptyset$. Let $X \subseteq V(D)$ contain a minimal S-T separator C. Let C' be an S-T separator that covers C and let $X' = (X \setminus C) \cup C'$. Let $u, v \in \overline{R}_D(S, X')$. If there exists a path from u to v in $D - X'$ then there exists a path from u to v in $D - X$.*

Proof. Suppose not. Then there exists a path P from u to v in $D - X'$ but no path from u to v in $D - X$, that is, $V(P) \cap X' = \emptyset$ and $V(P) \cap X \neq \emptyset$. Let $a \in V(P) \cap X$. Then $a \in X \setminus X' \subseteq C$. Let $R = R_D(S, X)$. Since C is a minimal S-T separator and is contained in X, we have that $C = N^+(R)$ and thus $a \in N^+(R)$. Since C' covers C, $R_D(S, X) \subseteq R_D(S, X')$. Also, since $a \notin X'$, it follows that $a \in R_D(S, X')$. Thus, there is a path from S to a in $D - X'$, which together with the subpath of P from a to v gives a walk (and eventually a path) from S to v in $D - X'$. This is a contradiction to the fact that $v \in \overline{R}_D(S, X')$. $\square$

3 The FPT algorithm for rooted $\mathcal{H}$-SCC Deletion

In this section, we consider the $\mathcal{H}$-SCC deletion problem when all the graphs in $\mathcal{H}$ are rooted (the R-$\mathcal{H}$-SCC problem) and prove Theorem 1. Towards the end of the section, we also exhibit the proof of Theorem 2.

Theorem 1. R-$\mathcal{H}$-SCC Deletion *can be solved in time* $2^{O(k^3 \log k)} \cdot n^{O(h)}$.

We use the standard technique of Iterative Compression ([14]) to reduce the task of solving our instance of R-$\mathcal{H}$-SCC Deletion to that of solving at most $2^{k+1}n$ instances of the Disjoint R-$\mathcal{H}$-SCC Deletion Compression (R-$\mathcal{H}$-SCC DC) problem, where we are given a digraph D and a solution W of size at most $k+1$ and the goal is to compute a solution of size at most k that is disjoint from W, if one exists. The rest of the section is devoted to the proof of the following lemma, which as described above proves Theorem 1.

Lemma 7. *There is an algorithm that, given an instance* (D, W, k) *of* R-$\mathcal{H}$-SCC DC, *runs in time* $2^{O(k^3 \log k)} \cdot n^{O(h)}$ *and either correctly concludes that there is no solution for this instance disjoint from* W *or outputs a solution disjoint from* W.

3.1 Reduction to Nice Instances of the Partitioned Compression Version

We further solve R-$\mathcal{H}$-SCC DC by making $2^{O(k \log k)}$ calls to a subroutine that is allowed to assume the existence of a specific type of solution for instances of the R-$\mathcal{H}$-SCC Partitioned Compression (R-$\mathcal{H}$-SCC PC) problem, which is described below. An instance of R-$\mathcal{H}$-SCC PC is a tuple $(D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$, where W is a solution for the instance $(D, k+1)$ of $\mathcal{H}$-SCC and $\mathcal{S}$ is an ordered partition of W. A set $X \subseteq V(D)$ is said to be a *solution* for the instance $(D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$ of R-$\mathcal{H}$-SCC PC if X is a solution for the instance (D, k) of $\mathcal{H}$-SCC, $X \cap W = \emptyset$, and X intersects all S_i-S_j paths in D, for every $j > i$. Observe that every solution of the instance $(D, \mathcal{S}, W, k)$ of R-$\mathcal{H}$-SCC PC, is also a solution of (D, W, k) of R-$\mathcal{H}$-SCC DC. We now define a special kind of solution for R-$\mathcal{H}$-SCC PC, which we call a *nice solution*, and as we will see soon, it turns out that it is enough to look for nice solutions for our purpose.

Definition 2 (Nice Instances). Let $(D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$ be an instance of R-$\mathcal{H}$-SCC PC. A solution X for this instance is said to be *nice* if for every subgraph $F \subseteq D$ such that F is isomorphic to a graph in $\mathcal{H}$, and each $i \in [q]$, one of the following holds:

1. X intersects $V(F)$,

2. $r(F) \notin R(S_i, X)$, or

3. $\exists v \in V(F)$ such that there is no $\{v\}$-S_i path in $D - X$.

Observe from the definition above, that if X is a solution of R-$\mathcal{H}$-SCC PC such that after its deletion each S_i is in exactly one strong component, then X is a nice solution.

Observation 2. *Let* $(D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$ *be an instance of* R-$\mathcal{H}$-SCC PC. *If* X *is a nice solution for this instance, then for any* $X' \subseteq X$, $X \setminus X'$ *is a nice solution for the instance* $(D - X', \mathcal{S}, W, k - |X'|)$.

Observation 3. *Let* $X' \subseteq V(D) \setminus W$. *If* X *is a nice solution for* $(D - X', \mathcal{S}, W, k - |X'|)$ *then* $X \cup X'$ *is a nice solution for* $(D, \mathcal{S}, W, k)$.

We now show that for our purposes, it is enough to look for nice solutions.

Lemma 8. (D, W, k) *is a yes-instance of* R-$\mathcal{H}$-SCC DC *if and only if* $(D, \mathcal{S} = (S_1, \ldots, S_q),$ $W, k)$ *is a nice yes-instance of* R-$\mathcal{H}$-SCC PC, *for some ordered partition* $\mathcal{S} = (S_1, \ldots, S_q)$ *of* W.

Proof. For the forward direction, suppose (D, W, k) is a yes-instance of R-$\mathcal{H}$-SCC DC. Let $X \subseteq V(D) \setminus W$ be an $\mathcal{H}$-SCC solution for D of size at most k. Let $C_1, \ldots, C_q$ be the set of strongly connected components of $D - X$ as they appear in some topological ordering of the strongly connected components of $D - X$. For each $i \in [q]$, let $S_i = W \cap C_i$. Since $W \subseteq V(D) \setminus X$, $(\mathcal{S} = (S_1, \ldots, S_q))$ is an ordered partition of W. From the construction above, note that X is a solution to the instance $(D, \mathcal{S}, W, k)$ of R-$\mathcal{H}$-SCC PC. We now prove that X is also nice. Let $F \subseteq D$ such that F is isomorphic to a graph in $\mathcal{H}$. Fix $i \in [q]$. We will prove that either $V(F) \cap X \neq \emptyset$ or X contains either an S_i-$\{r(F)\}$ separator or an $\{u\}$-S_i separator for some vertex $u \in V(F)$. If $X \cap V(F) \neq \emptyset$ or there is no S_i-$\{r(F)\}$ path in $D - X$ we are done. Otherwise, $r(F)$ is reachable from S_i in $D - X$. Since $r(F)$ is the root of F and $F \subseteq D - X$, all the vertices of $V(F)$ are reachable from S_i in $D - X$. For the sake of contradiction, suppose that for each $u \in V(F)$, there is a path from u to S_i in $D - X$. Then F should be in the same connected component as S_i in $D - X$. This contradicts that X is a solution.

For the backward direction, suppose that there is a nice solution for $(D, \mathcal{S} = (S_1, \ldots, S_q),$ $W, k)$. Since every nice solution is a solution for the instance (D, W, k) of R-$\mathcal{H}$-SCC DC, we are done. □

Since the number of ordered partitions of a set W is at most $|W|^{|W|}$ and the size of W in an instance (D, W, k) of R-$\mathcal{H}$-SCC DC is $k+1$, we conclude from Lemma 8 that to solve R-$\mathcal{H}$-SCC DC, it is enough to solve $2^{O(k \log k)}$ nice instances of R-$\mathcal{H}$-SCC PC. Towards this, in order to prove Lemma 7 (and hence Theorem 1), it is enough to prove the following lemma.

Lemma 9. *There is an algorithm that, given an instance* $\mathcal{I} = (D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$ *of* R-$\mathcal{H}$-SCC PC, *runs in time* $O^*(2^{O(k^3 \log k)})$ *and either correctly concludes that there is no nice solution for* $(D, \mathcal{S}, W, k)$ *or outputs some solution for* $(D, \mathcal{S}, W, k)$.

The upcoming subsections are devoted to the proof of Lemma 9. Recall that the most challenging aspect in our strategy (see overview of our algorithm for R-$\mathcal{H}$ SCC DELETION in Section 1) is the identification of appropriate 'branchable' forbidden subgraphs. Specifically, we need to identify particular subgraphs F such that the natural exhaustive branchings reduce some measure depending on the parameter. The way we identify such subgraphs is the following: the algorithm will maintain a set Q such that Q is root-$\mathcal{H}$-free, with $Q = \emptyset$ initially. The algorithm will try to 'grow' the set Q until the entire graph is covered by Q. Initially, when $Q = \emptyset$, we try to grow it to the set $R_{\min}(S, T)$, the closest minimum S-T separator. We prove that if we find a forbidden subgraph whose root lies in $R_{\min}(S, T)$, that subgraph is good for us in the sense that all branches will drop our measure. Once all roots have been removed from $R_{\min}(S, T)$, we set $Q = R_{\min}(S, T)$. Then we recurse and grow Q further towards T. To formalize the above strategy in Section 3.4, we next describe two crucial tools, in the form of pushing lemmas.

3.2 Setting up the Required Notations and Machinery

Let $(D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$ be an instance of R-$\mathcal{H}$-SCC PC. Let X be some solution for this instance. Suppose, for instance, one had a hold on the set of vertices, say R, that are in the strong components containing S_1 in $D - X$. Then, one could argue that there is some other solution that picks an important R-$(W \setminus S_1)$ separator of size at most k. In this case, one can branch of these important sets. Unfortunately, the above mentioned set R is far from being found efficiently. However growing on this idea, our first pushing lemma, Lemma 11,

says that even if one is able to find a "weaker" set, viz. some superset of the vertices that are reachable from S_1 after the deletion of the solution, do not contain a graph of $\mathcal{H}$ inside them and their out-neighbourhood forms a minimum S_1-$(W \setminus S_1)$, then one can always construct another solution that picks the out-neighbours of this set.

Lemma 10. *Let X be an $\mathcal{H}$-deletion set of D. Let $F \subseteq D$ be isomorphic to some graph in $\mathcal{H}$. Suppose there exists a set of paths $\mathcal{P}$ containing a path P_{xy} from x to y in D for each $(x,y) \in V(F) \times V(F)$ (equivalently, F is contained in a single strong component of D). Then there exists a vertex $a \in X$ such that for every $v \in V(F)$, there exists a path from v to a and a path from a to v that is contained in $\bigcup_{(x,y)\in V(F)\times V(F)} P_{xy}$.*

Proof. Consider the collection $\mathcal{P}$ of paths described in the lemma. Since X is an $\mathcal{H}$-deletion set of D, F is not contained inside one strongly connected component of $D - X$. Thus, there exists $x, y \in V(F)$, such that $P_{xy} \in \mathcal{P}$ is not entirely contained in $D - X$, that is, $V(P_{xy}) \cap X \neq \emptyset$. Let $a \in V(P_{xy}) \cap X$. Consider the subpath, say $P' \subseteq P_{xy}$, from x to a. Since $a \in X$, P' is the desired path from x to a in D contained inside $\bigcup_{(x,y)\in V(F)\times V(F)} P_{xy}$. For any other $v \in V(F) \setminus \{x\}$, consider the path P_{vx} described in the lemma. P_{vx} together with P' give a walk (and eventually a path) from v to a (and hence X) that is contained inside $\bigcup_{(x,y)\in V(F)\times V(F)} P_{xy}$. A symmetric argument shows that we can also get a path from a to v, for each $v \in V(F)$, that is contained inside $\bigcup_{(x,y)\in V(F)\times V(F)} P_{xy}$. $\qquad \square$

Lemma 11 (Pushing-Routine-1)**.** *Let $\mathcal{I} = (D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$ be an instance of R-$\mathcal{H}$-SCC PC. Consider a $\mathcal{H}$-free set $S_1 \subseteq Q \subseteq V(D) \setminus (W \setminus S_1)$ such that $N^+(Q)$ is a minimum S_1-$(W \setminus S_1)$ separator. Let X be a solution of $\mathcal{I}$ such that $R_D(S_1, X) \cap N^+(Q) = \emptyset$. Then, there is a solution X' for $\mathcal{I}$ that contains $N^+(Q)$.*

Proof. Let $R = R_D(S_1, X)$. We will first show that $N^+(Q)$ covers $N^+(R)$. Observe that both $N^+(R)$ and $N^+(Q)$ are S_1-$(W \setminus S_1)$ separators in D. Also, since $R \cap (W \setminus S_1) = \emptyset$, $N^+(R)$ is a minimal S_1-$(W \setminus S_1)$ contained in X. Since $R_D(S_1, X) \cap N^+(Q) = \emptyset$, $R_D(S_1.R) \subseteq R_D(S_1, Q)$. Thus, $N^+(Q)$ covers $N^+(R)$.

We will now construct a set X' that contains $N^+(Q)$, and prove that it is also a solution for $\mathcal{I}$. Let $X' = (X \setminus N^+(R)) \cup N^+(Q)$. Since both $N^+(R)$ and $N^+(Q)$ are S_1-$W \setminus S_1$ separators in D and $N^+(Q)$ is a minimum one, $|N^+(Q)| \leq |N^+(R)|$. Thus, $|X'| \leq |X|$. Also, since $X \cap W = \emptyset$ and $N^+(Q) \cap W = \emptyset$ (because $N^+(Q)$ is a minimum S_1-$(W \setminus S_1)$ separator), we conclude that $X' \cap W = \emptyset$. Also observe that $R_D(S_1, X') \subseteq Q$.

Suppose that there exists $F \subseteq D - X'$ which is isomorphic to some graph in $\mathcal{H}$ and which is contained in a single strongly connected component of $D - X'$. Then, for each $(x,y) \in V(F) \times V(F)$, there exists paths P_{xy} that are entirely contained in $D - X'$. We will first prove that for each $v \in V(F)$, $v \in \overline{R}_D(S_1, X')$. Suppose not. Let $v \in V(F)$ be such that $v \in R_D(S_1, X')$. Then since the paths $P_{x,y}$ (for each $(x,y) \in V(F) \times V(F)$) are in $D - X'$, this implies $V(F) \subseteq R_D(S_1, X') \subseteq Q$. This contradicts that Q is $\mathcal{H}$-free. Since, for all $(x,y) \in V(F) \times V(F)$, $x, y \in \overline{R}_D(S_1, X')$, P_{xy} is a path from x to y in $D - X'$ and $N^+(Q)$ covers $N^+(R)$, from the construction of X' and Lemma [6], for all $(x,y) \in V(F) \times V(F)$, there exists a path from x to y in $D - X$. This implies that F is a strongly connected graph in $D - X$, and therefore present in some strongly connected component of $D - X$. This contradicts that X is a solution to $\mathcal{I}$.

We will now show that for each $i, j \in [q]$, $i < j$, there is no path from S_i to S_j in $D - X'$. Since $N^+(Q) \subseteq X'$ and $N^+(Q)$ is an S_1-$(W \setminus S_1)$ separator in D, there is no path from S_1 to S_j, $j > 1$. Consider any pair S_i, S_j, $i < j$. Note that $S_i, S_j \subseteq W \setminus S_1$. Also, $S_i, S_j \in \overline{R}_D(S_1, X')$. For the sake of contradiction, suppose there exists a vertex $a \in S_i$ and $b \in S_j$ such that there is a path from a to b in $D - X'$. Since $a, b \in \overline{R}_D(S_1, X')$, and $N^+(Q)$ covers $N^+(R)$, from the

construction of X' and Lemma 6, we conclude that there is a path from a to b, and hence from S_i to S_j, in $D - X$ too. This contradicts that X is a solution to $\mathcal{I}$. $\qquad\square$

For our second pushing lemma, we first borrow definitions of shadows for directed graphs from [4]. Note that what we do with the concept of shadows in our article is very different from the way it has been used so far. More specifically, we do not resort to the shadow removal technique that has often been an effective technique to design FPT algorithms for directed cut problems. In fact, for the general problems that we consider, it is not at all clear how the shadow removal technique could be helpful. Let $(D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$ be an instance of R-$\mathcal{H}$-SCC PC and let $X \subseteq V(D) \setminus W$. Then, F-Shadow$(X)$ denotes the set of those vertices $u \notin X$ such that there is no $\{u\}$-W path in D-X. Similarly, R-Shadow(X) denotes the set of those vertices $u \notin X$ such that there is no W-$\{u\}$ path in D-X. F-Shadow(X) is called the *forward shadow of X with respect to W* and R-Shadow(X) is called the *reverse shadow of X with respect to W*. Notice that with these definitions, we have that if X is a nice solution for the instance $(D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$, then for any subgraph $F \subseteq D$ that is isomorphic to a graph in $\mathcal{H}$ and any $i \in [q]$, either X intersects $V(F)$ or $r(F) \notin R(S_i, X)$ or $V(F) \cap$ F-Shadow$(X) \neq \emptyset$. Moreover, when $q = 1$, this implies that X intersects $V(F)$ or $r(F) \in$ R-Shadow(X) or $V(F) \cap$ F-Shadow$(X) \neq \emptyset$. Our second pushing lemma, guarantees the existence of a set to branch on, provided we have identified some vertex in the forward or reverse shadow of X with respect to W. For a digraph D, D^{rev} denotes the digraph obtained by reversing the direction of all arcs in D.

Lemma 12. *Let $\mathcal{I} = (D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$ be an instance of R-$\mathcal{H}$-SCC PC and let X be a solution for this instance. Let $u \in V(D)$ be a vertex in F-Shadow(X) (resp. R-Shadow(X)). Then, there is a solution for $\mathcal{I}$ that contains an important $\{u\}$-W separator in D (resp. an important $\{u\}$-W separator in D^{rev}).*

Proof. We prove the case when $u \in$ F-Shadow(X). The case when $u \in$ R-Shadow(X) is symmetrical. Since u is in the forward shadow of X w.r.t. W, there is no $\{u\}$-W path in $D - X$ and thus, X is a $\{u\}$-W separator. Let $R = R_D(\{u\}, X)$. Since R is exactly the set of vertices reachable from u after removing X, it follows that $N^+(R) \subseteq X$. Moreover, since $u \in R$ and $R \cap W = \emptyset$ it follows that $N^+(R)$ is a $\{u\}$-W separator. Thus there exists an important $\{u\}$-W separator C that covers $N^+(R)$ such that $|C| \leq |N^+(R)|$. Let $X' = (X \setminus N^+(R)) \cup C$. Clearly $|X'| \leq |X|$. Also, since $X \cap W = \emptyset$ and $C \cap W = \emptyset$ (because C is a $\{u\}$-W separator), $X' \cap W = \emptyset$. We now prove that that X' is a solution for $\mathcal{I}$.

Suppose that there exists $F \subseteq D - X'$ which is isomorphic to some graph in $\mathcal{H}$ and which is contained in a single strongly connected component of $D - X'$. Then, for each $(x, y) \in V(F) \times V(F)$, there exists paths P_{xy} that are entirely contained in $D - X'$. We will first prove that for each $v \in V(F)$, $v \in \overline{R}_D(\{u\}, X')$. Suppose not. Let $v \in V(F)$ be such that $v \in R_D(\{u\}, X')$. In this case, first note that $v \notin W$ because $u \in$ F-Shadow(X). Since, W is a solution to D for the $\mathcal{H}$-SCC problem, from Lemma 10, there exists a path from v to W in $\bigcup_{(x,y) \in V(F) \times V(F)} P_{xy}$. Since the paths P_{xy} are in $D - X'$, we conclude that there is a path from v to W in $D - X'$. Also, since $v \in R_D(\{u\}, W)$, we conclude that there is a path from u to W in $D - X'$, thereby contradicting that $u \in$ F-Shadow(X). Thus, we now have that for all $(x, y) \in V(F) \times V(F)$, $x, y \in \overline{R}_D(\{u\}, X')$. Since C covers $N^+(R)$ and there is a path $P_{x,y}$ from x to y in $D - X'$, from the construction of X' and Lemma 6, there is also a path from x to y in $D - X$. This implies that F is a strongly connected in $D - X$ and hence is contained in some strongly connected component of $D - X$. This contradicts that X is a solution to $\mathcal{I}$.

We will now show that for each $i, j \in [q]$, $i < j$, there is no path from S_i to S_j in $D - X'$. Consider any pair S_i, S_j, $i < j$. For the sake of contradiction, suppose there exists a vertex

$a \in S_i$ and $b \in S_j$ such that there is a path from a to b in $D - X'$. Since $S_i, S_j \subseteq W$ and $u \in \mathsf{F\text{-}Shadow}(X)$, $a, b \in \overline{R}_D(\{u\}, X')$. Since C covers $N^+(R)$, from the construction of X' and Lemma 6, there is a path from a to b, and hence from S_i to S_j, in $D - X$. This contradicts that X is a solution to $\mathcal{I}$. $\qquad \square$

As a consequence of Lemma 12 and Lemma 5, we have the following.

Lemma 13 (PUSHING-ROUTINE-2). *There is an algorithm that, given an instance $\mathcal{I} = (D, \mathcal{S}, W, k)$ of R-$\mathcal{H}$-SCC PC and a vertex $u \in V(D)$ such that either there is a u-W path or a W-u path in D, runs in time $4^k \cdot n^{O(1)}$ and outputs a non-empty set $Z \subseteq V(D)$ of size at most $4^k \cdot 2k$ with the following property: if there is a solution X for $\mathcal{I}$ such that $u \in \mathsf{F\text{-}Shadow}(X) \cup \mathsf{R\text{-}Shadow}(X)$, then there is a solution X' for $\mathcal{I}$ such that $X' \cap Z \neq \emptyset$.*

Proof. Given the instance $\mathcal{I}$ and $u \in V(D)$, let $\mathcal{F}_1$ be the family of important $\{u\}$-W separators of size at most k in D and let $\mathcal{F}_2$ be the family of important $\{u\}$-W separators of size at most k in D^{rev}. Let $Z = \cup_{X \in \mathcal{F}_1 \cup \mathcal{F}_2} X$. Note that since there is either a u-W path or a W-u path, Z must be non-empty. Then from Lemma 12, there is a solution X' of $\mathcal{I}$ such that $X' \cap Z \neq \emptyset$. Also, from Lemma 5, $|Z| \leq 4^k \cdot 2k$. $\qquad \square$

3.3 Solving instances of R-$\mathcal{H}$-SCC PC with a trivial partition

Towards the proof of Lemma 9, we first find a set $\widehat{Z}$ such that $\widehat{Z}$ intersects some solution for $\mathcal{I} = (D, \mathcal{S}, W, k)$ (if one exists) and $|\widehat{Z}| = O(h) \cdot 2^{O(k^2 \log k)}$. Observe that, having such a set $\widehat{Z}$ at hand, one can proceed with a branching algorithm that branches on the vertices of $\widehat{Z}$. We call the set $\widehat{Z}$, the *branch set* for the instance $\mathcal{I}$. The rest of the section is devoted to computing a branch set for $\mathcal{I}$ of the desired size.

Lemma 14. *Given an instance $\mathcal{I} = (D, \mathcal{S}, W, k)$ of R-$\mathcal{H}$-SCC PC, there is an algorithm, that runs in time $2^{O(k^2 \log k)} \cdot n^{O(h)}$ and outputs a branch set for $\mathcal{I}$ of size $\emptyset(h) \cdot 2^{O(k^2 \log k)}$ if $\mathcal{I}$ has a nice solution.*

The algorithm of Lemma 14 has two parts: in this section we design a simple algorithm when $q = 1$ by exploiting the structure of a nice solution to identify a vertex that belongs to the shadow of the solution, thereby allowing the applicability of Lemma 13 to find a branch set. The second part, when $q > 1$, is tricky. We design a recursive algorithm for the proof of Lemma 14 when $q > 1$ in Section 3.4.

Lemma 15. *There is an algorithm that, given an instance $\mathcal{I} = (D, \mathcal{S} = (S_1), W, k)$ of R-$\mathcal{H}$-SCC PC, runs in time $2^{O(k^2)} \cdot n^{O(h)}$ and either correctly concludes that there is no nice solution for $\mathcal{I}$ or outputs a solution for $\mathcal{I}$.*

Proof. The algorithm proceeds by checking if there is any graph, say $F \subseteq D$ such that F is isomorphic to some graph in $\mathcal{H}$ and is contained in a single strong component of D. If there is no such graph, then $\emptyset$ is a solution to $\mathcal{I}$. Otherwise, let F be such a graph. Suppose there exists a nice solution, say X, for $\mathcal{I}$. Then from the definition of a nice solution, X either contains some vertex of F, or contains an W-$\{r(F)\}$ separator, in which case $r(F) \in \mathsf{R\text{-}Shadow}(X)$, or contains an $\{v\}$-W separator for some $v \in V(F)$, in which case $v \in \mathsf{F\text{-}Shadow}(X)$. By Lemma 10, there is a v-W and W-v path for every $v \in V(F)$, thus we can call Lemma 13 on vertex v. For each $v \in V(F)$, let Z_v be the set outputted by the algorithm of Lemma 13 when given input $(\mathcal{I}, v)$. Let $Z = V(F) \cup \bigcup_{v \in V(F)} Z_v$. Note that $|Z| \leq h + h(4^k \cdot 2k)$. From the arguments above and Lemma 13, there exists a solution that contains some vertex of Z. The algorithm branches on the set Z, that is, for each $v \in Z$, the algorithm creates an instance $(D - \{v\}, \mathcal{S}, W, k-1)$. The

branching algorithm stops when either $k < 0$ (in which case we conclude that it is a no-instance) or when the there is no subgraph isomorphic to a graph in $\mathcal{H}$ (in which case it is a yes-instance). Since, if there exists a nice solution, then there exists a solution that contains some vertex of Z, the correctness of the branching algorithm follows. For the running time analysis, since the branching factor is at most $h + h(4^k \cdot 2k)$ and the branching depth is at most k, we get the desired running time. $\qquad\square$

3.4 Solving general instances of R-$\mathcal{H}$-SCC PC

We now design a recursive algorithm for Lemma 14 for the case when $q > 1$. To maintain the recursive invariants, we enhance the instance of R-$\mathcal{H}$-SCC PC.

Definition 3 (Extended Instance of R-$\mathcal{H}$-SCC PC). An instance $\mathcal{I}^{\text{ext}} = (D, \mathcal{S} = (S_1, \ldots, S_q), W, k, S, T, Q, N_Q)$ is called an *extended instance of* R-$\mathcal{H}$-SCC PC if the following holds:

1. $(D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$ is an instance of R-$\mathcal{H}$-SCC PC,

2. $S_1 \subseteq S \subseteq V(D) \setminus (W \setminus S_1)$ and $W \setminus S_1 \subseteq T$,

3. either $Q = \emptyset$ or, $S \subseteq Q \subseteq V(D) \setminus T$ such that Q is root-$\mathcal{H}$-free and $N^+(Q)$ is a minimum S-T separator in D, and

4. $N_Q \subseteq N^+(Q)$.

Definition 4 (Solution of an extended instance of R-$\mathcal{H}$-SCC PC). For an extended instance $\mathcal{I}^{\text{ext}} = (D, \mathcal{S} = (S_1, \ldots, S_q), W, k, S, T, Q, N_Q)$ of R-$\mathcal{H}$-SCC PC, $X \subseteq V(D) \setminus W$ is said to be a solution for $\mathcal{I}^{\text{ext}}$ if the following holds.

1. X is a nice solution for $(D, \mathcal{S}, W, k)$,

2. X is an S-T separator,

3. $S \subseteq R_D(S_1, X)$,

4. $N_Q \cap R_D(S, X) = \emptyset$ and,

5. $X \cap (S \cup T) = \emptyset$.

The idea behind extending an instance of R-$\mathcal{H}$-SCC PC in the way defined earlier is to get the situation closer to the applicability of Lemma 11. In fact, as we will see, this will form the base case for our arguments. To be more specific, the sets S, T defined in the definition correspond to the set of vertices that one has guessed to be reachable and unreachable, respectively from S_1 in $D - X$, where X is a solution for the original instance. Thus, any solution for the original instance is an S-T separator. Note that, since X is a solution for $(D, \mathcal{S}, W, k)$, the set S_1 itself is reachable from S_1 and $W \setminus S_1$ is unreachable from S_1 in $D - X$. Therefore we could assume that $S_1 \subseteq S$ and $(W \setminus S_1) \subseteq T$. The set Q in the extended instance is such that $N^+(Q)$ is a minimum S-T separator and Q itself is root-$\mathcal{H}$-free (and hence $\mathcal{H}$-free). The set N_Q is meant to be the subset of $N^+(Q)$ that one has guessed to be unreachable from S in $D - X$. The algorithm aims to slowly "grow" Q using Lemma 1 until we find a subgraph F in D that is isomorphic to some graph in $\mathcal{H}$ and whose root lies in Q (i.e. Q is no longer root-$\mathcal{H}$-free). Then using the fact that a nice solution exists, one can construct the desired branch set by branching of instances with a smaller, appropriately defined, measure.

Observation 4. *If* $(D, \mathcal{S}, W, k)$ *has a nice solution to the problem* R-$\mathcal{H}$-SCC PC, *then* $(D, \mathcal{S}, W, k, S_1, W \setminus S_1, \emptyset, \emptyset)$ *has a solution.*

Observe that, from Observation 4, Lemma 16 implies Lemma 14.

Lemma 16. *Given an extended instance* $\mathcal{I}^{ext} = (D, \mathcal{S} = (S_1, \ldots, S_q), W, k, S, T, Q, N_Q)$ *of* R-$\mathcal{H}$-SCC PC, *there is an algorithm that runs in time* $2^{O(k^2 \log k)} \cdot n^{O(h)}$, *and returns a branch set of* $(D, \mathcal{S}, W, k)$ *of size* $O(h) \cdot 2^{O(k^2 \log k)}$, *if* $\mathcal{I}^{ext}$ *has a solution.*

For the convenience of arguments, we further enhance an extended instance of R-$\mathcal{H}$-SCC PC. The idea behind this is to avoid asking that N_Q has to be unreachable from S in $D - X$ where X is a solution of the extended instance. As we see below, a slight modification to the digraph D and T help us achieve this, thereby easing the arguments used in the final proof.

Definition 5 (Auxiliary Instance of R-$\mathcal{H}$-SCC PC)**.** Given an extended instance $\mathcal{I}^{ext} = (D, \mathcal{S} = (S_1, \ldots, S_q), W, k, S, T, Q, N_Q)$ of R-$\mathcal{H}$-SCC PC, we define an auxiliary instance $\mathcal{I}^{aux} = (D^{aux}, \mathcal{S} = (S_1, \ldots, S_q), W, k, S, T^{aux}, Q, N_Q)$ of R-$\mathcal{H}$-SCC PC as follows: D^{aux} is a supergraph of D that is obtained from D by adding a new vertex t^{aux} in D and adding arcs (u, t^{aux}), for each $u \in N_Q$; $T^{aux} = T \cup \{t^{aux}\}$.

Definition 6 (Solution of an auxiliary instance of R-$\mathcal{H}$-SCC PC)**.** Let $\mathcal{I}^{aux} = (D^{aux}, \mathcal{S} = (S_1, \ldots, S_q), W, k, S, T^{aux}, Q, N_Q)$ be an auxiliary instance of R-$\mathcal{H}$-SCC PC obtained from $\mathcal{I}^{ext} = (D, \mathcal{S} = (S_1, \ldots, S_q), W, k, S, T, Q, N_Q)$, then $X \subseteq V(D) \setminus W$ is said to be a solution for $\mathcal{I}^{aux}$ if the following holds:

1. X is a nice solution for $(D, \mathcal{S}, W, k)$,

2. X is an S-T^{aux} separator in D^{aux},

3. $S \subseteq R_{D^{aux}}(S_1, X)$ and,

4. $X \cap (S \cup T^{aux}) = \emptyset$.

We will now show that a solution to an extended instance of R-$\mathcal{H}$-SCC PC is also a solution for the corresponding auxiliary version.

Lemma 17. *Let* $\mathcal{I}^{aux} = (D^{aux}, \mathcal{S} = (S_1, \ldots, S_q), W, k, S, T^{aux}, Q, N_Q)$ *be an auxiliary instance of* R-$\mathcal{H}$-SCC PC *obtained from* $\mathcal{I}^{ext} = (D, \mathcal{S} = (S_1, \ldots, S_q), W, k, S, T, Q, N_Q)$. *If* X *is a solution for* $\mathcal{I}^{ext}$ *then* X *is also a solution for* $\mathcal{I}^{aux}$.

Proof. Let X be a solution for $\mathcal{I}^{ext}$. Then X is already a nice solution for $(D, \mathcal{S}, W, k)$ and $X \cap (S \cup T^{aux}) = \emptyset$ follows trivially. Since t^{aux} is a vertex with no out-neighbours, there cannot be any path that pass through t^{aux} and have endpoints not equal to t^{aux}, thus there cannot be any $S - T$ paths passing through t^{aux} and no $S_1 - \{v\}$ paths through t^{aux} for any vetex $v \notin R_D(S_1, X)$, it follows that X is an $S - T$ cut in D^{aux} and $S \subseteq R_D(S_1, X) \subseteq R_{D^{aux}}(S_1, X)$. Moreover, since N_Q is unreachable from S in $D - X$, it follows that t^{aux} is unreachable from S in $D - X$ and thus X is a $S - T^{aux}$ separator in D^{aux}. $\qquad \square$

From Lemma 17, Lemma 18 implies Lemma 16.

Lemma 18 (FIND-BRANCH-SET)**.** *Given an auxiliary instance* $\mathcal{I}^{aux} = (D^{aux}, \mathcal{S} = (S_1, \ldots, S_q), W, k, S, T^{aux}, Q, N_Q)$ *of* R-$\mathcal{H}$-SCC PC *where* $q > 1$, *there is an algorithm that runs in time* $2^{O(k^2 \log k)} \cdot n^{O(h)}$ *and returns a branch set of* $(D, \mathcal{S}, W, k)$ *of size* $\emptyset(h) \cdot 2^{O(k^2 \log k)}$ *if* $\mathcal{I}^{aux}$ *has a solution.*

Proof. Let $\mathcal{I} = (D, \mathcal{S}, W, k)$. We will design a recursive algorithm FIND-BRANCH-SET. To analyse the depth of recursion, we associate a measure μ with an instance $\mathcal{I}^{\text{aux}} = (D^{\text{aux}}, \mathcal{S} = (S_1, \ldots, S_q), W, k, S, T^{\text{aux}}, Q, N_Q)$ as $\mu(\mathcal{I}^{\text{ext}}) = k^2 + k - \lambda_{D^{\text{aux}}}(S, T^{\text{aux}})^2 - |N_Q|$. For the sake of convenience, we will denote $\lambda_{D^{\text{aux}}}(S, T^{\text{aux}})$ by $\lambda(\mathcal{I}^{\text{aux}})$. In what follows, we give an exhaustive list of cases, and say what the algorithm outputs in each such case, give a proof of correctness for the same, point out the branching width of the recursive calls and argue that the measure μ drops for each of the instances called in each of the recursive calls.

Base Case: Observe that for any auxiliary instance $\mathcal{I}^{\text{aux}}$, if $\lambda(I^{\text{aux}}) > k$, then $\mathcal{I}_{\text{aux}}$ has no solution. If $k \leq 0$, then check if any strong component of D has a graph isomorphic to some graph in $\mathcal{H}$. If it does, then $\mathcal{I}^{\text{aux}}$ has no solution, otherwise, return $\emptyset$. Another case that is handled as a base case is when either $\mu(\mathcal{I}^{\text{aux}}) \leq 0$ or $|N_Q| = \lambda(\mathcal{I}^{\text{aux}})$. If $\mu(\mathcal{I}^{\text{aux}}) \leq 0$, we first claim that $|N_Q| = |N^+(Q)| = \lambda(\mathcal{I}^{\text{aux}})$. Since $|N^+(Q)| = \lambda(\mathcal{I}^{\text{aux}})$, it is enough to prove that $|N_Q| = \lambda(\mathcal{I}^{\text{aux}})$. Since $N_Q \subseteq N^+(Q)$, we have that $|N_Q| \leq \lambda(\mathcal{I}^{\text{aux}})$. For the sake of contradiction, suppose that $|N_Q| < \lambda(\mathcal{I}^{\text{aux}})$. Since $\mu(\lambda(\mathcal{I}^{\text{aux}})) \leq 0$, we have that $k^2 + k \leq \lambda(\mathcal{I}^{\text{aux}})^2 + \lambda(\mathcal{I}^{\text{aux}})$. This implies that $|N_Q| \geq \lambda(\mathcal{I}^{\text{aux}})$, which is a contradiction. Thus, we have that $|N_Q| = \lambda(\mathcal{I}^{\text{aux}})$. In this case, FIND-BRANCH-SET$(\mathcal{I}^{\text{aux}})$ returns $N^+(Q)$. We now prove that $N^+(Q)$ is indeed a branch set for $\mathcal{I}^{\text{aux}}$. First observe that, in this case $N_Q = N^+(Q)$. From the construction of D^{aux}, there is an arc (u, t^{aux}) for each $u \in N_Q$. Thus, in any solution X of $\mathcal{I}^{\text{aux}}$, $N_Q \cap R_{D^{\text{aux}}}(S, X) = \emptyset$, that is, $N^+(Q) \cap R_{D^{\text{aux}}}(S, X) = \emptyset$. Since any solution X of $\mathcal{I}^{\text{aux}}$ is also a solution for $\mathcal{I}$, we have that there exists a solution X to $\mathcal{I}$, such that $N^+(Q) \cap R_{D^{\text{aux}}}(S, X) = \emptyset$. Then, observe that $\mathcal{I}, Q, N^+(Q), X$ satisfy the properties of Lemma 11, and hence there exists a solution to $\mathcal{I}$ that contains a vertex of $N^+(Q)$. That is, $N^+(Q)$ is a branch set for $\mathcal{I}$. Note that the size of the set outputted in this case is $\lambda(\mathcal{I}^{\text{aux}}) \leq k$. We now proceed to the recursive cases.

Case 1: $[Q = \emptyset]$: The algorithm first computes the unique minimum closest S-T^{aux} separator C. It then checks if there exists $F \subseteq D$ such that F is isomorphic to some graph in $\mathcal{H}$ and $r(F) \in R_D(S, C)$.

Case 1.1: $[\nexists$ **a subgraph** $F \subseteq D$ **such that** F **is isomorphic to a graph in** $\mathcal{H}$ **and** $r(F) \in R_D(S, C)]$: In this case, the algorithm returns FIND-BRANCH-SET$(D^{\text{aux}}, \mathcal{S}, W, k, S, T^{\text{aux}}, R_D(S, C), \emptyset)$.

Correctness: Note that in this case Q is root-$\mathcal{H}$-free and $N^+(Q)$ is a minimum S-T^{aux} separator in D^{aux}. Thus, X is a solution for the auxiliary instance $(D^{\text{aux}}, \mathcal{S}, W, k, S, T^{\text{aux}}, R_D(S, C), \emptyset)$.

Branching width: It is 1.

Measure drop: Let $\mathcal{I}'^{\text{aux}} = (D^{\text{aux}}, \mathcal{S}, W, k, S, T, R_D(S, C), \emptyset)$. Since the branching width is 1, in this case it is enough to prove that this case cannot occur more than n times and the measure does not increase whenever this case arises. Since in the new instance $\mathcal{I}'^{\text{aux}}$, the size of Q has strictly increased, as it was an empty set before, the resulting instance $\mathcal{I}'^{\text{aux}}$ does not fall into this case again (as we will see later that in all the cases the set Q only grows). Since k, N_Q remains the same in both the instances, and $\lambda(\mathcal{I}'^{\text{aux}}) \geq \lambda(\mathcal{I}^{\text{aux}})$ because $T^{\text{aux}} \cup \{r(F)\} \supseteq T^{\text{aux}}$, we conclude that $\mu(\mathcal{I}'^{\text{aux}}) \leq \mu(\mathcal{I}^{\text{aux}})$.

Case 1.2: $[\exists F \subseteq D$ **that is isomorphic to a graph in** $\mathcal{H}$, **and** $r(F) \in Q']$: In this case, the algorithm returns $V(F) \cup$ FIND-BRANCH-SET$(D^{\text{aux}}, \mathcal{S}, W, k, S, T^{\text{aux}} \cup \{r(F)\}, \emptyset, \emptyset) \cup \bigcup_{u \in V(F)}$ PUSHING-ROUTINE-2$(D, \mathcal{S}, W, k, u)$.

Correctness: Let X be a solution for $\mathcal{I}^{\text{aux}}$. Since by definition, X is a nice solution of $\mathcal{I}$, either $V(F) \cap X \neq \emptyset$, or X contains an S_1-$\{r(F)\}$ separator or if it does not satisfy either of the above two conditions, then it must contain a $\{u\}$-S_1 separator, for some $u \in V(F)$. In the first case, since the returned set contains $V(F)$, we are done. In the second case, if X contains an S_1-$\{r(F)\}$ separator and $S \subseteq R_D(S_1, X)$ (from the definition of an auxiliary solution), then X must also contain an S-$\{r(F)\}$ separator. Thus, in this case X must also be a

solution to Find-Branch-Set($D^{\mathrm{aux}}, \mathcal{S}, W, k, S, T^{\mathrm{aux}} \cup \{r(F)\}, \emptyset, \emptyset$). From induction hypothesis, ($D^{\mathrm{aux}}, \mathcal{S}, W, k, S, T^{\mathrm{aux}} \cup \{r(F)\}, \emptyset, \emptyset$) returns a branch set for ($D, \mathcal{S}, W, k$), and we are done. In the last case, if neither of the above two cases hold, then X contains a $\{u\}$-S_1 separator for some $u \in V(F)$. We will prove that u in in the forward shadow of X with respect to W. Suppose, for the sake of contradiction, that there is a path from u to a vertex $v \in W$ in $D - X$. Note that $v \notin S_1$ as X contains a $\{u\} - S_1$ separator. Since $r(F)$ is reachable from S_1 in $D - X$, X is disjoint from $V(F)$, and there is a path from $r(F)$ to u contained in $V(F)$ as F is rooted, it follows that u is reachable from S_1 in $D - X$. Furthermore, there is a path from u to v in $D - X$ and thus $v \in W \setminus S_1$ is reachable from S_1 in $D - X$. This is a contradiction to the fact that X is a solution for $\mathcal{I}$. Therefore, it follows that u is in the forward shadow of X with respect to W. Moreover, since there is an S_1-$\{r(F)\}$ path in D, by rooted-ness there is an S_1-u path and since $S_1 \subseteq W$, there is a W-u path in D. By Lemma 13, Pushing-Routine-2($D, \mathcal{S}, W, k, u$) returns a branch set for $\mathcal{I}$.

Branching width: It is 1.

Measure drop: Let $\mathcal{I}'^{\mathrm{aux}} = (D^{\mathrm{aux}}, \mathcal{S}, W, k, S, T^{\mathrm{aux}} \cup \{r(F)\}, \emptyset, \emptyset)$. Since the branching width is 1 and in the new instance $\mathcal{I}'^{\mathrm{aux}}$ the size of T^{aux} grows, this case does not happen more than n number of times because, if $T^{\mathrm{aux}} \setminus t^{\mathrm{aux}}$ becomes equal to $V(D) \setminus S_1$, then there is a no solution to the problem and we stop. Again, since the branching width is 1, in this case, we only need to show that $\mu(\mathcal{I}'^{\mathrm{aux}}) \le \mu(\mathcal{I}^{\mathrm{aux}})$. Since k, N_Q remains the same in both the instances, and $\lambda(\mathcal{I}'^{\mathrm{aux}}) \ge \lambda(\mathcal{I}^{\mathrm{aux}})$ because $T^{\mathrm{aux}} \cup \{r(F)\} \supseteq T^{\mathrm{aux}}$, we conclude that $\mu(\mathcal{I}'^{\mathrm{aux}}) \le \mu(\mathcal{I}^{\mathrm{aux}})$.

Case 2: $[Q \ne \emptyset]$: The algorithm proceeds by invoking Lemma 1 and either finds a separator C' that tightly covers $N^+(Q)$ in D^{aux} or concludes that $N^+(Q)$ is the furthest minimum S-T^{aux} separator in D^{aux}.

Case 2.1: $[N^+(Q)$ **is the furthest minimum** S-T^{aux} **separator in** $D^{\mathrm{aux}}]$: In this case, the algorithm returns $N^+(Q) \cup \bigcup_{v \in N^+(Q) \setminus N_Q}$ Find-Branch-Set($D^{\mathrm{aux}}, \mathcal{S}, W, k, S \cup \{v\}, T^{\mathrm{aux}}, \emptyset, \emptyset$).

Correctness: Let $\mathcal{I}_v^{\mathrm{aux}} = (D^{\mathrm{aux}}, \mathcal{S}, W, k, S \cup \{v\}, T^{\mathrm{aux}}, \emptyset, \emptyset)$. Let X be a solution for $\mathcal{I}^{\mathrm{aux}}$. Suppose there exists $v \in N^+(Q) \setminus N_Q$ that is reachable from S in $D^{\mathrm{aux}} - X$, then observe that X is also a solution for $\mathcal{I}_v^{\mathrm{aux}}$. Thus, Find-Branch-Set($\mathcal{I}_v^{\mathrm{aux}}$) returns a branch set for $\mathcal{I}$. Now we look at the case when no vertex in $N^+(Q) \setminus N_Q$ is reachable from S in $D^{\mathrm{aux}} - X$. Recall that N_Q cannot be reachable from S in $D^{\mathrm{aux}} - X$. Thus the entirety of $N^+(Q)$ is unreachable from S in $D^{\mathrm{aux}} - X$. In this case, observe that $\mathcal{I}, Q, N^+(Q), X$ satisfy the conditions for Lemma 11, and thus, $N^+(Q)$ is a branch set for $\mathcal{I}$.

Branching width: It is at most $|N^+(Q)| = \lambda(\mathcal{I}^{\mathrm{aux}}) \le k$.

Measure drop: For each $v \in N^+(Q) \setminus N_Q$, we show that $\mu(\mathcal{I}_v^{\mathrm{aux}}) < \mu(\mathcal{I}^{\mathrm{aux}})$. From Lemma 3, the $\lambda(\mathcal{I}_v^{\mathrm{aux}}) \ge \lambda(\mathcal{I}^{\mathrm{aux}}) + 1$. However the size of N_Q in the new instance decreases to 0. Thus, the drop in μ is at least $(k^2 + k - \lambda(\mathcal{I}^{\mathrm{aux}})^2 - |N_Q|) - (k^2 + k - (\lambda(\mathcal{I}^{\mathrm{aux}}) - 1)^2 - 0) = \lambda(\mathcal{I}^{\mathrm{aux}})^2 - |N_Q| - (\lambda(\mathcal{I}^{\mathrm{aux}}) + 1)^2 = 2\lambda(\mathcal{I}^{\mathrm{aux}}) - |N_Q|$. Since $|N_Q| \le \lambda$, the drop is $\ge \lambda + 1$.

Case 2.2: $[N^+(Q)$ **is not the furthest minimum** S-T^{aux} **separator in** $D^{\mathrm{aux}}]$: From Lemma 1, the algorithm first finds a separator C' that tightly covers $N^+(Q)$. Let $Q' = R_D(S, C')$ that is $C' = N(Q')$. It then checks if there exists $F \subseteq D$ such that F is isomorphic to some graph in $\mathcal{H}$ and $r(F) \in Q'$.

Case 2.2.1: $[\nexists$ **a subgraph isomorphic to a graph in** $\mathcal{H}$ **whose root is in** $Q']$: In this case, the algorithm returns Find-Branch-Set($D^{\mathrm{aux}}, \mathcal{S}, W, k, S, T^{\mathrm{aux}}, Q', N_Q$).

Correctness: Let $\mathcal{I}'^{\mathrm{aux}} = (D^{\mathrm{aux}}, \mathcal{S}, W, k, S, T^{\mathrm{aux}}, Q', N_Q)$. From construction, $N(Q')$ is a minimum $S - T^{\mathrm{aux}}$ separator and Q' is root-$\mathcal{H}$-free. Also since $N^+(Q')$ covers $N^+(Q)$ and is an S-T^{aux} separator and for each $v \in N_Q$, (v, t^*) is an arc in D^{aux}, it follows that $N_Q \subseteq N^+(Q')$. Thus, if X is a solution to $\mathcal{I}^{\mathrm{aux}}$, then it is also a solution to $\mathcal{I}'^{\mathrm{aux}}$.

Branching width: It is 1.

Measure drop: Since the branching width is 1 and $Q' \supset Q$, it is enough to prove that the

measure does not increase. This is indeed the case, as k, N_Q remain the same in both the instances. Also, since S and T^{aux} remain the same, $\lambda(\mathcal{I}^{\text{aux}}) = \lambda(\mathcal{I}'^{\text{aux}})$.

Case 2.2.2: $[\exists F \subseteq D$ **that is isomorphic to a graph in** $\mathcal{H}$ **and** $r(F) \in Q']$**:** In this case, the algorithm returns $V(F) \cup \text{Find-Branch-Set}(D^{\text{aux}}, \mathcal{S}, W, k, S \cup (N^+(Q) \setminus N_Q), T^{\text{aux}} \cup \{r(F)\}, \emptyset, \emptyset) \cup \bigcup_{v \in N^+(Q) \setminus N_Q} \text{Find-Branch-Set}(D^{\text{aux}}, \mathcal{S}, W, k, S, T^{\text{aux}}, Q, N_Q \cup \{v\}) \cup \bigcup_{v \in V(F)}$ $\text{Pushing-Routine-2}(D, \mathcal{S}, W, k, u)$.

Correctness: Let X be a solution of $\mathcal{I}^{\text{aux}}$. Since X is a nice solution of $\mathcal{I}$, either $X \cap V(F) \neq \emptyset$ or, X contains an S_1-$\{r(F)\}$ separator or, X contains a $\{v\}$-S_1 separator, for some $v \in V(F)$. In the first case, since $V(F)$ is present in the set returned, we are done. In the second case, since S is reachable from S_1 in $D^{\text{aux}} - X$, it follows that X is an S-$\{T \cup r(F)\}$ separator. If there exists a vertex $v \in N^+(Q) \setminus N_Q$ that is unreachable from S in $D^{\text{aux}} - X$, then observe that X is also a solution for $\text{Find-Branch-Set}(D^{\text{aux}}, \mathcal{S}, W, k, S, T^{\text{aux}}, Q, N_Q \cup \{v\})$. Thus, we are done. Otherwise, $N^+(Q) \setminus N_Q$ is reachable from S in $D^{\text{aux}} - X$. In this case, X is also a solution for $\text{Find-Branch-Set}(D^{\text{aux}}, \mathcal{S}, W, k, S \cup (N^+(Q) \setminus N_Q), T^{\text{aux}} \cup \{r(F)\}, \emptyset, \emptyset)$, and hence, we are done. In the third case, X contains a $\{v\}$-S_1 separator, for some $v \in V(F)$. Using a similar argument as in **Case 1**, it follows that u is in the forward-shadow of X with respect to W and that there is a W-u path in D. Thus, from Lemma 13, $\text{Pushing-Routine-2}(D, \mathcal{S}, W, k, u)$ returns a branch set for $\mathcal{I}$.

Branching width: It is at most $|N^+(Q)| + 1 \leq \lambda(\mathcal{I}^{\text{aux}}) + 1 \leq k + 1$.

Measure drop: Consider the instance $\mathcal{I}'^{\text{aux}} = (D^{\text{aux}}, \mathcal{S}, W, k, S \cup (N^+(Q) \setminus N_Q), T^{\text{aux}} \cup \{r(F)\}, \emptyset, \emptyset)$. We show that $\mu(\mathcal{I}'^{\text{aux}}) < \mu(\mathcal{I}^{\text{aux}})$. From Lemma 4, the minimum $(S \cup (N^+(Q) \setminus N_Q))$-$(T \cup \{r(F), t^{\text{aux}}\})$ separator is of size greater than $\lambda(\mathcal{I}^{\text{aux}})$. Thus, $\lambda(\mathcal{I}'^{\text{aux}}) > \lambda(\mathcal{I}^{\text{aux}})$. However the N_Q (the last variable in the instance) for $\mathcal{I}'^{\text{aux}}$ is an empty set. Therefore μ drops by at least $(k^2 + k - \lambda(\mathcal{I}^{\text{aux}})^2 - |N_Q|) - (k^2 + k - (\lambda(\mathcal{I}^{\text{aux}}) + 1)^2 - 0) = \lambda(\mathcal{I}^{\text{aux}})^2 - |N_Q| - (\lambda(\mathcal{I}^{\text{aux}}) + 1)^2 = 2\lambda(\mathcal{I}^{\text{aux}}) + 1 - |N_Q|$. Since $|N_Q| \leq \lambda(\mathcal{I}^{\text{aux}})$, the drop is at least $\lambda(\mathcal{I}^{\text{aux}}) + 1$. For each $v \in V(F)$, consider the instance $\mathcal{I}_v^{\text{aux}} = (D^{\text{aux}}, \mathcal{S}, W, k, S, T^{\text{aux}}, Q, N_Q \cup \{v\})$. We now show that $\mu(\mathcal{I}_v^{\text{aux}}) < \mu(\mathcal{I}^{\text{aux}})$. Since $|N_Q \cup \{v\}| = |N_Q| + 1$ and $\lambda((I)_v^{\text{aux}}) = \lambda(\mathcal{I}^{\text{aux}})$, we conclude that the measure drops by one in this case.

This concludes the description of the recursive algorithm together with its correctness. To bound the number of nodes in the recursion tree, since the maximum branching width of the recursion tree is at most $k + 1$ and the depth of the recursion tree is at most $\mu(\mathcal{I}^{\text{aux}}) = k^2 + k - \lambda(\mathcal{I}^{\text{aux}})^2 - |N_Q| \leq k^2 + k$, we conclude that the number of nodes in the recursion tree is $(k+1)^{k^2+k} = 2^{O(k^2 \log k)}$. Since the size of the set returned at the leaf nodes of the recursion tree is at most k and at each level of the recursion tree a set of size at most $h + kh + 1$ is added to the set obtained from the recursive calls, we conclude that the size of the set that the algorithm outputs is at most $(h + kh + 1) \cdot 2^{O(k^2 \log k)} = \emptyset(h) \cdot 2^{O(k^2 \log k)}$. $\qquad\square$

Given an instance $(D, \mathcal{S}, W, k)$ of R-$\mathcal{H}$-SCCPC, the algorithm of Lemma 9 creates an extended instance $\mathcal{I}^{\text{ext}} = (D, \mathcal{S}, W, k, S_1, W \setminus S_1, \emptyset, \emptyset)$ of R-$\mathcal{H}$-SCCPC and calls Find-Branch-Set on $\mathcal{I}^{\text{aux}}$, which is an auxiliary instance of $\mathcal{I}^{\text{ext}}$. The proof of Lemma 9 then follows from Lemma 18. As already argued, Lemma 9 implies Theorem 1. This completes the description and proof of our FPT algorithm for R-$\mathcal{H}$-SCC Deletion. Observe that the only place where we incur a $n^{O(h)}$ factor in the running time of this algorithm is for checking whether there exists a subgraph of D isomorphic to a graph in $\mathcal{H}$. If this can be done in time $g(h) \cdot n^{O(1)}$, then our algorithm will run in time $g(k, h) \cdot n^{O(1)}$. In particular, if $\mathcal{H}$ comprises of only the $(d+1)$-out-star, the algorithm will run in time that is FPT in k and d (Theorem 2).

3.5 Proof of Lemma 9, Theorems 1 and 2

Given an instance $(D, \mathcal{S}, W, k)$ of R-$\mathcal{H}$-SCCPC, the algorithm of Lemma 9 creates an extended instance $\mathcal{I}^{\text{ext}} = (D, \mathcal{S}, W, k, S_1, W \setminus S_1, \emptyset, \emptyset)$ of R-$\mathcal{H}$-SCCPC and calls FIND-BRANCH-SET on $\mathcal{I}^{\text{aux}}$, which is an auxiliary instance of $\mathcal{I}^{\text{ext}}$. The proof of Lemma 9 then follows from Lemma 18. As already argued, Lemma 9 implies Theorem 1. This completes the description and the proof of our FPT algorithm for R-$\mathcal{H}$-SCC DELETION. Observe that the only place where we incur a $n^{O(h)}$ factor in the running time of this algorithm is for checking whether there exists a subgraph of D isomorphic to a graph in $\mathcal{H}$. If this can be done in time $g(h) \cdot n^{O(1)}$, then our algorithm will run in time $g(k, h) \cdot n^{O(1)}$. In particular, if $\mathcal{H}$ comprises of only the $(d+1)$-out-star, the algorithm will run in time that is FPT in k and d (Theorem 2).

4 The FPT algorithm for PATH $\mathcal{H}$-SCC DELETION

In this section, we consider the $\mathcal{H}$-SCC deletion problem when $\mathcal{H}$ contains a directed path (the PATH $\mathcal{H}$-SCC problem) and prove Theorem 3.

Theorem 3. P-$\mathcal{H}$-SCC DELETION *can be solved in time* $2^{O(k^3 \log k)} \cdot h^{O(k)} \cdot 2^{O(h^6)} \cdot n^{O(h^3)}$.

The overall picture of the arguments used is summarized here. Observe that what makes the $\mathcal{H}$-SCC problem tricky is the fact that in the problem the goal is to delete a set of vertices thereby excluding the graphs in $\mathcal{H}$ as subgraphs in the *strongly connected components* of the resulting graph. On the other hand, hitting/excluding the graphs of $\mathcal{H}$ as subgraphs in the *whole graph* is relatively easy, as one can find a subgraph of D isomorphic to a digraph in $\mathcal{H}$, and then branch on its vertices to will go to the solution. We call the later problem as the $\mathcal{H}$-HITTING problem.

In what follows, we begin by showing that for every family of digraphs $\mathcal{H}$, there exists another family $\mathcal{H}^*$, such that $\mathcal{H}$-SCC problem is equivalent to the $\mathcal{H}^*$-HITTING problem. The observation seems to be a good news but comes with own set of challenges, viz. the family $\mathcal{H}^*$ might not be a finite family, therefore the simple branching algorithm described above might simply fail. Despite this nature of $\mathcal{H}^*$ in the general case, we show that when $\mathcal{H}$ contains a directed path, then the family $\mathcal{H}^*$, as described above, exhibits certain nice properties (which we elaborate later in the section), that can be exploited to yield the FPT algorithm for the PATH $\mathcal{H}$-SCC problem.

4.1 The Design of $\mathcal{H}^*$

Given $\mathcal{H}$, we will now define the family $\mathcal{H}^*$, such that $\mathcal{H}$-SCC problem reduces to the $\mathcal{H}^*$-HITTING problem. Towards this, given a digraph H, we define a class of strongly connected supergraphs of H, which we call the class of *path completions of H*, denoted by $PC(H)$. Intuitively, this is the class of digraphs obtained by adding paths between the vertices of H to make the resulting graph strongly connected.

Definition 7. Let H be a digraph. Then the *path completions of H*, denoted by $PC(H)$, is a class of strongly connected supergraphs of H, defined as follows. A supergraph $H^* \in PC(H)$ if $H^* = H \cup P_1 \cup \ldots \cup P_\ell$, where each P_i is a directed path with end-points in $V(H)$. Also, the vertex set of the paths P_i are not necessarily disjoint. The collection $\{P_i : i \in [\ell]\}$ is called the *witnessing collection of paths* for H^*. Note that there could be more than one witnessing collection of paths for a digraph.

Note that, for any digraph H, the family $PC(H)$ could be potentially infinite. Thus, finding the collection $PC(H)$ is hard, but, as we will see later, checking if a digraph $H^* \in PC(H)$ is

fairly easy, in the sense that one can check this in time proportional to the size of H^*. We now refine this class $PC(H)$ to get a class that serves our purpose and avoids "redundancy".

Definition 8. Let H be a digraph. The *good path completions of H*, denoted by $GPC(H)$, is a subset of $PC(H)$, such that, for each $H^* \in GPC(H)$, $H^* = H \cup P_1 \cup \ldots \cup P_\ell$ such that the pair of end-points of the paths P_i are distinct. Thus, $\ell \leq |V(H)|^2$. For a family of digraphs $\mathcal{H}$, let $GPC(\mathcal{H}) = \cup_{H \in \mathcal{H}} GPC(H)$.

The key insight of having the above definition is that for any strongly connected supergraph of H, say $\widehat{H}$, there exists $H^* \in GPC(H)$, such that H^* is a subgraph of $\widehat{H}$. Also, the number of paths required to be added to H to make it H^* is bounded as a function of the size of H. The former claim is formalized below.

Lemma 19. *Let H be a digraph and let $\widehat{H} \supseteq H$ such that $\widehat{H}$ is strongly connected. Then there exists $H^* \in GPC(H)$ such that $H^* \subseteq \widehat{H}$.*

Proof. We will construct the digraph H^* iteratively. Begin by initialising $H^* := H$. If H^* is not strongly connected, then identify a pair $(u, v) \in V(H) \times V(H)$ such that there is no path from u to v in H^*. Since $\widehat{H}$ is strongly connected, there exists a u to v path, say $P_{u,v}$, in $\widehat{H}$. Update $H^* := H^* \cup P_{u,v}$.

From the construction above, clearly $H^* = H \cup P_1 \cup \ldots \cup P_\ell$, where the end-points of the paths P_i are in $V(H)$. What remains to prove now is that H^* is strongly connected. Let us split the vertex set of $V(H^*)$ into two parts: $V(H)$ containing the vertices of H, and $V(P)$ containing the vertices in the paths P_i that are not in $V(H)$, that is $V(P) = V(H^*) \setminus V(H)$. From the construction of H^*, it is clear that, for any two vertices $u, v \in V(H)$, there is both a u to v, and v to u path in H^*. Since the endpoints of the paths P_i are in $V(H)$, for every vertex in $V(P)$, there is path to a vertex in $V(H)$ and from a vertex in $V(H)$. Thus, every vertex of $V(P)$ can reach all other vertices of $V(H^*)$. This proves that H^* is strongly connected. $\qquad\square$

The above lemma together with the definitions stated above helps us to build the relation between the $\mathcal{H}$-SCC problem and a corresponding $\mathcal{H}^*$-HITTING problem. We show that the $\mathcal{H}^*$ corresponding to the family $\mathcal{H}$ is $GPC(\mathcal{H})$. This is formalized below.

Lemma 20. *Given a digraph D and an integer k, $X \subseteq V(D)$ is a solution to the instance (D, k) of the problem $\mathcal{H}$-SCC if and only if it is a solution to the instance (D, k) of the $GCP(\mathcal{H})$-HITTING problem.*

Proof. For the forward direction, let X be a solution to the instance (D, k) of $\mathcal{H}$-SCC. Suppose, for the sake of contradiction, that there exists digraphs H, H^*, such that $H \in \mathcal{H}$ and $H^* \in GPC(H)$, and $H^* \subseteq D - X$. Since H^* is strongly connected (from the definition of GPC), there exists a strongly connected component, say C, of $D - X$ such that $H^* \subseteq C$. Since $H \subseteq H^*$ (from the definition of GPC), we conclude that $H \subseteq C$, thereby contradicting that X is a solution for the problem $\mathcal{H}$-SCC.

For the backward direction, suppose that X is a solution to the instance (D, k) of the $GCP(\mathcal{H})$-HITTING problem. For the sake of contradiction, suppose that there exists some strongly connected component, say C, of $D - X$ and some $H \in \mathcal{H}$ and $H \subseteq C$. From Lemma 19, there exists a subgraph $H^* \in GPC(H)$ such that $H \subseteq H^* \subseteq C$. Since $C \subseteq D - X$ and $H^* \subseteq C$, we conclude that $H^* \subseteq D - X$, thereby contradicting that X is a solution for the $GPC(\mathcal{H})$-HITTING problem. $\qquad\square$

4.2 Discovering the structure of $GPC(\mathcal{H})$

As discussed earlier, even though for hitting problems it is relatively easier to design FPT algorithms, we cannot exploit Lemma 20 forthright as the family $GPC(\mathcal{H})$ could contain digraphs of very large size and hence could be potentially infinite. Next, we split the family $GPC(\mathcal{H})$ into two parts such that one is a finite collection of digraphs of bounded size, and hence, exploitable by means of a branching algorithm, while the other part, which could be a potentially infinite collection, has a very special exploitable structure. Towards formalizing the above intuition, let $P \in \mathcal{H}$ be some directed path of length p. Such a path exists because we work with the PATH $\mathcal{H}$-SCC problem. Let $GPC(\mathcal{H}) = \mathcal{H}_p^+ \uplus \mathcal{H}_p^-$, such that for all the digraphs in $\mathcal{H}_p^-$ there exists some witnessing collection of paths where all the witnessing paths have length at most $p-1$. Then, $\mathcal{H}_p^+ = GPC(\mathcal{H}) \setminus \mathcal{H}_p^-$. Recall that $h = \max_{H \in \mathcal{H}} |V(H)|$.

4.2.1 The finite sub-collection: $\mathcal{H}_p^-$

Lemma 21 concludes that the family $\mathcal{H}_p^-$ is finite.

Lemma 21. *If $H^* \in \mathcal{H}_p^-$, then $|V(H^*)| \leq h + (p-1)h^2$.*

Proof. Consider any $H^* \in \mathcal{H}_p^-$. Since, $\mathcal{H}_p^- \subseteq GPC(\mathcal{H})$, $H^* \in GPC(H)$, for some $H \in \mathcal{H}$. Also, since $H^* \in \mathcal{H}_p^-$, there exists a witnessing collection of paths, say $P_1, \ldots, P_\ell$ such that $H^* = H \cup P_1 \cup \ldots \cup P_\ell$ and the length of each P_i is at most $p-1$. Also, from the definition of GPC, $\ell \leq |V(H)|^2 \leq h^2$. Thus, $|V(H^*)| \leq h + (p-1)h^2$. $\square$

From Lemma 21 one can derive an easy algorithm for computing the family $\mathcal{H}_p^-$.

Lemma 22. *Given the family of digraphs $\mathcal{H}$, the family $\mathcal{H}_p^-$ can be computed in $2^{O(h^6)}$ time.*

Proof. From Lemma 21, note that $\mathcal{H}_p^-$ contains only digraphs of size at most $h + (p-1)h^2$ which are in $GPC(\mathcal{H})$. Thus, to enumerate the family $\mathcal{H}_p^-$ it is enough to enumerate all digraphs of size at most $h + (p-1)h^2$, and then for each of them check whether it is in $GPC(\mathcal{H})$. In order to check if a digraph H^* is in $GPC(\mathcal{H})$, first check whether H^* is strongly connected followed by guessing the partition of the digraph H^* into at most $h^2 + 1$ parts, say $H \cup P_1 \cup \ldots \cup P_\ell$, where $|V(H)| \leq h$ and $\ell \leq |V(H)|^2$, and checking if $H \in \mathcal{H}$ and the paths P_i are of size at most $p-1$ and have distinct pair of end-points in $V(H)$.

Since the number of digraphs on at most $h + (p-1)h^2$ vertices is at most $2^{(h+(p-1)h^2)^2}$ and $p \leq h$, and each of the steps described above takes time at most $2^{O(h^6)}$, the running time follows. $\square$

4.2.2 Making the instance $\mathcal{H}^-$-free

Here, we design a branching algorithm that takes an instance (D, k) of $\mathcal{H}$-SCC and returns an equivalent instance (D', k') of $\mathcal{H}$-SCC such that D' has no digraph in $\mathcal{H}_p^-$ as a subgraph. We call such a digraph $\mathcal{H}_p^-$-free. Also, $k' \leq k$.

Lemma 23. *Let (D, k) be an instance of $\mathcal{H}$-SCC. In time $2^{O(h^6)} \cdot h^{O(k)} \cdot n^{O(h^3)}$, one can either conclude that (D, k) is a no-instance of $\mathcal{H}$-SCC or output at most h^k instances $\{(D_1, k_1), \ldots, (D_q, k_q)\}$ of $\mathcal{H}$-SCC such that for each $i \in [q]$, D_i is $\mathcal{H}_p^-$-free and $k_i \leq k$, and (D, k) is a yes-instance if and only if there exists $i \in [q]$ such that (D_i, k_i) is a yes-instance.*

Proof. Let (D, k) be an instance of $\mathcal{H}$-SCC. Compute the family $\mathcal{H}_p^-$ using Lemma 22. Suppose D contains a subgraph, say F isomorphic to some graph in $\mathcal{H}_p^-$. Since $\mathcal{H}_p^- \subseteq GPC(\mathcal{H})$, from Lemma 20, there exists a solution X of the instance (D, k) of $\mathcal{H}$-SCC such that X contains

some vertex of F. For each $v \in V(F)$, the algorithm branches in the following instances: $(D - \{v\}, k-1)$. Since there exists a solution containing some vertex of v, (D, k) is a yes-instance if and only if at least one of $(D - \{v\}, k-1)$, for $v \in V(F)$ is a yes-instance. The branching algorithm stops when $k \leq 0$ or when the resulting digraph has no subgraph isomorphic to a digraph in $\mathcal{H}_p^-$. When $k \leq 0$, if the resulting digraph has a subgraph isomorphic to a digraph in $\mathcal{H}_p^-$, then report that (D, k) is a no-instance of $\mathcal{H}$-SCC. This completes the description of the algorithm. For the running time analysis, since the size of the digraphs in $\mathcal{H}_p^-$ is at most $h + ph^2$, $p \leq h$ and there are at most $2^{(h+(p-1)h^2)^2}$ graphs in $\mathcal{H}_p^-$s, we can check whether there exists a subgraph F isomorphic to a graph in $\mathcal{H}_p^-$ in time $2^{(h+(p-1)h^2)^2} \cdot n^{O(h^3)}$. Since the branching algorithm stops when $k \leq 0$, we get the following recurrence: $T(k) \leq \sum_{i \in [h]} T(k-1)$, $T(0) = 1$, where $T(k)$ denotes the number of leaves in the branching tree rooted at the instance with budget parameter k. By induction, one can prove that $T(k) \leq h^k$. This yields the desired running time. $\qquad \square$

Henceforth, we assume that the instance (D, k) of $\mathcal{H}$-SCC is such that D is $\mathcal{H}_p^-$-free.

4.2.3 The structure of the potentially infinite sub-collection: $\mathcal{H}_p^+$

Recall that $\mathcal{H}_p^+$ is a collection of digraphs in $GPC(\mathcal{H})$ which have a witnessing collection of paths where at least one path has length strictly more than p.

Lemma 24. *For each $H^* \in \mathcal{H}_p^+$, there exists a subgraph $H' \subseteq H^*$ such that $H' \in GPC(P)$. (Recall P is a directed path in $\mathcal{H}$ that we fixed.)*

Proof. Since $H^* \in \mathcal{H}_p^+$, let H^* be a path-completion of $H \in \mathcal{H}$ where one of the witnessing paths has length at least p. That is, $H^* = H \cup P_1 \cup \ldots \cup P_\ell$, where there exists $i \in [\ell]$ such that $|V(P_i)| \geq p$. Without loss of generality, let $i = 1$. Since the length of P is p, P is a subpath of P_1. Let P be a path from u to v in H^*. Since H^* is strongly connected, there exists another path say P' from v to u. Then $P^* = P \cup P'$ is a closed walk in H^*. Observe that P^* is strongly connected. Also by construction, $P^* \in GPC(P)$ (with P' being the witnessing path). Since $P^* \subseteq H^*$, we are done. $\qquad \square$

Combining Lemmas 24 and 20 we get the following lemma.

Lemma 25. *Let (D, k) be an instance of PATH $\mathcal{H}$-SCC such that $P \in \mathcal{H}$ is a directed path of length p and D is $\mathcal{H}_p^-$-free. Then, (D, k) is a yes-instance of PATH $\mathcal{H}$-SCC if and only if it is a yes-instance of $GPC(P)$-HITTING.*

Proof. Recall that $GCP(\mathcal{H}) = \mathcal{H}_p^- \uplus \mathcal{H}_p^+$. For the forward direction, let X be a solution to the instance (D, k) of PATH $\mathcal{H}$-SCC. Since $P \in \mathcal{H}$, from Lemma 20, X is also a solution to the instance (D, k) of $GPC(P)$-HITTING. For the backward direction, let X be a solution to the instance (D, k) of $GPC(P)$-HITTING. We first prove that X is also a solution of $\mathcal{H}_p^+$-HITTING. This follows from Lemma 24. Since X is a solution of $\mathcal{H}_p^+$-HITTING, and D is $\mathcal{H}_p^-$-free, X is also a solution of $\mathcal{H}$-SCC from Lemma 20. This proves the lemma. $\qquad \square$

Combining Lemmas 25 and 20, we get the following lemma.

Lemma 26. *Let D be a $\mathcal{H}_p^-$-free digraph. Then, (D, k) is a yes-instance of $\mathcal{H}$-SCC if and only if it is a yes-instance of $\{P\}$-SCC.*

Proof. From Lemma 26, (D, k) is a yes-instance of $\mathcal{H}$-SCC if and only if it is a yes-instance of $GPC(P)$-HITTING. Also, from Lemma 20, (D, k) is a yes-instance of $GPC(P)$-HITTING if and only if it is a yes-instance of $\{P\}$-SCC. Combining both the statements above, we prove the lemma. $\qquad \square$

4.3 Proof of Theorem 3

Let (D, k) be an instance of PATH $\mathcal{H}$-SCC and let $P \in \mathcal{H}$ be a directed path of length p. Let $h = \max_{H \in \mathcal{H}} |V(H)|$. From Lemma 23, in time $2^{O(h^6)} \cdot h^{O(k)} \cdot n^{O(h^3)}$, we get a set of instances $\{(D_1, k_1), \ldots, (D_q, k_q)\}$, such that for each $i \in [q]$, D_i is $\mathcal{H}_p^-$-free and $k_i \leq k$, and (D, k) is a yes-instance if and only if for some $i \in [q]$, (D_i, k_i) is a yes-instance. From Lemma 26, we conclude that it is enough to solve the $\{P\}$-SCC problem on these instances to obtain the solution. Since P is a rooted digraph, from Theorem 1, the problem can further be solved in $2^{O(k^3 \log k)} \cdot n^{O(p)}$. Thus, we get an algorithm with running time $2^{O(k^3 \log k)} \cdot h^{O(k)} \cdot 2^{O(h^6)} \cdot n^{O(h^3)}$.

5 Faster FPT algorithms

5.1 Faster FPT algorithm for 1-OUT-REGULAR DELETION

In this section, we give an algorithm for ROOTED $\mathcal{H}$-SCC DELETION. that runs in time $O^*(2^{O(k \log k)})$ when $\mathcal{H}$ contains only the out-directed 2-star, that is we prove Theorem 4.

Theorem 4. 1-OUT-REGULAR VERTEX DELETION *can be solved in time* $2^{O(k \log k)} \cdot n^{O(1)}$.

In the following (Definition 9, Observation 5, Lemma 28- 30), fix $\tau = (D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$ to be an instance of R-$\mathcal{H}$-SCC PCC. Recall that a solution for τ is a set $X \subseteq V(D) \backslash W$ of size at most k that intersects all S_i-S_j paths in D for every $j > i$ such that every non-trivial strongly connected component of $D - X$ is $\mathcal{H}$-free and hence, is a cycle.

We have the following specialization of Definition 2 to our current choice of $\mathcal{H}$.

Definition 9. A solution X for the instance τ is said to be *nice* if for every triple $u, v, w \in V(D)$ such that $v, w \in N^+(u)$ and for every $i \in [q]$, one of the following holds.

1. X intersects $\{u, v, w\}$.

2. $u \notin R(S_i, X)$.

3. There is no v-S_i path in $D - X$ or no w-S_i path in $D - X$.

Using Lemma 8, Theorem 4 can be obtained as a consequence of the following lemma.

Lemma 27. *There is an algorithm that, given* τ, *runs in time* $2^{O(k \log k)} \cdot n^{O(1)}$ *and either returns a solution for* (D, k) *or correctly concludes that there is no nice solution for* τ.

The rest of Section 5.1 is therefore devoted to proving Lemma 27. We assume without loss of generality that for every $v \in V(D)$, either v lies in a strongly connected component of D that intersects S_1 and contains at least one vertex of out-degree at least 2 or v can reach $W \setminus S_1$ in D. This can be ensured by a straightforward preprocessing routine that computes the strongly connected components of D and deletes vertices that do not satisfy these properties. The correctness of this step follows from the fact that the deleted vertices do not participate in minimal solutions for the given instance and moreover, the non-existence of a nice solution in the reduced instance is not affected by adding back the deleted vertices. We begin with the following simple observation regarding graphs with maximum out-degree 1.

Observation 5. *Let* $R \subseteq V(D)$ *be such that every vertex in* R *has out-degree at most 1 in* D *and let* $Z \subseteq N^+[R]$. *Then, each vertex of* R *can reach at most one vertex of* Z *via paths whose internal vertices (if there are any) are contained in* $R \setminus Z$.

Proof. If this were not true, then there would be a vertex in R with at least two out-neighbours in D, a contradiction to the premise. $\qquad\square$

In particular, the above observation implies that each vertex of R can reach at most one vertex of $N^+(R)$ via paths whose internal vertices are contained in R. Motivated by Observation 5, we define the following notation.

Definition 10. Let $S_1 \subseteq R \subseteq V(D)$ be such that every vertex in R (i) is reachable from S_1 in $D[R]$ and (ii) has out-degree at most 1 in D. For every $s \in S_1$, we define $close(s, R) = (R \setminus S_1) \cap N^-(s)$. The notation is extended to subsets of S_1 in a natural way.

As a consequence of Observation 5, we have that each $s \in S_1$ can reach at most one vertex of $close(S_1, R)$ via a path internally vertex-disjoint from $close(S_1, R)$ (and hence also disjoint from S_1). Therefore, $|close(S_1, R)| \leq |S_1|$.

Fix a set R satisfying the premise of this observation. For every $v \in R \setminus S_1$ that can reach a vertex $s \in S_1$ (which must then be unique) via a path contained in R and internally vertex-disjoint from S_1, we denote by $\partial(v)$ the singleton set containing the unique vertex of $close(s, R)$ that lies on this v-s path in $D[R]$. For every $v \in R \setminus S_1$ that can reach a vertex of $N^+(R)$ via a path internally vertex-disjoint from $N^+(R) \cup S_1$, we denote by $\partial(v)$ the singleton set containing this unique vertex of $N^+(R)$. For every other $v \in R \setminus S_1$, it must be the case that v cannot reach reach a vertex of $N^+(R)$ via a path internally vertex-disjoint from $N^+(R)$ and so, we set $\partial(v) = \emptyset$.

In other words, for every $v \in R \setminus S_1$, we define $\partial(v)$ as follows. We consider the unique maximal path contained in $D[R]$ that starts at v and is disjoint from S_1. Suppose that this path terminates at the vertex w. If $N^+(w)$ is empty or only comprises v, then $\partial(v)$ is defined as $\emptyset$. Otherwise, $\partial(v) = N^+(w)$.

For a set $R' \subseteq R$, we denote by $\partial(R')$ the set $\bigcup_{v \in R'} \partial(v)$. Notice that for every $R' \subseteq R$, $\partial(R') \cap R \subseteq close(S_1, R)$ and $|\partial(R')| \leq |R'|$.

The following lemma forms the crux of the correctness of our main algorithm. The lemma identifies a pair of vertex subsets in the graph such that if there is a certain kind of nice solution for τ, then, we may assume that for either of these sets, the intersection of the nice solution with the set is one of only a bounded (in k) number of possibilities.

Lemma 28. *Let $T \supseteq W \setminus S_1$, $\mathcal{L} \subseteq V(D) \setminus T \cup \{S_1\}$ and let C be a minimal S_1-T separator in D that is disjoint from $\mathcal{L}$. Let $R = R(S_1, C)$. Suppose that every vertex in R has out-degree at most 1 in D and suppose that there is a nice solution X for τ that is an S_1-T separator and an $\mathcal{L}$-$T \cup \{S_1\}$ separator. Then the following statements hold:*

1. *There is a nice solution X' for τ that is an S_1-T separator, an $\mathcal{L}$-$T \cup \{S_1\}$ separator and moreover, $X' \cap R \subseteq close(S_1, R)$.*

2. *There is a nice solution X' for τ that is an S_1-T separator, an $\mathcal{L}$-$T \cup \{S_1\}$ separator and moreover, $X' \cap R_{max}(\mathcal{L}, T \cup \{S_1\}) = \emptyset$.*

3. *If $c \in V(D)$ is reachable from S_1 and can reach S_1 in $D - X$, then X is also a nice solution for the instance $\tau' = (D, \mathcal{S} = (S_1 \cup \{c\}, \ldots, S_q), W, k)$ that is an $S_1 \cup \{c\}$-T separator and an $\mathcal{L}$-$T \cup \{S_1\} \cup \{c\}$ separator.*

Proof. We begin by observing that R satisfies the properties in the premise of Observation 5 and Definition 10.

Consider the first statement. Suppose that $X_C = X \cap R$ is non-empty. We claim that $X' = (X \setminus X_C) \cup \partial(X_C)$ is also a nice solution for this instance that is an S_1-T separator and an $\mathcal{L}$-$T \cup \{S_1\}$ separator. Since $\partial(X_C)$ is no larger than X_C, it follows that $|X'| \leq |X|$.

We now argue that X' is a nice solution, an S_1-T separator and an $\mathcal{L}$-$T \cup \{S_1\}$ separator. If not, then it must be the case that either (i) there is a closed walk $\mathcal{W}$ in $D - X'$ that induces a subgraph containing a vertex of out-degree at least 2 or (ii) there is an S_i-S_j path $\mathcal{W}$ in $D - X'$, where $i < j$ (implying that X' is not a solution for τ) or (iii) there is an S_1-T path $\mathcal{W}$ in $D - X'$ or (iv) there is a $\mathcal{L}$-$T \cup \{S_1\}$ path $\mathcal{W}$ in $D - X'$ or (v) there is an $i \in [q]$ and a triple $u, v, w \in V(D)$ with $v, w \in N^+(u)$ such that X' is disjoint from $\{u, v, w\}$ and $D - X'$ contains an S_i-u path $\mathcal{W}_u$, a v-S_i path $\mathcal{W}_v$ and a w-S_i path $\mathcal{W}_w$, implying that X' is not a *nice* solution for τ (see Definition 9).

Observe that in each of the cases (i) to (iii), $\mathcal{W}$ must contain some $x \in X_C$ and some $y \in V(D) \setminus R$ such that $\mathcal{W}$ contains an x-y subwalk. Therefore, we conclude that in each of these cases, $\mathcal{W}$ must intersect $\partial(x)$ for some $x \in X_C$, a contradiction since $\partial(x) \in X'$.

We now consider Case (iv). In this case as well, $\mathcal{W}$ must intersect X_C at a vertex x. Moreover, if $\mathcal{W}$ contains an x-y walk for some $y \in V(D) \setminus R$, then the same argument as above implies a contradiction. Hence, we may assume that $\mathcal{W}$ is an $\mathcal{L}$-S_1 path and moreover, the subpath of $\mathcal{W}$ from x to S_1 is contained in R. However, the unique vertex of $N^-(S_1)$ that comprises the set $\partial(x)$, must also be contained in this subpath, a contradiction since $\partial(x) \subseteq X'$.

We now consider Case (v). By the premise of the lemma and the fact that u has out-degree at least 2, we have that $u \notin R$. If u is not reachable from S_i in $D - X$, then we have a contradiction along the same lines as that used before. That is, we have a path $\mathcal{W}$ in $D - X'$ from some $x \in X_C$ to some $y \in V(D) \setminus R$. Moreover, the same argument also implies that $i = 1$ (since $R \subseteq V(D) \setminus T$ and $T \supseteq W \setminus S_1$). That is, $D - X'$ contains an S_1-u path $\mathcal{W}_u$, a v-S_1 path $\mathcal{W}_v$ and a w-S_1 path $\mathcal{W}_w$. We have already concluded that $D - X$ also contains an S_1-u path. Without loss of generality, suppose that $D - X$ does not contain the path $\mathcal{W}_v$. Let x be the last vertex of X_C that lies on this path when traversing it from v to S_1. Then, the subpath of $\mathcal{W}_v$ from x to S_1 also intersects $\partial(x)$, a contradiction since $\partial(x) \subseteq X'$. This completes the proof of the first statement.

We now prove the second statement. Let $\hat{R} = R_{max}(\mathcal{L}, T \cup \{S_1\})$ and $\hat{C} = C_{max}(\mathcal{L}, T \cup \{S_1\})$. Suppose that $X_{\hat{C}} = X \cap \hat{R}$ is non-empty. Furthermore, let $J = \hat{C} \setminus R(\mathcal{L}, X)$. That is, J comprises those vertices of $\hat{C}$ that are *not* reachable from $\mathcal{L}$ after deleting X. Since $\hat{C}$ is a minimum $\mathcal{L}$-$T \cup \{S_1\}$ separator, it must be the case that $|X_{\hat{C}}| \geq |J|$. This bound on $|J|$ can be immediately inferred from the existence of a set of $|\hat{C}|$ pairwise internally vertex disjoint $\mathcal{L}$-$T \cup \{S_1\}$ paths.

We claim that $X' = (X \setminus X_{\hat{C}}) \cup J$ is also a nice solution for this instance that is an S_1-T separator and an $\mathcal{L}$-$T \cup \{S_1\}$ separator. Since $|X_{\hat{C}}| \geq |J|$, it follows that $|X'| \leq |X|$.

It remains to be argued that X' is a nice solution, an S_1-T separator and an $\mathcal{L}$-$T \cup \{S_1\}$ separator. If this were not the case, then one of the five cases (i)–(v) enumerated earlier (see proof of the first statement of this lemma) must hold in $D - X'$. In each of these cases, we infer the presence of an x-y walk in $D - X'$ where $x \in X_{\hat{C}}$ and $y \in T \cup \{S_1\}$. But this implies the presence of an x'-y walk in $D - X$ where $x' \in \hat{C} \setminus J$. However, by the definition of J, we have that $\hat{C} \setminus J \subseteq R(\mathcal{L}, X)$. This is a contradiction to X being an $\mathcal{L}$-$T \cup \{S_1\}$ separator. This completes the argument for the second statement.

We now consider the third statement. If X is not a solution for the tuple τ', then it must be the case that there is a c-d path in $D - X$ for some $d \in \bigcup_{i \in [q] \setminus \{1\}} S_i$. Since c is reachable from S_1 in $D - X$, this implies an S_1-d path, a contradiction to X being a solution for τ. On the other hand, suppose that X is not a *nice* solution for τ'. Then, there is an $i \in [q]$ and a triple $u, v, w \in V(D)$ with $v, w \in N^+(u)$ such that X is disjoint from $\{u, v, w\}$ and $D - X$ contains an $S_1 \cup \{c\}$-u path $\mathcal{W}_u$, a v-$S_1 \cup \{c\}$ path $\mathcal{W}_v$ and a w-$S_1 \cup \{c\}$ path $\mathcal{W}_w$. Since c is both reachable from S_1 and can reach S_1 in $D - X$, we conclude that $D - X$ contains an S_1-u path, a v-S_1 path and a w-S_1 path. This contradicts the premise that X is a nice solution for τ. This completes the proof of the lemma. $\qquad \square$

We now provide a subroutine that computes a set of vertices upon which our main algorithm will be able to branch exhaustively while strictly making progress in each branch.

Lemma 29. *Let $T \supseteq W \setminus S_1$, There is an algorithm that, given τ and T, runs in polynomial time and performs one of the following operations:*

1. *Outputs a $u_1 \in R_{min}(S_1, T)$ and $u_2, u_3 \in N_D^+(u)$.*

2. *Correctly concludes that $R_{max}(S_1, T)$ has no vertex with out-degree at least 2 in D.*

3. *Outputs a minimum S_1-T separator C such that $R(S_1, C)$ has no vertex with out-degree at least 2 in D and moreover, for some $u_1 \in C$, there exist $u_2, u_3 \in N_D^+(u_1) \setminus T$.*

Proof. We first compute $R_{min}(S_1, T)$ and $R_{max}(S_1, T)$ and check whether there exist (i) $u_1 \in R_{min}(S_1, T)$, $u_2, u_3 \in N_D^+(u)$ or (ii) $u \in R_{max}(S_1, T)$ such that u has out-degree at least 2 in D. If the answer to (i) is affirmative, then we return u_1, u_2, u_3. Similarly, if the answer to (ii) is negative, then we return the same.

We now greedily compute a minimum S_1-T separator C such that $R(S_1, C)$ has no vertex with out-degree at least 2 in D and $R(S_1, C)$ is maximal subject to the out-degree constraint on the vertices contained within. Recall that we are in the case where $C \neq R_{max}(S_1, T)$. Therefore, $C_{max}(S_1, T)$ covers C. But this implies that there is some $u_1 \in C \cap R_{max}(S_1, T)$. Notice that u_1 has no out-neighbours in T.

Therefore, it suffices to prove that u_1 has out-degree at least 2 in D. Suppose to the contrary that u_1 has at most one out-neighbour in D. Since C is a minimal S_1-T separator, it follows that u_1 has at least one out-neighbour in D. Therefore, we have that u_1 has exactly one out-neighbour in D, call it u_2. We now observe that $C' = (C \setminus \{u_1\}) \cup \{u_2\}$ is also a minimum S_1-T separator, $R(S_1, C') \supset R(S_1, C)$ and every vertex in $R(S_1, C')$ has out-degree at most 1 in D. This contradicts our choice of C, completing the proof of the lemma. $\square$

We are now ready to present the main algorithm of this section. Specifically, we obtain Lemma 27 as a consequence of the following lemma.

Lemma 30. *Let $T \supseteq W \setminus S_1, \mathcal{L} \subseteq V(D) \setminus T \cup \{S_1\}$. There is an algorithm that, given the tuple $(\tau, T, \mathcal{L})$, runs in time $2^{O(k \log k)} \cdot n^{O(1)}$ and either returns a solution for (D, k) or correctly concludes that there is no nice solution for τ that is also an $\mathcal{L}$-$T \cup \{S_1\}$ separator and an S_1-T separator.*

Proof. We begin by checking whether $k < 0$ or $\lambda(\mathcal{L}, T \cup \{S_1\}) > k$ or $\lambda(S_1, T) > k$ or there is a strongly connected component of $D[W]$ that contains a vertex of out-degree at least 2. If any of these checks return an affirmative answer, then we return NO and terminate. Moreover, if $k = 0$ and $(D, 0)$ is a no-instance of 1-Out-Regular SCC Deletion then we return NO and terminate. If $k \geq 0$ and $(D, 0)$ is a yes-instance of 1-Out-Regular SCC Deletion then we return $\emptyset$ and terminate. Henceforth, we assume that $k \geq 1$, $\lambda(\mathcal{L}, T \cup \{S_1\}) \leq k$, $\lambda(S_1, T) \leq k$ and every strongly connected component of $D[W]$ has maximum out-degree at most 1. To provide an intuitive description of our algorithm, we fix a hypothetical nice solution $X^\star$ for τ that is also an $\mathcal{L}$-$T \cup \{S_1\}$ separator and an S_1-T separator (if one exists).

Case I: $\mathcal{L} \neq \emptyset$. If $\mathcal{L}$ is non-empty and $\lambda(\mathcal{L}, T \cup \{S_1\}) = 0$, then we set $\mathcal{L} = \emptyset$ and recurse.

Otherwise, we compute $C_{max} = C_{max}(\mathcal{L}, T \cup \{S_1\})$. We now branch by picking a vertex in C_{max} and either guessing it to be in $X^\star$ or guessing that it is reachable from $\mathcal{L}$ in $D - X^\star$. The correctness of this branching follows from the fact that there is always a nice solution of the required type that is disjoint from $R_{max}(\mathcal{L}, T \cup \{S_1\})$ (Lemma 28 (2)).

23

Formally, we do the following. We pick a vertex $c \in C_{max}$ and recurse on the tuple $(\tau_c, T, \mathcal{L})$ where, $\tau_c = (D - \{c\}, \mathcal{S}, W, k - 1)$. If this call returns a set Z, then we return the set $Z \cup \{c\}$. Otherwise, we recurse on the tuple $(\tau, T, \mathcal{L} \cup \{c\})$ and return its output.

Analysis. Observe that in the first call, the budget k drops by 1 (with $\lambda(\mathcal{L}, T \cup \{S_1\})$ and $\lambda(S_1, T)$ dropping by at most 1) while in the second call, $\lambda(\mathcal{L}, T \cup \{S_1\})$ increases by at least 1 (Lemma 3).

Case II: $\mathcal{L} = \emptyset$. We now describe the algorithm when $\mathcal{L}$ is empty.

Case II.(a): $T = \emptyset$. This also implies that $q = 1$, i.e., $S_1 = W$. Therefore, we may assume that every strongly connected component of D intersects S_1 and moreover, contains at least one vertex with at least 2 out-neighbours in the same strongly connected component. If the instance is not already solved and is a yes-instance, then there must exist vertices u_1, u_2, u_3 such that $u_2, u_3 \in N^+(u_1)$ and $\{u_1, u_2, u_3\} \setminus S_1 \neq \emptyset$. If such a triple of vertices do not exist, then we return NO as a nice solution for this tuple cannot exist (see Definition 9). Otherwise, we branch by guessing one of $\{u_1, u_2, u_3\}$ to be contained in $X^\star$ (subject to disjointness of the vertex from S_1) or by guessing one of $\{u_1, u_2, u_3\}$ to be unreachable from S_1 in $D - X^\star$ or by guessing that they are all reachable from S_1 in $D - X^\star$, in which case we guess that at least one of $\{u_1, u_2, u_3\}$ cannot reach $T \cup \{S_1\}$ in $D - X^\star$ (also subject to disjointness from S_1). The correctness of the branching follows from Definition 9 and the fact that $X^\star$ is a nice solution for τ.

Formally, we recurse on the tuples: $\mathcal{C}_1 = (\tau_1, \emptyset, \emptyset)$, $\mathcal{C}_2 = (\tau_2, \emptyset, \emptyset)$, $\mathcal{C}_3 = (\tau_3, \emptyset, \emptyset)$, $\mathcal{C}_4 = (\tau_4, \{u_1\}, \emptyset)$, $\mathcal{C}_5 = (\tau_5, \{u_2\}, \emptyset)$, $\mathcal{C}_6 = (\tau_6, \{u_3\}, \emptyset)$, $\mathcal{C}_7 = (\tau, \emptyset, \{u_1\})$, $\mathcal{C}_8 = (\tau, \emptyset, \{u_2\})$, $\mathcal{C}_9 = (\tau, \emptyset, \{u_3\})$ defined as follows. For each $i \in [3]$, $\tau_i = (D - \{u_i\}, \mathcal{S}, W, k - 1)$ if $u_i \notin S_1$ and τ_i is a trivial no-instance otherwise. Notice that in the correct branch (say, Branch i), $X^\star \setminus \{u_i\}$ is a nice solution for τ_i.

Similarly, $\tau_{3+i} = (D, (S_1, \{u_i\}), S \cup \{u_i\}, k)$ if $u_i \notin S_1$ and τ_{3+i} is a trivial no-instance otherwise. Notice that in the correct branch (say, Branch $3 + i$), $X^\star$ is a nice solution for τ_{3+i} that is an S_1-u_i separator. Finally, for each $i \in [3]$, $\tau_{6+i} = \tau$ if $u_i \notin S$ and τ_{6+i} is a trivial no-instance otherwise. Let the respective outputs be denoted by $Z_1, \ldots, Z_9$. We return NO if all calls return NO. If any of the calls $\{\mathcal{C}_i \mid i \in [3]\}$ returns a set Z_i, then we return $Z_i \cup \{u_i\}$ (i is chosen to be the least such value) and terminate. If any of the calls $\{\mathcal{C}_{3+i} \mid i \in [6]\}$ return a set Z_i, then we return the same set (with i chosen to be the least such value) and terminate.

Analysis. Observe that in the first three calls, the budget k drops by 1 (with no change in $\lambda(\mathcal{L}, T \cup \{S_1\})$ and $\lambda(S_1, T)$) while in the next three calls, $\lambda(S_1, T)$ increases by at least 1 and in the final three calls, $\lambda(\mathcal{L}, T \cup \{S_1\})$ increases by at least 1.

Case II.(b): $T \neq \emptyset$. If $\lambda(S_1, T) = 0$ (recall that $\mathcal{L} = \emptyset$ and so, $\lambda(\mathcal{L}, T \cup \{S_1\}) = 0$), then we solve the problem independently on the subgraph induced by the strongly connected components intersecting S_1 and that induced by the remaining vertices.

Otherwise, we begin by running the polynomial-time algorithm of Lemma 29 to either (i) correctly conclude that every vertex in $R_{max} = R_{max}(S_1, T)$ has out-degree at most 1 in D or (ii) compute a $u_1 \in R_{min} = R_{min}(S_1, T)$ and $u_2, u_3 \in N_D^+(u)$, or (iii) compute a minimum S_1-T separator C such that $R = R(S_1, C)$ has no vertex with out-degree at least 2 in D, and vertices $u_1 \in C$, $u_2, u_3 \in N_D^+(u_1) \setminus T$.

Case II.(b).(i): We first guess a vertex of $close(S_1, R_{max})$ to be contained in $X^\star$. That is, for every $c \in close(S_1, R_{max})$, we recurse on the tuple $\widehat{\mathcal{C}}_c = (\tau_c, T, \emptyset)$, where $\tau_c =$

$(D - \{c\}, \mathcal{S}, W, k - 1)$. We will later prove that this branching factor is $O(k)$ because the size of S_1, and hence, that of $close(S_1, R_{max})$ will always be bounded by $O(k)$. Let $\{Z_c \mid c \in close(S_1, R_{max})\}$ denote the outputs of these branches. If for any $c \in close(S_1, R_{max})$, Z_c is a set, then we return $Z_c \cup \{c\}$ and terminate. Notice that in the correct branch (say, Branch-c), $X^\star \setminus \{c\}$ is a nice solution for τ_c that is also an S_1-T separator in $D - \{c\}$.

Finally, in the branch where $close(S_1, R_{max})$ is guessed to be disjoint from the solution, we pick a vertex $c \in C_{max}(S_1, T)$ and branch by either guessing it to be in $X^\star$ or by guessing c to be reachable from S_1 in $D - X^\star$. The exhaustiveness follows from Lemma 28 (1). In the latter case, we have two further branches where we either guess that c (which, in this guess, cannot reach T in $D - X^\star$) cannot reach S_1 or that c can reach S_1 in $D - X^\star$. The latter two branches are executed by setting $\mathcal{L} = \{c\}$ or by adding c to S_1 respectively. The correctness of adding c to S_1 follows from Lemma 28 (3). Formally, we recurse on the tuples $\mathcal{C}_1 = (\tau_1, T, \emptyset)$, $\mathcal{C}_2 = (\tau, T, \{c\})$ and $\mathcal{C}_3 = (\tau_2, T, \emptyset)$, where $\tau_1 = (D - \{c\}, \mathcal{S}, W, k - 1)$ and $\tau_2 = (D, (S_1 \cup \{c\}, S_2, \ldots, S_q), W \cup \{c\}, k)$. This is now an exhaustive branching due to Lemma 28 (1). Let Z_1, Z_2, Z_3 denote the respective outputs of the three final recursions. If Z_2 or Z_3 is a set, then we return this set (chosen arbitrarily) and terminate. On the other hand, if Z_1 is a set, then we return $Z_1 \cup \{c\}$ and terminate. Finally, if all recursive outputs are NO, then we return NO and terminate.

Analysis. We remark that this case contains the only branch where vertices are added to S_1. Recall that S_1 is initially bounded by k. We will prove later that the depth of the search tree is bounded by $O(k)$, implying that this branch can only be taken $O(k)$ times, and $|close(S_1, R_{max})| = O(k)$ as required (regardless of the value of S_1 at any given stage of the algorithm), thus bounding the branching factor. Moreover, in the first $|close(S_1, R_{max})| + 1$ branches, the budget k drops by 1 while $\lambda(S_1, T)$ decreases by at most 1. In the penultimate branch, $\lambda(\mathcal{L}, T \cup \{S_1\})$ strictly increases and in the final branch, $\lambda(S_1, T)$ strictly increases (Lemma 3).

Case II.(b).(ii): In this case, we branch by either guessing one of $\{u_1, u_2, u_3\}$ to be in $X^\star$ (subject to disjointness from S_1) or by guessing that u_1 is unreachable from S_1 in $D - X^\star$ (in which case we add u_1 to T) or by guessing that $\{u_1, u_2, u_3\}$ are all reachable from S_1 in $D - X^\star$ but at least one of $\{u_1, u_2, u_3\}$ cannot reach $S_1 \cup T$ in $D - X^\star$ (again, subject to disjointness from S_1). This branching is exhaustive due to Definition 9 and the fact that $X^\star$ is a nice solution for τ.

Formally, we recurse on the following tuples: $\mathcal{C}_1 = (\tau_1, T, \emptyset)$, $\mathcal{C}_2 = (\tau_2, T, \emptyset)$, $\mathcal{C}_3 = (\tau_3, T, \emptyset)$, $\mathcal{C}_4 = (\tau_4, T \cup \{u_1\}, \emptyset)$, $\mathcal{C}_5 = (\tau_5, T, \{u_1\})$, $\mathcal{C}_6 = (\tau, T, \{u_2\})$, $\mathcal{C}_7 = (\tau, T, \{u_3\})$ defined as follows. For each $i \in \{1, 2, 3\}$, $\tau_i = (D - \{u_i\}, \mathcal{S}, W, k - 1)$ if $u_i \notin S_1$ and τ_i is a trivial no-instance otherwise. $\tau_4 = \tau$ if $u_1 \notin S_1$ and τ_4 is a trivial no-instance otherwise. Finally, for each $i \in \{1, 2, 3\}$, $\tau_{4+i} = \tau$ if $u_i \notin S_1$ and τ_{4+i} is a trivial no-instance otherwise.

Let the outputs of these recursive calls be denoted by $Z_1, \ldots, Z_7$ respectively. We return NO if all calls return NO. If any of the calls $\{\mathcal{C}_i \mid i \in [3]\}$ returns a set Z_i, then we return $Z_i \cup \{u_i\}$ (i is chosen to be the least such value) and terminate. If any of the calls $\{\mathcal{C}_i \mid i \in \{4, 5, 6, 7\}\}$ returns a set Z_i, then we return the same set (with i chosen to be least possible) and terminate.

Analysis. In the first three branches, the budget k decreases by 1 while $\lambda_D(S_1, T)$ decreases by at most 1. In $\mathcal{C}_4$, $\lambda(S_1, T)$ increases by at least 1 (Lemma 3) and in the final three branches, $\lambda_D(\mathcal{L}, T \cup S_1)$ increases by at least 1. We may assume that every vertex in D can reach $T \cup S_1$. Otherwise, they can be deleted as they do not participate in S_1-T paths or in strongly connected components containing a vertex of out-degree at

least 2 (within the same strongly connected component).

> **Case II.(b).(iii):** In this case, we execute a combination of the branchings used in Case II.(b).(i) and Case II.(b).(ii). That is, we first guess a vertex of $close(S_1, R)$ to be contained in $X^\star$. That is, for every $c \in close(S_1, R)$, we recurse on the tuple $\widehat{\mathcal{C}}_c = (\tau_c, T, \emptyset)$, where $\tau_c = (D - \{c\}, \mathcal{S}, W, k - 1)$. In the remaining branch, we may assume that $close(S_1, R)$ is disjoint from $X^\star$ and hence, by Lemma 28 (1), $X^\star$ is also disjoint from R.
>
> We now branch by either guessing one of $\{u_1, u_2, u_3\}$ to be in $X^\star$ or by guessing that $\{u_1, u_2, u_3\}$ are all reachable from S_1 in $D - X^\star$ but at least one of $\{u_1, u_2, u_3\}$ cannot reach $S_1 \cup T$ in $D - X^\star$. The correctness follows from Definition 9 and Lemma 28 (1) and the analysis is identical to that used in the previous two cases.

This completes the description of the algorithm.

Correctness and running time analysis. The correctness follows from the terminating conditions of the algorithm outlined in the beginning, plus the fact that (a) every branching step has been argued to be exhaustive and (b) the measure $3k - \lambda(S_1, T) - \lambda(\mathcal{L}, T \cup S_1)$ strictly decreases in every recursive call. Since the algorithm has $O(k)$ branches at any step, we conclude that the running time of this algorithm is $2^{O(k \log k)} \cdot n^{O(1)}$. This completes the proof of the lemma. $\square$

We obtain Lemma 27 by invoking Lemma 30 on the tuple $((D, \mathcal{S}, W, k), T = W \setminus S_1, \mathcal{L} = \emptyset)$. This completes our proof of Theorem 4.

5.2 Faster FPT algorithm for Bounded Size SCC Deletion

In this section, we give an algorithm for Rooted $\mathcal{H}$-SCC Deletion that runs in time $2^{O(k \log s)} \cdot n^{O(1)}$ in the special case where $\mathcal{H}$ is the set of all arborescences on exactly $s + 1$ vertices, that is we prove Theorem 5.

Theorem 5. *BSSCVD can be solved in time $2^{O(k(\log k + \log s))} \cdot n^{O(1)}$.*

In the following, fix $\tau = (D, \mathcal{S} = (S_1, \ldots, S_q), W, k)$ to be an instance of Rooted $\mathcal{H}$-SCC Deletion. Here, $\mathcal{H}$ is the set of all arborescences on $s + 1$ vertices. Using Lemma 8, Theorem 5 can be obtained as a consequence of the following lemma.

Lemma 31. *There is an algorithm that, given τ, runs in time $2^{O(k \log s)} \cdot n^{O(1)}$ and either returns a solution for (D, k) or correctly concludes that there is no nice solution for τ.*

The proof of the following lemma is identical to that of Lemma 28 (2).

Lemma 32. *Let $\mathcal{L} \subseteq V(D) \setminus W$. Suppose that there is a nice solution X for τ that is an $\mathcal{L}$-W separator. Then, there is a nice solution X' for τ that is an $\mathcal{L}$-W separator and moreover, $X' \cap R_{max}(\mathcal{L}, W) = \emptyset$.*

Lemma 33. *Let $\mathcal{L} \subseteq V(D) \setminus W$. There is an algorithm that, given the tuple $(\tau, \mathcal{L})$, runs in time $2^{O(k \log s)} \cdot n^{O(1)}$ and either returns a solution for (D, k) or correctly concludes that there is no nice solution for τ that is also an $\mathcal{L}$-W separator.*

Proof. This algorithm closely resembles parts of the algorithm of Lemma 30. Therefore, we only give a high level sketch outlining the differences. Fix a nice solution $X^\star$ that is also an $\mathcal{L}$-W separator (if one exists).

The base cases and Case I ($\mathcal{L} \neq \emptyset$) are identical to those in Lemma 30, with the correctness of the latter following from Lemma 32. In Case II ($\mathcal{L} = \emptyset$), we pick an arborescence on $s + 1$

vertices that is rooted at a vertex $c \in S_1$ and and branch by guessing one of these vertices to be in $X^\star$, in which case we delete it and recurse, or by guessing that one of these vertices cannot reach W in $D - X^\star$, in which case we add it to $\mathcal{L}$ and recurse. This is correct because any arborescence on $s + 1$ vertices that is reachable from a vertex of S_1 implies the existence of an arborescence on $s + 1$ vertices that is rooted at a vertex of S_1. Notice that the branching factor in either case is bounded by $2(s + 1)$ and in each branch, the measure $2k - \lambda(\mathcal{L}, W)$ strictly decreases. To achieve the increase in $\lambda(\mathcal{L}, W)$ in the second set of $s + 1$ branches of Case II (where $\mathcal{L}$ changes from $\emptyset$ to a singleton), we note that any vertex of D that cannot reach W in D cannot be part of either a cycle or an S_i-S_j path for some i, j and hence, can be removed without affecting the presence of a nice solution. This completes the proof of the lemma. □

We obtain Lemma 31 by invoking Lemma 33 on the tuple $((D, \mathcal{S}, W, k), \mathcal{L} = \emptyset)$. This completes our proof of Theorem 5.

6 Conclusions

We have initiated the study of the parameterized complexity of $\mathcal{H}$-SCC Deletion problem, where the objective is to compute a maximum subdigraph where no strongly connected component contains a forbidden subgraph from the family $\mathcal{H}$. This problem is a natural generalization of the classic Directed Feedback Vertex Set problem. We have obtained fixed-parameter algorithms for this problem when $\mathcal{H}$ either contains at least one path or only contains rooted graphs. Furthermore, we have demonstrated that for a pair of previously studied special cases of this problem, one can obtain faster fixed-parameter algorithms by using our general strategy tailored to these special cases.

Our algorithms are fixed-parameter tractable parameterized by k for fixed families $\mathcal{H}$ (that contain a path or only rooted digraphs) and a fixed-parameter algorithm for this problem parameterized by both k and $d_{\mathcal{H}}$ (size of the largest graph in $\mathcal{H}$) is unlikely to exist in general.

Our work identifies some natural directions for future research. In particular, can we completely characterize those finite families $\mathcal{H}$ for which this problem is fixed-parameter tractable? Furthermore, one could ask: for which infinite families $\mathcal{H}$ does this problem admit a fixed-parameter algorithm? Recently Göke, Marx and Mnich [7] gave a fixed-parameter algorithm for the case where $\mathcal{H}$ is the set of all cycles of length *at least* a given s.

References

[1] Karthekeyan Chandrasekaran and Sahand Mozaffari. Odd multiway cut in directed acyclic graphs. In *12th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 12:1–12:12. Springer, 2017. 1

[2] Jianer Chen, Yang Liu, Songjian Lu, Barry O'sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 177–186, 2008. 1, 2, 4, 5

[3] Rajesh Chitnis, Marek Cygan, Mohammataghi Hajiaghayi, and Dániel Marx. Directed subset feedback vertex set is fixed-parameter tractable. *ACM Transactions on Algorithms (TALG)*, 11(4):1–28, 2015. 1

[4] Rajesh Chitnis, Mohammad Taghi Hajiaghayi, and Dániel Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. *SIAM Journal on Computing*, 42(4):1674–1696, 2013. 1, 3, 9

[5] Marek Cygan, Fedor V Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*, volume 4. Springer, 2015. 5

[6] Alexander Göke, Dániel Marx, and Matthias Mnich. Parameterized algorithms for generalizations of directed feedback vertex set. In *International Conference on Algorithms and Complexity*, pages 249–261. Springer, 2019. 1, 2, 3, 27

[7] Alexander Göke, Dániel Marx, and Matthias Mnich. Hitting long directed cycles is fixed-parameter tractable. In *47th International Colloquium on Automata, Languages and Programming (ICALP), to appear*, 2020. 27

[8] Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Magnus Wahlström. Fixed-parameter tractability of multicut in directed acyclic graphs. *SIAM Journal on Discrete Mathematics*, 29(1):122–144, 2015. 1

[9] Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *53rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 450–459, 2012. 5

[10] Daniel Lokshtanov, MS Ramanujan, and Saket Saurabh. When recursion is better than iteration: a linear-time algorithm for acyclicity with few error vertices. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1916–1933. SIAM, 2018. 1, 5

[11] Daniel Lokshtanov, MS Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2181–2200. SIAM, 2020. 1

[12] Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006. 3, 4, 5

[13] Dániel Marx. Important separators and parameterized algorithms. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 5–10. Springer, 2011. 5

[14] Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004. 6