FPT-approximation for FPT Problems

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

Abstract

Over the past decade, many results have 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 have seen no progress over the course of 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. 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 Set.
- Additionally, we present a combinatorial lemma that yields a partition of the vertex set of a graph to roughly equal sized sets so that the removal of each set reduces its treewidth substantially, which may be of independent interest. For several graph problems, use this lemma to design $c^w n^{\mathcal{O}(1)}$-time $(1+\epsilon)$-approximation algorithms that are faster than known SETH lower bounds, where w is the treewidth of the input graph. Examples of such problems include 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 bi-criteria approximation algorithms for all problems mentioned in the first bullet.

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

*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

Lokshtanov and Zehavi acknowledge support from United States-Israel Binational Science Foundation (BSF) grant no. 2018302. Lokshtanov is also supported by NSF award CCF-2008838. Zehavi is also supported by Israel Science Foundation (ISF) grant no. 1176/18. Saurabh is supported by funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416) and also acknowledges the support of Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.

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 by 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 $f(k) \cdot |I|^{\mathcal{O}(1)}$, where $|I|$ is the size of the input and f is an arbitrary computable function. 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 [17, 19, 25, 58, 64, 66, 26].

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 correctly concludes 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 in polynomial time.

Some of the notable problems that have been shown to admit FPT-approximation algorithms include Vertex Minimum Bisection [22], k-Path Deletion [43], Max k-Vertex Cover in d-uniform hypergraphs [62, 53], k-Way Cut [28, 29, 34, 52] and Steiner Tree parameterized by the number of non-terminals [20]. These are just representative examples [16, 18, 20, 2, 22, 27, 28, 29, 34, 36, 42, 43, 53, 54, 55, 52, 59, 62, 65]. 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 [11, 33, 67, 14, 45, 46, 3]. For a comprehensive overview of the state of the art on Parameterized Approximation, we refer to the recent survey by Feldmann et al. [23], and the surveys by Kortsarz [36] and by Marx [55].

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 for problems that are FPT?

Indeed, these problems hold a lot of promise and remain hitherto unexplored from the perspective of FPT-approximation, with exceptions that are few and far between [8, 9, 24, 40, 56] (this list is not exhaustive). 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 $2^{\mathcal{O}(k^3)}n$ [4]. 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 $\mathcal{O}(k)$ [61, 41, 60, 1]. In fact, in 2013, the first constant-factor FPT-approximation with running time $2^{\mathcal{O}(k)}n$ was obtained [5]. The main idea of the paper is to replicate this success of Treewidth for other combinatorial problems.

With this goal in mind, in this paper, we expand the study of FPT-approximation and initiate a systematic study of FPT-approximation for problems that are FPT. We 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.

Since we plan to design FPT-approximations for FPT problems, two important questions that we need to address are: (a) which problems within FPT are interesting to study from this perspective; and (b) for which running times and factors of approximation should one aim?

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. Moreover, 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. It is well known that it admits an algorithm with running time $2^{\mathcal{O}(k \log k)}n^{\mathcal{O}(1)}$, and the parameter dependence has not been improved since 2007, when Chen et al. [13] resolved this longstanding open question in the area of parameterized complexity, although the dependence on the input size has been improved [50]. Similarly, a decade ago Marx and Razgon [57] and Bousquet et al. [7], 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 [57]). However, there has been no improvement over the function $f(k) = 2^{\mathcal{O}(k^3)}$ in the last ten years. These are examples of problems for which there has been little to no progress in a long time. Other examples are those problems for which we have 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. [6] 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. [47] 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 [10, 32, 38, 47, 48].

We believe that studying the aforementioned central problems in parameterized complexity in the realm of FPT-approximation will lead to the development of new algorithm design methodologies.

1.1.2 What 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 $2^{\mathcal{O}(k \log k)}n^{\mathcal{O}(1)}$ [13], and the best known polynomial-time approximation algorithm has factor $\mathcal{O}(\log n \log \log n)$ [21]. Furthermore, under Unique Games Conjecture (UGC), the problem does not admit any constant-factor approximation algorithm [30, 63, 31]. Similarly, Multicut on undirected graphs [57], 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 $\mathcal{O}(\log n)$ [44]. Assuming UGC, Chawla et al. [12] showed that the problem does not admit any constant-factor approximation algorithm. A stronger version of UGC leads to a lower bound of $\Omega(\sqrt{\log \log n})$ [12]. 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 $\delta > 0$ [47]. In the realm of polynomial-time approximation algorithms, the problem admits a simple factor-2 approximation, and assuming UGC, this approximation factor cannot be improved to $2-\eta$, for any fixed $\eta > 0$ [35]. For this problem, a desirable FPT-approximation algorithm will be an FPT-AS (FPT-approximation scheme). That is, for every $\epsilon > 0$, a factor-$(1+\epsilon)$ approximation algorithm 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 the treewidth (denoted by w) of the input graph, including 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 Transversal (DOCT) and Multicut. In particular, 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. [51] 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. 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 [17] 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 that is a solution of size at most $2k + 1$, 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[1] 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,

[1] That is, a set of vertices that hits all paths that start at a vertex in W_1 and end at a vertex in W_2.

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 [39] and in the case of DOCT, the special case can be 2-approximated by solving max-flow in a bounded number of auxiliary graphs. 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 aforementioned 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.

Before proceeding, we briefly remark on the main similarities and differences between the factor-2 approximation for DOCT in this paper and that of Lokshtanov et al. [51]. Lokshtanov et al. also begin with the iterative compression method and assume that the input is a pair (D, k) and a set $W \subseteq V(D)$ of size at most $2k + 1$ which intersects all directed odd cycles in D. However, in order to test for the existence of a solution S of size at most k disjoint from W (and compute a factor-2 approximation thereof), they guess the partition of W into the strongly connected components of $D - S$ and guess an ordering between these specific strongly connected components. Following this, they use an FPT algorithm for the Skew Multicut problem [13] to compute a set of size at most $|S|$ that hits all cycles in D that intersect at least two of these distinct strongly connected components of $D - S$. The residual problem is a disjoint union of several instances of the special case where one may assume the existence of a solution whose deletion keeps the vertices in each guessed partition of W in the same strongly connected component. However, since they have already spend a cost of $|S|$ at this point, they need an exact algorithm for the residual problem, for which they use the shadow-removal technique [15]. Their running time of $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(1)}$ is dominated by the time required to implement the shadow-removal technique, which we sidestep in this paper.

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. 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 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 [17], even though their unweighted variants have been 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.

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 to select a non-negative value k, and ask for a solution S such that $w(S) \leq k$. This, however, is not interesting 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$\neq$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 [30, 63, 31]. 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*). We interpret ϕ as a characterization of the space of all feasible solutions for an input graph G. That is, for a graph G and a vertex (or edge) subset S of G, $\phi_\Pi(G, S)$ returns *true* if S is a feasible solution and *false* otherwise. 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 $\phi_\Pi(G, X)$ 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.

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 $\frac{1}{\epsilon} \cdot 2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.*

These algorithms build upon a novel scheme to reduce the weighted problem to an unweighted instance and then invoke 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 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 we can iterate over all choices for γ. 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 refer to 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, the digraph obtained from D by making these copies, *must respect the vertex bundles*. That is either it includes all of Z_v or it is disjoint from 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 multiplicative cost of $(1 + \epsilon)$ to the approximation factor, 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. [37] 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 $k^{k/\epsilon} \cdot 2^{\mathcal{O}(k^3 \log k)} n^{\mathcal{O}(1)}$*

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. In the rest of this paper, we present the details of our single-exponential-time factor-2 approximation algorithms for the unweighted cut problems.

2 Preliminaries

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 When the (di)graph G is clear from the context, we let $n = |V(G)|$ and $m = |E(G)|$. 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 digraph D is *bidirected* if 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.

2.2 Optimization and Parameterized Complexity

DEFINITION 2.1. *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 note 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

$y \in \text{sol}(x)$ such that $\text{cost}(x,y) \leq \text{opt}(x) \cdot c(k)$. The definition for maximization problems is symmetric. When the parameterization is by solution size or value, we define FPT-approximation as follows.

DEFINITION 2.2. *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 $\text{opt}(x) \leq k$, runs in time $f(k)\cdot|x|^{\mathcal{O}(1)}$ and computes a $y \in \text{sol}(x)$ such that $\text{cost}(x,y) \leq k \cdot c(k)$. For inputs not satisfying $\text{opt}(x) \leq k$, the output can be arbitrary.*

The definition for maximization problems is symmetric. 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 [49] that will suffice for our purposes.

DEFINITION 2.3. *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 $\frac{k'}{\mathsf{opt}(I')} \leq \frac{k}{\mathsf{opt}(I)}$.*[2] *Given I, I' and a solution S' for I',* **lift** *outputs a solution S for 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 Factor-2 approximations in $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$-time

In this section, we present the first single-exponential-time factor-2 FPT-approximations for some well-studied cut problems – (SUBSET) DFVS, UNDIRECTED MULTICUT and DIRECTED ODD CYCLE TRANSVERSAL.

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 $\mathsf{conn}_D(X)$ and drop the subscript if D is clear from the context.*

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

[2] 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 [49]).

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. *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. 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. [17] *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 $x \in \mathcal{I}$, $\text{sol}(x)$ denotes the set of $\mathcal{F}$-transversals in x. Moreover, for every feasible solution y, $\text{cost}(x, y)$ 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.*

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. ∎

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 [15], 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 [15]. 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.

Note that 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.*

Due to this claim, 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. □

Given the above claim, we make use of a standard important-separator branching routine (see [17] 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. □

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'.*

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 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 $2k + 1$ 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 Alg-Disjoint-SFVS, we have the required factor-2 FPT-approximation for SUBSET DFVS.

We now proceed to describe Algorithm Alg-Disjoint-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.

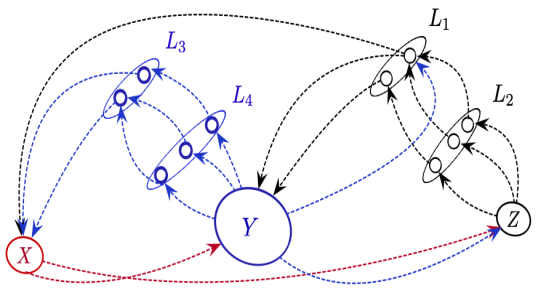

Figure 1: An illustration of the sets $X \uplus Y \uplus Z = W$ and the separators $L_1, \ldots, L_4$. The dotted arrows represent paths.

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 $\emptyset$, 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 $(D[\mathsf{rel}(Z, Q)], T, Z, i)$. 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.

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

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

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.

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: $|M_\ell| \leq |W|/3$ 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 $|W|/3 \leq |X \cup Y|, |Z| \leq 2|W|/3$. 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 $2(i_1+i_3+i_4) \leq 2|S| \leq 2k$ 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 $|X|, |Z| \leq 2|W|/3$. 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, $|X|, |Z|, |X \cup Y| \leq 2|W|/3$.

We now argue by induction on k and r that $T(k,r) \leq 2^{9k+5r}$. Indeed, the base cases are satisfied and we assume $r, k > 1$. Now,

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

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. □

As a corollary of Theorem 3.1, we get our factor-2 FPT-approximation for DFVS.

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 [57, 7] 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 [57] and Bousquet et al. [7] 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 a special case.

Note that 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 [39]. 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. [39] *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 one 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 3.3. *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 no pair in $\mathcal{P}$ is reachable from s in $D' - S'$ for some $S' \subseteq V(D)$, then S' is a $\mathcal{T}$-multicut in D.*

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 each $N \in \{Z, Y\}$, $I[Q, N, i]$ denotes the tuple $(D[\mathsf{rel}(N, Q)], \mathcal{T}, Z, i)$, i.e., 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) $S_Z \leftarrow$ Alg-Disjoint-BiMC$(I[Q, Z, i_2])$.

(ii) $S_Y \leftarrow$ Alg-Disjoint-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) $S_Z \leftarrow$ Alg-Disjoint-BiMC$(I[Q, Z, i_2])$.

(ii) $S_Y \leftarrow$ Alg-Strict-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. The correctness and running time analysis are similar to those in the proof of Theorem 3.1 and hence we omit the details. □

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. *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. [51] *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.

Note that 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. [51] 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 $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(1)}$. 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 3.4. *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.*

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 3.4 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 3.4 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 $1 \leq i \leq k$, $Q \subseteq V(D)$ and $(X, Y, Z) \in \mathcal{P}$.

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

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

Main loop: For every $(X, Y, Z) \in \mathcal{P}$ such that $|W|/3 \leq |X \cup Y|, |Z| \leq 2|W|/3$ 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 $L_1 \in \mathcal{L}^{i_1}[Z \to XY]$, $L_2 \in \mathcal{L}^{i_1}[XY \leftarrow Z]$, $L_3 \in \mathcal{L}^{i_2}[Y \to X, L_1, L_2]$, $L_4 \in \mathcal{L}^{i_2}[X \leftarrow Y, L_1, L_2]$. Set $Q = \bigcup_{q \in [4]} L_q$.

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) $S_Z \leftarrow$ 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. The correctness and running time analysis are similar to those in the proof of Theorem 3.1. □

4 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. 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

[1] E. Amir, *Approximation algorithms for treewidth*, Algorithmica, 56 (2010), pp. 448–479.

[2] R. Belmonte, M. Lampis, and V. Mitsou, *Parameterized (approximate) defective coloring*, in 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France, R. Niedermeier and B. Vallée, eds., vol. 96 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 10:1–10:15.

[3] A. Bhattacharyya, É. Bonnet, L. Egri, S. Ghoshal, Karthik C. S., B. Lin, P. Manurangsi, and D. Marx, *Parameterized intractability of even set and shortest vector problem*, CoRR, abs/1909.01986 (2019).

[4] H. L. Bodlaender, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317.

[5] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk, *A $c^k n$ 5-approximation algorithm for treewidth*, SIAM J. Comput., 45 (2016), pp. 317–378.

[6] M. Bonamy, L. Kowalik, J. Nederlof, M. Pilipczuk, A. Socala, and M. Wrochna, *On directed feedback vertex set parameterized by treewidth*, in Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings, A. Brandstädt, E. Köhler, and K. Meer, eds., vol. 11159 of Lecture Notes in Computer Science, Springer, 2018, pp. 65–78.

[7] N. Bousquet, J. Daligault, and S. Thomassé, *Multicut is FPT*, SIAM J. Comput., 47 (2018), pp. 166–207.

[8] L. Brankovic and H. Fernau, *Parameterized approximation algorithms for hitting set*, in Approximation and Online Algorithms - 9th International Workshop, WAOA 2011, Saarbrücken, Germany, September 8-9, 2011, Revised Selected Papers, R. Solis-Oba and G. Persiano, eds., vol. 7164 of Lecture Notes in Computer Science, Springer, 2011, pp. 63–76.

[9] ——, *A novel parameterised approximation algorithm for minimum vertex cover*, Theor. Comput. Sci., 511 (2013), pp. 85–108.

[10] Y. Cao and D. Marx, *Chordal editing is fixed-parameter tractable*, Algorithmica, 75 (2016), pp. 118–137.

[11] P. Chalermsook, M. Cygan, G. Kortsarz, B. Laekhanukit, P. Manurangsi, D. Nanongkai, and L. Trevisan, *From gap-eth to fpt-inapproximability: Clique, dominating set, and more*, in 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017, C. Umans, ed., IEEE Computer Society, 2017, pp. 743–754.

[12] S. Chawla, R. Krauthgamer, R. Kumar, Y. 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, IEEE Computer Society, 2005, pp. 144–153.

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

[14] Y. Chen and B. Lin, *The constant inapproximability of the parameterized dominating set problem*, SIAM J. Comput., 48 (2019), pp. 513–533.

[15] R. H. Chitnis, M. Cygan, M. T. Hajiaghayi, and D. Marx, *Directed subset feedback vertex set is fixed-parameter tractable*, ACM Trans. Algorithms, 11 (2015), pp. 28:1–28:28.

[16] R. H. Chitnis, M. Hajiaghayi, and G. Kortsarz, *Fixed-parameter and approximation algorithms: A new look*, in Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers, G. Z. Gutin and S. Szeider, eds., vol. 8246 of Lecture Notes in Computer Science, Springer, 2013, pp. 110–122.

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

[18] E. D. Demaine, M. T. Hajiaghayi, and K. 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, IEEE Computer Society, 2005, pp. 637–646.

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

[20] P. Dvorák, A. E. Feldmann, D. Knop, T. Masarík, T. Toufar, and P. Veselý, *Parameterized approximation schemes for steiner trees with small number of steiner vertices*, in 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France, R. Niedermeier and B. Vallée, eds., vol. 96 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 26:1–26:15.

[21] G. Even, J. Naor, B. Schieber, and M. Sudan, *Approximating minimum feedback sets and multicuts in directed graphs*, Algorithmica, 20 (1998), pp. 151–174.

[22] U. Feige and M. Mahdian, *Finding small balanced separators*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006, J. M. Kleinberg, ed., ACM, 2006, pp. 375–384.

[23] A. E. Feldmann, Karthik C. S., E. Lee, and P. Manurangsi, *A survey on approximation in parameterized complexity: Hardness and algorithms*, Algorithms, 13 (2020), p. 146.

[24] M. R. Fellows, A. Kulik, F. A. Rosamond, and H. Shachnai, *Parameterized approximation via fidelity preserving transformations*, J. Comput. Syst. Sci., 93 (2018), pp. 30–40.

[25] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Springer, Berlin, 2006.

[26] F. V. Fomin, D. Lokshtanov, S. Saurabh, and M. Zehavi, *Kernelization: theory of parameterized preprocessing*, Cambridge University Press, 2019.

[27] F. Grandoni, S. Kratsch, and A. Wiese, *Parameterized approximation schemes for independent set of rectangles and geometric knapsack*, in 27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany, M. A. Bender, O. Svensson, and G. Herman, eds., vol. 144 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 53:1–53:16.

[28] A. Gupta, E. Lee, and J. Li, *Faster exact and approximate algorithms for k-cut*, in 59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018, M. Thorup, ed., IEEE Computer Society, 2018, pp. 113–123.

[29] ———, *An FPT algorithm beating 2-approximation for k-cut*, in Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018, A. Czumaj, ed., SIAM, 2018, pp. 2821–2837.

[30] V. Guruswami, J. Håstad, R. Manokaran, P. Raghavendra, and M. Charikar, *Beating the random ordering is hard: Every ordering csp is approximation resistant*, SIAM Journal on Computing, 40 (2011), pp. 878–914.

[31] V. Guruswami and E. Lee, *Simple proof of hardness of feedback vertex set*, Theory Comput., 12 (2016), pp. 1–11.

[32] B. M. P. Jansen, D. Lokshtanov, and S. Saurabh, *A near-optimal planarization algorithm*, in Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, C. Chekuri, ed., SIAM, 2014, pp. 1802–1811.

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

[34] K. Kawarabayashi and B. Lin, *A nearly 5/3-approximation FPT algorithm for min-*k*-cut*, in Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020, S. Chawla, ed., SIAM, 2020, pp. 990–999.

[35] S. Khot and O. Regev, *Vertex cover might be hard to approximate to within 2-epsilon*, J. Comput. Syst. Sci., 74 (2008), pp. 335–349.

[36] G. Kortsarz, *Fixed-parameter approximability and hardness*, in Encyclopedia of Algorithms, 2016, pp. 756–761.

[37] S. Kratsch, S. Li, D. Marx, M. Pilipczuk, and M. Wahlström, *Multi-budgeted directed cuts*, Algorithmica, (2019), pp. 1–21.

[38] S. Kratsch, M. Pilipczuk, M. Pilipczuk, and M. Wahlström, *Fixed-parameter tractability of multicut in directed acyclic graphs*, SIAM J. Discret. Math., 29 (2015), pp. 122–144.
[39] S. Kratsch and M. Wahlström, *Representative sets and irrelevant vertices: New tools for kernelization*, J. ACM, 67 (2020), pp. 16:1–16:50.
[40] A. Kulik and H. Shachnai, *Analysis of two-variable recurrence relations with application to parameterized approximations*, CoRR, abs/1911.02653 (2019).
[41] J. Lagergren, *Efficient parallel algorithms for graphs of bounded tree-width*, J. Algorithms, 20 (1996), pp. 20–44.
[42] M. Lampis, *Parameterized approximation schemes using graph widths*, in Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I, J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, eds., vol. 8572 of Lecture Notes in Computer Science, Springer, 2014, pp. 775–786.
[43] E. Lee, *Partitioning a graph into small pieces with applications to path transversal*, Math. Program., 177 (2019), pp. 1–19.
[44] F. T. Leighton and S. Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, J. ACM, 46 (1999), pp. 787–832.
[45] B. Lin, *The parameterized complexity of the k-biclique problem*, J. ACM, 65 (2018), pp. 34:1–34:23.
[46] ———, *A simple gap-producing reduction for the parameterized set cover problem*, in 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece, C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, eds., vol. 132 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 81:1–81:15.
[47] D. Lokshtanov, D. Marx, and S. Saurabh, *Known algorithms on graphs of bounded treewidth are probably optimal*, ACM Trans. Algorithms, 14 (2018), pp. 13:1–13:30.
[48] ———, *Slightly superexponential parameterized problems*, SIAM J. Comput., 47 (2018), pp. 675–702.
[49] D. Lokshtanov, F. Panolan, M. S. Ramanujan, and S. 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, 2017, pp. 224–237.
[50] D. Lokshtanov, M. S. Ramanujan, and S. Saurabh, *When recursion is better than iteration: A linear-time algorithm for acyclicity with few error vertices*, in Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018, A. Czumaj, ed., SIAM, 2018, pp. 1916–1933.
[51] D. Lokshtanov, M. S. Ramanujan, S. Saurabh, and M. Zehavi, *Parameterized complexity and approximability of directed odd cycle transversal*, in Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020, S. Chawla, ed., SIAM, 2020, pp. 2181–2200.
[52] D. Lokshtanov, S. Saurabh, and V. Surianarayanan, *A parameterized approximation scheme for min k-cut*, CoRR, to appear in FOCS 2020, abs/2005.00134 (2020).
[53] P. Manurangsi, *A note on max k-vertex cover: Faster fpt-as, smaller approximate kernel and improved approximation*, in 2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA, J. T. Fineman and M. Mitzenmacher, eds., vol. 69 of OASICS, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 15:1–15:21.
[54] D. Marx, *Minimum sum multicoloring on the edges of planar graphs and partial k-trees*, in Approximation and Online Algorithms, Second International Workshop, WAOA 2004, Bergen, Norway, September 14-16, 2004, Revised Selected Papers, G. Persiano and R. Solis-Oba, eds., vol. 3351 of Lecture Notes in Computer Science, Springer, 2004, pp. 9–22.
[55] ———, *Parameterized complexity and approximation algorithms*, Comput. J., 51 (2008), pp. 60–78.
[56] D. Marx and I. Razgon, *Constant ratio fixed-parameter approximation of the edge multicut problem*, Inf. Process. Lett., 109 (2009), pp. 1161–1166.
[57] D. Marx and I. Razgon, *Fixed-parameter tractability of multicut parameterized by the size of the cutset*, SIAM J. Comput., 43 (2014), pp. 355–388.
[58] R. Niedermeier, *Invitation to Fixed-Parameter Algorithms*, Oxford University Press, 2006.
[59] M. Pilipczuk, E. J. van Leeuwen, and A. Wiese, *Approximation and parameterized algorithms for geometric independent set with shrinking*, in 42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark, K. G. Larsen, H. L. Bodlaender, and J. Raskin, eds., vol. 83 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 42:1–42:13.
[60] B. A. Reed, *Finding approximate separators and computing tree width quickly*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada, S. R. Kosaraju, M. Fellows, A. Wigderson, and J. A. Ellis, eds., ACM, 1992, pp. 221–228.
[61] N. Robertson and P. D. Seymour, *Graph minors .xiii. the disjoint paths problem*, J. Comb. Theory, Ser. B, 63 (1995), pp. 65–110.
[62] P. Skowron and P. Faliszewski, *Chamberlin-courant rule with approval ballots: Approximating the maxcover*

problem with bounded frequencies in FPT time, J. Artif. Intell. Res., 60 (2017), pp. 687–716.

[63] O. Svensson, *Hardness of vertex deletion and project scheduling*, in Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, Springer, 2012, pp. 301–312.

[64] V. V. Vazirani, *Approximation algorithms*, Springer, 2001.

[65] A. Wiese, *A (1+epsilon)-approximation for unsplittable flow on a path in fixed-parameter running time*, in 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland, I. Chatzigiannakis, P. Indyk, F. Kuhn, and A. Muscholl, eds., vol. 80 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 67:1–67:13.

[66] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*, Cambridge University Press, 2011.

[67] M. Wlodarczyk, *Parameterized inapproximability for steiner orientation by gap amplification*, in 47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference), A. Czumaj, A. Dawar, and E. Merelli, eds., vol. 168 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 104:1–104:19.