FPT-approximation for FPT Problems

Daniel Lokshtanov[*], Pranabendu Misra[†], M. S. Ramanujan[‡], Saket Saurabh[§], and Meirav Zehavi[¶]

Abstract

Over the past decade, a flurry of results has focused on the design of parameterized approximation algorithms for W[1]-hard problems. However, there are fundamental problems within the class FPT for which the best known algorithms run in time $2^{\mathsf{poly}(k)}n^{\mathcal{O}(1)}$ or $c^k n^{\mathcal{O}(1)}$, and have seen no progress over the decade; some of them have even been proved not to admit algorithms that run in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ under the Exponential Time Hypothesis (ETH) or $(c-\epsilon)^k n^{\mathcal{O}(1)}$ under the Strong ETH (SETH). In this paper, we expand the study of FPT-approximation and initiate a systematic study of FPT-approximation for problems that are FPT. That is, we design FPT-approximation algorithms for problems that are FPT, with running times that are significantly faster than the corresponding best known FPT-algorithm, and while achieving approximation ratios that are significantly better than what is possible in polynomial time.

- We present a general scheme to design $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$-time 2-approximation algorithms for cut problems. In particular, we exemplify it for Directed Feedback Vertex Set, Directed Subset Feedback Vertex Set, Directed Odd Cycle Transversal and Undirected Multicut.
- Further, we extend our scheme to obtain FPT-time $\mathcal{O}(1)$-approximation algorithms for weighted cut problems, where the objective is to obtain a solution of size at most k and of minimum weight. Here, we present two approaches. The first approach achieves $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$-time constant-factor approximation, which we exemplify for all problems mentioned in the first bullet. The other leads to an FPT-approximation Scheme (FPT-AS) for Weighted Directed Feedback Vertex.
- Additionally, we present a combinatorial lemma that yields a partition of the vertex set of a graph to roughly equal sized sets that the removal of each reduces its treewidth substantially, which may be of independent interest. We exemplify it by designing $c^w n^{\mathcal{O}(1)}$-time $(1+\epsilon)$-approximation algorithms for graph problems where w is the treewidth of the input graph, faster than known SETH lower bounds. We exemplify it for Vertex Cover, Component Order Connectivity, Bounded-Degree Vertex Deletion and $\mathcal{F}$-Packing for any family $\mathcal{F}$ of bounded sized graphs.
- Lastly, we present a general reduction of problems parameterized by treewidth to their versions parameterized by solution size. Combined with our first scheme, we exemplify it to obtain $c^w n^{\mathcal{O}(1)}$-time $(2+\epsilon)$-approximation algorithms for all problems mentioned in the first bullet.

[*]University of California, Santa Barbara, USA. `daniello@ucsb.edu`

[†]Max Planck Institute for Informatics, Germany. `pmisra@mpi-inf.mpg.de`

[‡]University of Warwick, UK `r.maadapuzhi-sridharan@warwick.ac.uk`

[§]Department of Informatics, University of Bergen, Norway and Institute of Mathematical Sciences, HBNI, India, `saket@imsc.res.in`

[¶]Ben-Gurion University of the Negev, Israel. `meiravze@bgu.ac.il`

1 Introduction

Two algorithmic paradigms that have seen immense success in dealing with NP-hard problems are Approximation Algorithms and Parameterized Complexity. In Approximation Algorithms, we design algorithms that run in polynomial time and output a solution with a provable guarantee on its quality. There is also a well-developed theory of hardness of approximation, which allows us to trace the boundaries of tractability for approximation algorithms.

On the other hand, the goal of parameterized complexity is to find ways of solving NP-hard problems more efficiently than brute force: here the aim is to restrict the combinatorial explosion to a parameter that is hopefully much smaller than the input size. Formally, a *parameterization* of a problem is the assignment of an integer k to each input instance, and we say that a parameterized problem is *fixed-parameter tractable (FPT)* if there is an algorithm that solves the problem in time $f(k) \cdot |I|^{O(1)}$, where $|I|$ is the size of the input and f is an arbitrary computable function depending on the parameter k only. Just as NP-hardness is used as evidence that a problem probably is not polynomial time solvable or polynomial time approximable, there exists a hierarchy of complexity classes above FPT, and showing that a parameterized problem is hard for one of these classes gives evidence that the problem is unlikely to be FPT. In fact, assuming well-known assumptions such as Exponential Time Hypothesis (ETH) or Strong Exponential Time Hypothesis (SETH), we can obtain qualitative lower bounds for FPT algorithms, that is, lower bounds on $f(k)$ in the running time of any FPT algorithm for some specific problem. For more background on Approximation Algorithms and Parameterized Complexity, the reader is referred to the monographs and books [CFK+15, DF99, FG06, Nie06, Vaz01, WS11, FLSZ19].

There is a plethora of problems for which, simultaneously, the non-existence of polynomial-time algorithms with certain approximation ratios, as well as intractability within Parameterized Complexity, are known. These intractabilities together motivate the desire for algorithms that runs in FPT-time, for the parameter in which the problem is intractable, and at the same time beat the lower bounds on hardness of approximation that are proven for polynomial-time algorithms. This leads to the world of FPT-approximation, which has been an extremely active area of research in the last five years. For a minimization problem parameterized by the solution size k, a factor-α FPT-approximation algorithm is an algorithm that runs in time $f(k) \cdot n^{\mathcal{O}(1)}$, and either returns that there is no solution of size at most k or returns a solution of size at most αk. One can similarly define the notion of parameterized approximation for maximization problems. When the parameter is structural (e.g., the treewidth of the input graph), a factor-α approximation FPT-algorithm is just a factor-α approximation algorithm that runs in FPT-time rather than polynomial time.

Some of the notable problems that have been shown to admit FPT-approximation algorithms include Vertex Minimum Bisection [FM06], k-Path Deletion [Lee19], Max k-Vertex Cover in d-uniform hypergraphs [SF17, Man19], k-Way Cut [GLL18a, GLL18b, KL20, LSS20] and Steiner Tree parameterized by the number of non-terminals [DFK+18]. These are just representative examples (and far away from exhaustive) [CHK13, DHK05, DFK+18, BLM18, FM06, GKW19, GLL18a, GLL18b, KL20, Kor16, Lam14, Lee19, Man19, Mar04, Mar08, LSS20, PvLW17, SF17, Wie17]. On the other hand, several basic problems are shown to be even hard in terms of FPT-approximation. The main ones include Set Cover, Dominating Set, Independent Set, Clique, Biclique and Steiner Orientation [CCK+17, KLM19a, Wlo20, CL19, Lin18, Lin19, BBE+19]. For a comprehensive overview of the state of the art on Parameterized Approximation, we refer to the recent survey by Feldmann et al. [FKLM20], and the surveys by Kortsarz [Kor16] and by Marx [Mar08].

1.1 Our Context and Questions

For all the parameterized problems mentioned above for which FPT-approximation algorithms were developed, the parameter used to measure the running time of the algorithm is one with respect to which the problem is known to be W-hard. In other words, most known FPT-approximation algorithms are for problems that are intractable within Parameterized Complexity. A natural question is:

What about FPT-approximation of problems that are FPT?

Indeed, these problems hold a lot of promise and remain hitherto unexplored in the light of FPT-approximation, with exceptions that are few and far between [BF11, BF13, FKRS18, KS19, MR09]. Our guiding example in this regard is Treewidth. From as early as 1990, it is well known that given a graph G and an integer k, we can test whether the graph has treewidth at most k in time $2^{\mathcal{O}(k^3)}n$ [Bod96]. While this algorithm has stood the test of time and remains the best-known algorithm for the problem, several faster parameterized algorithms have been designed that either return that the treewidth of G is more than k or return a tree decomposition of width $\mathcal{O}(k)$ [RS95, Lag96, Ree92, Ami10]. In fact, in 2013, the first constant-factor FPT-approximation with running time $2^{\mathcal{O}(k)}n$ was obtained [BDD+16]. The main idea of the paper is to replicate this success of Treewidth to other combinatorial problems. In particular, we wish to achieve the following.

In this paper, we expand the study of FPT-approximation and initiate a systematic study of FPT-approximation for problems that are FPT! That is, design FPT-approximation algorithms for problems that are FPT, with a running time that is significantly faster than the corresponding FPT-algorithm (for an exact decision version of the problem), and that achieves approximation ratios that are better than one can provably achieve in polynomial time.

A natural choice of problems for FPT-approximation is W-hard problems. However, given the fact that in this paper, we plan to design FPT-approximation for FPT problems, two important questions that we need to address are: (a) which problems within FPT; and (b) what kind of running time and factor of approximation are we looking for.

1.1.1 Which Problems Within FPT?

A central problem in parameterized algorithms is to obtain algorithms with running time $f(k)n^{\mathcal{O}(1)}$, such that f is a computable function of the parameter k that grows as slowly as possible. In the last three decades, several problems have been shown to admit such algorithms or shown that no such algorithms can exist under a plausible assumption. In fact, several problems have been shown to admit algorithms with running time of the form $c^k n^{\mathcal{O}(1)}$; however, still, there is a plethora of problems for which the best known algorithms run in time $2^{\mathsf{poly}(k)}n^{\mathcal{O}(1)}$, and have seen no progress over more than a decade. Also, there are problems for which we can show lower bounds on $f(k)$ under ETH or SETH (or other plausible conjectures). In our opinion, these problems are the most natural candidates for designing FPT-approximation algorithms

For illustration, consider Directed Feedback Vertex Set (DFVS) parameterized by the solution size.[1] It is well known that it admits an algorithm with running time $2^{\mathcal{O}(k \log k)}n^{\mathcal{O}(1)}$, and this has not been improved since 2007, when Chen et al. [CLL+08] resolved this longstanding open question in the area of Parameterized Complexity, except for the dependence on the input size [LRS18]. Similarly, a decade back Marx and Razgon [MR14] and Bousquet et al. [BDT18], independently, settled the parameterized complexity of Multicut on undirected graphs parameterized by the solution size, by designing an algorithm with running time $2^{\mathcal{O}(k^3)}n^{\mathcal{O}(1)}$ (this is the running time of the algorithm given in [MR14]). However, there has been no improvement over $f(k) = 2^{\mathcal{O}(k^3)}$ for this algorithm, in the last ten years. These are examples of problems for which there has been no progress in a long time. The other examples would be those

[1]The definitions of all problems considered in this paper can be found in Appendix 7.

for which we have known lower bounds on $f(k)$—for example, DFVS and VERTEX COVER parameterized by the treewidth w of the input graph (in case of a directed graph, the treewidth of its underlying undirected graph). Indeed, Bonamy et al. [BKN+18] showed that assuming ETH, DFVS does not admit an algorithm with running time $2^{o(w \log w)} n^{\mathcal{O}(1)}$. Moreover, Lokshtanov et al. [LMS18a] showed that assuming SETH, there is no algorithm for VERTEX COVER running in time $(2-\delta)^w n^{\mathcal{O}(1)}$, for any fixed constant $\delta > 0$. The field of Parameterized Algorithms is full of such examples [CM16, JLS14, KPPW15, LMS18a, LMS18b].

We believe that studying the aforementioned central problems in Parameterized Complexity in the realm of FPT-approximation will lead to the development of new methodologies in the design of Parameterized Algorithms.

1.1.2 What Kind of Approximation Ratios and Running Times?

As stated above, we must aim to design an FPT-approximation algorithm that achieves an approximation factor that is not possible (under a plausible complexity theoretic assumption) in polynomial time, and the function $f(k)$ in the running time should be asymptotically better than the best known bound to solve the exact decision version of the problem. For example, DFVS parameterized by the solution size admits an algorithm with running time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ [CLL+08], and the best known polynomial time approximation algorithm has factor $\mathcal{O}(\log n \log \log n)$ [ENSS98]. Furthermore, under Unique Games Conjecture (UGC), the problem does not admit any constant-factor approximation algorithm [GL16]. Similarly, MULTICUT on undirected graphs [MR14], parameterized by the solution size admits an algorithm with running time $2^{\mathcal{O}(k^3)} n^{\mathcal{O}(1)}$, and an approximation algorithm with factor $\mathcal{O}(\log n)$ [LR99]. Assuming UGC, Chawla et al. [CKK+05] showed that the problem does not admit any constant-factor approximation algorithm. A stronger version of UGC leads to a hardness result of $\Omega(\sqrt{\log \log n})$ [CKK+05]. Thus, for DFVS and MULTICUT on undirected graphs, a desirable outcome will be a constant-factor FPT-approximation algorithm running in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.

For VERTEX COVER parameterized by treewidth (w), an FPT-algorithm with running time $\mathcal{O}(2^w n)$ is known; on the other hand, assuming SETH, there is no algorithm running in time $(2-\delta)^w n^{\mathcal{O}(1)}$, for any fixed constant $\delta > 0$ [LMS18a]. In the realm of polynomial time approximation algorithm, the problem admits a simple factor-2 approximation, and assuming UGC, this approximation factor cannot be improved to $2-\eta$, for any fixed $\eta > 0$ [KR08]. For this problem, a desirable FPT-approximation algorithm will be an FPT-AS (FPT-approximation scheme). That is, for every $\epsilon > 0$, design an $(1+\epsilon)$-FPT-AS running in time $(2-g(\epsilon))^w n^{\mathcal{O}(1)}$, for some function g.

1.2 Our Results and Methods

We classify our algorithmic results into following three classes based on the methods involved in solving each of them.

- Basic cut problems such as DFVS, SUBSET DFVS, DIRECTED ODD CYCLE TRANSVERSAL, and MULTICUT.
- Problems parameterized by treewidth, w, of the input graph, such as all aforementioned problems, as well as VERTEX COVER, TRIANGLE PACKING (or, more generally, $\mathcal{F}$-PACKING), and more.
- Weighted versions of cut problems such as WEIGHTED DFVS and WEIGHTED MULTICUT.

1.2.1 Two-extremal Separator Technique and Cut Problems

Our main technical results for cut problems are single-exponential-time factor-2 FPT-approximation algorithms for basic cut-problems such as DFVS, SUBSET DFVS, DIRECTED ODD CYCLE

Problem Name	**FPT** $f(k)$	**Polytime APR**	**FPT-Apx** $f(k)$	**FPT-APR**	**Last Dev. or Lower Bound**
DFVS	$k^{\mathcal{O}(k)}$	$\mathcal{O}(\log n \log\log n)$	$2^{\mathcal{O}(k)}$	2	2007
Subset DFVS	$2^{\mathcal{O}(k^3)}$	$\mathcal{O}(\log n \log\log n)$	$2^{\mathcal{O}(k)}$	2	2012
Directed Odd Cycle Transversal	W[1]-hard	NPR	$2^{\mathcal{O}(k)}$	2	
Multicut	$2^{\mathcal{O}(k^3)}$	$\mathcal{O}(\log n)$	$2^{\mathcal{O}(k)}$	2	2010
DFVS/tw(w)	$w^{\mathcal{O}(w)}$	$\mathcal{O}(\log n \log\log n)$	$2^{\mathcal{O}(\frac{w}{\epsilon})}$	$(2+\epsilon)$	ETH
Subset DFVS/tw(w)	$w^{\mathcal{O}(w)}$	$\mathcal{O}(\log n \log\log n)$	$2^{\mathcal{O}(\frac{w}{\epsilon})}$	$(2+\epsilon)$	ETH
Directed Odd Cycle Transveral/tw(w)	$w^{\mathcal{O}(w)}$	NPR	$2^{\mathcal{O}(\frac{w}{\epsilon})}$	$(2+\epsilon)$	ETH
Multicut/tw(w)	para-NP-hard	$\mathcal{O}(\log n)$	$2^{\mathcal{O}(\frac{w}{\epsilon})}$	$(2+\epsilon)$	
Vertex Cover/tw(w)	2^w	2	$2^{(1-\epsilon)w}$	$(1+\epsilon)$	SETH
Component Order Connectivity/tw(w)	ℓ^w	$\mathcal{O}(\log \ell)$	$\ell^{(1-\frac{\epsilon}{2\ell-1})w}$	$(1+\epsilon)$	SETH
Bounded-Degree Vertex Deletion/tw(w)	$(d+2)^w$	$\mathcal{O}(\log d)$	$(d+2)^{(1-\frac{\epsilon}{d^3+4d^2+5d+1})w}$	$(1+\epsilon)$	SETH
Triangle Packing/tw(w)	2^w	$\frac{3}{2}$	$2^{(1-\frac{\epsilon}{3})w}$	$(1+\epsilon)$	SETH
Weighted DFVS	NPR	$\mathcal{O}(\log n \log\log n)$	$2^{\mathcal{O}(k)}$	$(4, 8(1+\epsilon))$	OPEN
Weighted Subset DFVS	NPR	$\mathcal{O}(\log n \log\log n)$	$2^{\mathcal{O}(k)}$	$(4, 8(1+\epsilon))$	OPEN
Weighted DFVS	NPR	$\mathcal{O}(\log n \log\log n)$	$k^{k/\epsilon}2^{k^3 \log k}$	$(1, 1+\epsilon)$	OPEN
Weighted Multicut	NPR	$\mathcal{O}(\log n)$	$2^{\mathcal{O}(k)}$	$(4, 8(1+\epsilon))$	NPR
Weighted DOCT	W[1]-hard	NPR	$2^{\mathcal{O}(k)}$	$(4, 8(1+\epsilon))$	NPR

Table 1: A summary of our results. Here, NPR stands for *no previous result.* Note that the FPT-approximation results given in this table are proved in this paper.

Transversal (DOCT) and Multicut. In particular, in Section 3, we prove the following results.

Theorem 1.1. Directed Feedback Vertex Set*,* Subset Directed Feedback Vertex Set*,* Directed Odd Cycle Transversal *(DOCT), and* Multicut *have $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$-time factor-2 approximation algorithms.*

Lokshtanov et al. [LRSZ20] gave the first factor-2 FPT-approximation for DOCT. However, the running time of their algorithm is $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(1)}$, which we improve to a single-exponential FPT running time.

To obtain our results, we give a general technique for FPT-approximating cut problems that could be applicable to further problems. Here, we illustrate this technique by describing its application to the DFVS problem. Although in Section 3, we directly design our algorithm for Subset DFVS and derive the algorithm for DFVS as a corollary, we believe that the application of this technique to DFVS provides a more insightful description. Recall that in the optimization version of DFVS, one is given a digraph D and the goal is to find a vertex set S of smallest size such that $D - S$ is acyclic. A factor-2 FPT-approximation for DFVS is an algorithm that, given the pair (D, k), runs in time $f(k)n^{\mathcal{O}(1)}$ for some computable f and if D has a solution of size at most k, then it outputs a solution of size at most $2k$.

Our starting point is the well-known iterative compression method [CFK+15] which guarantees that in order to obtain our result for DFVS, it is sufficient for us to give an algorithm that, given the pair (D, k) and a vertex set W of size at most $2k$, runs in time $2^{\mathcal{O}(k+|W|)}n^{\mathcal{O}(1)}$ and if D has a solution of size at most k disjoint from W, then it outputs a solution of size at most $2k$. In the base case, when $|W| = 1$, this can be solved trivially in polynomial time. Hence, suppose that $|W| > 1$ and let $S^\star$ be a smallest solution (with size at most k) in D that is disjoint from W. Then, it is straightforward to see that there is an ordering $w_1, \ldots, w_r$ of the vertices in W such that in the graph $D - S^\star$, there is no directed path from w_i to w_j for every $j < i$. Let $W_2 = \{w_1, \ldots, w_{\lfloor r/2 \rfloor}\}$ and $W_1 = \{w_{\lfloor r/2 \rfloor + 1}, \ldots, w_r\}$. Then, $S^\star$ is a W_1-W_2 separator of size at most k in D. In particular, there is a minimal subset $S_1^\star \subseteq S^\star$ that is a W_1-W_2 separator in D. Now, let X_{pre} and X_{post} be W_1-W_2 separators in D such that X_{pre} is "closer" than $S_1^\star$ is to W_1 and X_{post} is "closer" than $S_1^\star$ is to W_2. Formally, the set of vertices reachable from W_1 after deleting X_{pre} is a subset of that reachable from W_1 after deleting $S_1^\star$. Similarly, the set of vertices reachable from W_1 after deleting X_{post} is a superset of that reachable from W_1 after deleting $S_1^\star$. Then, we show that deleting $X_{\mathsf{pre}} \cup X_{\mathsf{post}}$ from D makes $S_1^\star$ irrelevant, i.e., $S^\star \setminus S_1^\star$ is a solution for $D - (X_{\mathsf{pre}} \cup X_{\mathsf{post}})$. In other words, we can reduce the size of an optimal solution by $|S_1^\star|$ by paying a cost of at most $|X_{\mathsf{pre}} \cup X_{\mathsf{post}}|$. This raises the question – "Can we come up with small enough X_{pre}, X_{post}, say of size at most $|S_1^\star|$ each?". We observe that indeed, this is possible by considering these separators to be *important* W_1-W_2 *separators* (see Section 3 for the formal definition), out of which one is pushed as close as possible to W_1 and the other is pushed as close as possible to W_2. It is well-known that the number of such extremal separators of size at most k pushed closest to each side is at most 4^k, and hence we have 16^k choices for X_{pre} and X_{post}, which can therefore be guessed in FPT-time. Once these are guessed, we delete both separators from the input, in the process decreasing the size of the optimal solution by at least $1/2 \cdot |X_{\mathsf{pre}} \cup X_{\mathsf{post}}|$, and then recurse independently on two subinstances – one induced by the vertices in the strong components containing W_1 and the other induced by the vertices in the strong components containing W_2. This division of the problem can be done since no cycle can intersect both these instances once a W_1-W_2 separator has been removed. Now, we recursively solve the same problem on two inputs, each of which has at most half the number of vertices of W from the original input. Analysing the resulting recurrence gives us the required running time bound.

We apply the same high level "two-extremal" separator approach to the other cut problems we consider. In fact, this approach works for any problem where the goal is to hit a family of *strongly connected subgraphs*. However, a major difference between DFVS and the other problems is the following. We know that the strongly connected components of $D - S^\star$ (in the above example) are singletons, implying that we can continue the divide-and-conquer approach till we hit the base case. However, for the other problems, two issues crop up: (i) It may very well be the case that there is a unique strongly connected component containing the set W (a given solution for the respective problem) after removing the hypothetical solution $S^\star$. (ii) The vertices of W could be broken up across the strong components of $S^\star$ in a very imbalanced way. We overcome Issue (i) by designing a subroutine to efficiently 2-approximate the solution in cases where there is a unique strongly connected component containing W in $D - S^\star$. This subroutine is problem-specific and can take different forms for different problems. For instance, in the case of Subset DFVS, we solve this special case using branching on important separators, in the case of Multicut (which we phrase as a directed cut problem by moving to bidirected graphs), this case is solved by a reduction to the Digraph Pair Cut problem [KW20] and in the case of DOCT, the special case can be 2-approximated in polynomial time by solving max-flow in an auxiliary graph. In order to overcome Issue (ii), we devise a 3-way divide and conquer where, in each of the (at most) three subinstances that are generated in each step, either the number of vertices of W drops by a constant fraction or we can directly use the subroutine designed for the afore-mentioned special case, avoiding the need for further recursion on this instance. A careful analysis of the recurrence relation give us the required time bound for these problems.

1.2.2 Results for Parameterization by Treewidth

For parameterization by the treewidth w of the input graph, we present three general theorems, and derive a host of results for specific problems as corollaries. The first two theorems, derived from a new combinatorial lemma that may be of independent interest (described below), "break" SETH-based bounds at an (arguably) negligible cost of an ϵ factor in approximation. The third theorem allows us to combine the results given in Section 3 to obtain constant-factor single-exponential (in w) time approximation algorithms for the problems studied in that section—such as Directed Feedback Vertex Set—which do not admit single-exponential (in w) time exact algorithms under the ETH.[2] Notice that here we consider these problems when the parameter is w rather than k, yet the algorithms for the parameterization by k will come in handy. Briefly, the idea of the proof is to identify "not too many" bags (so that their removal is not costly), such that the subinstances derived by their removal have optimum that is "not too large" compared to the treewidth w (so that they can be efficiently solved) yet "not too small" (as to compensate for the cost of the bags removed). So, as consequences of a more general (our third) theorem, we have the following.

Theorem 1.2. *For every fixed constant $\epsilon > 0$, each of the following problems admits a $(4+\epsilon)$-approximation algorithm that runs in time $2^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(1)}$:* Directed (Subset) Feedback Vertex Set*,* Directed Odd Cycle Transversal*,* Undirected Multicut.

Roughly speaking, our first theorem states that any vertex deletion problem that admits an α-approximate ck-vertex kernel and which can be solved in time $\mathcal{O}(b^w n^p)$, admits an $\alpha(1+\epsilon)$-approximation algorithm that runs in time $\mathcal{O}(b^{(1-\frac{\epsilon}{c-1})w+o(w)} n^p + n^{\mathcal{O}(1)})$. Moreover, the second theorem essentially states that for any fixed graph family $\mathcal{F}$ (where the maximum size of a graph in $\mathcal{F}$ is d) such that the corresponding $\mathcal{F}$-Packing problem can be solved in time $\mathcal{O}(b^w n^p)$, the $\mathcal{F}$-Packing problem also admits a $(1+\epsilon)$-approximation algorithm that runs in time $\mathcal{O}(b^{(1-\frac{\epsilon}{d})w+o(w)} n^p + n^{\mathcal{O}(1)})$. It is known that each of the following problems admits an $\mathcal{O}(b^w n)$-time algorithm: Vertex Cover where $b = 2$; Component Order Connectivity where $b = \ell$; Bounded-Degree Vertex Deletion where $b = (d+2)$; Triangle Packing where $b = 2$. Moreover, all of these constants b are known to be tight under the SETH for their respective problems! As consequences of our two theorems, we derive the following.

Theorem 1.3. *For every fixed constant $\epsilon > 0$, each of the following problems admits a $(1+\epsilon)$-approximation algorithm that runs in time $b^{w+o(w)} n + n^{\mathcal{O}(1)}$:* Vertex Cover *where $b = 2^{1-\epsilon}$* Component Order Connectivity *where $b = \ell^{1-\frac{\epsilon}{2\ell-1}}$;* Bounded-Degree Vertex Deletion *where $b = (d+2)^{1-\frac{\epsilon}{d^3+4d^2+5d+1}}$;* $\mathcal{F}$-Packing *for every graph family $\mathcal{F}$ that consists of graphs on at most d vertices where $b = b_{\mathcal{F}}^{1-\frac{\epsilon}{d}}$, where $b_{\mathcal{F}}$ is the best known constant such that* $\mathcal{F}$-Packing *is solvable in time $\mathcal{O}(b_{\mathcal{F}}^w n)$. For example, for* Triangle Packing *$b_{\mathcal{F}} = 2$.*

So, for example, we can approximate Vertex Cover within factor $1\frac{1}{3}$ in time $1.588^w n + n^{\mathcal{O}(1)}$.

The proof of both theorems is based on a combinatorial lemma that yields a partition of the vertex set of a graph to roughly equal sized sets that the removal of each reduces its treewidth substantially, which may be of independent interest. When being applied, for vertex deletion problems, we note that there exists a part that has "large" intersection with an (unknown) optimal solution, and furthermore that part is small (as all parts are, being disjoint and of equal size, and considered after applying a linear-vertex kernel), and hence we can just take it into our solution at modest cost. For packing problems, we note that there exists a part that has "small" intersection with an (unknown) optimal solution, and hence we can just be discard it at modest cost. Very briefly, the proof of the combinatorial lemma itself is based on a greedy computation of a proper coloring of the graph when each bag of its tree decomposition is turned into a clique. By using more colors than "necessary", we are able to argue that no color is used "too many" times. Then, having computed this coloring, a packing argument concerning its color classes yields the combinatorial lemma.

1.2.3 FPT-approximations for Weighted Problems

In the above discussions, the focus was primarily on *unweighted* problems. However, it is often the case that a problem instance is presented with certain costs or weight function, and the objective is to find a solution of minimum (or maximum) weight. Such types of optimization

[2]See Section 4 for references for the results mentioned here.

problems are a central object of study in approximation algorithms. However, Parameterized Complexity has so far primarily focused on *unweighted problems*, although FPT algorithms are known for several weighted problems such as Weighted Steiner Tree. The parameterized complexity of many other problems such as Weighted DFVS and Weighted Multicut remain longstanding open problems [CFK+15], even though their unweighted variants are known to be FPT for a long time.

In this paper, we present methods and techniques to develop approximation algorithms for weighted graph problems, that we exemplify via Weighted (Subset) DFVS, Weighted DOCT, and Weighted Multicut. We remark that our methods may also be applicable to other weighted problems for which the unweighted version admits an FPT (approximation) algorithm. Moreover, they yield approximation algorithms that essentially have the same running time as the algorithm for the unweighted problem, and only a slightly worse approximation ratio. In other words, *our algorithms have running times are significantly faster than the best (known) FPT algorithms (for the unweighted problem), while achieving approximation ratios that are significantly better than what is possible in polynomial time.*

To describe our results in more detail, let us focus on the example of Weighted DFVS. Here we are given a directed graph G, a weight function $w : V(G) \to \mathbb{Q}$, and the objective is to find a subset $S \subseteq V(G)$ such that $w(S) = \sum_{v \in S} w(v)$ is minimized and $G - S$ is a directed acyclic graph (DAG). Let us begin by discussing the parameterization of weighted problems.

Parameterization for Weighted Problems. A natural way to parameterize Weighted DFVS is select a non-negative value k, and ask for a solution S such that $w(S) \leq k$. This however is unlikely to work since we can reduce DFVS to Weighted DFVS by assigning every vertex a weight of $\frac{1}{k}$, and ask for a solution of weight at most 1. Clearly, unless P≠NP, we do not expect an FPT algorithm for this problem. Thus, parameterizing Weighted DFVS by the value of the weight is not meaningful. A more suitable choice is the cardinality of the solution, i.e. the number of vertices in it. That is, given a directed graph D, a weight function $w : V(G) \to \mathbb{Q}$ and a non-negative integer k, we seek a set $S \subseteq V(G)$, such that $D - S$ is a DAG, $|S| \leq k$ and $w(S)$ is minimized. We remark that it is a longstanding open problem whether Weighted DFVS is FPT parameterized by the solution cardinality k. Furthermore, this problem does not admit a constant-factor approximation algorithm in polynomial time, even in the unweighted setting [GL16]. We present algorithms that are substantial improvements on both fronts.

Let Opt_k denote the weight $w(S_k^{OPT})$, where S_k^{OPT} is a minimum weight solution of cardinality at most k. Note that Opt_k could be much larger than $\mathsf{OPT} = \min_{k \in \mathbb{N}} \mathsf{Opt}_k$, and conversely any solution of weight OPT could have much larger cardinality than k. In a parameterized algorithm, we are only interested in solutions whose cardinality is bounded by k, while in an approximation algorithm our objective is to approximate OPT irrespective of the solution cardinality. Taking our cue from both these approaches, we define a notion of *bi-criteria* FPT-approximation. To state it formally, we require a few additional definitions. A problem Π on graphs is associated with a predicate $\phi_\Pi(G, S)$ (also called a *graph property*), that for a graph G and a vertex (or edge) subset S of G returns *true* if S is feasible solution and *false* otherwise. We interpret ϕ as a characterization of the space of all feasible solutions for an input graph G. Then let $\mathcal{H}_k$ be the collection of those subsets $X \subseteq V(G)$ (or $X \subseteq E(G)$), such that $|X| \leq k$, and $\phi_\Pi(G, S)$ is *true*. Further, let $w : V(G) \to \mathbb{R}^+$ be a weight function on the vertices (similarly for edges). Then, we define $\mathsf{Opt}_k = \min_{X \in \mathcal{H}_k} w(X)$, and $\mathsf{OPT} = \min_{k \in [n]} \mathsf{Opt}_k$.

Definition 1.1. *Let Π be a weighted parameterized graph minimization problem. For $\alpha, \beta > 0$, we say that Π admits an (α, β)-FPT-approximation algorithm, if given an instance (G, w, k) of Π, there exists an algorithm running in time $f(k) \cdot n^{\mathcal{O}(1)}$ such that, if $\mathcal{H}_k$ is non-empty, then it returns a set S of size at most αk (i.e. $S \in \mathcal{H}_{\alpha k}$), such that $\phi_\Pi(G, S)$ is true and $w(S) \leq \beta \cdot \mathsf{Opt}_k$, otherwise the output is arbitrary.*

Our FPT-approximation algorithms are guaranteed to output a solution that is (α, β)-

approximate should the given instance admit a solution of cardinality k; otherwise the output is arbitrary.

We prove the following results.

Theorem 1.4. *For every $\epsilon > 0$,* Weighted DFVS*,* Weighted Multicut *and* Weighted DOCT *admit a $(4, 8(1+\epsilon))$-FPT-approximation algorithm running in time $\frac{1}{\epsilon} \cdot 2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.*

We remark that the above algorithms provide constant-factor approximations in single-exponential FPT time; and are comparable to our results for the unweighted case.

These algorithms build upon a novel scheme to reduce the weighted problem to an unweighted instance and then invoking the FPT (approximation) algorithm for the unweighted problem on it. Let us discuss our methods via the example of DFVS. Consider an instance (D, w, k), and suppose that it admits a solution of cardinality k. A trivial reduction to the unweighted version of the problem is as follows: create $w(v)$ copies for each vertex $v \in V(D)$ (assuming for now that the weights are integral). This reduction however is not very helpful since the value of OPT (and Opt_k) might not be a function of k, and hence the unweighted instance is not amenable to an FPT (approximation) algorithm. We present a more nuanced reduction that avoids this issue, at a small cost to cost to the approximation factor and it is essentially independent of the weights themselves.

The first step of our reduction is to consider weighted instances where the weights are integral and upper-bounded by an integer M. Given such an instance (D, w, k), suppose that we know the value of Opt_k, and let $\gamma = \lceil \frac{\mathsf{Opt}_k}{k} \rceil$. Note that, we do not actually need to know the value of Opt_k and γ, since we know $\gamma \in [M]$ and will iterate over all choices for it. Next, we consider a new weight function w_γ that is obtained by rounding up the weight $w(v)$ of each vertex v to the nearest integral multiple of γ. We prove that the instance (D, w_γ, k) admits a solution of cost at most $2\mathsf{Opt}_k$. Then, from D and w_γ we construct an unweighted instance where for every vertex v we have $\frac{w_\gamma(v)}{\gamma}$ copies (note that this is an integer); we call the subset of copies of v as the *vertex bundle* for v, denoted by Z_v. Our key observation is that any minimal feedback vertex set for H *must respect the vertex bundles*, that is either it include all of Z_v or it is disjoint Z_v. From this we infer that if D admits a solution of cardinality k, then H admits a solution of cardinality $2k$. This means the unweighted instance $(H, 2k)$ may be approximated using an FPT algorithm that we discussed earlier. Further, we show that given a solution S' of cardinality $4k$ to this instance, we can map it back to a solution S of (D, w) such that $w(S) \leq 8\mathsf{Opt}_k$ and $|S| = |S'|$. Thus, for bounded weight instances we obtain a $(4, 8)$ FPT-approximation in single exponential FPT time.

The second step is to reduce from the general weighted instances to bounded weight instances. Here we make use of a knapsack like rounding procedure, that given an $\epsilon > 0$, at a cost of a factor $(1+\epsilon)$ to the approximation cost, produces weighted instances of DFVS where the weights are integral and upper-bounded by $\lceil \frac{k}{\epsilon} \rceil$. Then, combined with the previous step, we obtain a $(4, 8(1+\epsilon))$ FPT-approximation in single exponential time. Our methods easily extend to Weighted Multicut and Weighted DOCT, and we believe they can be applied to several other problem.

Finally, we present another FPT-approximation for Weighted DFVS that is able to achieve a $(1, 1+\epsilon)$ FPT-approximation, but at the cost of a higher running time. This algorithm builds upon an algorithm for MultiBudgeted DFVS. In this problem, the vertex set $V(D)$ of the input digraph is partitioned into a number of classes $V_1 \uplus V_2 \ldots V_\ell$, and the objective is to find a solution S such that for each $i \in [\ell]$ $|S \cap V_i| \leq k_i$, where the numbers $k_1, k_2, \ldots, k_\ell$ are also a part of the input. An FPT algorithm for this problem was presented by Kratsch et.al. [KLM+19b] that runs in time $2^{\mathcal{O}(k^3 \log k)} n^{\mathcal{O}(1)}$ where $k = \sum_{i=1}^{\ell} k_i$. We combine this algorithm with the knapsack like rounding procedure to obtain the following theorem.

Theorem 1.5. *For every $\epsilon > 0$,* Minimum Weight DFVS *admits a $(1, 1+\epsilon)$-FPT-approximation algorithm running in time $k^{k/\epsilon} \cdot 2^{\mathcal{O}(k^3 \log k)} n^{\mathcal{O}(1)}$*

We remark that one of our main motivations to study the MultiBudgeted DFVS problem was as an intermediate step towards an FPT algorithm for Weighted DFVS. The above theorem is an *FPT-approximation Scheme* (FPT-AS) for Weighted DFVS. Further, the above technique can be applied to any problem for which a "multi-budgeted" algorithm can be designed. These results are presented in Section 5.

To assist the reader with navigating the paper, we have included a Table of Contents in the Appendix.

2 Preliminaries

For an integer ℓ, let $[\ell]$ denote the set $\{1, 2, \ldots, \ell\}$. Let $w : A \to \mathbb{R}$ be a "weight" function. For any subset $A' \subseteq A$, we define the weight of A' as $w(A') = \sum_{a \in A'} w(a)$.

2.1 Graph Notation

Given a (di)graph G, we let $V(G)$ and $E(G)$ ($A(G)$) denote its vertex set and its edge (arc) set, respectively. When it is clear from context, $n = |V(G)|$ and $m = |E(G)|$. Given a subset $S \subseteq V(G)$, $G[S]$ denotes the subgraph of G induced by S, that is, the subgraph on vertex set S and edge set $\{\{u, v\} \in E(G) : u, v \in S\}$. Given subsets $S \subseteq E(G)$ and $T \subseteq E(G)$, $G - (S \cup T)$ is the graph on vertex set $V(G) \setminus S$ and edge set $E(G) \setminus (T \cup \{\{u, v\} \in E(G) : \{u, v\} \cap S \neq \emptyset\})$. A subset $S \subseteq V(G)$ is a *connected set* if $G[S]$ is a connected graph. The contraction of an edge $\{u, v\} \in E(G)$ yields the graph on vertex set $V(G - \{u, v\}) \cup \{r\}$ for some new vertex r and edge set $E(G - \{u, v\}) \cup \{\{r, w\} : \{u, w\} \in E(G)$ or $\{v, w\} \in E(G)$ (or both)$\}$. A family $\mathcal{F}$ of graphs is *hereditary* if for every graph $G \in \mathcal{F}$ and subset $S \subseteq V(G)$, $G - S \in \mathcal{F}$. Given a rooted tree T and a vertex $v \in V(T)$, we let T_v denote the subtree of T rooted at v. We say that a graph H is a *minor* of a graph G if there exists a sequence of vertex deletions, edges deletions and edge contractions in G that yields a graph isomorphic to H.

A bidirected graph is a digraph D where, for every arc $(u, v) \in A(D)$, the arc (v, u) is also present in D. The operation of *identifying* a vertex set X in a digraph D is defined as follows. We create a new vertex x' and define a function f as follows: for every $v \in V(D) \setminus X$, $f(v) = v$ and for every $v \in X$, $f(v) = x'$. Now, for every arc $(x, y) \in A(D)$, we add the arc $(f(x), f(y))$ (if it is not already present in D). Finally, we delete X. The resulting digraph is said to be obtained from D by identifying the vertices in X. Notice that in general, the identification operation could lead to self-loops and parallel edges. For a pair of vertices $a, b \in V(D)$, an *a-b walk* denotes a directed walk in D that starts at a and ends in b.

Treewidth is a structural parameter indicating how much a graph resembles a tree. Formally,

Definition 2.1. *A* tree decomposition *of a graph G is a pair $\mathcal{T} = (T, \beta)$ of a tree T and $\beta : V(T) \to 2^{V(G)}$, such that*

1. *for any edge $\{x, y\} \in E(G)$ there exists a node $v \in V(T)$ such that $x, y \in \beta(v)$, and*
2. *for any vertex $x \in V(G)$, the subgraph of T induced by $\{v \in V(T) : x \in \beta(v)\}$ is a non-empty tree.*

The width *of $\mathcal{T}$ is $\max_{v \in V(T)}\{|\beta(v)|\} - 1$. The* treewidth *$w$ of G is the minimum width over all tree decompositions of G.*

We define the treewidth of a directed graph G as the treewidth of the underlying undirected graph of G.

Definition 2.2. *A tree decomposition (T, β) of a graph G is* nice *if for the root r of T, $\beta(r) = \emptyset$, and each node $v \in V(T)$ is of one of the following types.*

- **Leaf***: v is a leaf in T and $\beta(v) = \emptyset$.*

- **Forget**: *v has one child, u, and there is a vertex $x \in \beta(u)$ such that $\beta(v) = \beta(u) \setminus \{x\}$.*
- **Introduce**: *v has one child, u, and there is a vertex $x \in \beta(v)$ such that $\beta(v) \setminus \{x\} = \beta(u)$.*
- **Join**: *v has two children, u and w, and $\beta(v) = \beta(u) = \beta(w)$.*

For $v \in V(T)$, $\beta(v)$ is called the *bag* of v, and $\gamma(v)$ is the union of the bags of v and its descendants in T. Further, we denote $\mathcal{T}_v = (T_v, \beta_v)$ where β_v is the restriction of β to $V(T_v)$. According to standard practice in Parameterized Complexity with respect to problems parameterized by treewidth, w, we assume that every input instance is given to us along with a tree decomposition of width w.

2.2 Optimization and Parameterized Complexity

Definition 2.3. *An* NP-optimisation problem *is defined as a tuple* $(I, \mathrm{sol}, \mathrm{cost}, \mathrm{goal})$ *where: (i) I is the set of instances. (ii) For an instance $x \in I$,* $\mathrm{sol}(x)$ *is the set of feasible solutions for x, the length of each $y \in \mathrm{sol}(x)$ is polynomially bounded in $|x|$, and it can be decided in time polynomial in $|x|$ whether $y \in \mathrm{sol}(x)$ holds for given x and y. (iii) Given an instance x and a feasible solution y,* $\mathrm{cost}(x, y)$ *is a polynomial-time computable positive integer. (iv)* $\mathrm{goal} \in \{\max, \min\}$.

The objective of an optimization problem is to find an optimal solution z for a given instance x, that is a solution z with $\mathrm{cost}(x, z) = \mathrm{opt}(x) := \mathrm{goal}\{\mathrm{cost}(x, y) \mid y \in \mathrm{sol}(x)\}$.

If y is a solution for the instance x then the *performance ratio* of y is defined as $R(x, y) = \mathrm{cost}(x, y)/\mathrm{opt}(x)$ (if goal = min) and $\mathrm{opt}(x)/\mathrm{cost}(x, y)$ (if goal = max). For a real number $c > 1$ (or a function $c : \mathbb{N} \to \mathbb{N}$), we say that an algorithm is a c-approximation algorithm if it always produces a solution with performance ratio at most c (respectively, $c(x)$).

Let Π be an NP-hard problem. In the framework of Parameterized Complexity, each instance of Π is associated with a *parameter k*. Here, the goal is to confine the combinatorial explosion in the running time of an algorithm for Π to depend only on k. Formally, we say that Π is *fixed-parameter tractable (FPT)* if any instance (I, k) of Π is solvable in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where f is an arbitrary function of k. Parameterized Complexity also provides methods to show that a problem is unlikely to be FPT. The main technique is the one of parameterized reductions analogous to those employed in classical complexity. Here, the concept of W[1]-hardness replaces the one of NP-hardness. Parameterization by solution size (or value) means that we seek a solution of size (or value) at most (for minimization) or at least (for maximization) k where k, the parameter, is given as part of the input. We remind that with respect to graph problems parameterized by the treewidth of the input graph, w, we assume that every input instance is given to us along with a tree decomposition of width w. We remark that for other structural parameterizations, when the parameter is computable in polynomial time (e.g., the size of a maximum matching in the graph), the input instance does not have additional arguments, and the parameter is thus implicit.

When the parameter k is structural (e.g., treewidth), a factor-$c(k)$ FPT-approximation algorithm for X is an algorithm that, given input (x, k) (where k can be implicit), runs in time $f(k) \cdot |x|^{\mathcal{O}(1)}$ and computes a $y \in \mathrm{sol}(x)$ such that $\mathrm{cost}(x, y) \leq \mathrm{opt}(x) \cdot c(k)$. The definition for maximization problems is symmetric. When the parameterization is by solution size or value, we define FPT-approximation as follows.

Definition 2.4 (Standard FPT-approximation for minimization)**.** *Let $X = (I, sol, cost, goal)$ be a minimization problem. A standard factor-$c(k)$ FPT-approximation algorithm for X (where the parameterization is by solution size or value) is an algorithm that, given input (x, k) satisfying* $\mathrm{opt}(x) \leq k$, *runs in time $f(k) \cdot |x|^{\mathcal{O}(1)}$ and computes a $y \in \mathrm{sol}(x)$ such that* $\mathrm{cost}(x, y) \leq k \cdot c(k)$. *For inputs not satisfying* $\mathrm{opt}(x) \leq k$, *the output can be arbitrary.*

The definition for maximization problems is symmetric.

Remark 1. *In this paper, we will refer to standard FPT-approximation algorithms as simply FPT-approximation algorithms when the parameterization by solution size is clear. Moreover, for all unweighted graph minimization problems we consider, feasible solutions will be vertex or edge subsets and the cost of a solution will be the size of the set.*

To obtain (essentially) tight conditional lower bounds for the running times of algorithms, we rely on the well-known *Exponential-Time Hypothesis (ETH)* and *Strong Exponential-Time Hypothesis (SETH)*. To formalize the statements of ETH and SETH, first recall that given a formula φ in conjuctive normal form (CNF) with n variables and m clauses, the task of CNF-SAT is to decide whether there is a truth assignment to the variables that satisfies φ. In the p-CNF-SAT problem, each clause is restricted to have at most p literals. First, ETH asserts that 3-CNF-SAT cannot be solved in time $\mathcal{O}(2^{o(n)})$. Second, SETH asserts that for every fixed $\epsilon < 1$, there exists a (large) integer $p = p(\epsilon)$ such that p-CNF-SAT cannot be solved in time $\mathcal{O}((2-\epsilon)^n)$. We remark that ETH implies FPT$\neq$W[1], and that SETH implies ETH.

A companion notion to that of FPT is the one of a kernel. Formally, a *decision* parameterized problem Π is said to admit a *compression* if there exists a (not necessarily parameterized) problem Π' and a polynomial-time algorithm that given an instance (I, k) of Π, outputs an equivalent instance I' of Π' (that is, (I, k) is a yes-instance of Π if and only if I' is a yes-instance of Π') such that $|I'| \leq p(k)$ where p is some computable function that depends only on k. In case $\Pi' = \Pi$, we further say that Π admits a *kernel*. More broadly, to accommodate optimization and approximation, we rely on the more general notion of *lossy kernelization*. We define the notion of lossy kernelization in a more restricted way than [LPRS17] that will suffice for our purposes.

Definition 2.5 (Lossy kernelization (restricted version))**.** *Let Π be a parameterized minimization problem, parameterized by the solution size. Let $\alpha \geq 1$. An α-*approximate kernelization algorithm *for Π consists of two polynomial-time procedures:* **reduce** *and* **lift***. Given an instance I of Π with parameter k,* **reduce** *outputs another instance I' of Π with parameter k' such that $|I'| \leq f(k', \alpha), k' \leq k$, and where $\frac{k'}{\mathsf{opt}(I')} \leq \frac{k}{\mathsf{opt}(I)}$.*[3] *Given I, I' and a solution S' to I',* **lift** *outputs a solution S to I such that, if $\mathsf{opt}(I) \leq k$, then $\frac{|S|}{\mathsf{opt}(I)} \leq \alpha \frac{|S'|}{\mathsf{opt}(I')}$ (otherwise, S can be of any size).*

In case of a graph problem and when the output graph has $f(k)$ vertices, we say that the kernel (in the above definition) is an α-approximate $f(k)$-vertex kernel. When $f(k)$ is linear in k, we use the term α-approximate linear-vertex kernel.

3 Single-exponential Constant-factor approximations

In this section, we present the first single-exponential-time factor-2 FPT-approximations for some well-studied cut problems – (Subset) Directed Feedback Vertex Set, Undirected Multicut and Directed Odd Cycle Transversal (see Appendix for formal definitions).

3.1 Setting Up the Machinery

In Definitions 3.1–3.5, Observation 3.1, Proposition 3.1, fix a digraph D and disjoint $X, Y \subseteq V(D)$.

Definition 3.1. *We denote by $\mathsf{rel}_D(X, Y)$ the set of all vertices that lie in a strongly connected component of $D - Y$ intersected by X. We denote by $\mathsf{conn}_D(X, Y)$ the set of all vertices that lie on an x_1-x_2 walk in $D - Y$ for some $x_1, x_2 \in X$. When $Y = \emptyset$, we simply write, $\mathsf{rel}_D(X)$ and $\mathsf{conn}_D(X, Y)$ and drop the subscript if D is clear from the context.*

[3]Often, the requirement $|I'| \leq f(k', \alpha)$ is replaced by the more relaxed requirement $|I'| \leq f(k, \alpha)$. However, as all the (known) kernels we will use (as black boxes) have this property, we directly define it like this. Further, the requirement is implicit for the definition to be sensible (without using a π function as in [LPRS17]).

Notice that in the above definition, it is possible that $x_1 = x_2$ and hence $\mathsf{conn}_D(X, Y) \supseteq \mathsf{rel}_D(X, Y)$.

Definition 3.2 (Separators)**.** *A vertex set S disjoint from $X \cup Y$ is called an X-Y* separator *if there is no X-Y path in $D - S$. We say that S is a* minimal X-Y separator *if no strict subset of S is also an X-Y separator. We denote by $R_D(X, S)$ the set of vertices reachable from vertices of X via directed paths in $D - S$ and by $NR_D(X, S)$ the set of vertices* not *reachable from vertices of X in $D - S$. The subscript is ignored if the digraph D is clear from the context.*

Definition 3.3 (Covering and dominating relations between separators)**.** *Let S_1 and S_2 be X-Y separators. We say that S_2* covers *S_1 (denoted by $S_1 \sqsubseteq S_2$) if $R(X, S_1) \subseteq R(X, S_2)$ and we say that S_2* dominates *S_1 (denoted by $S_1 \preceq S_2$) if S_2 covers S_1 and $|S_2| \leq |S_1|$.*

When S_2 covers S_1, we also say that S_1 is covered by S_2.

Observation 3.1. *Let S_1 and S_2 be minimal X-Y separators such that $S_1 \sqsubseteq S_2$. Then, $S_2 \setminus S_1 \subseteq NR(X, S_1)$. That is, $S_2 \setminus S_1$ is unreachable from X in $D - S_1$. Similarly, $Y \subseteq NR(S_1 \setminus S_2, S_2)$. That is, Y is unreachable from $S_1 \setminus S_2$ in $D - S_2$.*

Definition 3.4 (Important separators)**.** *Let S be a minimal X-Y separator. We say that S is an* important X-Y separator closest to Y *if there is no X-Y separator S' that dominates S. Similarly, we say that S is an* important X-Y separator closest to X *if there is no X-Y separator S' that is dominated by S. Following standard terminology, we simply use the term important X-Y separator, then we are referring to one closest to Y.*

Proposition 3.1. [CFK+15] *The number of important X-Y separators of size at most k closest to Z (for each $Z \in \{X, Y\}$) is bounded by 4^k. Moreover, these can be enumerated in time $4^k(m + n)$.*

Definition 3.5. *Let D be a directed graph. A directed closed walk in D (a directed walk that starts and ends at the same vertex) with an odd number of edges is called a* directed odd closed walk*. For a set $T \subseteq V(D) \cup A(D)$, a directed closed walk in D is said to be a T*-closed walk *if it contains an element from T. A T-closed walk is called a T*-cycle *if it is a simple cycle. A set $S \subseteq V(D)$ is called a T*-sfvs *if it intersects every T-cycle in D.*

Let $\mathcal{F} = \{F_1, F_2, \ldots, , F_q\}$ be a fixed set of subgraphs of a digraph D such that $\mathcal{F}$-free subgraphs of D are closed under taking subgraphs. An *$\mathcal{F}$-transversal* in D is a set of vertices that intersects every $F_i \in \mathcal{F}$. The family $\mathcal{F}$ could be exponentially large, in which case it is implicitly defined. In our work, we are interested in problems that can be formulated as computing a smallest $\mathcal{F}$-transversal where the graphs in $\mathcal{F}$ are all strongly connected. We refer to this problem as SCC $\mathcal{F}$-Transversal.

The minimization version of SCC $\mathcal{F}$-Transversal is the tuple $(\mathcal{I}, \text{sol}, \text{cost}, \min)$, where, $\mathcal{I}$ is the set of digraphs, for every $x \in \mathcal{I}$, sol(x) denotes the set of $\mathcal{F}$-transversals in x. Moreover, for every feasible solution y, cost(x, y) denotes the size of the vertex set y. Recall that for every $c \in \mathbb{R}$, a (standard) factor-c FPT-approximation algorithm for SCC $\mathcal{F}$-Transversal is an algorithm that, on input (D, k), runs in time $f(k) \cdot n^{\mathcal{O}(1)}$ (for some computable f) and if there is an $\mathcal{F}$-transversal in D of size at most k, then it outputs an $\mathcal{F}$-transversal in D of size $\leq ck$.

Observation 3.2. Subset DFVS, Directed OCT, Bidirected Multicut *are special cases of $\mathcal{F}$-transversal.*

Proof. To cast an instance (D, T) of Subset DFVS as one of SCC $\mathcal{F}$-Transversal, take $\mathcal{F}$ to be the set of T-cycles. In the case of DOCT, we take $\mathcal{F}$ to be the set of all directed odd cycles in the input graph. Consider an instance $(D, \mathcal{T})$ of BiMC where D is a bidirected graph and take $\mathcal{F}$ to be the subgraphs induced by the vertex sets of the s_i-t_i paths in D where $(s_i, t_i) \in \mathcal{T}$. □

The following lemma is at the heart of the algorithms in this section.

Lemma 3.1. *Let $\widetilde{S}$ be an $\mathcal{F}$-transversal in D. Let $W = W_1 \uplus W_2$ be an $\mathcal{F}$-transversal in D such that for some $\emptyset \neq S \subseteq \widetilde{S}$, S is a minimal W_1-W_2 separator. Let X_{pre} and X_{post} be W_1-W_2 separators in D such that $X_{\mathsf{pre}} \sqsubseteq S \sqsubseteq X_{\mathsf{post}}$. Then, $\widetilde{S} \setminus S$ is an $\mathcal{F}$-transversal in the graph $D' = D - (X_{\mathsf{pre}} \cup X_{\mathsf{post}})$.*

Proof. Suppose that this is not the case. Then, there is a graph $F \in \mathcal{F}$ that is contained in $D'' = D' - (\widetilde{S} \setminus S)$. Since $\widetilde{S}$ and W are both $\mathcal{F}$-transversals in D, it follows that F is a strongly connected subgraph of D'' that intersects both S and W. This, in turn, implies that there is a closed walk in D'' that intersects some $s \in S \setminus (X_{\mathsf{pre}} \cup X_{\mathsf{post}})$ and some $w \in W$. Since, $X_{\mathsf{pre}} \sqsubseteq S \sqsubseteq X_{\mathsf{post}}$, we have that $S \setminus X_{\mathsf{pre}}$ is unreachable from W_1 in $D - X_{\mathsf{pre}}$ and W_2 is unreachable from $S \setminus X_{\mathsf{post}}$ in $D - X_{\mathsf{post}}$ (see Observation 3.1). This gives a contradiction to our assumption that there is a closed walk in D'' that contains s and w, completing the proof of the lemma. $\square$

As an immediate consequence of Lemma 3.1, we have the following.

Lemma 3.2. *Let $D, W_1, W_2, \widetilde{S}, S$ be as defined in Lemma 3.1. Then, there exists an important W_1-W_2 separator closest to W_1 of size at most $|S|$, call it X_{pre}, and an important W_1-W_2 separator closest to W_2 of size at most $|S|$, call it X_{post}, such that $\widetilde{S} \setminus S$ is an $\mathcal{F}$-transversal in $D' = D - (X_{\mathsf{pre}} \cup X_{\mathsf{post}})$.*

Therefore, if we knew W_1, W_2 and $|S|$, then we can guess X_{pre} and X_{post} and make some $|S|$ vertices of $\widetilde{S}$ "irrelevant" (and reducing the size of the optimal solution by $|S|$) by paying a cost of at most $2|S|$. This property forms the crux of our approximation algorithms for the special cases of SCC $\mathcal{F}$-Transversal considered in this section. However, embedding this idea into our algorithms is not straightforward and requires some care.

3.2 Subset DFVS

Using the reduction in [CCHM15], we work with the following equivalent formulation of Subset DFVS where the terminals are arcs instead of vertices, as is usually the case. We continue to refer to this problem as Subset DFVS instead of the term Edge Subset DFVS used in [CCHM15]. As proved in Observation 3.2, Subset DFVS is a special case of SCC $\mathcal{F}$-Transversal as one can simply take $\mathcal{F}$ to be the set of T-cycles. Moreover, notice that the existence of T-cycles is equivalent to the existence of T-closed walks. In order to design our FPT-approximation for Subset DFVS, we first consider a special case.

A factor-c FPT-approximation algorithm for Strict Subset DFVS is an algorithm that, on input (D, T, W, k) where (D, T) is an input to the minimization version of Subset DFVS and W is a T-sfvs in D, runs in time $f(k) \cdot n^{\mathcal{O}(1)}$ (for some computable f) and if there is a T-sfvs S in D of size at most k such that W is contained in a unique strongly connected component of $D - S$, then it outputs a T-sfvs in D of size at most ck. Otherwise, the output of the algorithm can be arbitrary.

Lemma 3.3. *There is a factor-1 FPT-approximation algorithm for* Strict Subset DFVS *with running time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. We call this algorithm* Alg-Strict-SFVS.

Proof. Let $I = (D, T, W, k)$ be the given input. We may assume that D is a strongly connected graph. Otherwise, we work individually over each strongly connected component. We first check whether there is an arc $(u,v) \in T$ such that $u, v \in W$. If yes, then we terminate the algorithm with an arbitrary output. This is correct since there is no T-sfvs S in D such that W is contained in a unique strongly connected component of $D - S$. Henceforth, we assume that $D[W]$ is an independent set.

Our next step is to construct a new tuple $I' = (D', T', w, k)$ where D' is obtained from D by identifying the vertices in W (with parallel arcs removed), w is the new vertex created in place of W by this operation and T' is obtained by updating T accordingly. That is, T' contains the arcs $\{(x,y) \in T \mid x, y \notin W\}$ plus the arcs $\{(w,y) \mid \exists x \in W, \exists (x,y) \in T\}$ and $\{(x,w) \mid \exists y \in W, \exists (x,y) \in T\}$. Since we are in the case where $D[W]$ is an independent set, there are no self-loops incident on w. Notice that D' is strongly connected since D is assumed to be strongly connected. We now have the following claim.

Claim 3.1. *The following statements hold.*

1. *w is a T-sfvs in D'.*
2. *Every T-sfvs S in D that is disjoint from W such that W is contained in a unique strongly connected component of $D - S$, is a T'-sfvs in D' that is disjoint from w.*
3. *Conversely, every T'-sfvs in D' disjoint from w is a T-sfvs in D.*

Proof. The first statement follows from the fact that a T'-cycle in D' that is disjoint from w would have to be a T-cycle in D that is disjoint from W, which cannot exist.

For the second statement, suppose for a contradiction that S is not a T'-sfvs in D'. Let H be a T'-closed walk in $D' - S$. We may assume without loss of generality that H is a simple cycle. Notice that we are guaranteed (in the premise) that there is a w_1-w_2 path in $D - S$ for every $w_1, w_2 \in W$. Moreover, in the first statement of the lemma, we have argued that H must intersect w. Let p and q denote the in-neighbor and out-neighbor of w respectively, in H. Then, $p \in N_D^-(w_1)$ and $q \in N_D^+(w_2)$ for some $w_1, w_2 \in W$. We choose w_1 and w_2 such that if $(p, w) \in T'$ (or $(w, r) \in T'$), then $(p, w_1) \in T$ (respectively, $(w_2, r) \in T$). By the construction of D' and T', such w_1 and w_2 exist. Let P denote a w_1-w_2 path in $D - S$. Then, replacing the path p, w, r in H with the walk obtained by concatenating the arc (p, w_1), the w_1-w_2 path P, and the arc (w_2, r) gives us a T-closed walk in $D - S$, a contradiction to our assumption that S is a T-sfvs in D.

For the final statement, suppose for a contradiction that for some T'-sfvs S in D' disjoint from w, there is a T-cycle H in $D - S$. Then, by identifying the vertices of W on H, we obtain a T'-closed walk in $D' - S$, a contradiction.

This completes the proof of the claim. $\square$

Due to the above claim, in the rest of the proof of the lemma, it is sufficient to describe an algorithm that, given $I' = (D', T', w, k)$, runs in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ and either outputs a T'-sfvs of size at most k disjoint from w (we refer to such sets as a solution for I' in the rest of the proof) or correctly concludes that one does not exist.

Claim 3.2. *Let S be a solution for I'. For every $(u,v) \in T'$, either $\{u,v\} \cap S \neq \emptyset$ or there is a solution for I' that contains an important x-w separator closest to w or an important w-x separator closest to w for some $x \in \{u,v\}$.*

Proof. Let S be a T'-sfvs in D' of size at most k disjoint from w and let C be the strongly connected component of $D' - S$ that contains w. We first observe that for every $e = (u,v) \in T'$, it cannot be the case that $\{u,v\} \subseteq C$. Otherwise, we would contradict S being a T'-sfvs in D'. This implies that either at least one of u or v is contained in S, or S intersects all w-x or x-w paths for some $x \in \{u,v\}$. It remains for us to argue that if S intersects all w-x or x-w paths for some $x \in \{u,v\}$, then there is a solution S' that contains an important w-x separator closest to w or an important x-w separator closest to w for some $x \in \{u,v\}$. We only argue the case where S intersects all w-x paths for some $x \in \{u,v\}$. The other case is analogous.

Let $\widehat{S} \subseteq S$ be a minimal w-x separator in D'. Since D' is strongly connected, it must be the case that $\widehat{S}$ is non-empty. Now, consider an important w-x separator $\widetilde{S}$ of size at most $|\widehat{S}|$ that is covered by $\widehat{S}$. We claim that $(S \setminus \widehat{S}) \cup \widetilde{S}$ is also a solution for I'. If this were not the case, then there would be a closed walk intersecting w and a vertex $s \in \widehat{S} \setminus \widetilde{S}$ that is disjoint from $\widetilde{S}$, a contradiction to Observation 3.1, which guarantees the absence of w-s paths in the graph $D' - \widetilde{S}$. This completes the proof of the claim. $\square$

Given the above claim, we make use of a standard important-separator branching routine (see [CFK+15] for an exposition) to obtain an algorithm that does the following: It picks an arc $(u, v) \in T'$, in the first two branches, it branches by deleting one of u, v and adding it to the solution. In the remaining four branches, it enumerates all important u-w separators closest to w, important v-w separators closest to w, important w-u separators closest to w and important w-v separators closest to w, each of size at most k and adds one of them to the solution. Finally, if there is a leaf at which the vertices added to the solution form a T-sfvs (which can be checked in polynomial-time) of size at most k, then we return such a solution. Otherwise, we terminate with an arbitrary output.

The correctness of the algorithm follows from Claim 3.1 and Claim 3.2. Indeed, from Claim 3.1, we have that for every T-sfvs S in D of size at most k such that W is contained in a unique strongly connected component of $D - S$, then S is a solution for I'. Moreover, Claim 3.2 guarantees that for every $(u, v) \in T'$, either one of u or v must be in S or our important separator branching procedure is correct.

Standard important separator analysis with a branching measure of $2k - \lambda(y, z)$ (where we are enumerating important y-z separators and $\lambda(y, z)$ denotes the size of smallest y-z separator) shows that we have a branching algorithm with branching vector $(2, 2, 1, 1, 1, 1)$, bounding the number of leaves in our search tree by γ^k (where $\gamma = 10 + 4\sqrt{6}$) and overall running time by $\gamma^k n^{\mathcal{O}(1)}$ since we only require polynomial time at each node. This completes the proof of the lemma. $\square$

Lemma 3.4. *Let D be a digraph, $T \subseteq A(D)$, and let W and S be disjoint T-sfvs in D. Let $\emptyset \neq W' \subseteq W$ be such that in $D - S$, there is a strongly connected component whose intersection with W is precisely W'. Consider the graph D' obtained from D by adding a bidirected clique on W' (i.e., we add an arc (w, w') for every $w, w' \in W'$ such that $(w, w') \notin A(D)$). Then, W and S are both T-sfvs in D'.*

Proof. It is straightforward to see that W remains a T-sfvs in D' since all arcs in $A(D') \setminus A(D)$ are incident on W. Suppose for a contradiction that S is not a T-sfvs in D' and consider a T-cycle H. Moreover, for every $(p, q) \in A(H) \setminus A(D)$, it must be the case that $p, q \in W'$, implying that there is a p-q path in $D - S$. Hence, we can replace every such arc with the corresponding path in $D - S$ to obtain a T-closed walk in $D - S$, a contradiction to S being a T-sfvs in D. This completes the proof of the lemma. $\square$

We are now ready to present the algorithm for Subset DFVS, which uses Algorithm Alg-Strict-SFVS as a subroutine.

Theorem 3.1. *There is a factor-2 FPT-approximation algorithm for* Subset DFVS *with running time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.*

Proof. By using the iterative compression technique, we reduce our goal to designing an algorithm that, on input (D, T, W, k), where (D, T) is an instance of Subset DFVS, $k \geq 0$ and W is a T-sfvs in D, runs in time $2^{\mathcal{O}(k+|W|)} n^{\mathcal{O}(1)}$ and if there is a T-sfvs S in D of size at most k that is disjoint from W, then it outputs a T-sfvs in D of size at most $2k$. Otherwise, the output of the algorithm can be arbitrary. Indeed, suppose that such an algorithm (which we call Algorithm Alg-Disjoint-SFVS) exists. Then, one can immediately obtain an algorithm Alg-Compression-SFVS

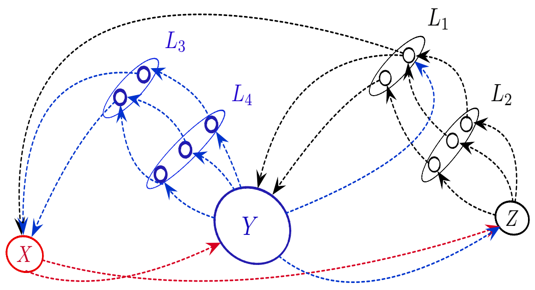

Figure 1: An illustration of the sets $X \uplus Y \uplus Z = W$ and the separators $L_1, \ldots, L_4$. The dotted arrows represent paths.

that, on input (D, T, W, k), runs in time $2^{\mathcal{O}(k+|W|)} n^{\mathcal{O}(1)}$ and if there is a T-sfvs S in D of size at most k that is *not necessarily disjoint from* W, then it outputs a T-sfvs in D of size at most $2k$.

Now, suppose that $V(D) = \{v_1, \ldots, v_n\}$ and for every $i \in [n]$, $V_i = \bigcup_{j=1}^{i} v_j$. Furthermore, for every $X \subseteq V(D)$, let $T[X] = \{(x, y) \in T \mid x, y \in X\}$. Then, we construct instances $I_1, \ldots, I_n$ where $I_i = (D[V_i], T[V_i], W_i, k)$, $W_1 = \{v_1\}$, for every $i > 1$, $W_i \leftarrow \{v_i\} \cup \mathsf{Alg\text{-}Compression\text{-}SFVS}(I_{i-1})$. Moreover, for the first occurrence of an i for which W_i is not a T-sfvs of size at most $2k$ in $D[V_i]$, we terminate and return an arbitrary vertex set. It is straightforward to see that assuming the correctness and claimed running time of $\mathsf{Alg\text{-}Disjoint\text{-}SFVS}$, we have the required factor-2 FPT-approximation for Subset DFVS.

We now proceed to describe Algorithm $\mathsf{Alg\text{-}Disjoint\text{-}SFVS}$. In the base case of this algorithm, $k \leq 1$ or $|W| = 1$. In either case, can solve the instance by brute force. If $k \leq 1$, then it is sufficient for us to check whether there is a T-cycle in D and if yes, whether there is a T-sfvs in D of size at most 1. If $k > 1, |W| = 1$, then we can simply return W. Hence, we assume that $k, |W| > 1$. Moreover, we assume that D is strongly connected. Otherwise, we can simply work with the subinstance induced by each strongly connected component.

Let $\mathcal{P}$ denote the set of all 3-partitions of W into sets (X, Y, Z). For every $\tau = (X, Y, Z) \in \mathcal{P}$, we define the following sets and tuples. Let $1 \leq i, j \leq k$.

- $\mathcal{L}^i[Z \to XY]$ denotes the set of all important Z-$X \cup Y$ separators of size at most i closest to $X \cup Y$.
- $\mathcal{L}^i[XY \leftarrow Z]$ denotes the set of all important Z-$X \cup Y$ separators of size at most i closest to Z.

Recall that both these sets have size at most 4^i (Proposition 3.1). When $i = 0$, we assume that these sets only contain $\emptyset$. Moreover, if Z or $X \cup Y$ is empty, then $\mathcal{L}^i[Z \to XY]$ and $\mathcal{L}^i[XY \leftarrow Z]$ are empty for every i.

In the following, let $1 \leq i, j \leq k$, $L_1 \in \mathcal{L}^j[Z \to XY]$, $L_2 \in \mathcal{L}^j[XY \leftarrow Z]$.

- $\mathcal{L}^i[Y \to X, L_1, L_2]$ denotes the set of all important Y-X separators of size at most i closest to X in $D - (L_1 \cup L_2)$.
- $\mathcal{L}^i[X \leftarrow Y, L_1, L_2]$ denotes the set of all important Y-X separators of size at most i closest to Y in $D - (L_1 \cup L_2)$.

When $i = 0$, we assume that these sets only contain $\emptyset$. Moreover, if X or Y is empty, then $\mathcal{L}^i[Y \to X, L_1, L_2]$ and $\mathcal{L}^i[X \leftarrow Y, L_1, L_2]$ are empty for every i.

To help readability, in the rest of proof, we will forgo the notation $T[Q]$ when referring to the arcs of T with both endpoints in Q, because the vertex set Q will always be clear from the context. Abusing notation in this way, we will continue to refer to the set of terminals as T even when referring to subinstances that do not contain some arcs in T. Now, for every $Q \subseteq V(D)$, we define the following:

- $I[Q, Z, i]$ denotes the tuple $(D[\mathsf{rel}(Z, Q)], T, Z, i)$. That is, the subinstance induced by those vertices that are in the strongly connected components intersected by Z after deleting Q. Similarly, we define the following tuples.
- $I[Q, XY, i]$ denotes $(D[\mathsf{rel}(X \cup Y, Q)], T, X \cup Y, i)$.
- $I[Q, X, i]$ denotes $(D[\mathsf{rel}(X, Q)], T, X, i)$.
- $\widetilde{I}[Q, Y, i]$ denotes (D', T, Y, i), where D' is the graph obtained from $D[\mathsf{rel}(Y, Q)]$ by adding a bidirected clique on Y. That is, we construct the subinstance induced by those vertices that are in the strongly connected components intersected by Y after deleting Q and then add an arc (w, w') for every $w, w' \in Y$ such that $(w, w') \notin A(D)$.

We are now equipped with the notation required to describe the rest of the algorithm. To ease readability, we interleave the steps of the algorithm and intuitive descriptions and observations related to these.

Main loop: For every $(X, Y, Z) \in \mathcal{P}$ such that $|W|/3 \leq |X \cup Y|, |Z| \leq 2|W|/3$ or $|Y| > |W|/3$, and for every i_1, i_2 such that $1 \leq i_1 \leq k$ and $k_1 = i_1 + i_2 \leq k$, we do the following:

Step 1: We guess $L_1 \in \mathcal{L}^{i_1}[Z \to XY]$, $L_2 \in \mathcal{L}^{i_1}[XY \leftarrow Z]$, $L_3 \in \mathcal{L}^{i_2}[Y \to X, L_1, L_2]$, $L_4 \in \mathcal{L}^{i_2}[X \leftarrow Y, L_1, L_2]$ (see Figure 1). Set $Q = \bigcup_{q \in [4]} L_q$.

That is, we guess a pair of important Z-$(X \cup Y)$ separators of size at most i_1 in D, one that is closest to Z and another that is closest to $X \cup Y$. Following this, we delete $L_1 \cup L_2$ and guess a pair of important Y-X separators of size at most i_2 in $D - (L_1 \cup L_2)$, one that is closest to Y and another that is closest to X. This guessing step is implemented as follows. Using the important separator enumeration algorithm [CFK+15], we obtain a branching algorithm that takes polynomial time in each step and produces at most $4^{i_1} \cdot 4^{i_1} \cdot 4^{i_2} \cdot 4^{i_2} = 2^{4(i_1+i_2)} = 2^{4k_1}$ leaves, where each leaf corresponds to a guess of L_1, L_2, L_3, L_4.

Notice that deleting L_1 and L_2 breaks up the original instance into two disjoint pieces comprising the vertices in $\mathsf{rel}_D(Z)$ and $\mathsf{rel}_D(X \cup Y)$. Additionally, deleting L_3 and L_4, breaks up the original instance into three disjoint pieces comprising the vertices in $\mathsf{rel}_D(Z)$, $\mathsf{rel}_D(X)$ and $\mathsf{rel}_D(Y)$. We will use this crucially in our algorithm as follows.

Step 2: If $|W|/3 \leq |X \cup Y|, |Z| \leq 2|W|/3$, then for every i_3, i_4 such that $i_3 + i_4 \leq k - k_1$, we recursively compute:

(i) $S_Z \leftarrow \textsf{Alg-Disjoint-SFVS}(I[Q, Z, i_3])$.
(ii) $S_{XY} \leftarrow \textsf{Alg-Disjoint-SFVS}(I[Q, XY, i_4])$.

If $\Delta = Q \cup S_Z \cup S_{XY}$ is a T-sfvs in D of size at most $2k$, then we return Δ.

Note that if Z or $X \cup Y$ is empty, then above two instances are empty. We allow Alg-Disjoint-SFVS to take empty instances with the promise that the output is always the empty set. We argue that the instances $I[Q, Z, i_3]$ and $I[Q, XY, i_4]$ are valid input instances to Alg-Disjoint-SFVS as follows. Notice that from the definition of L_1 and L_2 as Z-$(X \cup Y)$ separators in D, it follows that $X \cup Y$ is disjoint from $\mathsf{rel}(Z, L_1 \cup L_2)$ and Z is disjoint from $\mathsf{rel}(X \cup Y, L_1 \cup L_2)$. Moreover, $X \cup Y \cup Z$ is a T-sfvs in D. Hence, we conclude that Z and $X \cup Y$ are T-sfvs in $D[\mathsf{rel}(Z, Q)]$ and $D[\mathsf{rel}(X \cup Y, Q)]$ respectively, validating $I[Q, Z, i_3]$ and $I[Q, XY, i_4]$ as input instances to Alg-Disjoint-SFVS.

Step 3: If Step 2 does not apply and $|Y| > |W|/3$, then for every i_3, i_4, i_5 such that $i_3 + i_4 + i_5 = k - k_1$, we compute:

(i) $S_Z \leftarrow \textsf{Alg-Disjoint-SFVS}(I[Q, Z, i_3])$.
(ii) $S_X \leftarrow \textsf{Alg-Disjoint-SFVS}(I[Q, X, i_4])$.
(iii) $S_Y \leftarrow \textsf{Alg-Strict-SFVS}(\widetilde{I}[Q, Y, i_5])$.

If $\Delta = Q \cup S_Z \cup S_X \cup S_Y$ is a T-sfvs in D of size at most $2k$, then we return Δ.

The argument for the validity of $I[Q, Z, i_3]$ and $I[Q, X, i_4]$ as inputs to the recursive calls to Alg-Disjoint-SFVS follows along the same line as the arguments used following the previous step. That is, since Q is a Z-$(X \cup Y)$ separator and a Y-X separator, it follows that Z is a T-sfvs in $D[\mathsf{rel}(Z, Q)]$ and X is a T-sfvs in $D[\mathsf{rel}(X, Q)]$. Moreover, we have that Y is a T-sfvs in $D[\mathsf{rel}(Y, Q)]$. Now, since every arc in $A(D') \setminus A(D)$ is incident on Y, it follows that Y is also a T-sfvs in D'. This implies that $\widetilde{I}[Q, Y, i_5]$ is a valid input to Alg-Disjoint-SFVS.

If the algorithm completes iterating through the main loop without returning, then we return an arbitrary vertex set. This completes the description of the algorithm.

Correctness. The correctness is proved by induction on $|W|$. In the base case, $|W| = 1$, in which case, the algorithm works by brute-force and hence is correct. Now, we assume that $|W| > 1$. Suppose that there is a T-sfvs S in D of size at most k, such that $S \cap W = \emptyset$. Our aim is to show that the algorithm outputs a T-sfvs of size at most $2k$.

Let $(M_1, \ldots, M_r)$ denote the partition of W such that (i) each M_i is contained in a strongly connected component of $D - S$, and (ii) for every $\ell_1 > \ell_2$, there is no path in $D - S$ from $\mathsf{rel}_D(M_{\ell_1}, S)$ to $\mathsf{rel}_D(M_{\ell_2}, S)$. That is, S is an M_{ℓ_1}-M_{ℓ_2} separator for every $\ell_1 > \ell_2$. We now consider the following two cases.

Case 1: $|M_\ell| \le |W|/3$ for every $\ell \in [r]$. Let $\ell' \in [r]$ denote the least value such that $|W|/3 < \Sigma_{i=1}^{\ell'} |M_i|$. Then, $|W|/3 < \Sigma_{i=1}^{\ell'} |M_i| \le 2|W|/3$. Define $X = \emptyset$, $Y = \bigcup_{i=1}^{\ell'} M_i$ and $Z = \bigcup_{i=\ell'+1}^{r} M_i$. Then, we have that $|W|/3 \le |X \cup Y|, |Z| \le 2|W|/3$. Therefore, when considering the partition (X, Y, Z), **Step 2** would have been executed.

Let S_1 be a minimal subset of S that intersects all Z-$X \cup Y$ paths in D. Let $i_1 = |S_1|$. Since D is strongly connected, it follows that $i_1 > 0$. Lemma 3.2 guarantees that there exist $L_1 \in \mathcal{L}^{i_1}[Z \to XY]$ and $L_2 \in \mathcal{L}^{i_1}[XY \leftarrow Z]$ such that $S' = S \setminus S_1$ is a T-sfvs in $D - (L_1 \cup L_2)$. Let $i_2 = 0$. This implies that $L_3 = L_4 = \emptyset$. Let $Q = \bigcup_{q \in [4]} L_q$. Now, define:

- $S'_Z = S' \cap \mathsf{rel}(Z, Q)$, $i_3 = |S'_Z|$.
- $S'_{XY} = S' \cap \mathsf{rel}(X \cup Y, Q)$, $i_4 = |S'_{XY}|$.

Notice that S'_{XY} intersects all T-cycles in $D - Q$ that intersect $X \cup Y$ and S'_Z intersects all T-cycles in $D - Q$ that intersect Z. Conversely, one can obtain a T-sfvs in $D - Q$ by taking the union of any set that intersects all T-cycles in $D - Q$ that intersect $X \cup Y$ and any set that intersects all T-cycles in $D - Q$ that intersect Z. This is becaue $W = X \cup Y \cup Z$ is a T-sfvs in D.

Therefore, by the induction hypothesis, S_Z is a T-sfvs of size at most $2i_3$ in $D[\mathsf{rel}(Z, Q)]$ and S_{XY} is a T-sfvs of size at most $2i_4$ in $D[\mathsf{rel}(X \cup Y, Q)]$. As argued, the set $Q \cup S_Z \cup S_{XY} = L_1 \cup L_2 \cup S_Z \cup S_{XY}$ is a therefore a T-sfvs in D. Moreover, it has size at most $2(i_1 + i_3 + i_4) \le 2|S| \le 2k$ as required.

Case 2: There exists $\ell^\star \in [r]$ such that $|M_{\ell^\star}| > |W|/3$. Define $Y = M_{\ell^\star}$. If $\ell^\star = 1$, then define $X = \emptyset$. If $\ell^\star = r$, then define $Z = \emptyset$. Otherwise, define $X = \bigcup_{i=1}^{\ell^\star - 1} M_i$ and $Z = \bigcup_{i=\ell^\star+1}^{r} M_i$. Then, we have that $|X|, |Z| \le 2|W|/3$. Notice that we would have executed **Step 3** in this case.

Let S_1 be a minimal subset of S that intersects all Z-$X \cup Y$ paths in D. Let $i_1 = |S_1|$. Then, Lemma 3.2 guarantees that there exist $L_1 \in \mathcal{L}^{i_1}[Z \to XY]$ and $L_2 \in \mathcal{L}^{i_1}[XY \leftarrow Z]$ such that $S' = S \setminus S_1$ is a T-sfvs in $D - (L_1 \cup L_2)$. Now, let S_2 be a minimal subset of S' that intersects all Y-X paths in $D - (L_1 \cup L_2)$. Let $i_2 = |S_2|$. Then, Lemma 3.2 guarantees that there exist $L_3 \in \mathcal{L}^{i_2}[Y \to X, L_1, L_2]$ and $L_4 \in \mathcal{L}^{i_2}[X \leftarrow Y, L_1, L_2]$ such that $S'' = S' \setminus S_2$ is a T-sfvs in $D - \bigcup_{q \in [4]} L_q$. Let $Q = \bigcup_{q \in [4]} L_q$. Define:

- $S''_Z = S'' \cap \mathsf{rel}(Z, Q)$, $i_3 = |S''_Z|$.
- $S''_X = S'' \cap \mathsf{rel}(X, Q)$, $i_4 = |S''_X|$.
- $S''_Y = S'' \cap \mathsf{rel}(Y, Q)$, $i_5 = |S''_Y|$.

Then, we have that S''_Z is a T-sfvs of size at most i_3 in $D[\mathsf{rel}(Z, Q)]$, S''_X is a T-sfvs of size at most i_4 in $D[\mathsf{rel}(X, Q)]$. Lemma 3.4 implies that S''_Y and Y are both T-sfvs in D' where D' is the graph obtained from $D[\mathsf{rel}(Y, Q)]$ by adding a bidirected clique on Y. Due to this bidirected clique, it trivially holds that in $D' - S''_Y$, there is a unique strongly connected component intersected by Y. We also have that S''_Y has size at most i_5. Conversely, we have that the union of any three sets hitting all T-cycles in $D - Q$ passing through X, Y and Z respectively, is a T-sfvs in $D - Q$.

Therefore, by the induction hypothesis and correctness of Alg-Strict-SFVS (in the case of S''_Y), S_Z is a T-sfvs of size at most $2i_3$ in $D[\mathsf{rel}(Z, Q)]$, S_X is a T-sfvs of size at most $2i_4$ in $D[\mathsf{rel}(X, Q)]$, S_Y is a T-sfvs of size at most i_5 in D' where D' is the graph obtained from $D[\mathsf{rel}(Y, Q)]$ by adding a bidirected clique on Y. This also implies that S_Y is a T-sfvs of size at most i_5 in $D[\mathsf{rel}(Y, Q)]$. Then, the set $Q \cup S_X \cup S_Y \cup S_Z = \bigcup_{q \in [4]} L_q \cup S_X \cup S_Y \cup S_Z$ is a T-sfvs in D of size at most $2(i_1 + i_2 + i_3 + i_4 + i_5) \leq 2|S| \leq 2k$ as required.

This completes the proof of correctness.

Running time. We now analyze the running time taken by Alg-Disjoint-SFVS. The time spent in any single step of the algorithm is dominated by $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ (the running time of Alg-Strict-SFVS). Hence, in order to bound the running time, it suffices to bound the number of leaves generated in the branching. Let $T(k, r)$ denote the number of leaves generated by the instance (D, T, W, k), where $r = |W|$. From the description of the algorithm, we have the following recurrence:

$$T(k, r) \leq 3^r \cdot \sum_{k_1=1}^{k} 2^{5k_1} \cdot 2 \sum_{k_2+k_3 \leq k-k_1} T(k_2, \lfloor 2r/3 \rfloor) + T(k_3, \lfloor 2r/3 \rfloor).$$

$$T(1, r) = 1, T(k, 1) = 1.$$

The following is an intuitive description of this recurrence. There are 3^r 3-way partitions (X, Y, Z) of $|W|$. For each possible size k_1 (which is equal to $i_1 + i_2$) of the minimal part of a hypothetical optimal solution that intersects all Z-$X \cup Y$ paths (and if necessary, also all Y-X paths), there are at most $16^{k_1} \cdot k_1^2 \leq 2^{5k_1}$ choices of vertex sets of size at most $2k_1$ that comprise important separators and whose deletion reduces the size of the optimal solution by k_1. Having guessed and removed this set of size at most $2k_1$, we recursively call Alg-Disjoint-SFVS for increasing values of i_3, followed by calls to Alg-Disjoint-SFVS for increasing values of i_4, which is followed by the invocation of Lemma 3.3 (Alg-Strict-SFVS) with budget $i_5 = k - (k_1 - i_3 - i_4)$. This gives a total of at most 3 recursive calls to Alg-Disjoint-SFVS: (i) on subinstance corresponding to X (budget i_4), (ii) on subinstance corresponding to Z (budget i_3), (iii) on subinstance corresponding to $X \cup Y$ (budget i_5)). Moreover, $|X|, |Z|, |X \cup Y| \leq 2|W|/3$.

Claim 3.3. *$T(k, r) \leq 2^{9k+5r}$.*

Proof. The base cases are satisfied and we assume $r, k > 1$.

$$\begin{aligned} T(k, r) &\leq 3^r \cdot 2^{10/3r} \cdot \sum_{k_1=1}^{k} 2^{5k_1} \cdot 2 \sum_{i_3+i_4 \leq k-k_1} (2^{9i_3} + 2^{9i_4}) \\ &\leq 2^{5r} \cdot \sum_{k_1=1}^{k} 2^{5k_1} \cdot 2^{9(k-k_1)+3} \\ &\leq 2^{5r} \cdot 2^{9k} \cdot \sum_{k_1=1}^{k} 2^{-4k_1+3} \leq 2^{5r} \cdot 2^{9k}. \end{aligned}$$

This completes the proof of the claim. $\square$

Thus, we have concluded that Algorithm Alg-Disjoint-SFVS on input (D, T, W, k), where (D, T) is an instance of Subset DFVS and W is a T-sfvs in D, runs in time $2^{\mathcal{O}(k+|W|)}n^{\mathcal{O}(1)}$ and if there is a T-sfvs S in D of size at most k disjoint from W, then it outputs a T-sfvs in D of size at most $2k$. This completes the proof of Theorem 3.1. $\square$

As a corollary of Theorem 3.1, we obtain our factor-2 FPT-approximation for DFVS with running time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$.

3.3 Bidirected Multicut

For a digraph D and a set $\mathcal{T} = \{(s_i, t_i) \mid s_i, t_i \in V(D)\}$, we say that a path is a *$\mathcal{T}$-path* if it is an s_i-t_i path for some $(s_i, t_i) \in \mathcal{T}$. We say that a set $S \subseteq V(D)$ is a *$\mathcal{T}$-multicut* if there is no $\mathcal{T}$-path in $D - S$ and $\mathcal{T}[S]$ denotes the set $\{(s_i, t_i) \in \mathcal{T} \mid s_i, t_i \in S\}$. The classic Undirected Multicut problem [MR14, BDT18] is easily seen to be equivalent to the Bidirected Multicut (BiMC) problem.

Moreover, recall that BiMC is a special case of SCC $\mathcal{F}$-Transversal as one can simply take $\mathcal{F}$ to be the subgraphs induced by the vertex sets of the s_i-t_i paths in D where $(s_i, t_i) \in \mathcal{T}$. The results of Marx and Razgon [MR14] and Bousquet et al. [BDT18] on the fixed-parameter tractability of Undirected Multicut imply factor-1 FPT-approximation algorithms for BiMC. The result of Marx and Razgon in particular, implies a factor-1 FPT-approximation with running time $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(1)}$.

Our goal is to improve the running time to $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ at the cost of a factor-2 approximation. As we did for Subset DFVS, we first consider the following special case.

A factor-c FPT-approximation algorithm for Strict BiMC is an algorithm that, on input $(D, \mathcal{T}, W, k)$, runs in time $f(k, W) \cdot n^{\mathcal{O}(1)}$ (for some computable f) and if there is a $\mathcal{T}$-multicut S in D of size at most k such that W is contained in a unique strongly connected component of $D - S$, then it outputs a $\mathcal{T}$-multicut in D of size at most ck. Otherwise, the output of the algorithm can be arbitrary.

Towards designing such an algorithm, we recall the following definitions from [KW20]. Let D be a digraph, $s \in V(D)$ and $\{x, y\} \subseteq V(D)$ be a pair of vertices. We say that the *pair $\{x, y\}$ is reachable from s* if there exist paths from s to x and from s to y in D. These paths need not be disjoint. In the Digraph Pair Cut problem, we are given a directed graph D, a source vertex $s \in V(D)$, a set $\mathcal{P}$ of pairs of vertices, and a non-negative integer k. The task is to decide whether there exists a set $X \subseteq V(D) \setminus \{s\}$ such that $|X| \leq k$ and no pair in $\mathcal{P}$ is reachable from s in $D - X$.

Proposition 3.2. [KW20] *There is an algorithm that, given D, a source vertex $s \in V(D)$, a set $\mathcal{P}$ of pairs of vertices, and a non negative integer k, runs in time $2^k n^{\mathcal{O}(1)}$ and either correctly outputs a set $X \subseteq V(D) \setminus \{s\}$ such that $|X| \leq k$ and no pair in $\mathcal{P}$ is reachable from s in $D - X$ or correctly concludes that such a set does not exist.*

We are now ready to give our algorithm for Strict BiMC.

Lemma 3.5. *There is a factor-1 FPT-approximation algorithm for* Strict BiMC *with running time $2^k n^{\mathcal{O}(1)}$. We call this algorithm, Alg-Strict-BiMC.*

Proof. Let $(D, \mathcal{T}, W, k)$ be the input. We now construct a graph D' as follows. We add a new vertex s and make every vertex in W an out-neighbor of s. We set $\mathcal{P} = \emptyset$ and then, for every $(s_i, t_i) \in \mathcal{T}$, we add the pair (s_i, t_i) to $\mathcal{P}$. We now have the following claim.

Claim 3.4. *If S is a $\mathcal{T}$-multicut in D such that W is contained in a unique strongly connected component of $D - S$, then no pair in $\mathcal{P}$ is reachable from s in $D' - S$. Moreover, if $S' \subseteq V(D)$ is such that no pair in $\mathcal{P}$ is reachable from s in $D' - S'$, then S' is a $\mathcal{T}$-multicut in D.*

Proof. Consider the first statement and suppose for a contradiction that some pair $(s_i, t_i) \in \mathcal{P}$ is reachable from s in $D' - S$. Since s is a source, this implies that there exist $w_1, w_2 \in W$ such that s_i is reachable from w_1 in $D - S$ and t_i is reachable from w_2 in $D - S$. However, we know that D is bidirected and there is a strongly connected component of $D - S$ that contains W. Hence, we conclude that there is an s_i-t_i path in $D - S$, a contradiction.

Consider the second statement and the set S' in the premise. Suppose for a contradiction that S' is not a $\mathcal{T}$-multicut in D and there is an s_i-t_i path P in $D - S'$, where $(s_i, t_i) \in \mathcal{T}$. Since W is a $\mathcal{T}$-multicut, it follows that P contains a vertex $w \in W$. Since D is bidirected, it follows that there is a w-s_i and w-t_i path in $D' - S'$, implying an s-s_i and an s-t_i path in $D' - S'$. This is a contradiction to our choice of S'. This completes the proof of the claim. □

The algorithm follows for Strict BiMC from the above claim, which reduces our problem to Digraph Pair Cut and Proposition 3.2, which gives a $2^k n^{\mathcal{O}(1)}$-time algorithm for Digraph Pair Cut. This completes the proof of Lemma 3.5. □

Theorem 3.2. *There is a factor-2 FPT-approximation algorithm for* BiMC *with running time* $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.

Proof. By using the iterative compression technique (see proof of Theorem 3.1), we reduce our goal to designing an algorithm (called Alg-Disjoint-BiMC) that, on input $(D, \mathcal{T}, W, k)$, where $(D, \mathcal{T})$ is an instance of BiMC and W is a $\mathcal{T}$-multicut in D, runs in time $2^{\mathcal{O}(k+|W|)} n^{\mathcal{O}(1)}$ and if there is a $\mathcal{T}$-multicut S in D of size at most k disjoint from W, then it outputs a $\mathcal{T}$-mulicut in D of size at most $2k$. Otherwise, the output of the algorithm can be arbitrary.

We now proceed to describe Algorithm Alg-Disjoint-BiMC. The structure of the algorithm closely resembles that of Algorithm Alg-Disjoint-SFVS. Moreover, Algorithm Alg-Disjoint-BiMC is simpler since we work with bidirected graphs and these essentially behave like undirected graphs in our setting. Another consequence of working with bidirected graphs is that we only need to consider bipartitions of W instead of 3-partitions. We now proceed to the description of the algoroithm.

In the base case of this algorithm, $k = 1$ or $|W| = 1$. In either case, it is sufficient for us to check whether there is a $\mathcal{T}$-multicut in D of size at most 1, which can be done in polynomial time. Hence, we assume that $k, |W| > 1$. Moreover, since D is bidirected, every vertex-induced subgraph of D is strongly connected.

Let $\mathcal{P}$ denote the set of all bipartitions of W into sets (Y, Z). For every $\tau = (Y, Z) \in \mathcal{P}$, we define the following sets and tuples. Let $1 \leq i, j \leq k$.

- $\mathcal{L}^i[Z \to Y]$ denotes the set of all important Z-Y separators of size at most i closest to Y.
- $\mathcal{L}^i[Y \leftarrow Z]$ denotes the set of all important Z-Y separators of size at most i closest to Z.

In the following, let $1 \leq i \leq k$ and $Q \subseteq V(D)$.

- For every $N \in \{Z, Y\}$, $I[Q, N, i]$ denotes the tuple $(D[\mathsf{rel}(N, Q)], \mathcal{T}, Z, i)$. That is, the subinstance induced by those vertices that are in the strongly connected components intersected by N after deleting Q.
- $\widetilde{I}[Q, Y, i]$ denotes $(D', \mathcal{T}, Y, i)$, where D' is the graph obtained from $D[\mathsf{rel}(Y, Q)]$ by adding a bidirected clique on Y (i.e., we add an arc (w, w') for every $w, w' \in Y$ such that $(w, w') \notin A(D)$).

We now describe the rest of the algorithm.

Main loop: For every $(Y, Z) \in \mathcal{P}$ such that $|W|/3 \leq |Y|, |Z| \leq 2|W|/3$ or $|Y| > |W|/3$, and for every i_1 such that $1 \leq i_1 \leq k$, we do the following:

Step 1: We guess $L_1 \in \mathcal{L}^{i_1}[Z \to Y]$ and $L_2 \in \mathcal{L}^{i_1}[Y \leftarrow Z]$ and set $Q = L_1 \cup L_2$.

Step 2: If $|W|/3 \leq |Y|, |Z| \leq 2|W|/3$, then for every i_2, i_3 such that $i_2 + i_3 \leq k - i_1$, we recursively compute:

(i) $S_Z \leftarrow \mathsf{Alg\text{-}Disjoint\text{-}BiMC}(I[Q, Z, i_2])$.
(ii) $S_Y \leftarrow \mathsf{Alg\text{-}Disjoint\text{-}BiMC}(I[Q, Y, i_3])$.

Step 3: If Step 2 does not apply and $|Y| > |W|/3$, then for every i_2, i_3 such that $i_2 + i_3 = k - i_1$, we compute:

(i) $S_Z \leftarrow \mathsf{Alg\text{-}Disjoint\text{-}BiMC}(I[Q, Z, i_2])$.
(ii) $S_Y \leftarrow \mathsf{Alg\text{-}Strict\text{-}BiMC}(\widetilde{I}[Q, Y, i_3])$.

Step 4: If $\Delta = Q \cup S_Z \cup S_Y$ is a $\mathcal{T}$-multicut in D of size at most $2k$, then we return Δ.

If the algorithm completes iterating through the main loop without returning, then we return an arbitrary vertex set. This completes the description of the algorithm.

Correctness. The correctness is proved by induction on $|W|$. In the base case, $|W| = 1$, in which case, the algorithm works by brute-force and hence is correct. Now, we assume that $|W| > 1$. Suppose that there is a $\mathcal{T}$-multicut S in D of size at most k, such that $S \cap W = \emptyset$. Our aim is to show that the algorithm outputs a $\mathcal{T}$-multicut of size at most $2k$.

Let $(M_1, \ldots, M_r)$ denote the partition of W such that (i) each M_i is contained in a strongly connected component of $D - S$, and (ii) for every $\ell_1 > \ell_2$, there is no path in $D - S$ from $\mathsf{rel}_D(M_{\ell_1}, S)$ to $\mathsf{rel}_D(M_{\ell_2}, S)$. That is, S is an M_{ℓ_1}-M_{ℓ_2} separator for every $\ell_1 > \ell_2$. Since we are dealing with bidirected graphs, we may assume without loss of generality that $|M_1| \geq |M_\ell|$ for every $\ell \in [r]$. We now consider the following cases.

Case 1: $|M_1| \leq |W|/3$. This implies that $|M_\ell| \leq |W|/3$ for every $\ell \in [r]$. Let $\ell' \in [r]$ denote the least value such that $|W|/3 < \Sigma_{i=1}^{\ell'} |M_i|$. Then, $|W|/3 < \Sigma_{i=1}^{\ell'} |M_i| \leq 2|W|/3$. Define $Y = \bigcup_{i=1}^{\ell'} M_i$ and $Z = \bigcup_{i=\ell'+1}^{r} M_i$. Then, we have that $|W|/3 \leq |Y|, |Z| \leq 2|W|/3$.

Let S_1 be a minimal subset of S that intersects all Z-Y paths in D. Let $i_1 = |S_1|$. Since D is strongly connected, it follows that $i_1 > 0$. Lemma 3.2 guarantees that there exist $L_1 \in \mathcal{L}^{i_1}[Z \to Y]$ and $L_2 \in \mathcal{L}^{i_1}[Y \leftarrow Z]$ such that $S' = S \setminus S_1$ is a $\mathcal{T}$-multicut in $D - (L_1 \cup L_2)$. Set $Q = L_1 \cup L_2$ and define:

- $S'_Z = S' \cap \mathsf{rel}(Z, Q)$, $i_2 = |S'_Z|$.
- $S'_Y = S' \cap \mathsf{rel}(Y, Q)$, $i_3 = |S'_Y|$.

Then, we have that S'_Z is a $\mathcal{T}$-multicut of size at most i_2 in $D[\mathsf{rel}(Z, Q)]$ and S'_Y is a $\mathcal{T}$-multicut of size at most i_3 in $D[\mathsf{rel}(Y, Q)]$. Therefore, by the induction hypothesis, S_Z is a $\mathcal{T}$-multicut of size at most $2i_2$ in $D[\mathsf{rel}(Z, Q)]$ and S_Y is a $\mathcal{T}$-multicut of size at most $2i_3$ in $D[\mathsf{rel}(Y, Q)]$.

Conversely, one can obtain a $\mathcal{T}$-multicut in $D - Q$ by taking the union of any set that intersects all $\mathcal{T}$-paths in $D - Q$ that intersect Y and any set that intersects all T-paths in $D - Q$ that intersect Z. Hence, the set $Q \cup S_Z \cup S_Y$ is a $\mathcal{T}$-multicut in D and it has size at most $2(i_1 + i_2 + i_3) \leq 2|S| \leq 2k$ as required.

Case 2: $|M_1| > |W|/3$. Define $Y = M_1$. If $r = 1$, then define $Z = \emptyset$. Otherwise, define $Z = \bigcup_{i=2}^{r} M_i$. Then, we have that $|Z| \leq 2|W|/3$.

Let S_1 be a minimal subset of S that intersects all Z-Y paths in D. Let $i_1 = |S_1|$. Then, Lemma 3.2 guarantees that there exist $L_1 \in \mathcal{L}^{i_1}[Z \to Y]$ and $L_2 \in \mathcal{L}^{i_1}[Y \leftarrow Z]$ such that $S' = S \setminus S_1$ is a $\mathcal{T}$-multicut in $D - (L_1 \cup L_2)$. Let $Q = L_1 \cup L_2$.

- Let $S'_Z = S' \cap \mathsf{rel}(Z, Q)$, $i_2 = |S'_Z|$.
- $S'_Y = S' \cap \mathsf{rel}(Y, Q)$, $i_3 = |S'_Y|$.

We have that S'_Z is a $\mathcal{T}$-multicut in $D[\mathsf{rel}(Z, Q)]$. Moreover, S'_Y and Y are $\mathcal{T}$-multicuts in $D[\mathsf{rel}(Y, Q)]$. This further implies that S'_Y and Y are both also $\mathcal{T}$-multicuts in D' where D' is the graph obtained from $D[\mathsf{rel}(Y, Q)]$ by adding a bidirected clique on Y. Due to this bidirected clique, it trivially holds that in $D' - S'_Y$, there is a unique strongly connected component intersected by Y. Therefore, by the induction hypothesis and correctness of Alg-Strict-BiMC (in the case of S'_Y), S_Z is a $\mathcal{T}$-multicut of size at most $2i_2$ in $D[\mathsf{rel}(Z, Q)]$, S_Y is a $\mathcal{T}$-multicut of size at most i_3 in $D[\mathsf{rel}(Y, Q)]$.

Conversely, we have that the union of any pair of sets hitting all $\mathcal{T}$-paths in $D - Q$ passing through Y and Z respectively is a $\mathcal{T}$-multicut in $D - Q$. Hence, the set $Q \cup S_Y \cup S_Z$ is a $\mathcal{T}$-multicut in D of size at most $2|S| \leq 2k$ as required.

This completes the proof of correctness.

Running time. We now analyze the running time taken by Alg-Disjoint-BiMC. The time spent in any single step of the algorithm is dominated by $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ (the running time of Alg-Strict-BiMC). Hence, in order to bound the running time, it suffices to bound the number of leaves generated in the branching. Let $T(k, r)$ denote the number of leaves generated by the instance (D, T, W, k), where $r = |W|$. From the description of the algorithm, we have the following recurrence:

$$T(k, r) \leq 2^r \cdot \sum_{i_1=1}^{k} 2^{4i_1} \cdot \sum_{i_2+i_3 \leq k-i_1} T(i_2, \lfloor 2r/3 \rfloor) + T(i_3, \lfloor 2r/3 \rfloor).$$

$$T(1, r) = 1, T(k, 1) = 1.$$

Using arguments similar to that in the proof of Theorem 3.1, we conclude that $T(k, r)$ is bounded by $2^{\mathcal{O}(k+r)}$.

Thus, we have concluded that Algorithm Alg-Disjoint-BiMC on input $(D, \mathcal{T}, W, k)$, runs in time $2^{\mathcal{O}(k+|W|)} n^{\mathcal{O}(1)}$ and if there is a $\mathcal{T}$-multicut S in D of size at most k disjoint from W, then it outputs a $\mathcal{T}$-multicut in D of size at most $2k$. This completes the proof of Theorem 3.2. $\square$

3.4 Directed OCT

For a digraph D, we denote say that S is a *directed odd cycle transversal* (or *doct*) in D if $D - S$ does not contain directed odd cycles.

Definition 3.6 (Directed Bipartite Double Cover)**.** *Let D be a digraph. We denote by $\widetilde{D}$ the* Directed Bipartite Double Cover *of D which is defined as follows. The vertex set of $\widetilde{D}$ is $\{v^a \mid v \in V(D)\} \cup \{v^b \mid v \in V(D)\}$. For every arc $(u, v) \in A(D)$, $\widetilde{D}$ has arcs (u^a, v^b) and (u^b, v^a). For a set $S \subseteq V(D)$, we define $\widetilde{S} = \{v^a \mid v \in S\} \cup \{v^b \mid v \in S\}$. For each $v \in V(D)$, we call v^a and v^b the copies of v in $\widetilde{D}$.*

Proposition 3.3. [LRSZ20] *A strongly connected digraph does not contain directed odd cycles if and only if the underlying undirected graph is bipartite.*

Recall that DOCT is a special case of SCC $\mathcal{F}$-Transversal as one can simply take $\mathcal{F}$ to be the set of all directed odd cycles. We also define the following.

A factor-c FPT-approximation algorithm for Strict DOCT is an algorithm that, on input (D, W, k) where W is a doct in D, runs in time $f(k, W) \cdot n^{\mathcal{O}(1)}$ (for some computable f) and if there is a doct S in D of size at most k such that the undirected graph underlying $D[\mathsf{conn}(W, S)]$ is bipartite, then then it outputs a doct in D of size at most ck. Otherwise, the output of the algorithm can be arbitrary.

The above definition is motivated by Proposition 3.3 and is a relaxation of the case where there is a doct S in D such that W is contained in a unique strongly connected component of $D - S$. Indeed, if W is contained in a unique strongly connected component of $D - S$, then the undirected graph underlying this strongly connected component, which is the same as the graph $D[\mathsf{conn}(W, S)]$, is bipartite.

Lokshtanov et al. [LRSZ20] gave a factor-1 FPT-approximation algorithm for Strict DOCT with running time $2^{\mathcal{O}(k^2+|W|\log|W|)}n^{\mathcal{O}(1)}$. They used this algorithm as a subroutine in their factor-2 FPT-approximation for DOCT running in time $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(1)}$. Here, we give a single-exponential-time factor-2 approximation for Strict DOCT that can be used to obtain a single-exponential-time factor-2 approximation for DOCT.

Lemma 3.6. *There is a factor-2 FPT-approximation algorithm for* Strict DOCT *with running time* $2^{|W|}n^{\mathcal{O}(1)}$. *We call this algorithm* Alg-Strict-DOCT.

Proof. Let $I = (D, W, k)$ be the input. We begin with the following claim.

Claim 3.5. *If there is a doct S in D such that the undirected graph underlying $D[\mathsf{conn}(W, S)]$ is bipartite, then there exists an $\alpha \subseteq W$ such that the following statements hold.*

1. *$\widetilde{S}$ (see Definition 3.6) intersects all $(\alpha^a \cup \beta^b) - (\alpha^b \cup \beta^a)$ paths in $\widetilde{D}$.*
2. *Conversely, for every $S' \subseteq V(D)$ such that $\widetilde{S'}$ intersects all $(\alpha^a \cup \beta^b) - (\alpha^b \cup \beta^a)$ paths in $\widetilde{D}$ is a doct in D.*

Proof. Fix S to be a doct in D such that the undirected graph underlying $D[\mathsf{conn}(W, S)]$ is bipartite. Let (α, β) denote a partition of W induced by this bipartition. Let $C = \mathsf{conn}(W, S)$. We now claim that for this choice of α, the statements in the claim hold. Notice that for every $X, Y \in \{\alpha, \beta\}$, $x \in X$ and $y \in Y$, there is no odd directed x-y walk in $D[C]$ (and hence also in $D - S$) if $X = Y$ and there is no even directed x-y walk in $D[C]$ (and hence also in $D - S$) if $X \neq Y$.

We now proceed to the proof of the statements. For the first statement, suppose that $\widetilde{S}$ is not an $(\alpha^a \cup \beta^b) - (\alpha^b \cup \beta^a)$ separator in $\widetilde{D}$ and let P be a directed x-y path in $\widetilde{D}$, where $x \in \alpha^a \cup \beta^b$ and $y \in \alpha^b \cup \beta^a$. Let x' and y' be the vertices in W that correspond to x and y respectively and suppose that $x' \in X$, $y' \in Y$ where $X, Y \in \{\alpha, \beta\}$. Then, P implies the existence of an odd directed x-y walk in $D[C]$ (and hence also in $D - S$) if $X = Y$ and an even directed x-y path in $D[C]$ (and hence also in $D - S$) if $X \neq Y$, which is a contradiction.

Consider the second statement and an S' as described in the premise. Suppose for a contradiction that S' is not a doct in D_α and let Q be a directed odd cycle in $D_\alpha - S'$. Since W is a doct in D, it follows that Q contains a vertex $w \in W$. Then, the construction of $\widetilde{D}$ implies the existence of a w^a-w^b path in $\widetilde{D} - \widetilde{S'}$, a contradiction.

This completes the proof of Claim 3.5. $\square$

Given the above claim, our algorithm is described as follows. Recall that $I = (D, W, k)$ is the input. Now, for every $\alpha \subseteq W$, we check whether there is a $(\alpha^a \cup \beta^b) - (\alpha^b \cup \beta^a)$ separator in

$\widetilde{D}$ of size at most $2k$ and compute one if it exists (call this $\widehat{S}_\alpha$). For every $\alpha \subseteq W$ and $\widehat{S}_\alpha$, we define $S_\alpha \subseteq V(D)$ as the set $\{v \mid \{v^a \cup v^b\} \cap \widehat{S}_\alpha \neq \emptyset\}$. That is, S_α comprises those vertices of $V(D)$ that "contribute a copy" to $\widehat{S}_\alpha$. Notice that for every $\alpha \subseteq W$, either S_α does not exist or has size at most $2k$. When then check whether there exists an $\alpha \subseteq W$ such that S_α is a doct in D. If there is such an α, then we return S_α. Otherwise, we return an arbitrary output and terminate.

The running time bound follows from the fact that for every $\alpha \subseteq W$, the time required to compute S_α (if it exists) and verify whether it is a doct in D is polynomial. For the correctness, recall that we only need our output to be correct only if there is a doct S in D such that the undirected graph underlying $D[\mathsf{conn}(W, S)]$ is bipartite. In this case, the second statement of Claim 3.5 guarantees that it is sufficient to compute any $S' \subseteq V(D)$ such that $\widetilde{S'}$ intersects all $(\alpha^a \cup \beta^b) - (\alpha^b \cup \beta^a)$ paths in $\widetilde{D}$ is a doct in D for some $\alpha \subseteq W$. The first statement of Claim 3.5 guarantees that there is indeed at least one such set. This completes the proof of Lemma 3.6. ☐

Theorem 3.3. *There is a factor-2 FPT-approximation algorithm for* DOCT *with running time* $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.

Proof. The algorithm for DOCT closely resembles that for SUBSET DFVS (Theorem 3.1) with the primary difference being the use of Algorithm Alg-Strict-DOCT as a subroutine instead of Algorithm Alg-Strict-SFVS (on an appropriate subinstance). We therefore use the same notation where possible, omit the running time analysis and only sketch the differences in the algorithm description and proof of correctness.

By using the iterative compression technique, we reduce our goal to designing an algorithm that, on input (D, W, k), where W is a doct in D, runs in time $2^{\mathcal{O}(k+|W|)} n^{\mathcal{O}(1)}$ and if there is a doct S in D of size at most k disjoint from W, then it outputs a doct in D of size at most $2k$. Otherwise, the output of the algorithm can be arbitrary.

We now proceed to describe this algorithm (Algorithm Alg-Disjoint-DOCT). In the base case of this algorithm, $k = 1$ or $|W| = 1$. In either case, the algorithm solves the instance by brute force. Hence, we assume that $k, |W| > 1$. Moreover, we assume that D is strongly connected. Otherwise, we can simply solve the subinstance induced by each strongly connected component.

Let $\mathcal{P}$ denote the set of all 3-partitions of W into sets (X, Y, Z). In the following, let $1 \le i \le k$, $Q \subseteq V(D)$ and $(X, Y, Z) \in \mathcal{P}$.

- $I[Q, Z, i]$ denotes the tuple $(D[\mathsf{rel}(Z, Q)], Z, i)$.
- $I[Q, XY, i]$ denotes the tuple $(D[\mathsf{rel}(X \cup Y, Q)], X \cup Y, i)$.
- $I[Q, X, i]$ denotes the tuple $(D[\mathsf{rel}(X, Q)], X, i)$.
- $I[Q, Y, i]$ denotes the tuple $(D[\mathsf{rel}(Y, Q)], Y, i)$.

We now proceed to the description of the rest of the algorithm.

Main loop: For every $(X, Y, Z) \in \mathcal{P}$ such that $|W|/3 \le |X \cup Y|, |Z| \le 2|W|/3$ or $|Y| > |W|/3$, and for every i_1, i_2 such that $1 \le i_1 \le k$ and $k_1 = i_1 + i_2 \le k$, we do the following:

Step 1: We guess $L_1 \in \mathcal{L}^{i_1}[Z \to XY]$, $L_2 \in \mathcal{L}^{i_1}[XY \leftarrow Z]$, $L_3 \in \mathcal{L}^{i_2}[Y \to X, L_1, L_2]$, $L_4 \in \mathcal{L}^{i_2}[X \leftarrow Y, L_1, L_2]$. Set $Q = \bigcup_{q \in [4]} L_q$.

Step 2: If $|W|/3 \le |X \cup Y|, |Z| \le 2|W|/3$, then for every i_3, i_4 such that $i_3 + i_4 \le k - k_1$, we recursively compute:

(i) $S_Z \leftarrow$ Alg-Disjoint-DOCT$(I[Q, Z, i_3])$.

(ii) $S_{XY} \leftarrow$ Alg-Disjoint-DOCT$(I[Q, XY, i_4])$.

If $\Delta = Q \cup S_Z \cup S_{XY}$ is a doct in D of size at most $2k$, then we return Δ.

In order to see that the instances $I[Q, Z, i_3]$ and $I[Q, XY, i_4]$ are valid input instances to Alg-Disjoint-DOCT, it is sufficient to argue that Z is a doct in $D[\mathsf{rel}(Z, Q)]$ and $X \cup Y$ is a doct in $D[\mathsf{rel}(X \cup Y, Q)]$. But this follows from the fact that $X \cup Y \cup Z$ is a doct and no directed odd cycle intersects both $X \cup Y$ and Z in $D - Q$.

Step 3: If Step 2 does not apply and $|Y| > |W|/3$, then for every i_3, i_4, i_5 such that $i_3 + i_4 + i_5 = k - k_1$, we recursively compute:

(i) $S_Z \leftarrow$ Alg-Disjoint-DOCT$(I[Q, Z, i_3])$.
(ii) $S_X \leftarrow$ Alg-Disjoint-DOCT$(I[Q, X, i_4])$.
(iii) $S_Y \leftarrow$ Alg-Strict-DOCT$(I[Q, Y, i_5])$.

If $\Delta = Q \cup S_Z \cup S_X \cup S_Y$ is a doct in D of size at most $2k$, then we return Δ.

If the algorithm completes iterating through the main loop without returning, then we return an arbitrary vertex set. This completes the description of the algorithm.

Correctness. The correctness is proved by induction on $|W|$. In the base case, $|W| = 1$, in which case, the algorithm works by brute-force and hence is correct. Now, we assume that $|W| > 1$. Suppose that there is a doct S in D of size at most k, such that $S \cap W = \emptyset$. Our aim is to show that the algorithm outputs a doct of size at most $2k$.

Let $(M_1, \ldots, M_r)$ denote the partition of W such that (i) each M_i is contained in a strongly connected component of $D - S$, and (ii) for every $\ell_1 > \ell_2$, there is no path in $D - S$ from $\mathsf{rel}(M_{\ell_1}, S)$ to $\mathsf{rel}(M_{\ell_2}, S)$.

Case 1: $|M_\ell| \leq |W|/3$ for every $\ell \in [r]$. Let $\ell' \in [r]$ denote the least value such that $|W|/3 < \Sigma_{i=1}^{\ell'} |M_i|$. Then, $|W|/3 < \Sigma_{i=1}^{\ell'} |M_i| \leq 2|W|/3$. Define $X = \emptyset$, $Y = \bigcup_{i=1}^{\ell'} M_i$ and $Z = \bigcup_{i=\ell'+1}^{r} M_i$. Then, we have that $|W|/3 \leq |X \cup Y|, |Z| \leq 2|W|/3$.

Let S_1 be a minimal subset of S that intersects all Z-$X \cup Y$ paths in D. Let $i_1 = |S_1|$. Since D is strongly connected, it follows that $i_1 > 0$. Lemma 3.2 guarantees that there exist $L_1 \in \mathcal{L}^{i_1}[Z \rightarrow XY]$ and $L_2 \in \mathcal{L}^{i_1}[XY \leftarrow Z]$ such that $S' = S \setminus S_1$ is a doct in $D - (L_1 \cup L_2)$. Let $i_2 = 0$, $L_3 = L_4 = \emptyset$, $Q = \bigcup_{q \in [4]} L_q$, and define:

- $S'_Z = S' \cap \mathsf{rel}(Z, Q)$, $i_3 = |S'_Z|$.
- $S'_{XY} = S' \cap \mathsf{rel}(X \cup Y, Q)$, $i_4 = |S'_{XY}|$.

Notice that S'_{XY} intersects all directed odd cycles in $D - Q$ that intersect $X \cup Y$ and S'_Z intersects all directed odd cycles in $D - Q$ that intersect Z. Conversely, one can obtain a doct in $D - Q$ by taking the union of any set that intersects all directed odd cycles in $D - Q$ that intersect $X \cup Y$ and any set that intersects all directed odd cycles in $D - Q$ that intersect Z. This is because $W = X \cup Y \cup Z$ is a doct in D.

Then, we have that S'_Z is a doct of size at most i_3 in $D[\mathsf{rel}(Z, Q)]$ and S'_{XY} is a doct of size at most i_4 in $D[\mathsf{rel}(X \cup Y, Q)]$. Therefore, by the induction hypothesis, S_Z is a doct of size at most $2i_3$ in $D[\mathsf{rel}(Z, Q)]$ and S_{XY} is a doct of size at most $2i_4$ in $D[\mathsf{rel}(X \cup Y, Q)]$. The set $Q \cup S_Z \cup S_{XY} = L_1 \cup L_2 \cup S_Z \cup S_{XY}$ is a therefore a doct in D and it has size at most $2(i_1 + i_3 + i_4) \leq 2|S| \leq 2k$ as required.

Case 2: There exists $\ell^\star \in [r]$ such that $|M_{\ell^\star}| > |W|/3$. Define $Y = M_{\ell^\star}$. If $\ell^\star = 1$, then define $X = \emptyset$. If $\ell^\star = r$, then define $Z = \emptyset$. Otherwise, define $X = \bigcup_{i=1}^{\ell^\star - 1} M_i$ and $Z = \bigcup_{i=\ell^\star+1}^{r} M_i$. Then, we have that $|X|, |Z| \leq 2|W|/3$.

Let S_1 be a minimal subset of S that intersects all Z-$X \cup Y$ paths in D. Let $i_1 = |S_1|$. Then, Lemma 3.2 guarantees that there exist $L_1 \in \mathcal{L}^{i_1}[Z \rightarrow XY]$ and $L_2 \in \mathcal{L}^{i_1}[XY \leftarrow Z]$ such that $S' = S \setminus S_1$ is a doct in $D - (L_1 \cup L_2)$. Now, let S_2 be a minimal subset of S'

that intersects all Y-X paths in D. Let $i_2 = |S_2|$. Then, Lemma 3.2 guarantees that there exist $L_3 \in \mathcal{L}^{i_2}[Y \to X, L_1, L_2]$ and $L_4 \in \mathcal{L}^{i_2}[X \leftarrow Y, L_1, L_2]$ such that $S'' = S' \setminus S_2$ is a doct in $D - \bigcup_{q\in[4]} L_q$. Let $Q = \bigcup_{q\in[4]} L_q$ and define:

- $S''_Z = S'' \cap \mathsf{rel}(Z, Q)$, $i_3 = |S''_Z|$.
- $S''_X = S'' \cap \mathsf{rel}(X, Q)$, $i_4 = |S''_X|$.
- $S''_Y = S'' \cap \mathsf{rel}(Y, Q)$, $i_5 = |S''_Y|$.

We have that S'_Z is a doct of size at most i_3 in $D[\mathsf{rel}(Z, Q)]$, S''_X is a doct of size at most i_4 in $D[\mathsf{rel}(X, Q)]$. Lemma 3.4 implies that S''_Y is a doct of size at most i_5 in $D[\mathsf{rel}(Y, Q)]$. Moreover, we observe that S''_Y is a doct in $D' = D[\mathsf{rel}(Y, Q)]$ with the additional property that the undirected graph underlying $D'[\mathsf{conn}(Y, S''_Y)]$ is bipartite. Indeed, we know from Proposition 3.3 that the undirected graph underlying $D[\mathsf{rel}(Y, S)]$ is bipartite and $\mathsf{conn}(Y, S''_Y \cup Q) \subseteq \mathsf{rel}(Y, S)$. Conversely, we have that the union of any three sets hitting directed odd cycle in $D - Q$ that intersect X, Y and Z respectively is a doct in $D - Q$.

Therefore, by the induction hypothesis and correctness of Alg-Strict-DOCT (in the case of S''_Y), S_Z is a doct of size at most $2i_3$ in $D[\mathsf{rel}(Z, Q)]$, S_X is a doct of size at most $2i_4$ in $D[\mathsf{rel}(X, Q)]$, S_Y is a doct of size at most i_5 in $D[\mathsf{rel}(Y, Q)]$. Then, the set $Q \cup S_X \cup S_Y \cup S_Z$ is a doct in D of size at most $2|S| \leq 2k$ as required.

This completes the proof of correctness and the proof of Theorem 3.3. □

4 FPT-approximation Algorithms Parameterized by Treewidth

In this section, we present two general reduction schemes—that can be used as black boxes—for (unweighted) graph problems Π parameterized by the treewidth of the input graph, denoted by w. We remark that for problems parameterized by treewidth, we assume that the input consists also of a tree decomposition of width w of the input graph, which is the standard assumption in the area (for more details, see, e.g., [CFK+15]). Moreover, we will assume that the input tree decomposition is a *nice tree decomposition* where the tree consists of at most $\mathcal{O}(w \cdot n)$ nodes. This can be done without loss of generality due to the following proposition

Proposition 4.1 (Lemma 7.2 in [CFK+15]; Lemma 14.23 in [FLSZ19])**.** *Let G be a graph, and let $\mathcal{T} = (T, \beta)$ be a tree decomposition of G of width w. Then, a nice tree decomposition $\mathcal{T}' = (T', \beta')$ of G of width at most w and where $|V(T')| \leq 16(w + 2)n$ can be computed in time $\mathcal{O}(w^2 \cdot (n + |V(T)|))$.*

Our main reduction scheme is given in Section 4.3. Towards that, in Section 4.1, we first present notions regarding the problems to which this scheme applies. Essentially, we can use the scheme in a black box manner to solve the entire class of $\mathcal{F}$-Packing problems where the family $\mathcal{F}$ consists of graphs that are of bounded size, as well as any deletion problem that admits a linear-vertex kernel when parameterized by the solution size k (rather than by w, the treewidth of the input graph) where the output graph is a minor of the input graph. Specifically, if the problem can be solved in time $\mathcal{O}(d^w \cdot n^q)$, then given any fixed $\epsilon > 0$, our scheme yields an $\mathcal{O}(c^w \cdot n^q + n^{\mathcal{O}(1)})$-time $(1 + \epsilon)$-approximation algorithm where c is significantly smaller than d (for maximization, $(1 - \epsilon)$). For some of the problems that fit our scheme, such as Vertex Cover and Triangle Packing, the best known d is also optimal under SETH, thus we break this lower bound at a modest cost in accuracy. Our scheme is based on a new combinatorial lemma, which we present in Section 4.2, and which we believe to be of independent interest. Roughly speaking, we prove that, given a graph G and a tree decomposition $\mathcal{T}$ of G of width w, we can very efficiently—that is, in polynomial time—partition the vertex set of G into p subsets of roughly the same size (for any p that we choose) so that the deletion of any of them reduces the width of $\mathcal{T}$ by roughly w/p.

Lastly, we give our second scheme in Section 4.4, where we show how to combine the results given in Section 3 to obtain constant-factor single-exponential (in w) time approximation algorithms for the problems studied in that section—such as DIRECTED FEEDBACK VERTEX SET—which do not admit single-exponential (in w) time exact algorithms under the ETH. Notice that here we consider these problems when the parameter is w rather than k, yet the algorithms for the parameterization by k will come in handy.

4.1 Tree Decomposition Maintenance and Composability

In Section 4.2, the deletion problems in which we will be interested will need to admit a linear-vertex kernel with an additional property, defined as follows.

Definition 4.1. *Let Π be a graph problem parameterized by the solution size k. We say that a kernelization algorithm of Π admits a* tree decomposition maintenance procedure *if, given any instance (I, k) of Π coupled with a tree decomposition $\mathcal{T}$ of the graph $G \in I$, a tree decomposition $\mathcal{T}'$ of the graph $G' \in I'$ whose width is at most that of $\mathcal{T}$ can be computed in polynomial time, where (I', k') is the output of the kernelization algorithm on (I, k).*

In particular, all problems where the output graph is a minor of the input graph (which is the case with many known kernels) admit a tree decomposition maintenance procedure, as we state below.

Lemma 4.1. *Let Π be a graph problem parameterized by the solution size k. Then, any kernelization algorithm of Π where the output graph is a minor*[4] *of the input graph admits a linear time tree decomposition maintenance procedure.*

Towards the proof of this lemma, we will make use of the following straightforward and well-known observation (see, e.g., [CFK+15]).

Observation 4.1 (Folklore). *Let G be a graph, and let $\mathfrak{s} = (s_1, s_2, \ldots, s_t)$ for some $t \in \mathbb{N}_0$ be a sequence of operations (consisting of vertex deletions, edge deletions and edge contractions) performed on G to obtain a graph G' that is a minor of G. Then, there exist subsets $D \subseteq V(G)$ and $W \subseteq E(G)$, and a collection $\mathcal{S}$ of connected subsets of $G - (D \cup W)$, such that G' equals the graph obtained from $G - (D \cup W)$ by contracting, for all $S \in \mathcal{S}$, the edges of any spanning tree of $(G - (D \cup W))[S]$. Moreover, $\mathcal{S}, D$ and W can be computed in linear time.*

We are now ready to prove Lemma 4.1.

Proof of Lemma 4.1. We now describe the tree decomposition maintenance procedure. To this end, let G be the input graph having a tree decomposition $\mathcal{T} = (T, \beta)$ of width at most w, and let $\mathfrak{s} = (s_1, s_2, \ldots, s_t)$ for some $t \in \mathbb{N}_0$ be the sequence of operations (consisting of vertex deletions, edge deletions and edge contractions) performed by the kernelization algorithm on G so as to obtain its output graph G' as a minor of G. Having the sequence $\mathfrak{s}$, we can compute in linear time the collection $\mathcal{S}$ and subsets D, W described in Observation 4.1. For every connected set $S \in \mathcal{S}$, let v_S denote the vertex in G' that is yielded by the contraction of the edges of some spanning tree of $(G-(D\cup W))[S]$. Then, we define a tree decomposition $\mathcal{T}' = (T', \beta')$ of G' as follows. We let $T' = T$, and for every $x \in V(T')$, we let $\beta'(x) = (\beta(x) \setminus (D \cup (\bigcup_{S\in\mathcal{S}} S))) \cup \{v_S : S \cap \beta(x) \neq \emptyset\}$. It is straightforward to verify that $\mathcal{T}'$ is a tree decomposition of G' of width at most that of $\mathcal{T}$, and it is clear that it can be computed in linear time. This completes the proof. □

[4]More precisely, we implicitly also require to be able to trace the vertex deletion, edge deletion and edge contraction operations in the input graph that yield the output graph.

Many known kernels for graph problems satisfy the requirement that the output graph is a (traceable) minor of the input graph. In particular, we highlight several such problems relevant to our main scheme in the following proposition.

Proposition 4.2. *Each of the following problems, when parameterized by the solution size k, admits a 1-approximate linear-vertex kernel of size at most ck where the output graph is a minor of the input graph:* Vertex Cover *for $c = 2$* [CKJ01, NT74]*;* Component Order Connectivity *for $c = 2\ell$* [KL16]*;* Bounded-Degree Vertex Deletion *for $c = d^3 + 4d^2 + 5d + 3$* [Xia17].

Thus, we have the following corollary of Lemma 4.1 and Proposition 4.2.

Corollary 1. *Each of the following problems, when parameterized by the solution size k, admits a 1-approximate linear-vertex kernel of size at most ck with a tree decomposition maintenance procedure:* Vertex Cover *for $c = 2$;* Component Order Connectivity *for $c = 2\ell k$;* Bounded-Degree Vertex Deletion *for $c = d^3 + 4d^2 + 5d + 3$.*

We remark that when the problems are parameterized by w (treewidth), k is not part of the input. So, when we will later describe our scheme, we may consider every possible $k \in \{0, 1, \ldots, n\}$, and, in fact, will only need to “care about” producing a $(1 + \epsilon)$-approximate solution in the iteration where k is the optimal solution size. We will do this in a slightly more clever way, so as not to incur an extra factor of n in the running time.

For a deletion problem to fit our scheme, we will also need it to satisfy a *composability property*, defined as follows (where we differentiate between minimization and maximization problems). We remark that we define composability so that it captures the problems solved in this paper, but, given other problems of interest, it might be possible to extend or modify it to capture these problems so that our scheme will still work.

Definition 4.2. *A minimization graph problem Π is* composable *if any instance I of Π, where $G \in I$ denotes the graph in I, and any subset $D \subseteq V(G)$, satisfy the following two properties:* (i) *Every solution $S^\star$ for I is a subset of $V(G)$, and $S^\star \setminus D$ is a solution for the subinstance of I induced by $G - D$;* (ii) *Given any solution S to the subinstance of I induced by $G - D$, $D \cup S$ is a solution for I.*

Definition 4.3. *A maximization graph problem Π is d-*composable *if any instance I of Π, where $G \in I$ denotes the graph in I, and any subset $D \subseteq V(G)$, satisfy the following two properties:* (i) *Every solution $\mathcal{S}^\star$ for I is a collection of pairwise disjoint subsets of $V(G)$, each of size at most d, and $\{S \in \mathcal{S}^\star : S \cap D = \emptyset\}$ is a solution for the subinstance of I induced by $G - D$;* (ii) *Any solution $\mathcal{S}$ for the subinstance of I induced by $G - D$ is also a solution for I.*

Now, we observe that for any hereditary family of graphs $\mathcal{F}$, the corresponding $\mathcal{F}$-Vertex Deletion problem is composable, as stated below.

Observation 4.2. *Let $\mathcal{F}$ be a hereditary graph family. Then, the $\mathcal{F}$-*Vertex Deletion *problem is composable.*

This directly yields that all the problems considered in Corollary 1 are composable, as well as additional problems that will be relevant to Section 4.4. Moreover, it is trivial that for any family $\mathcal{F}$ of graphs, each on at most d vertices, the $\mathcal{F}$-Packing problem is d-composable. Thus, we have the following corollary.

Corollary 2. *Each of the following problems is composable:* Vertex Cover*;* Component Order Connectivity*;* Bounded-Degree Vertex Deletion*;* Directed (Subset) Feedback Vertex Set*;* Directed Odd Cycle Transversal*;* Directed Multiway Cut*. Moreover, for every family $\mathcal{F}$ of graphs, each on at most d vertices, the $\mathcal{F}$-*Packing *problem is d-composable.*

4.2 Computation of Pairwise Disjoint "Treewidth Hitting Sets"

In this section, we prove a new combinatorial lemma, which might be of independent interest. Roughly speaking, we prove that, given a graph G and a tree decomposition $\mathcal{T}$ of G of width w, we can partition the vertex set of G into p subsets of roughly the same size (for any p that we choose) so that the deletion of any of them reduces the width of $\mathcal{T}$ by roughly w/p.

We formalize the notion of a hitting set that will play a central role in this section as follows. Here, the implicit assumption that $\mathcal{T}_D$ is a tree decomposition of $G - D$ is trivial to verify.

Definition 4.4. *Let G be a graph with a tree decomposition $\mathcal{T} = (T, \beta)$ of width w, $h \in \mathbb{N}$. A subset $D \subseteq V(G)$ is a $(\mathcal{T}, h)$-*hitting set *if the width of the tree decomposition $\mathcal{T}_D = (T_D, \beta_D)$ of $G - D$, defined as follows, is at most $w - h$: $T_D = T$ and for every node $x \in V(T_D)$, $\beta_D(x) = \beta(x) \setminus D$.*

Towards the computation of a collection of pairwise disjoint hitting sets with the properties that we require, we make use of an algorithm that, given a graph G and a tree decomposition $\mathcal{T}$ of G, properly colors the graph derived from G by transforming each bag of $\mathcal{T}$ into a clique. More directly, we define the colorings we are interested in as follows.

Definition 4.5. *Let G be a graph with a tree decomposition $\mathcal{T} = (T, \beta)$ of width w. For every $\ell \in \mathbb{N}$, a coloring $\mathsf{col} : V(G) \to \{1, 2, \ldots, w + \ell\}$ is $(\ell, \mathcal{T})$-*proper *if for every node $x \in V(T)$ and distinct vertices $u, v \in \beta(x)$, $\mathsf{col}(u) \neq \mathsf{col}(v)$.*

We now present a greedy algorithm to produce $(\ell, \mathcal{T})$-proper colorings with the required properties, called $\mathsf{ColorALG}$. Given a graph G and a tree decomposition $\mathcal{T} = (T, \beta)$, and an integer $\ell > w$, the algorithm works as follows.

1. Denote $V(T) = \{x_1, x_2, \ldots, x_t\}$ where $t = |V(T)|$ and the nodes are ordered in preorder.
2. Allocate a table M with an entry $M[i]$ for every $i \in \{1, 2, \ldots, t\}$.
3. Initialize $M[1] = f$ where $f : \beta(x_1) \to \{1, 2, \ldots, w + \ell\}$ is an arbitrary injective function.
4. For every $i \in \{1, 2, \ldots, t\}$:
   (a) If the parent y of x_i is a join node: $M[i] = M[i - 1]$.
   (b) If the parent y of x_i is an introduce node: $M[i] = M[i - 1]$.
   (c) If the parent y of x_i is a forget node: Let $f' = M[i-1]$ and $\{v\} = \beta(x_i) \setminus \beta(y)$. Let $c \in \{1, 2, \ldots, w+\ell\} \setminus \{f'(u) : u \in \beta(y)\}$ be a color such that $|\{v \in \mathsf{domain}(f') : f'(v) = c\}|$ is minimized (break ties arbitrarily). Define $f : \{v\} \cup \mathsf{domain}(f') \to \{1, 2, \ldots, w + \ell\}$ as follows: $f(v) = c$ and for every other vertex $u \in \mathsf{domain}(f')$, $f(u) = f'(u)$.
5. Output $M[t]$.

We first state the following immediate observation.

Observation 4.3. *The time complexity of $\mathsf{ColorALG}$ is $\mathcal{O}(|V(T)| \cdot (w + \ell))$.*

Lemma 4.2. *Given a graph G with a nice tree decomposition $\mathcal{T} = (T, \beta)$ and an integer $\ell \in \mathbb{N}$, the output col of $\mathsf{ColorALG}$ is a $(w + \ell, \mathcal{T})$-proper coloring.*

Proof. For every $i \in \{1, 2, \ldots, t\}$ where $t = |V(T)|$, denote $G_i = G[\bigcup_{j=1}^{i} \beta(x_j)]$. Moreover, let f_i denote the function stored at $M[x_i]$. We will say that f_i is a *partial $(\ell, \mathcal{T})$-proper coloring* if its domain is $V(G_i)$, and for every node $x_j \in V(T)$ where $j \leq i$ and distinct vertices $u, v \in \beta(x_j)$, $f_i(u) \neq f_i(v)$. We prove the following claim by induction on i.

Claim 4.1. *For every $i \in \{1, 2, \ldots, t\}$, f_i is a partial $(w + \ell, \mathcal{T})$-proper coloring.*

Proof of Claim 4.1. In the basis, where $i = 1$, f_1 is injective, which trivially implies the claim. Now, suppose that the claim is correct for $i - 1 \geq 1$, and let us prove it for i. Let y denote the parent of x_i in T. In case y is a join or introduce node, the correctness directly follows from the inductive hypothesis. Thus, we next suppose that y is a forget node. This means that $v \notin V(G_{i-1})$, and hence does not belong to the domain of f_{i-1}. Moreover, it is assigned (by f_i) a color c that is not assigned to any other vertex in $\beta(x_i)$ by f_{i-1}. Therefore, combined with the inductive hypothesis, we derive that the claim is correct for i. $\square$

The lemma follows from this claim by setting $i = t$, as then $G_i = G$. $\square$

Now, we prove that when ℓ is chosen appropriately, no color is used by ColorALG "too many" times.

Lemma 4.3. *Given a graph G with a nice tree decomposition $\mathcal{T} = (T, \beta)$ and $\ell \in \mathbb{N}$, the output* col *of* ColorALG *satisfies the following property: for every $c \in \{1, 2, \ldots, w + \ell\}$, $|\{v \in V(G) : \mathsf{col}(v) = c\}| \leq \lceil n/\ell \rceil$.*

Proof. For every $i \in \{1, 2, \ldots, t\}$ where $t = |V(T)|$, denote $G_i = G[\bigcup_{j=1}^{i} \beta(x_j)]$ and $n_i = |V(G_i)|$. Moreover, let f_i denote the function stored at $M[x_i]$. We prove the following claim by induction on i.

Claim 4.2. *For every $i \in \{1, 2, \ldots, t\}$, $|\{v \in V(G) : f_i(v) = c\}| \leq \lceil n_i/\ell \rceil$.*

Proof of Claim 4.2. In the basis, where $i = 1$ and $\lceil n_i/\ell \rceil = 1$, and as f_1 is injective, each color is indeed used at most once. Now, suppose that the claim is correct for $i - 1 \geq 1$, and let us prove it for i. Let y denote the parent of x_i in T. In case y is a join or introduce node, the correctness directly follows from the inductive hypothesis. Thus, we next suppose that y is a forget node. Let c be the color chosen to be assigned to $v \in \beta(x_i) \setminus \beta(y)$. For any other color, the correctness directly follows from the inductive hypothesis. So, it remains to prove that $|\{u \in V(G) : f_i(u) = c\}| \leq \lceil n_i/\ell \rceil$. Note that $|\{u \in V(G) : f_{i-1}(u) = c\}| + 1 = |\{u \in V(G) : f_i(u) = c\}|$ and $n_i = n_{i-1} + 1$, thus it suffices to prove that $|\{u \in V(G) : f_{i-1}(u) = c\}| \leq \lceil (n_{i-1} + 1)/\ell \rceil - 1$. Targeting a contradiction, suppose that this claim is false. By the choice of c (see description of ColorALG) and because $|\{1, 2, \ldots, w + \ell\} \setminus \{f_{i-1}(u) : u \in \beta(y)\}| \geq w + \ell - (w - 1) = \ell + 1$, this means that there exist at least $\ell + 1$ colors in $\{1, 2, \ldots, w + \ell\}$ that are each assigned by f_{i-1} at least $\lceil (n_{i-1} + 1)/\ell \rceil$ times. However, this yields that f_{i-1} assigns $(\ell + 1) \cdot \lceil (n_{i-1} + 1)/\ell \rceil > n_{i-1}$ colors (including repetitions), which is impossible as the size of its domain is n_{i-1} (see, e.g., the proof of Lemma 4.2). $\square$

The lemma follows from this claim by setting $i = t$, as then $G_i = G$ and $n_i = n$. $\square$

Now, we give a simple packing lemma, which will be used on top of ColALG.

Lemma 4.4. *Let $U = \{u_1, u_2, \ldots, u_n\}$ be a collection of n items, where item u_i has weight $w(u_i)$, and let $g \in \mathbb{N}$. Let $M = \max_{i=1}^{n} w(u_i)$. Then, there exists a partition $\bar{\mathbf{P}} = (P_1, P_2, \ldots, P_g)$ of U such that for every $i \in \{1, 2, \ldots, g\}$, $|P_i| \geq \lfloor \frac{n}{g} \rfloor$ and $w(P_i) \leq \frac{w(U)}{g} + M$. Moreover, such a $\bar{\mathbf{P}}$ can be computed in time $\mathcal{O}(n(\log n + \log g))$.*

Proof. Consider the following greedy algorithm. Reorder the items in U so that $w(u_1) \geq w(u_2) \geq \cdots \geq w(u_n)$. Initialize $P_1 = P_2 = \cdots = P_g = \emptyset$. Now, for $i = 1, 2, \ldots, \lceil n/g \rceil$, perform the following operations. Order the sets $P_1, P_2, \ldots, P_g$ as $P_{q_1}, P_{q_2}, \ldots, P_{q_g}$ so that $w(P_{q_1}) \leq P_{q_2} \leq \cdots \leq w(P_{q_g})$. Then, for every $t = 1, 2, \ldots, x$ where $x = g$ if $i \cdot g \leq n$ and $x = (n \mod g)$ otherwise (which may correspond only to the iteration where $i = \lceil n/g \rceil$): insert $u_{(i-1) \cdot g + t}$ into P_{q_t}. This completes the description of the algorithm.

Clearly, the algorithm can be executed in time $\mathcal{O}(n(\log n + \log g))$. Moreover, it is clear that $\bar{\mathbf{P}} = (P_1, P_2, \ldots, P_g)$ is a partition of U, and as in each iteration corresponding to i, one item exactly is inserted to each set in $\bar{\mathbf{P}}$, we immediately have that for every $i \in \{1,2,\ldots,g\}$, $|P_i| \geq \lfloor \frac{n}{g} \rfloor$. For every $s \in \{1,2,\ldots,n\}$, denote $U_s = \{u_1, u_2, \ldots, u_s\}$. To complete the proof, we prove the following claim using induction on i.

Claim 4.3. *For every $s \in \{1,2,\ldots,n\}$, after the s^{th} item is inserted, the maximum difference between the weight of any two sets in $\bar{\mathbf{P}}$ is M.*

Proof of Claim 4.3. In the basis, where $s = 1$, only one item has been inserted, and therefore the claim trivially holds. Now, suppose that the claim is correct for $s-1 \geq 1$, and let us prove it for s. Let $i = \lceil n/g \rceil$ and $t = (s \mod g) + 1$. So, u_s is inserted into P_{q_t}. From the inductive hypothesis and as only the weight of P_{q_t} is changed, for every $j, j' \in \{q_1, q_2, \ldots, q_g\} \setminus \{q_t\}$, $|w(P_{q_j}) - w(P_{q_{j'}})| \leq M$. So it remains prove that for every $q_j \in \{1,2,\ldots,g\} \setminus \{q_t\}$, $|w(P_{q_t}) - w(P_{q_j})| \leq M$. We consider two cases. First, suppose that $j > t$. Then, by the ordering in the i^{th} iteration, we have that $w(P_{q_t} \setminus \{u_s\}) \leq w(P_{q_j})$. Hence, $w(P_{q_t}) - w(u_s) \leq w(P_{q_j})$, which means that $w(P_{q_t}) \leq w(P_{q_j}) + M$. Moreover, from the inductive hypothesis, $w(P_{q_j}) \leq w(P_{q_t} \setminus \{u_s\}) + M$, and therefore $w(P_{q_j}) \leq w(P_{q_t}) + M$. Second, suppose that $j < t$. Let u_r be the last item inserted into P_{q_j}. So, by the ordering of U, $w(u_r) \geq w(u_s)$. Then, by the ordering in the i^{th} iteration, we have that $w(P_{q_j} \setminus \{u_r\}) \leq w(P_{q_t} \setminus \{u_s\})$. Hence, $w(P_{q_j}) - w(u_r) \leq w(P_{q_t}) - w(u_s)$, which means that $w(P_{q_j}) \leq w(P_{q_t}) + w(u_r) \leq w(P_{q_t}) + M$. Moreover, from the inductive hypothesis, $w(P_{q_t} \setminus \{u_s\}) \leq w(P_{q_j} \setminus \{u_r\}) + M$, and hence $w(P_{q_t}) \leq w(P_{q_j}) + w(u_s) - w(u_r) + M$. Therefore, because $w(u_r) \geq w(u_s)$, we have that $w(P_{q_t}) \leq w(P_{q_j}) + M$. This completes the proof of the claim. □

At the end of the algorithm, the average weight of a set in $\bar{\mathbf{P}}$ is $\frac{w(U)}{g}$, and hence there exists a set in $\bar{\mathbf{P}}$ whose weight is at most $\frac{w(U)}{g}$. Thus, by Claim 4.3 with $i = n$, the maximum weight of a set in $\bar{\mathbf{P}}$ is at most $\frac{w(U)}{g} + M$. This completes the proof of Lemma 4.4. □

We are now ready to prove our combinatorial result by using Observation 4.3, Lemma 4.2, Lemma 4.3 and Lemma 4.4.

Theorem 4.1. *Let G be a graph with a nice tree decomposition $\mathcal{T} = (T, \beta)$ of width w. For every $0 < \alpha < 1$ and $\ell \in \mathbb{N}$, there exists a partition $\bar{\mathbf{V}} = (V_1, V_2, \ldots, V_{\lceil \frac{1}{\alpha} \rceil})$ of $V(G)$ such that, for every $i \in \{1,2,\ldots,\lceil \frac{1}{\alpha} \rceil\}$:* (i) *$|V_i| \leq \lceil \alpha n \rceil + \lceil n/\ell \rceil$;* (ii) *$V_i$ is a $(\mathcal{T}, \lfloor \alpha(w+\ell) \rfloor - \ell)$-hitting set. Moreover, such a partition can be computed in time $\mathcal{O}(|V(T)| \cdot (w+\ell) + (w+\ell) \cdot (\log(w+\ell) + \log(\frac{1}{\alpha})))$.*

Proof. The algorithm works as follows. First, call ColorALG on $(G, \mathcal{T}, \ell)$ to obtain col, which, by Lemma 4.2, is an $(\ell, \mathcal{T})$-proper coloring. Then, let $U = \{u_1, u_2, \ldots, u_{w+\ell}\}$ be a collection of $w + \ell$ items, and for each $i \in \{1,2,\ldots,w+\ell\}$, define the weight of u_i, denoted by $w'(u_i)$, to be $|\{v \in V(G) : \mathsf{col}(v) = i\}|$. Let $g = \lceil \frac{1}{\alpha} \rceil$. Call the algorithm in Lemma 4.4 to obtain a partition $\bar{\mathbf{P}} = (P_1, P_2, \ldots, P_g)$ of U. Lastly, for every $i \in \{1,2,\ldots,\lceil \frac{1}{\alpha} \rceil\}$, define $V_i = \{v \in V(G) : u_{\mathsf{col}(v)} \in P_i\}$. So, $\bar{\mathbf{V}} = (V_1, V_2, \ldots, V_{\lceil \frac{1}{\alpha} \rceil})$ is a partition of $V(G)$.

By Observation 4.3 and Lemma 4.4, the time complexity is indeed $\mathcal{O}(|V(T)| \cdot (w+\ell) + (w+\ell) \cdot (\log(w+\ell) + \log(\frac{1}{\alpha})))$. Now, we prove that the two properties in the theorem statement hold. To this end, consider some $i \in \{1,2,\ldots,\lceil \frac{1}{\alpha} \rceil\}$. We first prove that Property (i) holds. For this purpose, first note that by Lemma 4.3, for every $j \in \{1,2,\ldots,w+\ell\}$, $|\{v \in V(G) : \mathsf{col}(v) = j\}| \leq \lceil n/\ell \rceil$. Therefore, the maximum weight M of an item in U is at most $\lceil n/\ell \rceil$. By Lemma 4.4, this means that the weight of every set in $\bar{\mathbf{P}}$ is at most $w(U)/g + M \leq \lceil \alpha n \rceil + \lceil n/\ell \rceil$. Since $|V_i|$ equals the weight of P_i, the proof that Property (i) holds is complete.

We now prove that Property (ii) holds. To this end, consider some node $x \in V(T)$. By Lemma 4.4, for every $i \in \{1,2,\ldots,\lceil \frac{1}{\alpha} \rceil\}$, $|P_i| \geq \lfloor \alpha(w+\ell) \rfloor$. Denote $P_i = \{u_{j_1}, u_{j_2}, \ldots, u_{j_{|P_i|}}\}$. Because col is an $(\ell, \mathcal{T})$-proper coloring, we know that it assigns to each vertex in $\beta(x)$ a different color, and therefore, to color $\beta(x)$, it must use all except for at most $w + \ell - |\beta(x)|$ colors from $\{1,2,\ldots,w+\ell\}$. In particular, it must use at least $|P_i| - (w + \ell - |\beta(x)|) \geq$

$\lfloor \alpha(w+\ell) \rfloor - (w - |\beta(x)| + \ell)$ colors from $\{j_1, j_2, \ldots, j_{|P_i|}\}$. Observe that, by the definition of V_i, V_i contains at least $\lfloor \alpha(w+\ell) \rfloor - (w - |\beta(x)| + \ell)$ vertices of $\beta(x)$. So, we have that

$$\begin{aligned} |\beta(x) \setminus V_i| &\leq |\beta(x)| - (\lfloor \alpha(w+\ell) \rfloor - (w - |\beta(x)| + \ell)) \\ &\leq w - (\lfloor \alpha(w+\ell) \rfloor - \ell). \end{aligned}$$

This completes the proof. $\square$

4.3 Deterministic Algorithms Improving Upon SETH-based Lower Bounds

Our scheme will be built upon Theorem 4.1. For its presentation for deletion problems, where we make use of a linear-vertex kernel, we need to compute a parameter k that closely approximates the size of an optimal solution. For this purpose, we will use the following lemma.

Lemma 4.5. *Let Π be a graph minimization problem that, for every $0 < \epsilon' < 1$, admits an $\mathcal{O}(T_{\epsilon'})$-time algorithm that, given an instance I of Π and $k \in \mathbb{N}$,*[5] *if $\mathsf{opt}(I) \leq k$, then it returns a solution for I of size at most $(1+\epsilon)k$.*[6] *Then, for every $0 < \epsilon < 1$ and $0 < \delta < \epsilon$, Π admits a $(1+\epsilon)$-approximation algorithm that runs in time $\mathcal{O}(\frac{1}{\delta}\log|I| \cdot T_{\epsilon'})$ where $\epsilon' = \epsilon - \delta$.*

Proof. The algorithm works as follows. For $i = 1, 2, \ldots, x$, where x is the smallest integer such that $(1+\frac{\delta}{2})^x \geq |I|$, call the algorithm in the supposition of the lemma on I with ϵ' and $k = (1+\frac{\delta}{2})^i$, and let S_i be its output. Lastly, among all sets S_i, $1 \leq i \leq x$, that are solutions, return one of minimum size.

First, notice that $x = \mathcal{O}(\frac{1}{\delta}\log|I|)$, and hence the time complexity of the algorithm we described is $\mathcal{O}(\frac{1}{\delta}\log|I| \cdot T_{\epsilon'})$. Now, notice that we consider an iteration where $\mathsf{opt}(I) \leq k \leq (1+\frac{\delta}{2}\mathsf{opt}(I)$. In that iteration, because $\mathsf{opt}(I) \leq k$ and $\frac{\delta}{2}\epsilon < \frac{\delta}{2}$, we are guaranteed to compute a solution whose size is at most

$$(1+\epsilon')k \leq (1+\epsilon')(1+\frac{\delta}{2})\mathsf{opt}(I) = (1+\epsilon-\delta)(1+\frac{\delta}{2})\mathsf{opt}(I) = (1+\epsilon+\frac{\delta}{2}\epsilon-\frac{\delta}{2}-\frac{\delta^2}{2})\mathsf{opt}(I) \leq (1+\epsilon)\mathsf{opt}(I).$$

Because the algorithm returns the solution of minimum size among all those it computes, it will returns a solution of size at most $(1+\epsilon)\mathsf{opt}(I)$. $\square$

We further adapt Lemma 4.5 as follows.

Lemma 4.6. *Let $c > 0$. Let Π be a parameterized graph minimization problem, parameterized by the treewidth of the input graph, that admits* (i) *an α-approximate ck-vertex kernel having a tree decomposition maintenance procedure that runs in polynomial time, and* (ii) *for every $0 < \epsilon' < 1$, an $\mathcal{O}(T_{\epsilon'}(I))$-time parameterized algorithm that, given an instance I of Π and $k \in \mathbb{N}$, if $\mathsf{opt}(I) \leq k$ and the graph $G \in I$ has at most ck vertices, then it returns a solution for I of size at most $(1+\epsilon')k$. Then, for every $0 < \epsilon < 1$ and $0 < \delta < \epsilon$, Π admits an $\alpha(1+\epsilon)$-approximation algorithm that runs in time $\mathcal{O}(\frac{1}{\delta}\log|I| \cdot T_{\epsilon'}(I) + n^{\mathcal{O}(1)})$ where $\epsilon' = \epsilon - \delta$.*

Proof. Due to Lemma 4.5, it suffices to prove that for every $0 < \epsilon' < 1$, Π admits an $\mathcal{O}(T_{\epsilon'}+n^{\mathcal{O}(1)})$-time algorithm that, given an instance I of Π and $k \in \mathbb{N}$, if $\mathsf{opt}(I) \leq k$, then it returns a solution for I of size at most $(1+\epsilon')k$. We now describe such an algorithm. To this end, consider some instance I of Π.

1. Call the reduction procedure of the kernelization algorithm given in supposition *(i)* along with the treewidth maintenance procedure to obtain an instance I' of Π and $k' \leq k$, where the graph $G' \in I'$ has at most ck' vertices.[7]

[5]We assume that maximum size of a solution for I is $|I|$.

[6]If $k < \mathsf{opt}(I)$, the algorithm can return anything.

[7]Notice that having the treewidth mainenance procedure is required here, else we obtain an instance of the unparameterized version of Π rather than of Π, which the algorithm given in supposition *(ii)* does not solve.

2. Call the parameterized algorithm given in supposition *(ii)* to solve I', and let S' denote the solution.

3. Call the lifting procedure of the kernelization algorithm given in supposition *(i)* with S' to obtain a solution S for I.

Clearly, the algorithm we have just described runs in time $\mathcal{O}(T_{\epsilon'}(I') + n^{\mathcal{O}(1)}) = \mathcal{O}(T_{\epsilon'}(I) + n^{\mathcal{O}(1)})$ (where $T_{\epsilon'}(I') \leq T_{\epsilon'}(I)$ since the reduction and tree decomposition maintenance procedures do not increase the measures of the input instance). Now, notice that $|S'| \leq (1+\epsilon')k' = (1+\epsilon')\frac{k'}{\mathsf{opt}(I')} \cdot \mathsf{opt}(I')$. Thus, because the lifting procedure incurs an extra factor of α, we have that $|S| \leq \alpha(1+\epsilon')\frac{k'}{\mathsf{opt}(I')} \cdot \mathsf{opt}(I)$. Recall that we consider only reduction procedures that do not increase the ratio between the parameter and the optimum. Thus, $\frac{k'}{\mathsf{opt}(I')} \leq \frac{k}{\mathsf{opt}(I)}$, which yields that $|S| \leq \alpha(1+\epsilon')k$. □

We are now ready to present the two main theorems of this subsection.

Theorem 4.2. *Let Π be a composable parameterized graph minimization problem, parameterized by the treewidth of the input graph, that admits* (i) *an α-approximate ck-vertex kernel having a tree decomposition maintenance procedure that runs in polynomial time, and* (ii) *a parameterized algorithm that runs in time $\mathcal{O}(b^w n^p)$. Then, for every fixed constant $\epsilon > 0$, Π admits an $\alpha(1+\epsilon)$-approximation algorithm that runs in time $\mathcal{O}(b^{(1-\frac{\epsilon}{c-1})w+o(w)}n^p + n^{\mathcal{O}(1)})$.*

Proof. First, notice that when $w \leq \log n$, then $\mathcal{O}(b^w n^p) = n^{\mathcal{O}(1)}$, and the proof is complete by simply using the algorithm given by supposition *(ii)*. Thus, we assume that $w > \log n$. In this case, due to the $2^{o(w)}$ factor in the time complexity, it suffices to attain $\mathcal{O}(b^{(1-\frac{\epsilon}{c-1})w+o(w)}n^p \log^{\mathcal{O}(1)} n + n^{\mathcal{O}(1)})$. Therefore, due to Lemma 4.6, it suffices to prove for every $0 < \epsilon < 1$, Π admits an $\mathcal{O}(b^{(1-\frac{\epsilon}{c-1})w+o(w)}n^p + n^{\mathcal{O}(1)})$-time parameterized algorithm that, given an instance I of Π and $k \in \mathbb{N}$, if $\mathsf{opt}(I) \leq k$ and the graph $G \in I$ has at most ck vertices, then it returns a solution for I of size at most $(1+\epsilon-\frac{1}{\log n})k$. We now describe such an algorithm, based on Theorem 4.1. Let G be the input graph (where $|V(G)| \leq ck$), having a tree decomposition $\mathcal{T} = (T, \beta)$ of width w. By Proposition 4.1, we may assume that $\mathcal{T}$ is a nice tree decomposition, and $|V(T)| \leq 16(w+2)n$. Then, the algorithm works as follows.

1. Denote $\alpha = \frac{\epsilon}{c-1} - \frac{q}{\log\log n}$ (where q is a fixed constant that will be determined later), and $\ell = \lceil \log\log n \rceil = o(w)$ (because $w > \log n$). Then, use the algorithm in Theorem 4.1 to compute a partition $\mathbf{\overline{V}} = (V_1, V_2, \ldots, V_{\lceil \frac{1}{\alpha} \rceil})$ of $V(G)$ such that, for every $i \in \{1, 2, \ldots, \lceil \frac{1}{\alpha} \rceil\}$: *(i)* $|V_i| \leq \lceil \alpha n \rceil + \lceil n/\ell \rceil$; *(ii)* V_i is a $(\mathcal{T}, \lfloor \alpha(w+\ell) \rfloor - \ell)$-hitting set.

2. For every $i \in \{1, 2, \ldots, \lceil \frac{1}{\alpha} \rceil\}$:

   (a) Denote $G_i = G - V_i$, and let $\mathcal{T}_i = \mathcal{T}_{V_i}$ (see Definition 4.4).

   (b) Call the parameterized algorithm given by supposition *(ii)* on the subinstance induced by G_i and $\mathcal{T}_i$. Let S_i denote the solution.

3. Let $\widehat{i} \in \{1, 2, \ldots, \lceil \frac{1}{\alpha} \rceil\}$ be such that $|S_{\widehat{i}} \cup V_{\widehat{i}}|$ is minimum. Return $S_{\widehat{i}} \cup V_{\widehat{i}}$.

First, notice that since Π is composable, $S_{\widehat{i}} \cup V_{\widehat{i}}$ is indeed a solution. We now consider the time complexity of the algorithm. By Theorem 4.1, Step 1 is executed in time $\mathcal{O}(|V(T)| \cdot (w+\ell) + (w+\ell) \cdot \log(w+\ell))$. For every $i \in \{1, 2, \ldots, \lceil \frac{1}{\alpha} \rceil\}$, because V_i is a $(\mathcal{T}, \lfloor \alpha(w+\ell) \rfloor - \ell)$-hitting set, the width of $\mathcal{T}_i$ is at most $w - (\lfloor \alpha(w+\ell) \rfloor - \ell) \leq (1-\frac{\epsilon}{c-1})w + o(w)$, and hence each iteration of Step 2 is executed in time $\mathcal{O}(b^{(1-\frac{\epsilon}{c-1})w+o(w)}n^p)$. Lastly, Step 3 is executed in linear time, hence overall the time complexity is within the required bound.

It remains to prove that $|S_{\widehat{i}} \cup V_{\widehat{i}}| \leq (1+\epsilon-\frac{1}{\log n})k$. To this end, consider some optimal hypothetical solution $S^\star$. Then, $|S^\star| \leq k$. By the pigeon-hole principle, there exists $i \in$

$\{1,2,\ldots,\lceil\frac{1}{\alpha}\rceil\}$, denoted by $i^\star$, such that $|S^\star\cap V_i|\geq\lfloor\alpha|S^\star|\rfloor$. Because $|S_{\widehat{i}}\cup V_{\widehat{i}}|\leq|S_{i^\star}\cup V_{i^\star}|$, $|V_{i^\star}|\leq\lceil\alpha n\rceil+\lceil n/\ell\rceil$ and $n\leq ck$, it suffices to prove that $|S_{i^\star}|\leq(1+\epsilon-\frac{1}{\log n})k-\lceil\alpha ck\rceil-\lceil ck/\ell\rceil$. Because Π is composable, $S^\star\setminus V_{i^\star}$ is a solution for the subinstance induced by $G_{i^\star}$ and $\mathcal{T}_{i^\star}$, and as $S_{i^\star}$ is an optimal solution for this subinstance, we have that $|S_{i^\star}|\leq|S^\star\setminus V_{i^\star}|\leq|S^\star|-|S^\star\cap V_{i^\star}|\leq k-\lfloor\alpha k\rfloor\leq(1-\alpha)k+1$. Hence, it suffices to prove that $(1-\alpha)k+1\leq(1+\epsilon-\frac{1}{\log n})k-\lceil\alpha ck\rceil-\lceil ck/\ell\rceil$. Now, note that

$$\begin{aligned}(1+\epsilon-\tfrac{1}{\log n})k-\lceil\alpha ck\rceil-\lceil ck/\ell\rceil &\geq(1+\epsilon-\tfrac{1}{\log n})k-\alpha ck-ck/\ell-2\\ &\geq(1+\epsilon-\tfrac{1}{\log n})k-\tfrac{\epsilon ck}{c-1}+\tfrac{qck}{\log\log n}-\tfrac{ck}{\lceil\log\log n\rceil}-2\\ &=(1-\tfrac{\epsilon}{c-1}+\tfrac{q}{\log\log n})k+\tfrac{q(c-1)k}{\log\log n}-\tfrac{ck}{\lceil\log\log n\rceil}-\tfrac{k}{\log n}-2\\ &\geq(1-\tfrac{\epsilon}{c-1}+\tfrac{q}{\log\log n})k+1\\ &=(1-\alpha)k+1.\end{aligned}$$

Here, the third inequality follows by choosing a large enough (but a fixed constant) q and since $ck\geq n$. This completes the proof. $\square$

We remark that the proof of our second theorem is similar (though not identical) to the proof of the first theorem, mainly as both are based on Theorem 4.1.

Theorem 4.3. *Let Π be a d-composable parameterized graph maximization problem, parameterized by the treewidth of the input graph, that admits a parameterized algorithm that runs in time $\mathcal{O}(b^wn^p)$. Then, for every fixed constant $\epsilon>0$, Π admits a $(1+\epsilon)$-approximation algorithm that runs in time $\mathcal{O}(b^{(1-\frac{\epsilon}{d})w+o(w)}n^p+n^{\mathcal{O}(1)})$.*

Proof. First, notice that when $w\leq\log n$, then $\mathcal{O}(b^wn^p)=n^{\mathcal{O}(1)}$, and the proof is complete by simply using the algorithm given by supposition *(ii)*. Thus, we assume that $w>\log n$. We now describe the algorithm, based on Theorem 4.1. Let G be the input graph, having a tree decomposition $\mathcal{T}=(T,\beta)$ of width w. By Proposition 4.1, we may assume that $\mathcal{T}$ is a nice tree decomposition, and $|V(T)|\leq16(w+2)n$. Then, the algorithm works as follows.

1. Denote $\alpha=\frac{\epsilon}{d}$ (where q is a fixed constant that will be determined later), and $\ell=\lceil\log\log n\rceil=o(w)$ (because $w>\log n$). Then, use the algorithm in Theorem 4.1 to compute a partition $\overline{\mathbf{V}}=(V_1,V_2,\ldots,V_{\lceil\frac{1}{\alpha}\rceil})$ of $V(G)$ such that, for every $i\in\{1,2,\ldots,\lceil\frac{1}{\alpha}\rceil\}$: *(i)* $|V_i|\leq\lceil\alpha n\rceil+\lceil n/\ell\rceil$; *(ii)* V_i is a $(\mathcal{T},\lfloor\alpha(w+\ell)\rfloor-\ell)$-hitting set.
2. For every $i\in\{1,2,\ldots,\lceil\frac{1}{\alpha}\rceil\}$:
   (a) Denote $G_i=G-V_i$, and let $\mathcal{T}_i=\mathcal{T}_{V_i}$ (see Definition 4.4).
   (b) Call the parameterized algorithm given by supposition *(ii)* on the subinstance induced by G_i and $\mathcal{T}_i$. Let $\mathcal{S}_i$ denote the solution.
3. Let $\widehat{i}\in\{1,2,\ldots,\lceil\frac{1}{\alpha}\rceil\}$ be such that $|\mathcal{S}_{\widehat{i}}|$ is maximum. Return $\mathcal{S}_{\widehat{i}}$.

First, notice that since Π is composable, $\mathcal{S}_{\widehat{i}}$ is indeed a solution. We now consider the time complexity of the algorithm. By Theorem 4.1, Step 1 is executed in time $\mathcal{O}(|V(T)|\cdot(w+\ell)+(w+\ell)\cdot\log(w+\ell))$. For every $i\in\{1,2,\ldots,\lceil\frac{1}{\alpha}\rceil\}$, because V_i is a $(\mathcal{T},\lfloor\alpha(w+\ell)\rfloor-\ell)$-hitting set, the width of $\mathcal{T}_i$ is at most $w-(\lfloor\alpha(w+\ell)\rfloor-\ell)\leq(1-\frac{\epsilon}{d})w+o(w)$, and hence each iteration of Step 2 is executed in time $\mathcal{O}(b^{(1-\frac{\epsilon}{d})w+o(w)}n^p)$. Lastly, Step 3 is executed in linear time, hence overall the time complexity is within the required bound.

It remains to prove that $|\mathcal{S}_{\widehat{i}}|\geq(1-\epsilon)\mathsf{opt}(I)$. To this end, consider some optimal hypothetical solution $\mathcal{S}^\star$. Then, $|\mathcal{S}^\star|=\mathsf{opt}(I)$. By the pigeon-hole principle and because Π is d-composable, there exists $i\in\{1,2,\ldots,\lceil\frac{1}{\alpha}\rceil\}$, denoted by $i^\star$, such that $|(\bigcup\mathcal{S}^\star)\cap V_i|\leq\lfloor\alpha d|\mathcal{S}^\star|\rfloor$. Because

$|\mathcal{S}_{\widehat{i}}| \geq |\mathcal{S}_{i^\star}|$, it suffices to prove that $|\mathcal{S}_{i^\star}| \geq (1-\epsilon)\mathsf{opt}(I)$. Because Π is composable, $\{S \in \mathcal{S}^\star : S \cap V_{i^\star} = \emptyset\}$ is a solution for the subinstance induced by G_i and $\mathcal{T}_i$, and as $\mathcal{S}_{i^\star}$ is an optimal solution for this subinstance, we have that $|\mathcal{S}_{i^\star}| \geq |\{S \in \mathcal{S}^\star : S \cap V_{i^\star} = \emptyset\}| \geq |\mathcal{S}^\star| - |(\bigcup \mathcal{S}^\star) \cap V_{i^\star}| \geq \mathsf{opt}(I) - \lfloor \alpha d \mathsf{opt}(I) \rfloor \geq (1-\alpha d)\mathsf{opt}(I)$. Hence, it suffices to prove that $(1-\epsilon)\mathsf{opt}(I) \leq (1-\alpha d)\mathsf{opt}(I)$. Now, note that

$$(1-\alpha d)\mathsf{opt}(I) = (1 - \frac{\epsilon}{d}d)\mathsf{opt}(I) = (1-\epsilon)\mathsf{opt}(I).$$

This completes the proof. □

Lastly, before presenting concrete applications, let us state known results regarding the parameterized complexity of the problems in Corollary 2 with respect to treewidth.

Proposition 4.3. *Each of the following problems admits a $\mathcal{O}(b^w n)$-time algorithm:* Vertex Cover *where $b = 2$* [CFK+15]*;* Component Order Connectivity *where $b = \ell$* [DDvtH16]*;* Bounded-Degree Vertex Deletion *where $b = (d+2)$;*[8] Triangle Packing *where $b = 2$* [vRBR09, vRBvL+18].

Based on Corollaries 1 and 2, and Proposition 4.3, we have the following as a corollary of Theorems 4.2 and 4.3.

Corollary 3. *For every fixed constant $\epsilon > 0$, each of the following problems admits a $(1+\epsilon)$-approximation algorithm that runs in time $b^{w+o(w)}n + n^{\mathcal{O}(1)}$:* Vertex Cover *where $b = 2^{1-\epsilon}$* Component Order Connectivity *where $b = \ell^{1-\frac{\epsilon}{2\ell-1}}$;* Bounded-Degree Vertex Deletion *where $b = (d+2)^{1-\frac{\epsilon}{d^3+4d^2+5d+1}}$; $\mathcal{F}$-*Packing *for every graph family $\mathcal{F}$ that consists of graphs on at most d vertices where $b = b_{\mathcal{F}}^{1-\frac{\epsilon}{d}}$, where $b_{\mathcal{F}}$ is the best known constant such that* $\mathcal{F}$-Packing *is solvable in time $\mathcal{O}(b_{\mathcal{F}}^w n)$. For example, for* Triangle Packing *$b_{\mathcal{F}} = 2$.*

Here, it is worthwhile to note that for all of the problems in Corollary 3, the stated b (in that corollary) is also *optimal* under the SETH. Indeed, this is the case for Vertex Cover [LMS18a], and because Vertex Cover is the special case of Component Order Connectivity where $\ell = 1$ and of Bounded-Degree Vertex Deletion where $d = 0$, the same holds for these two problems as well. Moreover, this is also the case for Triangle Packing [LMS18a].

We remark that the authors of [FLL+19] have recently proved that Cluster Vertex Deletion admits a $(1+\epsilon)$-approximate linear-vertex kernel,[9] and hence, our scheme also yields for this problem an $\mathcal{O}(\widehat{b}^{w+o(w)}n + n^{\mathcal{O}(1)})$-time $(1+\epsilon)$-approximate parameterized algorithm where $\widehat{b}$ is strictly smaller than the best known b for which the problem can be solved in time $\mathcal{O}(b^w n)$. As this result is yet unpublished, we have not included it in Corollary 3, but yet wanted to comment that the requirement in Theorem 4.2 to have an approximate kernel rather than an exact kernel does make it more powerful.

4.4 Deterministic Algorithms Improving Upon ETH-based Lower Bounds

We first prove the main theorem of this subsection. We remark that some of the arguments in this proof resemble those of [HKP20] (who used them to design approximate Turing kernels).

Theorem 4.4. *Let Π be a composable parameterized minimization graph problem that is closed under disjoint union and that, when parameterized by the solution size k, admits a c-approximation parameterized algorithm that runs in time $\mathcal{O}(b^k n^p)$. Then, for every fixed constant $\epsilon > 0$, Π admits a $(2c+\epsilon)$-approximation algorithm that runs in time $\mathcal{O}(b^{\frac{2cw}{\epsilon}} w \cdot n^{p+2})$.*

[8]Here, the algorithm is a straightforward extension of the algorithm for Vertex Cover, where for each vertex in the bag we store whether it is deleted, and if not, what is its degree (between 0 and d).

[9]It is an open problem whether this problem admits a (1-approximate) linear-vertex kernel [FLL+19].

Towards the proof of this theorem, we will prove the following lemma

Lemma 4.7. *Let Π be a composable parameterized graph minimization problem that, when parameterized by the solution size k, admits a c-approximation parameterized algorithm that runs in time $\mathcal{O}(b^k n^p)$. Then, for every fixed constant $\epsilon > 0$, there exists an $\mathcal{O}(|V(T)|b^{\frac{2cw}{\epsilon}} n^p)$-time algorithm that given an I instance of Π where $G \in I$ is the graph, having a tree decomposition $\mathcal{T} = (T, \beta)$ where T is rooted so that each node has at most two children, outputs either* (i) *a solution of size at most $c \cdot \mathsf{opt}(I)$, or* (ii) *a node $x \in V(T)$ such that the optimum of the subinstance of I induced $G[\gamma(x)]$ and $\mathcal{T}_x$ is at least $\frac{w}{\epsilon}$ along with a solution for this subinstance of size at most $\frac{2cw}{\epsilon} + w$ that contains $\beta(x)$.*

Proof. The algorithm works as follows. For every $y \in V(T)$, call the c-approximation parameterized algorithm in the supposition of the lemma on the subinstance of I induced by $G[\gamma(y)]$ and $\mathcal{T}_y$ with $k = \frac{w}{\epsilon}$, and let S_y denote its output; if S_y is not a solution for this subinstance, replace it by a dummy such that when its size is queried, it equals ∞. If $|S_r| \le \frac{cw}{\epsilon}$ where r is the root of T, then output S_r. Else, let x be some node in T such that $|S_x| > \frac{cw}{\epsilon}$, but for every child y of x in T, $|S_y| \le \frac{cw}{\epsilon}$ (observe that such a node x exists since $|S_r| \le \frac{cw}{\epsilon}$, and for every leaf ℓ of T, $|S_\ell| \le w$). Then, output x and $\beta(x) \cup \bigcup_y S_y$ where y ranges over the children of x in T (of which there are at most two). This completes the description of the algorithm. Clearly, its time complexity is $\mathcal{O}(|V(T)|b^{\frac{2cw}{\epsilon}} n^p)$.

First, consider the case where the algorithm returns S_r. So, if $\mathsf{opt}(I) \ge \frac{w}{\epsilon}$, then S_r, being of size at most $\frac{cw}{\epsilon}$ and verified to be a solution, is a c-approximate solution for I, and hence the algorithm complies with case *(i)* in the lemma. Else, $\mathsf{opt}(I) < \frac{w}{\epsilon} = k'$, then the correctness of the parameterized algorithm in the supposition implies that S_r must be a c-approximate solution for I, and hence the algorithm again complies with case *(i)*.

Now, consider the case where the algorithm returns x and $\beta(x) \cup \bigcup_y S_y$, which we will denote by S'. Then, because $|S_y| \le \frac{cw}{\epsilon}$ for each of the (at most two) children y of x in T, we have that $|S'| \le \frac{2cw}{\epsilon} + w$. Moreover, because Π is composable and as each S_y where y is a child of x was verified to be a solution for the subinstance of I induced $G[\gamma(y)]$ and $\mathcal{T}_y$ (else its size should have been ∞), S' is a solution for the subinstance of I induced $G[\gamma(x)]$ and $\mathcal{T}_x$, and clearly it contains $\beta(x)$. Lastly, because $|S_x| > \frac{cw}{\epsilon}$, the correctness of the parameterized algorithm in the supposition implies that the optimum of the subinstance of I induced by $G[\gamma(x)]$ and $\mathcal{T}_x$ is larger than $k = \frac{w}{\epsilon}$. □

We are now ready to prove Theorem 4.4.

Proof of Theorem 4.4. We first describe the algorithm. To this end, let G and $\mathcal{T}$ denote the input graph and tree decomposition of G of width w, respectively. By Proposition 4.1, we may assume that $\mathcal{T}$ is a nice tree decomposition, and $|V(T)| \le 16(w+2)n$. So, the algorithm works as follows.

1. Initialize $i = 0$, $S^0 = \emptyset$, $G^0 = G$ and $(\mathcal{T}^0 = (T^0, \beta^0)) = (\mathcal{T} = (T, \beta))$.
2. As long as $V(T^i) \neq \emptyset$:
   (a) Call the algorithm in Lemma 4.7 on the subinstance I^i of I induced by G^i and $\mathcal{T}^i$ to obtain either *(i)* a solution $\widehat{S}^i$ of size at most $c \cdot \mathsf{opt}(I^i)$, or *(ii)* a node $x^i \in V(T^i)$ such that the optimum of the subinstance of $\widehat{I}^i$ induced $G^i[\gamma^i(x^i)]$ and $\mathcal{T}^i_{x^i}$ is at least $\frac{w}{\epsilon}$ along with a solution $\widehat{S}^i$ to this subinstance of size at most $\frac{2cw}{\epsilon} + w$ that contains $\beta(x^i)$.
   (b) In case *(i)*, perform $S^{i+1} \Leftarrow S^i \cup \widehat{S}^i$, $G^{i+1} \Leftarrow (\emptyset, \emptyset)$ (i.e., the empty graph) and let $\mathcal{T}^{i+1}$ be its unique tree decomposition where the tree is empty. Increase i by 1.

(c) In case *(ii)*, perform $S^{i+1} \Leftarrow S^i \cup \widehat{S}^i$, $G^{i+1} \Leftarrow G^i - \gamma^i(x^i)$, and $\mathcal{T}^{i+1} \Leftarrow (T^{i+1}, \beta^{i+1})$ where $T^{i+1} = T^i - V(T^i_{x^i})$ and for every $y \in V(T^{i+1})$, $\beta^{i+1}(y) = \beta^i(y) \setminus \gamma^i(x^i)$. (It should be clear that the new $\mathcal{T}^{i+1}$ is a tree decomposition of the new G^{i+1}). Increase i by 1.

3. Output S^i.

Observe that $S^i = \bigcup_{j=1}^{i}(\beta^j(x^j) \cup (\widehat{S}^j \setminus \beta^j(x^j)))$, and it satisfies the following property: the set of connected components of $G - \bigcup_{j=1}^{i} \beta^j(x^j)$ has a partition $(\mathcal{C}^1, \mathcal{C}^2, \ldots, \mathcal{C}^i)$ (where each $\mathcal{C}^j$ is a collection, possibly empty, of connected components of $G - \bigcup_{j=1}^{i} \beta^j(x^j)$) such that for every $j \in \{1, 2, \ldots, i\}$, $\widehat{S}^j \setminus \beta^j(x^j)$ is a solution for the subinstance induced by $\mathcal{C}^j$ (because Π is composable). Hence, since Π is closed under disjoint union, S^i is a solution for I. Moreover, for every $j \in \{1, 2, \ldots, i\}$ (possibly excluding $j = i$, in which case we obtain a c-approximate solution for the corresponding subinstance), $\widehat{S}^j$ is of size at most $\frac{2cw}{\epsilon} + w$, while the optimum of the subinstance induced by $G[\beta^j(x^j) \cup \bigcup_{C \in \mathcal{C}^j} V(C)]$ (and hence, since Π is composable, also of the restriction of any optimal solution for I to this subinstace) is at least $\frac{w}{\epsilon}$. Thus, since these subinstances are pairwise disjoint, the approximation ratio is at most $(\frac{2cw}{\epsilon} + w)/(\frac{w}{\epsilon}) = (2c + \epsilon)$ as stated in the lemma.

By Lemma 4.7, the time complexity of each iteration is $\mathcal{O}(|V(T')| b^{\frac{2cw}{\epsilon}} n^p)$ for a subtree T' of T (and where n is actually smaller than the original n). So, as there can be at most n iterations, the total time complexity is $\mathcal{O}(n|V(T)| b^{\frac{2cw}{\epsilon}} n^p) = \mathcal{O}(b^{\frac{2cw}{\epsilon}} w n^{p+2})$. This completes the proof. $\square$

As a corollary of Corollary 2, Theorem 4.4 and our results in Section 3, we derive the following.

Corollary 4. *For every fixed constant $\epsilon > 0$, each of the following problems admits a $(4 + \epsilon)$-approximation algorithm that runs in time $2^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(1)}$:* Directed (Subset) Feedback Vertex Set*,* Directed Odd Cycle Transversal*,* Undirected Multicut*.*

Here, it is worthwhile to note that for two of the problems in Corollary 4, the stated b is also known to be *optimal* under the ETH. Indeed, this is the case for Directed Feedback Vertex Set [BKN+18], which implies that this is the case also for Directed Odd Cycle Transveral because the former can be easily reduced to the latter while keeping the treewidth the same (unless the treewidth was 1, in which case it can increase to 2): for every arc (u, v) in the graph, add a new vertex $w_{u,v}$ and the arcs (u, w_{uv}) and (w_{uv}, v).

5 Approximating Weighted Instances

In this section, we present FPT-approximation algorithms for some well studied problems, such as (Subset) Directed Feedback Vertex Set, Bidirected Multicut and Directed Odd Cycle Transversal. More formally, assuming that the given instance admits a solution S_k^{OPT}, that is a minimum weight solution of cardinality at most k, our algorithm computes an (α, β)-approximate solution S to this instance, that is $|S| \leq \alpha k$ and $w(S) \leq \beta w(S_k^{OPT})$, for some constants α and β. We present some general techniques that will likely be useful for FPT-approximation for other weighted problems.

5.1 Approximating Bounded-Weight Instances

In this section we consider weighted instances (D, w) of the class of SCC $\mathcal{F}$-Transversal problems, where $w : V(D) \to [M]$ for some integer M. Let k be some integer and let S_k^{OPT} be a minimum weight solution such that $|S_k^{OPT}| \leq k$. Our goal is to obtain an (α, β)-approximate solution in time $M \cdot c^k n^{\mathcal{O}(1)}$, for some constants α, β and c. Note that this is an FPT-approximation algorithm parameterized by k.

Our approach is to reduce weighted instances to unweighted instances by introducing a number of copies of a vertex v, proportional to its weight, and then applying the FPT-approximation algorithms for the unweighted problems. Note that if we simply introduce $w(v)$ copies for each vertex $v \in V(D)$, then it leads to an unweighted instance with roughly $M \cdot |V(D)|$ vertices, and a solution on k vertices in the weighted instance would map to one with roughly Mk vertices in the unweighted instance. Hence, the running time of the approximation algorithms would be an exponential function of Mk, and the approximation factor would involve M as well. Here, we describe a more nuanced approach, which reduces the dependence on M in the running time to just linear, and completely eliminates it from the approximation factor. We begin by analyzing the effect of introducing copies of a vertex on a digraph.

5.1.1 Vertex Bundles and Separators

Let (D, w) be a weighted digraph with positive integer weights on the vertices. Let γ be an integer, and let w_γ be a weight function defined as by rounding up $w(v)$ to the nearest multiple of γ. That is

$$w_\gamma(v) = q\gamma, \quad \text{where } q = \lceil w(v)/\gamma \rceil$$

Observation 5.1. *Let $X \subseteq V(D)$ be a subset of vertices. Then $w_\gamma(X) \leq w(X) + |X|\gamma$.*

Let H be a digraph constructed from D and w_γ as follows: for every vertex $v \in V(D)$ introduce $w_\gamma(v)/\gamma = \lceil w(v)/\gamma \rceil$ vertices to H, denoted by Z_u; and for every arc $(u, v) \in A(D)$, introduce the arcs $\{(x, y) \mid x \in Z_u, y \in Z_v\}$ to H. We call $Z_v \subseteq V(H)$ the **vertex bundle** of $v \in V(D)$, and observe that these bundles partition $V(H)$.

The following lemma shows that minimal separators in H have an "all-or-nothing" property with respect to each bundle.

Lemma 5.1. *Let X, Y be disjoint vertex subsets in H. Let $S \subseteq V(H)$ be a minimal X, Y-separator in H. Then for every $v \in V(D)$ either $Z_v \subseteq S$ or $Z_v \cap S = \emptyset$.*

Proof. Suppose not; i.e. for some $v \in V(D)$, we have $u, u' \in Z_v$ such that $u \in S$ and $u' \notin S$. Note that, by definition $S \cap (X \cup Y) = \emptyset$. First suppose that $Z_v \cap X \neq \emptyset$, and since $X \cap S = \emptyset$, we can assume that $u' \in X \cap Z_v$. Since S is a minimal separator, there is a path P from X to Y such that $V(P) \cap S = \{u\}$. Then consider the path P' obtained, by taking the subpath P from u to Y, and substituting u' for u. Observe that P' is a path from X to Y that is not intercepted by S. But this is a contradiction. A similar argument applies in the case where $Z_v \cap Y \neq \emptyset$. Finally, consider the case when $Z_v \cap (X \cup Y) = \emptyset$. Since S is a minimal separator, there is a path P from X to Y in H such that $V(P) \cap S = \{u\}$. Then observe that the path P' from X to Y obtained from P by replacing u with u' is not intercepted by S, and hence S is not a separator. This is also a contradiction. $\square$

Corollary 5. *Let X, Y be disjoint vertex subsets in H. Let $S \subseteq V(H)$ be a minimal X, Y-separator in H. Then,*

- *For every vertex $v \in V(D)$, if $(X \cup Y) \cap Z_v \neq \emptyset$ then $Z_v \cap S = \emptyset$.*
- *If S is an x-y separator in H for vertices $x, y \in V(H)$, then it is also a Z_u-Z_v separator where $x \in Z_u, y \in Z_v$.*

5.1.2 Subset DFVS

We present an FPT-approximation for Subset Directed Feedback Vertex Set (Subset DFVS), parameterized by $M + k$, where the weights are integers upper-bounded by an integer M and the solution size is upper-bounded by k. Our input is a digraph D with weights $w : V(D) \to [M]$ where M is an integer, a subset of terminal arcs $T \subseteq A(D)$ and an integer k.

Here, we study a somewhat more general version of the problem where we are additionally given a subset of blacklisted vertices $B \subseteq D(V)$, and out goal is to compute a solution disjoint from it. Introducing blacklisted vertices will be helpful later in this section when we consider general weight functions. Let us fix an integer parameter γ whose value will be decided later. From w and γ we define the weight function w_γ, as described earlier. Then we construct the (unweighted) digraph H from D, γ and w_γ. We define $T_H = \{(u', v') \in A(H) \mid u' \in Z_u, v' \in Z_v, (u, v) \in T\}$ as the subset of terminal arcs in H.

Observation 5.2. *If C is a T_H-cycle in H, then $C_D = D[\{v \in V(D) \mid Z_v \cap V(C) \neq \emptyset\}]$ is strongly connected and contains a T-cycle. Conversely, if C' is a T-cycle in D then $C'_H = H[\{$ an arbitrary vertex $u \in Z_v \mid v \in V(C')\}]$ is a T_H-cycle in H.*

Lemma 5.2. *If $S \subset V(H)$ is an inclusion-wise minimal solution for (H, T_H), then for every vertex $v \in V(D)$, either $Z_v \subseteq S$ or $Z_v \cap S = \emptyset$*

Proof. Suppose not; and for some vertex $v \in V(D)$ there exist $u, u' \in Z_v$ such that $u \in S$ and $u' \notin S$. Then, as S is a minimal T_H-sfvs in H, there exists a T_H-cycle C such that $V(C) \cap S = \{u\}$. Now observe that the cycle C' obtained from C by substituting u' for u is not intercepted by S, and it is also a T_H-cycle by the construction of H. But this is a contradiction. $\square$

Observation 5.3. *Let S be a solution for (D, T) such that $|S| \leq k$. Then (H, T_H) has a solution S' of size at most $\lceil w(S)/\gamma \rceil + k$.*

The following lemma, which is the main lemma of this section presents a $(4, 8)$-FPT-approximation for Subset DFVS in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time.

Lemma 5.3. *Let $M > 0$ be an integer. Let (D, w, B, T, k) be an instance of* Subset DFVS *where D is a digraph on n vertices, $w : V(D) \to [M]$ is a weight function, $T \subseteq A(D)$ is a subset of terminal arcs, $B \subseteq V(D)$ is a subset of blacklisted vertices and k is an integer. Suppose that this instance admits a solution of cardinality k. Then there is an algorithm that runs in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)} \cdot M$ and outputs $S \subseteq V(D) \setminus B$ such that $S \cap C \neq \emptyset$ for every T-cycle C, $|S| \leq 4k$ and $w(S) \leq 8w(S_k^{OPT})$ where $S_k^{OPT} \subseteq V(D) \setminus B$ is a solution of minimum weight for (D, T) such that $|S_k^{OPT}| \leq k$.*

Proof. Let w' be the weight function on $V(D)$ that is defined as follows:

$$w'(v) = \begin{cases} 10Mk & \text{if } v \in B \\ w(v) & \text{otherwise} \end{cases}$$

Let $\mathsf{Opt}_k = w'(S_k^{OPT})$, and observe that $\mathsf{Opt}_k \leq Mk$. Let $\gamma = \lceil \mathsf{Opt}_k / k \rceil$. Note that while we don't know the value of Opt_k (and hence γ), we know that $\gamma \in [M]$. We will iterate over each choice for γ, and do the following.

Consider the weight function $w'_\gamma : V(D) \to [M]$ constructed from w' and γ (as described above). Let H be the digraph constructed from D, w'_γ and γ. Along with the terminal arc subset T_H constructed from T, we obtain an instance (H, T_H) of the problem. If (D, T) admits a solution of cardinality k and weight Opt_k, then by Observation 5.3, there is a minimal solution $S_H \subseteq \bigcup_{v \in S_k^{OPT}} Z_v$ of cardinality at most $\lceil \mathsf{Opt}_k/\gamma \rceil + k \leq 2k$ for (H, T_H). Further, consider a solution S' of cardinality at most $4k$ to the instance (H, T_H). For each blacklisted vertex $v \in B$ the vertex bundle $Z_v \subseteq V(H)$ contains at least $\lceil w'(v)/\gamma \rceil \geq \lceil w'(v)/M \rceil > 4k$ vertices. Hence by Lemma 5.2, for every blacklisted vertex $v \in B$, we have $Z_v \cap S' = \emptyset$.

Our next step is to apply Theorem 3.1 to the instance $(H, T_H, 2k)$, and obtain a solution S' of cardinality at most $4k$.[10] We can assume S' is a minimal solution, and hence by Lemma 5.2

[10]More precisely, S' is a solution for the instance (H, T_H) of cardinality $4k$ assuming that (D, T) admits a solution of cardinality k.

for every $v \in V(D)$, either $Z_v \subseteq S'$ or $Z_v \cap S' = \emptyset$. let $S = \{v \in V(D) \mid Z_v \subseteq S'\}$. It is immediate from the construction of (H, T_H) that $S \subseteq V(D) \setminus B$ is a solution for (D, T) and $|S| \le 4k$. Further, $w(S) = w'(S) \le w'_\gamma(S) = \gamma|S'| \le \gamma 4k \le 4\mathsf{Opt}_k + 4k \le 8\mathsf{Opt}_k$. Here, we use the fact that $\gamma \le 1 + \mathsf{Opt}_k/k$, and $\mathsf{Opt}_k \ge k$ as $w(v) \ge 1 \, \forall v \in V(D)$.

We iterate over each choice of $\gamma \in [M]$, and obtain a solution S_γ of (D, T) corresponding to it. We then output the one of the least weight. The correctness of this algorithm, and the bounds on its cardinality and weight are immediate from Theorem 3.1 and the above discussion. It only remains to bound the running time of this algorithm. We iterate over $|M|$ choices for γ, and for each choice we spend polynomial time on constructing the instance $(H, T_H, 2k)$ and then $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ (using Theorem 3.1) to compute an approximate solution. Hence the overall running time is $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)} \cdot M$. □

5.1.3 Bidirected Multicut

We present an FPT-approximation Bidirected Multicut, parameterized by $M + k$, where the weights are integers upper-bounded by an integer M and the solution size is upper-bounded by k. Our input is a digraph D with weights $w : V(D) \to [M]$ where M is an integer, a subset collection of terminal pairs $T \subseteq V(D) \times V(D)$, a subset of blacklisted vertices $B \subseteq D(V)$ and an integer k. Let us fix an integer parameter γ whose value will be decided later. From w and γ we define the weight function w_γ, as described earlier. Then we construct the (unweighted) digraph H from D, γ and w_γ. We define $T_H = \{(u', v') \in A(H) \mid u' \in Z_u, v' \in Z_v, (u, v) \in T\}$ as the subset of terminal arcs in H. The following observations and lemmas are proved in the same way as before.

Observation 5.4. *If P is a T_H-path in H, then $P_D = D[\{v \in V(D) \mid Z_v \cap V(C) \neq \emptyset\}]$ is a walk in D that contains T-path. Conversely, if P' is a T-path in D then $P'_H = H[\{$ an arbitrary vertex $v \in Z_v \mid v \in V(C')\}]$ is a T_H-path in H.*

Lemma 5.4. *If $S \subset V(H)$ is an inclusion-wise minimal solution for (H, T_H), then for every vertex $v \in V(D)$, either $Z_v \subseteq S$ or $Z_v \cap S = \emptyset$*

Observation 5.5. *Let S be a solution for (D, T) such that $|S| \le k$. Then H, T_H has a solution S' of size at most $\lceil w(S)/\gamma \rceil + k$.*

From Theorem 3.2, we obtain the following, by setting $\gamma = \left\lceil w(S_k^{OPT})/k \right\rceil$, where $S_k^{OPT} \subseteq V(D) \setminus B$ is a minimum weight solution for (D, T) such that $|S_k^{OPT}| \le k$.

Lemma 5.5. *Let $M > 0$ be an integer. Let (D, w, B, T, k) be an instance of* Bidirected Multicut *where D is a digraph, $w : V(D) \to [M]$ is a weight function, $T \subseteq V(D) \times V(D)$ is a subset of terminal pairs, $B \subseteq V(D)$ is a subset of blacklisted vertices and k is an integer. Suppose that this instance admits a solution of cardinality k. Then there is an algorithm that runs in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)} \cdot M$ and outputs $S \subseteq V(D) \setminus B$ such that $S \cap P \neq \emptyset$ for every T-path P, $|S| \le 4k$ and $w(S) \le 8w(S_k^{OPT})$ where $S_k^{OPT} \subseteq V(D) \setminus B$ is a solution of minimum weight for (D, T) such that $|S_k^{OPT}| \le k$.*

5.1.4 Directed OCT

We present an FPT-approximation for Directed OCT, parameterized by $M + k$, where the weights are integers upper-bounded by M and the solution size is upper-bounded by k. Our input is a digraph D with weights $w : V(D) \to [M]$ where M is an integer, a subset of blacklisted vertices $B \subseteq D(V)$ and an integer k. Let us fix an integer parameter γ whose value will be decided later. From w and γ we define the weight function w_γ, as described earlier. Then we construct the (unweighted) digraph H from D, γ and w_γ.

Observation 5.6. *If C is an odd-cycle in H, then $D[\{v \in V(D) \mid Z_v \cap V(C) \neq \emptyset\}]$ contains an odd-cycle. Conversely, if C' is an odd-cycle in D then $C'_H = H[\{$ an arbitrary vertex $v \in Z_v \mid v \in V(C')\}]$ is a odd-cycle in H.*

Proof. Given the odd-cycle C in H, suppose that there are two vertex $u, u' \in V(C)$ such that $u, u' \in Z_v$ for some $v \in V(D)$. Observe that as $|C|$ is odd, one of the two sub-paths of C, either from u to u' or from u' to u is of odd length. From this odd-length sub-path, by substituting u for u', we can obtain an odd cycle C' in H. We repeat this process until we obtain a odd-cycle C^* in H such that $|V(C^*) \cap Z_v| \leq 1$ for every $v \in V(D)$. Now it is clear that $C^*_D = D[\{v \in V(D) \mid Z_v \cap V(C) \neq \emptyset\}]$ is an odd-cycle in D. The reverse direction is trivial. □

The following observations and lemmas can be proved in the same way as before.

Lemma 5.6. *If $S \subset V(H)$ is an inclusion-wise minimal solution for (H, T_H), then for every vertex $v \in V(D)$, either $Z_v \subseteq S$ or $Z_v \cap S = \emptyset$*

Observation 5.7. *Let S be a solution for (D, T) such that $|S| \leq k$. Then H, T_H has a solution S' of size at most $\lceil w(S)/\gamma \rceil + k$.*

From Theorem 3.3, we obtain the following, by setting $\gamma = \left\lceil w(S_k^{OPT})/k \right\rceil$, where $S_k^{OPT} \subseteq V(D) \setminus B$ is a minimum weight solution for D such that $|S_k^{OPT}| \leq k$.

Lemma 5.7. *Let $M > 0$ be an integer. Let (D, w, B, k) be an instance of* Directed OCT *where D is a digraph, $w : V(D) \to [M]$ is a weight function, $B \subseteq V(D)$ is a subset of blacklisted vertices and k is an integer. Suppose that this instance admits a solution of cardinality k. Then there is an algorithm that runs in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)} \cdot M$ and outputs $S \subseteq V(D) \setminus B$ such that $S \cap C \neq \emptyset$ for every directed odd cycle C in D, $|S| \leq 4k$ and $w(S) \leq 8w(S_k^{OPT})$ where $S_k^{OPT} \subseteq V(D) \setminus B$ is a solution of minimum weight in D such that $|S_k^{OPT}| \leq k$.*

5.2 Approximating General Weighted Instances

Let us now discuss approximating instances with a general weight function. The following lemma describes a general procedure that reduces a weighted instance (D, w) of a SCC $\mathcal{F}$-Transversal problem to an instance of bounded weight at a small cost to the approximation factor. Here we assume that the weights are real numbers, and an arithmetic operation can be done in $\mathcal{O}(1)$ time. The following lemma allows us to reduce from general weighted instances to instances of bounded weight.

Lemma 5.8. *Let $\mathcal{F}$ be a family of digraphs, and let (D, w, k) be an instance of a* SCC $\mathcal{F}$-Transversal *problem where $w : V(D) \to \mathbb{R}^+$ is a weight function on the vertices.*

Let M_H be an integer and let $\alpha, \beta > 1$ be constants. Let $\mathsf{Algo}_{\alpha,\beta}$ be an algorithm that given an instance (H, w_H) of the problem along with a set $B \subseteq V(H)$ of blacklisted vertices, where $w_H(v) \in [M_H]$ for all $v \in V(H)$, runs in time $f(M_H, k)g(n)$ and outputs a solution $X \subseteq V(H) \setminus B$ such that $|X| \leq \alpha |X_k^{OPT}|$ and $w_H(X) \leq \beta \cdot w_H(X_k^{OPT})$, where $X_k^{OPT} \subseteq V(H) \setminus B$ is a minimum weight solution for (H, w_H) and $|X_k^{OPT}| \leq k$.

Suppose that the instance (D, w, k) admits a solution of cardinality k. Then, for every constant $\epsilon > 0$, in time $f(k/\epsilon, k)g(n) \cdot n$ we can compute a solution S for (D, w) such that $|S| \leq \alpha |S_k^{OPT}|$ and $w(S) \leq \beta(1 + \epsilon) w(S_k^{OPT})$, where S_k^{OPT} is a minimum weight solution such that $|S_k^{OPT}| \leq k$.

Proof. Let S_k^{OPT} be a solution for (D, w) minimizing $w(S_k^{OPT})$ such that $|S_k^{OPT}| \leq k$. Let $M = \max_{v \in S_k^{OPT}} w(v)$. Note that, while M is unknown to us, it is the weight of one of the vertices in $V(D)$ and hence there are at most n possible values for it. We will iterate over each choice of M.

Let us fix a choice M and suppose that it is correct, i.e. $M = \max_{v \in S_k^{OPT}} w(v)$. Then we compute an approximate solution corresponding to M as follows. First, let us define a new weight function w' on $V(D)$ as follows.

$$w'(v) = \begin{cases} \lceil w(v) \cdot k/M\epsilon \rceil & \text{if } \lceil w(v) \cdot k/M\epsilon \rceil \leq \lceil k/\epsilon \rceil \\ \lceil k/\epsilon \rceil + 1 & \text{otherwise} \end{cases}$$

Let us consider the instance (D, w') with $B = \{v \in V(H) \mid w'(v) = \lceil k/\epsilon \rceil + 1\}$ as the set blacklisted vertices. Let $X_k^{OPT} \subseteq V(D) \setminus B$ be a minimum weight solution such that $|X_k^{OPT}| \leq k$. Note that, as $S_k^{OPT} \cap B = \emptyset$ by the definition of M, S_k^{OPT} is also a solution for (D, w') of cardinality k, and hence $w'(S_k^{OPT}) \geq w'(X_k^{OPT})$. Let us apply the algorithm $\mathsf{Algo}_{\alpha,\beta}$ to (D, w') with B as the subset of blacklisted vertices. It returns a solution S such that $|S| \leq \alpha k$, $S \cap B \leq \emptyset$ and $w'(S) \leq \beta w'(X_k^{OPT})$. Then S is also a solution for (D, w) and we can bound its weight as follows.

$$\begin{aligned} w(S) &\leq M\epsilon/k \cdot w'(S) \\ &\leq M\epsilon/k \cdot \beta w'(X_k^{OPT}) \\ &\leq M\epsilon/k \cdot \beta w'(S_k^{OPT}) \\ &= M\epsilon/k \cdot \beta \sum_{v \in S_k^{OPT}} \lceil w(v) \cdot k/M\epsilon \rceil \\ &\leq \beta M\epsilon + \beta \sum_{v \in S_k^{OPT}} w(v) \\ &\leq \beta(1+\epsilon) w(S_k^{OPT}) \end{aligned}$$

Next, observe that, in (D, w'), $w'(v) \in [\lceil k/\epsilon \rceil + 1]$ for every vertex v. Hence, assuming that our choice of M was correct, the algorithm $\mathsf{Algo}_{\alpha,\beta}$ runs in time $f(k/\epsilon, k)g(n)$ and returns a solution S such that $w(S) \leq \beta(1+\epsilon)w(S_k^{OPT})$ and $|S| \leq \alpha k$.

To compute the approximate solution for (D, w), we proceed as follows. For each vertex $v \in V(D)$ we consider the case where v is the maximum weight vertex in some minimum weight solution of cardinality k for (D, w). Let $M_v = w(v)$, and corresponding to it we have the weight function w_v and the subset of blacklisted vertices Z_v, as defined above. Then by applying the algorithm $\mathsf{Algo}_{\alpha,\beta}$ to (D, w_v) and Z_v, we obtain a solution $S_v \subseteq V(D) \setminus Z_v$ in time $f(k/\epsilon, k)g(n)$. Therefore in time $f(k/\epsilon, k)g(n) \cdot n$ we obtain a collection $\{S_v \mid v \in V(D)\}$ of solutions for (D, w). Let $S \in \{S_v \mid v \in V(D)\}$ be one that minimizes $w(S)$. It is clear that S is a solution for (D, w) and $w(S) \leq \beta(1+\epsilon)w(S_k^{OPT})$, and we output S as the required approximate solution. This completes the proof of this lemma. □

Using the above lemma, for every $\epsilon > 0$ we obtain an $(4, 8(1+\epsilon))$ FPT-approximation algorithms for Subset-DFVS that runs in $\frac{1}{\epsilon} \cdot 2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time.

Theorem 5.1. *Let (D, T, w, k) be an instance of* Subset Directed Feedback Vertex Set *where D is a digraph, $w : V(D) \to \mathbb{R}^+$ is a weight function, $T \subseteq A(D)$ is the subset of terminal arcs, and k is an integer. Suppose that this instance admits a solution of cardinality k. Then for every constant $\epsilon > 0$, in time $\frac{1}{\epsilon} \cdot 2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$, we can obtain a solution S to this instance such that $w(S) \leq 8(1+\epsilon)w(S_k^{OPT})$, and $|S| \leq 4k$ where S_k^{OPT} is a minimum weight solution such that $|S_k^{OPT}| \leq k$.*

Proof. We apply Lemma 5.8 to (D, w, k), along with the algorithm of Lemma 5.3. Note that the algorithm of Lemma 5.3 is invoked on at most n instances, where the maximum weight of vertex is upper-bounded by $M = \lceil k/\epsilon \rceil + 1$. Hence, this algorithm runs in time $\frac{1}{\epsilon} \cdot 2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ on such instances. And assuming that D admits a solution of cardinality k, one of these instances leads to an 8-approximate solution of cardinality at most $4k$. Therefore, Lemma 5.8 outputs a $8(1+\epsilon)$-approximation solution for (D, w, k). The correctness follows from Lemma 5.3 and Lemma 5.8, and the overall running time is $\frac{1}{\epsilon} \cdot 2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. □

A similar approach, using Lemma 5.5 and Lemma 5.7, respectively, leads to the following theorems.

Theorem 5.2. *Let (D, T, w, k) be an instance of* Bidirected Multicut *where D is a digraph, $w : V(D) \to \mathbb{R}^+$ is a weight function, $T \subseteq V \times V$ is the collection of terminal pairs and k is an integer. Suppose that this instance admits a solution of cardinality k. Then for every constant $\epsilon > 0$, in time $\frac{1}{\epsilon} \cdot 2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$, we can obtain a solution S to this instance such that $w(S) \leq 8(1+\epsilon) w(S_k^{OPT})$ and $|S| \leq 4k$, where S_k^{OPT} is a minimum weight solution such that $|S_k^{OPT}| \leq k$.*

Theorem 5.3. *Let (D, w, k) be an instance of* Directed Odd Cycle Transversal *where D is a digraph, $w : V(D) \to \mathbb{R}^+$ is a weight function, and k is an integer. Suppose that this instance admits a solution of cardinality k. Then for every constant $\epsilon > 0$, in time $\frac{1}{\epsilon} \cdot 2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$, we can obtain a solution S to this instance such that $w(S) \leq 8(1+\epsilon) w(S_k^{OPT})$ and $|S| \leq 4k$, where S_k^{OPT} is a minimum weight solution such that $|S_k^{OPT}| \leq k$.*

5.3 An Improved FPT-approximation for DFVS

For certain problems we can obtain a $(1, (1+\epsilon))$ FPT-approximation at the cost of higher running time. In this section we design such an algorithm for DFVS. We require the following algorithm for a variant called MultiBudgeted DFVS [KLM+19b].

Proposition 5.1 ([KLM+19b]). [11] *Let D be a digraph, and let $V(D) = V_0 \uplus V_1 \uplus V_2 \ldots \uplus V_\ell$ be a partition of $V(D)$ into $\ell + 1$ classes. Let $k_1, k_2, \ldots, k_\ell$ be positive integers, and let $k = \sum_{i=1}^{\ell} k_i$. Then, in time $2^{k^3 \log k}(|V(D)| + |A(D)|)$ we can obtain a feedback vertex set S of D such that $S \cap V_0 = \emptyset$ and $|S \cap V_i| \leq k_i$ for every $i \in [\ell]$, if one exists.*

From Proposition 5.1 we obtain the following lemma.

Lemma 5.9. *Let $M > 0$ be an integer. Let (D, w, B, k) be an instance of* Directed Feedback Vertex Set *where D is a digraph, $w : V(D) \to [M]$ is a weight function, B is a subset of blacklisted vertices and k is an integer. Suppose that this instance admits a solution of cardinality k. Then in time $\binom{M+k}{k} \cdot 2^{k^3 \log k} n^{\mathcal{O}(1)}$, we can obtain a solution $S \subseteq V(D) \setminus B$ to this instance of minimum weight and cardinality at most k.*

Proof. For $i \in [M]$, let $V_i = \{v \in V(D) \setminus B \mid w(v) = i\}$. Consider a minimum weight solution $S' \subseteq V(D) \setminus B$ to this instance such that $|S'| \leq k$. For $i \in [M]$ let $k'_i = |S' \cap V_i|$, and consider the tuple $(k'_1, k'_2, \ldots, k'_M)$. Observe that, at most k of these numbers are positive, and $w(S') = \sum_{i=1}^{M} k_i i$. There are at most $\binom{k+M}{k}$ possibilities for this tuple. We iterate over each choice, and apply Proposition 5.1 as follows.

Given $k_1, k_2, \ldots, k_\ell$, observe that at most k of these numbers are non-zero, and let $(i_1, i_2, \ldots, i_\ell)$ be the indices such that $k_{i_j} \geq 1$ for every $j \in [\ell]$. Here $\ell \leq k$. Let $V_0 = B \cup (\bigcup_{p \in [M],\ k_p = 0} V_p)$. Observe that $V_0 \uplus V_{i_1} \uplus V_{i_2} \ldots \uplus V_{i_\ell}$ is a partition of $V(D)$ and $\sum_{j=1}^{\ell} k_{i_j} = k$. We apply Proposition 5.1 to the instance $(D, (V_0 \uplus V_{i_1} \ldots \uplus V_{i_\ell}), (k_{i_1}, \ldots, k_{i_\ell}))$ and in time $2^{k^2 \log k} n^{\mathcal{O}(1)}$ obtain a solution S such that $S \cap V_0 = \emptyset$ and $|S \cap V_{i_j} \leq k_{i_j}$, assuming that one such solution exists.

[11]The result is actually stated for the Directed Feedback Arc Set problem, but there is an easy transformation from DFVS to DFAS that is approximation preserving: split each vertex v into two vertices v_{in} and v_{out}. Further, we introduce a class of vertices, V_0, that must be excluded from the solution. This can be enforced by introducing a directed cycle on 3 new vertices, adding them to V_0, setting $k_0 = 1$ and then applying the algorithm.

We iterate over all choices for the tuple $(k_1, k_2, \ldots, k_M)$, and for each we obtain a solution S of cardinality at most k and weight at most $\sum_{i=1}^{M} k_i i$, if one such solution exists. Amongst them, we output the one of minimum weight. The correctness of this algorithm is clear from the above discussion, and observe that the running time of this algorithm is upper-bounded by $\binom{M+k}{k} \cdot 2^{k^3 \log k} n^{\mathcal{O}(1)}$. $\square$

From Lemma 5.9 and Lemma 5.8 we obtain the following theorem.

Theorem 5.4. *Let (D, w, k) be an instance of* Directed Feedback Vertex Set *where D is a digraph, $w : V(D) \to \mathbb{R}^+$ is a weight function, and k is an integer. Suppose that this instance admits a solution of cardinality k. Then for every constant $\epsilon > 0$, in time $k^{k/\epsilon} \cdot 2^{k^3 \log k} n^{\mathcal{O}(1)}$, we can obtain a solution S to this instance such that $w(S) \leq (1+\epsilon) w(S_k^{OPT})$, and $|S| \leq k$ where S_k^{OPT} is a minimum weight solution such that $|S_k^{OPT}| \leq k$.*

Proof. We only need to observe that Lemma 5.8 invokes Lemma 5.9 on at most n instances where the maximum weight is upper-bounded by $\lceil k/\epsilon \rceil + 1$. Then one of these instances gives us a solution S of cardinality at most k, and weight $(1+\epsilon) w(S_k^{OPT})$ where S_k^{OPT} is the minimum weight solution of cardinality at most k to this instance. $\square$

6 Conclusion

The area of FPT-approximation has been booming in the last decade, enjoying a flurry of results. Notably, almost all of these results are for W[1]-hard problems. However, there are fundamental problems within the class FPT itself which the field of FPT-approximation has so far largely overlooked. In this paper, we took a systematic approach towards this study and designed FPT-approximation algorithms for problems that are in FPT. That is, we designed FPT-approximation algorithms for problems that are FPT, with running times that are significantly faster than the corresponding best known FPT-algorithm, and while achieving approximation ratios that are significantly better than what is possible in polynomial time. We addressed several fundamental problems such as Directed Feedback Vertex Set, Weighted Directed Feedback Vertex Set, Directed Odd Cycle Transversal, Undirected Multicut, Weighted Undirected Multicut, parameterized by the solution size. We also considered graph problems parameterized by the treewidth of the input graph and considered problems such as Vertex Cover, Component Order Connectivity, and $\mathcal{F}$-Packing for any family $\mathcal{F}$ of bounded sized graphs. Finally, we presented general reductions of problems parameterized by treewidth to their versions parameterized by solution size, as well as for weighted problems to their unweighted counterparts. We conclude the paper with several open problems. For this, let us fix a constant $\epsilon > 0$.

1. Do Directed Feedback Vertex Set, and Undirected Multicut, parameterized by the solution size, admit a $(1+\epsilon)$ approximation algorithm running in time $g(\epsilon)^k n^{\mathcal{O}(1)}$?
2. Does Planar Vertex Deletion, parameterized by the solution size, admit a constant-factor approximation algorithm running in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$?
3. Does Chordal Vertex Deletion, parameterized by the solution size, admit a constant-factor approximation algorithm running in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$?
4. Does Directed Feedback Vertex Set, parameterized by the treewidth of the input graph (w), admit a $(1+\epsilon)$ approximation algorithm running in time $g(\epsilon)^w n^{\mathcal{O}(1)}$?
5. Does Planar Vertex Deletion, parameterized by the treewidth of the input graph (w), admit a $(1+\epsilon)$ approximation algorithm running in time $g(\epsilon)^w n^{\mathcal{O}(1)}$? Here, even a constant-factor approximation algorithm running in time $2^{\mathcal{O}(w)} n^{\mathcal{O}(1)}$ would be interesting.
6. Does Feedback Vertex Set, parameterized by the treewidth of the input graph (w), admit a $(1+\epsilon)$ approximation algorithm running in time $c^w n^{\mathcal{O}(1)}$ where c is a fixed constant smaller than 3?

References

[Ami10] Eyal Amir. Approximation algorithms for treewidth. *Algorithmica*, 56(4):448–479, 2010. 2

[BBE$^+$19] Arnab Bhattacharyya, Édouard Bonnet, László Egri, Suprovat Ghoshal, Karthik C. S., Bingkai Lin, Pasin Manurangsi, and Dániel Marx. Parameterized intractability of even set and shortest vector problem. *CoRR*, abs/1909.01986, 2019. 1

[BDD$^+$16] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. 2

[BDT18] Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. *SIAM J. Comput.*, 47(1):166–207, 2018. 2, 20

[BF11] Ljiljana Brankovic and Henning Fernau. Parameterized approximation algorithms for hitting set. In Roberto Solis-Oba and Giuseppe Persiano, editors, *Approximation and Online Algorithms - 9th International Workshop, WAOA 2011, Saarbrücken, Germany, September 8-9, 2011, Revised Selected Papers*, volume 7164 of *Lecture Notes in Computer Science*, pages 63–76. Springer, 2011. 2

[BF13] Ljiljana Brankovic and Henning Fernau. A novel parameterised approximation algorithm for minimum vertex cover. *Theor. Comput. Sci.*, 511:85–108, 2013. 2

[BKN$^+$18] Marthe Bonamy, Lukasz Kowalik, Jesper Nederlof, Michal Pilipczuk, Arkadiusz Socala, and Marcin Wrochna. On directed feedback vertex set parameterized by treewidth. In Andreas Brandstädt, Ekkehard Köhler, and Klaus Meer, editors, *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, volume 11159 of *Lecture Notes in Computer Science*, pages 65–78. Springer, 2018. 3, 38

[BLM18] Rémy Belmonte, Michael Lampis, and Valia Mitsou. Parameterized (approximate) defective coloring. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 10:1–10:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 1

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. 2

[CCHM15] Rajesh Hemant Chitnis, Marek Cygan, Mohammad Taghi Hajiaghayi, and Dániel Marx. Directed subset feedback vertex set is fixed-parameter tractable. *ACM Trans. Algorithms*, 11(4):28:1–28:28, 2015. 13

[CCK$^+$17] Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-eth to fpt-inapproximability: Clique, dominating set, and more. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 743–754. IEEE Computer Society, 2017. 1

[CFK$^+$15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. 1, 4, 7, 12, 15, 17, 27, 28, 36

[CHK13] Rajesh Hemant Chitnis, MohammadTaghi Hajiaghayi, and Guy Kortsarz. Fixed-parameter and approximation algorithms: A new look. In Gregory Z. Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 110–122. Springer, 2013. 1

[CKJ01] Jianer Chen, Iyad A Kanj, and Weijia Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001. 29

[CKK+05] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, pages 144–153. IEEE Computer Society, 2005. 3

[CL19] Yijia Chen and Bingkai Lin. The constant inapproximability of the parameterized dominating set problem. *SIAM J. Comput.*, 48(2):513–533, 2019. 1

[CLL+08] Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), 2008. 2, 3

[CM16] Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016. 3

[DDvtH16] Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. 36

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, New York, 1999. 1

[DFK+18] Pavel Dvorák, Andreas Emil Feldmann, Dusan Knop, Tomás Masarík, Tomas Toufar, and Pavel Veselý. Parameterized approximation schemes for steiner trees with small number of steiner vertices. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 26:1–26:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 1

[DHK05] Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Ken-ichi Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 637–646. IEEE Computer Society, 2005. 1

[ENSS98] Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998. 3

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Springer, Berlin, 2006. 1

[FKLM20] Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020. 1

[FKRS18] Michael R. Fellows, Ariel Kulik, Frances A. Rosamond, and Hadas Shachnai. Parameterized approximation via fidelity preserving transformations. *J. Comput. Syst. Sci.*, 93:30–40, 2018. 2

[FLL$^+$19] Fedor V. Fomin, Tien-Nam Le, Daniel Lokshtanov, Saket Saurabh, Stéphan Thomassé, and Meirav Zehavi. Subquadratic kernels for implicit 3-hitting set and 3-set packing problems. *ACM Trans. Algorithms*, 15(1):13:1–13:44, 2019. 36

[FLSZ19] Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019. 1, 27

[FM06] Uriel Feige and Mohammad Mahdian. Finding small balanced separators. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 375–384. ACM, 2006. 1

[GKW19] Fabrizio Grandoni, Stefan Kratsch, and Andreas Wiese. Parameterized approximation schemes for independent set of rectangles and geometric knapsack. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 1

[GL16] Venkatesan Guruswami and Euiwoong Lee. Simple proof of hardness of feedback vertex set. *Theory Comput.*, 12(1):1–11, 2016. 3, 7

[GLL18a] Anupam Gupta, Euiwoong Lee, and Jason Li. Faster exact and approximate algorithms for k-cut. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 113–123. IEEE Computer Society, 2018. 1

[GLL18b] Anupam Gupta, Euiwoong Lee, and Jason Li. An FPT algorithm beating 2-approximation for k-cut. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2821–2837. SIAM, 2018. 1

[HKP20] Eva-Maria C. Hols, Stefan Kratsch, and Astrid Pieterse. Approximate turing kernelization for problems parameterized by treewidth. *CoRR (To Appear in ESA)*, abs/2004.12683, 2020. 36

[JLS14] Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1802–1811. SIAM, 2014. 3

[KL16] Mithilesh Kumar and Daniel Lokshtanov. A 2lk kernel for l-component order connectivity. In *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, pages 20:1–20:14, 2016. 29

[KL20] Ken-ichi Kawarabayashi and Bingkai Lin. A nearly 5/3-approximation FPT algorithm for min-k-cut. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 990–999. SIAM, 2020. 1

[KLM19a] Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. *J. ACM*, 66(5):33:1–33:38, 2019. 1

[KLM$^+$19b] Stefan Kratsch, Shaohua Li, Dániel Marx, Marcin Pilipczuk, and Magnus Wahlström. Multi-budgeted directed cuts. *Algorithmica*, pages 1–21, 2019. 8, 44

[Kor16] Guy Kortsarz. Fixed-parameter approximability and hardness. In *Encyclopedia of Algorithms*, pages 756–761. 2016. 1

[KPPW15] Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Magnus Wahlström. Fixed-parameter tractability of multicut in directed acyclic graphs. *SIAM J. Discret. Math.*, 29(1):122–144, 2015. 3

[KR08] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. 3

[KS19] Ariel Kulik and Hadas Shachnai. Analysis of two-variable recurrence relations with application to parameterized approximations. *CoRR*, abs/1911.02653, 2019. 2

[KW20] Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. 5, 20

[Lag96] Jens Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *J. Algorithms*, 20(1):20–44, 1996. 2

[Lam14] Michael Lampis. Parameterized approximation schemes using graph widths. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 775–786. Springer, 2014. 1

[Lee19] Euiwoong Lee. Partitioning a graph into small pieces with applications to path transversal. *Math. Program.*, 177(1-2):1–19, 2019. 1

[Lin18] Bingkai Lin. The parameterized complexity of the k-biclique problem. *J. ACM*, 65(5):34:1–34:23, 2018. 1

[Lin19] Bingkai Lin. A simple gap-producing reduction for the parameterized set cover problem. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 81:1–81:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 1

[LMS18a] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. 3, 36

[LMS18b] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. *SIAM J. Comput.*, 47(3):675–702, 2018. 3

[LPRS17] Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 224–237, 2017. 11

[LR99] Frank Thomson Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999. 3

[LRS18] Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. When recursion is better than iteration: A linear-time algorithm for acyclicity with few error vertices. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1916–1933. SIAM, 2018. 2

[LRSZ20] Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2181–2200. SIAM, 2020. 4, 23, 24

[LSS20] Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for min k-cut. *CoRR, to appear in FOCS 2020*, abs/2005.00134, 2020. 1

[Man19] Pasin Manurangsi. A note on max k-vertex cover: Faster fpt-as, smaller approximate kernel and improved approximation. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, volume 69 of *OASICS*, pages 15:1–15:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 1

[Mar04] Dániel Marx. Minimum sum multicoloring on the edges of planar graphs and partial k-trees. In Giuseppe Persiano and Roberto Solis-Oba, editors, *Approximation and Online Algorithms, Second International Workshop, WAOA 2004, Bergen, Norway, September 14-16, 2004, Revised Selected Papers*, volume 3351 of *Lecture Notes in Computer Science*, pages 9–22. Springer, 2004. 1

[Mar08] Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. 1

[MR09] Dániel Marx and Igor Razgon. Constant ratio fixed-parameter approximation of the edge multicut problem. *Inf. Process. Lett.*, 109(20):1161–1166, 2009. 2

[MR14] Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM J. Comput.*, 43(2):355–388, 2014. 2, 3, 20

[Nie06] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006. 1

[NT74] George L Nemhauser and Leslie Earl Trotter. Properties of vertex packing and independence system polyhedra. *Mathematical programming*, 6(1):48–61, 1974. 29

[PvLW17] Michal Pilipczuk, Erik Jan van Leeuwen, and Andreas Wiese. Approximation and parameterized algorithms for geometric independent set with shrinking. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, volume 83 of *LIPIcs*, pages 42:1–42:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. 1

[Ree92] Bruce A. Reed. Finding approximate separators and computing tree width quickly. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 221–228. ACM, 1992. 2

[RS95] Neil Robertson and Paul D. Seymour. Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. 2

[SF17] Piotr Skowron and Piotr Faliszewski. Chamberlin-courant rule with approval ballots: Approximating the maxcover problem with bounded frequencies in FPT time. *J. Artif. Intell. Res.*, 60:687–716, 2017. 1

[Vaz01] Vijay V. Vazirani. *Approximation algorithms.* Springer, 2001. 1

[vRBR09] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, pages 566–577, 2009. 36

[vRBvL+18] Johan M. M. van Rooij, Hans L. Bodlaender, Erik Jan van Leeuwen, Peter Rossmanith, and Martin Vatshelle. Fast dynamic programming on graph decompositions. *CoRR*, abs/1806.01667, 2018. 36

[Wie17] Andreas Wiese. A (1+epsilon)-approximation for unsplittable flow on a path in fixed-parameter running time. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 67:1–67:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. 1

[Wlo20] Michal Wlodarczyk. Parameterized inapproximability for steiner orientation by gap amplification. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 104:1–104:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. 1

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, 2011. 1

[Xia17] Mingyu Xiao. On a generalization of nemhauser and trotter's local optimization theorem. *J. Comput. Syst. Sci.*, 84:97–106, 2017. 29

7 Problem Definitions

We now define all the problems mentioned in the paper.

Directed Feedback Vertex Set
Input: A directed graph G.
Question: Find a minimum size subset $S \subseteq V(G)$ such that $G - S$ is a directed acyclic graph.

Weighted Directed Feedback Vertex Set
Input: A directed graph G, a weight function $w : V(G) \to \mathbb{Q}^+$ and an integer k.
Question: Find a minimum weight subset $S \subseteq V(G)$ of size at most k (if it exists) such that $G - S$ is a directed acyclic graph.

Directed Feedback Arc Set
Input: A directed graph G.
Question: Find a minimum size subset $S \subseteq A(G)$ (of arcs) such that $G - S$ is a directed acyclic graph.

Directed Subset Feedback Vertex Set
Input: A directed graph G, and a subset $T \subseteq V(G)$.
Question: Find a minimum size subset $S \subseteq V(G)$ such that $G - S$ does not have any directed cycle that contains at least one vertex from T.

Weighted Directed Subset Feedback Vertex Set
Input: A directed graph G, a subset $T \subseteq V(G)$, a weight function $w : V(G) \to \mathbb{Q}^+$ and an integer k.
Question: Find a minimum weight subset $S \subseteq V(G)$ such that $G - S$ does not have any directed cycle that contains at least one vertex from T.

Directed Odd Cycle Transversal
Input: A directed graph G.
Question: Find a minimum size subset $S \subseteq V(G)$ such that $G - S$ does not contain any directed odd cycle.

Weighted Directed Odd Cycle Transversal
Input: A directed graph G, a weight function $w : V(G) \to \mathbb{Q}^+$ and an integer k.
Question: Find a minimum weight subset $S \subseteq V(G)$ of size at most k (if it exists) such that $G - S$ does not contain any directed odd cycle.

Multicut
Input: A graph G, and a set of pairs $(s_i, t_i)_{i=1}^{\ell}$.
Question: Find a minimum size subset $S \subseteq V(G)$ such that for every $i \in [\ell]$, vertices s_i and t_i lie in different connected components of $G - S$.

Weighted Multicut
Input: A graph G, a weight function $w : V(G) \to \mathbb{Q}^+$ and an integer k.
Question: Find a minimum weight subset $S \subseteq V(G)$ of size at most k (if it exists) such that for every $i \in [\ell]$, vertices s_i and t_i lie in different connected components of $G - S$.

Edge Multicut
Input: A graph G, and a set of pairs $(s_i, t_i)_{i=1}^{\ell}$.
Question: Find a minimum weight subset $S \subseteq E(G)$ such that for every $i \in [\ell]$, vertices s_i and t_i lie in different connected components of $G - S$.

Vertex Cover
Input: A graph G.
Question: Find a minimum size subset $S \subseteq V(G)$ such that $G - S$ is edgeless.

Component Order Connectivity
Input: A graph G and an integer ℓ.
Question: Find a minimum size subset $S \subseteq V(G)$ such that every connected component of $G - S$ is of size at most ℓ.

Bounded-Degree Vertex Deletion
Input: A graph G and an integer d.
Question: Find a minimum size subset $S \subseteq V(G)$ such that every vertex in $G - S$ has degree at most d.

Triangle Packing
Input: A graph G.
Question: Find a maximum size collection of pairwise vertex-disjoint triangles in G.

Let $\mathcal{F}$ be a family of graphs.

$\mathcal{F}$-Packing
Input: A graph G.
Question: Find a maximum size collection of pairwise vertex-disjoint subgraphs of G, each isomorphic to an element of $\mathcal{F}$.

$\mathcal{F}$-Vertex Deletion
Input: A graph G.
Question: Find a a minimum size subset $S \subseteq V(G)$ such that there do not exist a subgraph of $G - S$ and an element of $\mathcal{F}$ that are isomorphic.

Contents