Solving Connectivity Problems Parameterized by Treewidth in Single Exponential Time

MAREK CYGAN, University of Warsaw
JESPER NEDERLOF, Utrecht University
MARCIN PILIPCZUK and MICHAŁ PILIPCZUK, University of Warsaw
JOHAN M. M. VAN ROOIJ, Utrecht University
JAKUB ONUFRY WOJTASZCZYK, Google Inc.

For the vast majority of local problems on graphs of small treewidth (where, by local we mean that a solution can be verified by checking separately the neighbourhood of each vertex), standard dynamic programming techniques give $c^{\mathbf{tw}}|V|^{O(1)}$ time algorithms, where **tw** is the treewidth of the input graph $G = (V, E)$ and c is a constant. On the other hand, for problems with a global requirement (usually connectivity) the best–known algorithms were naive dynamic programming schemes running in at least $\mathbf{tw}^{\mathbf{tw}}$ time.

We bridge this gap by introducing a technique we named Cut&Count that allows to produce $c^{\mathbf{tw}}|V|^{O(1)}$ time Monte-Carlo algorithms for most connectivity-type problems, including Hamiltonian Path, Steiner Tree, Feedback Vertex Set and Connected Dominating Set. These results have numerous consequences in various fields, like parameterized complexity, exact and approximate algorithms on planar and H-minor-free graphs and exact algorithms on graphs of bounded degree. The constant c in our algorithms is in all cases small, and in several cases we are able to show that improving those constants would cause the Strong Exponential Time Hypothesis to fail. In all these fields we are able to improve the best-known results for some problems. Also, looking from a more theoretical perspective, our results are surprising since the equivalence relation that partitions all partial solutions with respect to extendability to global solutions seems to consist of at least $\mathbf{tw}^{\mathbf{tw}}$ equivalence classes for all these problems. Our results answer an open problem raised by Lokshtanov, Marx and Saurabh [SODA'11].

In contrast to the problems aimed at *minimizing* the number of connected components that we solve using Cut&Count as mentioned above, we show that, assuming the Exponential Time Hypothesis, the aforementioned gap cannot be bridged for some problems that aim to *maximize* the number of connected components like Cycle Packing.

CCS Concepts: • **Theory of computation** → Graph algorithms analysis; **Parameterized complexity and exact algorithms**; *Algorithm design techniques*; • **Mathematics of computing** → **Paths and connectivity problems**; *Graph algorithms;*

An extended abstract of this article was presented at FOCS 2011.
Authors' addresses: M. Cygan, M. Pilipczuk, and M. Pilipczuk, Institute of Informatics, University of Warsaw, ul. Banacha 2, Warsaw 02-097, Poland; emails: {cygan, malcin, michal.pilipczuk}@mimuw.edu.pl; J. Nederlof, Department of Information and Computing Sciences, Utrecht University, Princetonplein 5, 3584, CC Utrecht, The Netherlands; email: j.nederlof@uu.nl; J. M. M. van Rooij, Department of Information and Computing Sciences, Utrecht University, Princetonplein 5, 3584, CC Utrecht, The Netherlands; email: J.M.M.vanRooij@uu.nl; J. O. Wojtaszczyk, Google Inc., Warsaw, Poland; email: onufry@google.com.

1549-6325/2022/02-ART17 $15.00
https://doi.org/10.1145/3506707

Additional Key Words and Phrases: Treewidth, connectivity problems, feedback vertex set, steiner tree, hamilton path/cycle, Isolation Lemma

ACM Reference format:
Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. 2022. Solving Connectivity Problems Parameterized by Treewidth in Single Exponential Time. *ACM Trans. Algorithms* 18, 2, Article 17 (February 2022), 31 pages.
https://doi.org/10.1145/3506707

1 INTRODUCTION AND NOTATION

The notion of *treewidth* was introduced independently by Rose in 1974 [73] (under the name of partial k-tree) and in 1984 by Robertson and Seymour [72], and in many cases proved to be a good measure of the intrinsic difficulty of various NP-hard problems on graphs, and a useful tool for attacking those problems. Many of them can be efficiently solved through dynamic programming if we assume the input graph to have bounded treewidth.

The interest in algorithms for graphs of bounded treewidth stems from their utility: such algorithms are used as sub-routines in a variety of settings. Amongst them prominent are approximation algorithms [3, 17, 32, 40] and parametrized algorithms [35, 42] for a vast number of problems on planar, bounded-genus and H-minor-free graphs, including Vertex Cover, Dominating Set, and Independent Set. There are also applications in parametrized algorithms in general graphs [64, 76] for problems like Connected Vertex Cover and Cutwidth and in exact algorithms [42, 66] such as Minimum Maximal Matching and Dominating Set.

In many cases, where the problem to be solved is "local" (loosely speaking this means that the property of the object to be found can be verified by checking separately the neighbourhood of each vertex), matching upper and lower bounds for the runtime of the optimal solution are known. For instance for the aforementioned $2^{\mathbf{tw}(G)}|V|^{O(1)}$ algorithm for Vertex Cover there is a matching lower bound—unless the Strong Exponential Time Hypothesis (see [52]) fails, there is no algorithm for Vertex Cover running faster than $(2-\varepsilon)^{\mathbf{tw}(G)}$ for any $\varepsilon > 0$ (see [61]).

On the other hand, when the problem involves some sort of a "global" constraint—e.g., connectivity—the best known algorithms usually have a runtime on the order of $2^{O(\mathbf{tw}(G)\log\mathbf{tw}(G))}|V|^{O(1)}$. In these cases, the typical dynamic programming routine has to keep track of all the ways in which the solution can traverse the corresponding separator of the tree decomposition, that is $\Omega(l^l)$ on the size l of the separator, and therefore of treewidth. This obviously implies weaker results in the applications mentioned above. This problem was observed, for instance, by Dorn, Fomin and Thilikos, [35, 36] and by Dorn et al. in [37], and the question whether the known $2^{O(\mathbf{tw}(G)\log\mathbf{tw}(G))}|V|^{O(1)}$ parametrized algorithms for Hamiltonian Path, Connected Vertex Cover and Connected Dominating Set are optimal was explicitly asked by Lokshtanov, Marx, and Saurabh [62].

The $2^{O(\mathbf{tw}(G)\log\mathbf{tw}(G))}|V|^{O(1)}$ dynamic programming routines for connectivity problems were thought to be optimal because of the following reasoning. Each node x of a tree decomposition of a graph G decomposes G into two subgraphs $G_1$and G_2, whose intersection consists of the vertices in a separator B_x (called a bag) that is associated with x. The subgraph G_1 corresponds to the subtree below x, whereas G_2 corresponds to the part of the graph that has not yet been handled. One then considers how a solution in the entire graph G can intersect the subgraph G_1 and classifies the partial solutions in G_1 based on their behaviour with respect to the separator B_x: which pairs of vertices of the separator B_x are connected by the partial solution in G_1, and which still need to be connected by the part of the solution that will be found in G_2? This leads to $2^{\Theta(\mathbf{tw}(G)\log\mathbf{tw}(G))}$

equivalence classes of partial solutions, as witnessed for example by the number of perfect matchings on $|B_x|$ vertices (and $|B_x| \leq \mathbf{tw}(G)+1$). This approach to dynamic programming was expected to be optimal as it was believed that an algorithm may have to store at least one partial solution for each connectivity pattern on the separator B_x, in order to find a global solution if one exists. From this point of view the results of this article come as a significant surprise, and follow-up research shows that it suffices to maintain a single-exponential amount of information to obtain a correct algorithm. In some sense, the classification of partial solutions based on the connectivity pattern on the separator is too refined and a courser partition can still be used to find an optimal solution.

1.1 Our Results

In this article, we introduce a technique we named "Cut&Count". Briefly stated, we first reduce the original problem to the task of counting possibly disconnected "cut solutions" modulo 2 by (i) making sure that the number of disconnected cut solutions is always even and (ii) using randomization to guarantee with high probability that the number of connected cut solutions is odd if and only if there is a solution. The reduction is performed in such a way that counting cut solutions is a local problem and can be done sufficiently fast by standard dynamic programming.

For most problems involving a global constraint our technique gives a randomized algorithm with runtime $c^{\mathbf{tw}(G)}|V|^{O(1)}$. In particular, we are able to give such algorithms for the three problems mentioned in [62], as well as for all the other sample problems mentioned in [36]. Moreover, the constant c is in all cases well defined and small. The randomization we mention comes from the usage of the Isolation Lemma [65]. This gives us Monte Carlo algorithms with a one-sided error. The formal statement of a typical result is as follows:

Theorem 1.1. *There exists a randomized algorithm, which given in a graph $G = (V, E)$, a tree decomposition of G of width t and a number k in $3^t|V|^{O(1)}$ time either states that there exists a connected vertex cover of size at most k in G, or that it could not verify this hypothesis. If there indeed exists such a cover, the algorithm will return "unable to verify" with probability at most* $1/2$.

We call an algorithm as in Theorem 1.1 an algorithm with *false negatives*. We see similar results for a number of other global problems. As the exact value of c in the $c^{\mathbf{tw}(G)}$ expression is often important and highly non-trivial to obtain, we gather the results in the second column of Table 1.

For a number of these results, we have matching lower bounds published in [28], such as the following one:

Theorem 1.2. *Unless the Strong Exponential Time Hypothesis is false, there do not exist a constant $\varepsilon > 0$ and an algorithm that given an instance $(G = (V, E), T, k)$ together with a path decomposition of the graph G of width p solves the* Steiner Tree *problem in $(3-\varepsilon)^p|V|^{O(1)}$ time.*

Since each path decomposition is also a tree decomposition a lower bound for pathwidth implies the same lower bound for treewidth. We have such matching lower bounds for several other problems presented in the third column of Table 1. We feel that the results for Connected Vertex Cover, Connected Dominating Set, Connected Feedback Vertex Set, and Connected Odd Cycle Transversal are of particular interest here and should be compared to the algorithms and lower bounds for the analogous problems without the connectivity requirement. For instance in the case of Connected Vertex Cover the results show that the increase in running time to $3^{\mathbf{tw}(G)}|V|^{O(1)}$ from the $2^{\mathbf{tw}(G)}|V|^{O(1)}$ algorithm of [67] for Vertex Cover is not an artifact of the Cut&Count technique, but rather an intrinsic characteristic of the problem. We see a similar increase of the base constant by one for the other three mentioned problems.

We have found Cut&Count to fail for two maximization problems: Cycle Packing and Max Cycle Cover. We believe this is an example of a more general phenomenon—problems that ask to

Table 1. Summary of Our Results for Treewidth and Pathwidth Parametrizations

Problem name	Algorithms param. by treewidth	Lower bounds
Steiner Tree	$3^{\mathbf{tw}(G)}$	$3^{\mathbf{pw}(G)}$
Feedback Vertex Set	$3^{\mathbf{tw}(G)}$	$3^{\mathbf{pw}(G)}$
Connected Vertex Cover	$3^{\mathbf{tw}(G)}$	$3^{\mathbf{pw}(G)}$
Connected Dominating Set	$4^{\mathbf{tw}(G)}$	$4^{\mathbf{pw}(G)}$
Connected Feedback Vertex Set	$4^{\mathbf{tw}(G)}$	$4^{\mathbf{pw}(G)}$
Connected Odd Cycle Transversal	$4^{\mathbf{tw}(G)}$	$4^{\mathbf{pw}(G)}$
Undirected/Directed Min Cycle Cover	$4^{\mathbf{tw}(G)}/6^{\mathbf{tw}(G)}$	
Undirected/Directed Longest Path (Cycle)	$4^{\mathbf{tw}(G)}/6^{\mathbf{tw}(G)}$	
Exact k-Leaf Spanning Tree	$4^{\mathbf{tw}(G)}$	$4^{\mathbf{pw}(G)}$
Exact k-Leaf Outbranching	$6^{\mathbf{tw}(G)}$	
Maximum Full Degree Spanning Tree	$4^{\mathbf{tw}(G)}$	
Graph Metric Travelling Salesman Problem	$4^{\mathbf{tw}(G)}$	
(Directed) Cycle Packing		$2^{\Omega(\mathbf{pw}(G)\log \mathbf{pw}(G))}$
(Directed) Max Cycle Cover		$2^{\Omega(\mathbf{pw}(G)\log \mathbf{pw}(G))}$
Maximally Disconnected Dominating Set		$2^{\Omega(\mathbf{pw}(G)\log \mathbf{pw}(G))}$

For the sake of presentation in each entry we skip the $|V|^{O(1)}$ multiplicative term.

maximize (instead of minimizing) the number of connected components in the solution seem more difficult to solve than the problems that ask to minimize (including problems where we demand that the solution forms a single connected component). As an evidence we have presented lower bounds for the time complexity of solutions to such problems in [28], proving that $c^{\mathbf{tw}(G)}$ solutions of these problems are unlikely:

Theorem 1.3. *Unless the Exponential Time Hypothesis is false, there does not exist a* $2^{o(p \log p)}|V|^{O(1)}$ *algorithm solving* Cycle Packing *or* Max Cycle Cover *(either in the directed and undirected setting). The parameter p denotes the width of a given path decomposition of the input graph.*

To further verify this intuition, we investigated an artificial problem (the Maximally Disconnected Dominating Set), in which we ask for a dominating set with the largest possible number of connected components, and indeed we found a similar phenomenon.

1.2 Previous Work

The Cut&Count technique has two main ingredients. The first is an algebraic approach, where we assure that objects we are not interested in are counted an even number of times, and then do the calculations in $\mathbb{Z}_2$ (or for example any other field of characteristic 2), which causes them to disappear. This line of reasoning goes back to Tutte [77], and was previously used by Björklund [10] and Björklund et al. [12].

The second is the idea of defining the connectivity requirement through cuts, which is frequently used in approximation algorithms via linear programming relaxations. In particular, cut based constraints were used in the Held and Karp relaxation for the Travelling Salesman Problem from 1970 [49, 50] and appear up to now in the best known approximation algorithms, for example in the recent algorithm for the Steiner Tree problem by Byrka et al. [20]. To the best of our knowledge the idea of defining problems through cuts was never used in the exact and parameterized settings.

A number of articles circumvent the problems stemming from the lack of single exponential algorithms parametrized by treewidth for connectivity–type problems. For instance in the case of parametrized algorithms, sphere cuts [35, 37] (for planar and bounded genus graphs) and Catalan structures [36] (for H-minor-free graphs) were used to obtain $2^{O(\sqrt{k})}|V|^{O(1)}$ algorithms for a number of problems with connectivity requirements. To the best of our knowledge, however, no attempt to attack the problem directly was published before; indeed the non-existence of $2^{o(\mathbf{tw}(G) \log \mathbf{tw}(G))}|V|^{O(1)}$ algorithms was deemed to be more likely.

For classical graph problems the base of the exponent for treewidth parametrization was improved a few times. For example, Alber et al. [1] gave a $4^{\mathbf{tw}(G)}|V|^{O(1)}$ time algorithm for Dominating Set, improving over the algorithm of Telle and Proskurowski [75]. Later, van Rooij et al. [79] observed that one could use a generalisation of fast subset convolution [11] to improve the running time of algorithms on graphs of bounded treewidth. Their results include a $3^{\mathbf{tw}(G)}|V|^{O(1)}$ algorithm for Dominating Set, matching the space bound of the naive approach; see also [78].

1.3 Consequences of the Cut&Count Technique

As already mentioned, algorithms for graphs of bounded treewidth have a number of applications in various branches of algorithmics. Thus, it is not a surprise that the results obtained by our technique give a large number of corollaries.

We would like to emphasize that the strength of the Cut&Count technique shows not only in the quality of the results obtained in various fields, which are frequently better than the previously best known ones, achieved through a plethora of techniques and approaches, but also in the ease in which new strong results can be obtained.

1.3.1 Solution Size Parametrizations. Let us recall the definition of the Feedback Vertex Set problem:

Feedback Vertex Set
Input: An undirected graph G and an integer k
Question: Is it possible to remove k vertices from G so that the remaining vertices induce a forest?

This problem is on Karp's original list of 21 NP-complete problems [56]. It has also been extensively studied from the parametrized complexity point of view. Let us recall that in the **fixed-parameter setting** (**FPT**) the problem comes with a parameter k, and we are looking for a solution with time complexity $f(k)n^{O(1)}$, where n is the input size and f is some function (usually exponential in k). Thus, we seek to move the intractability of the problem from the input size to the parameter.

There is a long sequence of FPT algorithms for Feedback Vertex Set [4, 14, 22, 31, 38, 39, 47, 55, 69, 70]. The best—so far—result in this series is the deterministic $3.83^k k|V|^2$ result of Cao, Chen and Liu [21].[1] Our technique gives an improvement of their result:

Theorem 1.4. *There exists a Monte Carlo algorithm with constant one-sided error probability that solves the* Feedback Vertex Set *problem in a graph $G = (V, E)$ in $3^k|V|^{O(1)}$ time and polynomial space.*

We give similar improvements for Connected Vertex Cover (from the $2.4882^k|V|^{O(1)}$ deterministic algorithm of [8] to our randomized $2^k|V|^{O(1)}$ algorithm) and Connected Feedback

[1]After the extended abstract of this article was published, parameterized algorithms for Feedback Vertex Set were improved to run in time $3.62^k n^{O(1)}$ deterministically by Pilipczuk and Kociumaka [58], and in $2.70^k n^{O(1)}$ time probabilistically by Li and Nederlof [60].

Table 2. Table Summarizes the Running Times of Solution Size Parametrizations in Comparison with Previous Work

Problem name	Algorithms param. by solution size	Previous best algorithms param. by solution size
Feedback Vertex Set	3^k	3.83^k [21]
Connected Vertex Cover	2^k	2.4882^k [8]
Connected Feedback Vertex Set	3^k	46.2^k [63]

In each entry, we skip the $|V|^{O(1)}$ multiplicative term. Note that all our algorithms are randomized, while the previous works are all deterministic algorithms.

Vertex Set (from the $46.2^k|V|^{O(1)}$ deterministic algorithm of [63] to our randomized $3^k|V|^{O(1)}$ algorithm). This is summarized in Table 2.

1.3.2 Parametrized Algorithms for H-minor-free Graphs. A large branch of applications of algorithms parametrized by treewidth is the bidimensionality theory, used to find subexponential algorithms for various problems in H-minor-free graphs. In this theory, we use the theorem of Demaine et al. [33], which ensures that any H-minor-free graph either has treewidth bounded by $C\sqrt{k}$, or a $2\sqrt{k} \times 2\sqrt{k}$ grid as a minor. In the latter case we are assumed to be able to answer the problem in question (for instance a $2\sqrt{k} \times 2\sqrt{k}$ grid as a minor guarantees that the graph does not have a Vertex Cover or Connected Vertex Cover smaller than k). Thus, we are left with solving the problem with the assumption of bounded treewidth. In the case of, for instance, Vertex Cover, a standard dynamic programming algorithm suffices, thus giving us a $2^{O(\sqrt{k})}$ algorithm to check whether a graph has a vertex cover no larger than k. In the case of Connected Vertex Cover, however, the standard dynamic programming routine gives a $2^{O(\sqrt{k}\log k)}$ complexity—thus, we lose a logarithmic factor in the exponent.

There were a number of attempts to deal with this problem, taking into account the structure of the graph, and using it to deduce some properties of the tree decomposition under consideration. The latest and most efficient of those approaches is due to Dorn, Fomin and Thilikos [36], and exploits the so-called Catalan structures. The approach deals with most of the problems mentioned in our article, and is probably applicable to the remaining ones. Thus, the gain here is not in improving the running times (though our approach does improve the constants hidden in the big-O notation these are rarely considered to be important in the bidimensionality theory), but rather in simplifying the proof—instead of delving into the combinatorial structure of each particular problem, we are back to a simple framework of applying the Robertson–Seymour theorem and then following up with a dynamic programming algorithm on the obtained tree decomposition.

1.3.3 Exact Algorithms for Graphs of Bounded Degree. Another application of our methods can be found in the field of solving problems with a global constraint in graphs of bounded degree. The problems that have been studied in this setting are mostly local in nature (such as Vertex Cover, see, e.g., [19]); however global problems such as the Travelling Salesman Problem and Hamiltonian Cycle have also received considerable attention [13, 41, 45, 46, 53].

Throughout the following, we let n denote the number of vertices of the given graph. The starting point is the following theorem by Fomin et al. [42]:

Theorem 1.5 ([42]). *For any $\varepsilon > 0$ there exists an integer N_ε such that for any graph G with $n > N_\varepsilon$ vertices,*

$$\mathbf{pw}(G) \leq \frac{1}{6}n_3 + \frac{1}{3}n_4 + \frac{13}{30}n_5 + n_{\geq 6} + \varepsilon n,$$

where n_i is the number of vertices of degree i in G for any $i \in \{3, \ldots, 5\}$ and $n_{\geq 6}$ is the number of vertices of degree at least 6.

This theorem is constructive, and the corresponding path decomposition (and, consequently, tree decomposition) can be found in polynomial time. Combining this theorem with our results gives randomized algorithms running faster than 2^n time for graphs of maximum degree 3, 4, and (in the case of the $3^{\mathbf{tw}(G)}$ and $4^{\mathbf{tw}(G)}$ algorithms) 5.

Furthermore, Björklund [9] suggested a simple modification of our $4^{\mathbf{tw}(G)}|V(G)|^{O(1)}$ time algorithm for Hamiltonian Cycle in the case of path decompositions of cubic graphs, leading to $3^{\mathbf{pw}(G)}$ dependency on pathwidth in that case (see Section 4.3.1 of [24] for the proof). Consequently, we get the following theorem which improves over previously best results: the deterministic $O(1.251^n)$ algorithm of Iwama and Nakashima [53] for maximum degree three, and the randomized $O(1.657^n)$ algorithm of Björklund [10] for maximum degree four. Corollary 1.6 was later improved in [27] to $O(1.16^n)$ and $O(1.51^n)$, respectively.

Corollary 1.6. *There exists a Monte Carlo algorithm with constant one-sided error probability that solves the* Hamiltonian Cycle *problem in $O(1.201^n)$ time for cubic graphs and $O(1.588^n)$ for graphs of maximum degree* 4.

1.3.4 Exact Algorithms on Planar Graphs. Recall that n denotes the number of vertices of the given graph. We begin with a consequence of the work of Fomin and Thilikos [44]:

Proposition 1.7. *For any planar graph G, $\mathbf{tw}(G) + 1 \leq \frac{3}{2}\sqrt{4.5n} \leq 3.183\sqrt{n}$. Moreover, a tree decomposition of such width can be found in polynomial time.*

As a direct consequence, we immediately obtain $O(c^{\sqrt{n}})$ algorithms, with small constants c, for solving problems with a global constraint on planar graphs. For the Hamiltonian Cycle problem on planar graphs we obtain the following result:

Corollary 1.8. *There exists a Monte Carlo algorithm with constant one-sided error probability that solves the* Hamiltonian Cycle *problem on planar graphs in $O(4^{3.183\sqrt{n}}) = O(2^{6.366\sqrt{n}})$ time.*

To the best of our knowledge, the best algorithm known before was the $O(2^{6.903\sqrt{n}})$ of Bodlaender et al. [37]. Similarly, we obtain an $O(2^{6.366\sqrt{n}})$ algorithm for Longest Cycle on planar graphs which improves the $O(2^{7.223\sqrt{n}})$ algorithm of [37]. After our results, Cut&Count and the follow up rank-based approach where used in combination with branch decompositions leading to $O(2^{5.036\sqrt{n}})$ time randomized algorithms and $O(2^{6.570\sqrt{n}})$ time deterministic algorithms [68] for these problems.

In the same way, we obtain—as in the previous subsections—well-behaved $c^{\sqrt{n}}$ algorithms for all connectivity problem mentioned in this article.

1.4 Further Developments

Since the extended abstract of this article was published [29], the study of connectivity problems parameterized by treewidth has been very active. Bodlaender et al. [15] have shown two approaches in this line of research. In the first approach, a matrix with rows and columns indexed by partial solutions is analyzed and its rank is upper bounded via a formula inspired by the Cut&Count technique. It was shown that this can be combined with a Gaussian elimination algorithm to give a deterministic $c^{\mathbf{tw}(G)}|V(G)|^{O(1)}$ time algorithm for connectivity problems, which can be seen as a derandomization of this work, yet by using a different approach. The algorithms of [15] provide worse constants c in the base of the exponential function, nevertheless, can handle arbitrary real weights in the weighted variants of problems. A second approach of [15] borrows some ideas from the proof of the Matrix Tree Theorem and allows solving counting variants of some of the problems. For example a $15^{\mathbf{tw}(G)}|V(G)|^{O(1)}$ time algorithm is presented, which counts the number of Hamiltonian cycles in a graph when given its tree decomposition.

On the other hand, Fomin et al. [43] have given an explanation of why the $c^{\mathbf{tw}(G)}$ dependency on treewidth can be reached for connectivity problems via matroid theory. They gave a single exponential time algorithm computing representative families for linear matroids. One of the immediate consequences of their work is $c^{\mathbf{tw}(G)}$ dependency on treewidth for connectivity problems using deterministic algorithms.

The gap between the best known upper and lower bounds for the Hamiltonian Cycle problem parameterized by pathwidth was closed in [27], by showing a $(2+\sqrt{2})^{\mathbf{pw}(G)}|V(G)|^{O(1)}$ time algorithm together with a $(2+\sqrt{2}-\epsilon)^{\mathbf{pw}(G)}|V(G)|^{O(1)}$ lower bound based on the Strong Exponential Time Hypothesis. The algorithmic part of [27] is based on the rank based approach from [15], tailor made for the Hamiltonian Cycle setting. Curticapean et al. [23] showed a gap between the complexity of a counting version and the regular version by showing a $(6-\epsilon)^{\mathbf{pw}(G)}|V(G)|^{O(1)}$ lower bound for counting the number of Hamiltonian cycles.

Finally, after publishing our initial results, Cut&Count was applied in the context of various different problems and graph-width parameters. For example, it was applied to r-Dominating Set parameterised by treewidth by Borradaile and Le [18]. Regarding other graph-width parameters, Cut&Count was applied to branchwidth by Pino et al. [68], to treedepth by Hegerfeld and Kratsch [48], and to cliquewidth, $\mathbb{Q}$-rankwidth, rankwidth, and MIM-width by Bergougnoux [5] (see also Bergougnoux and Kanté [6, 7]).

In Section 6, we give examples of currently open problems related to the study of connectivity problems parameterized by treewidth.

1.5 Organization of the Article

As the reader might have already noticed, there is a quite a large amount of material covered by the set of our results. To keep the volume of the article reasonable, we focus only on the algorithmic part of the results. All the other proofs can be found in the extended version of the article on arXiv.org [28] and the dissertation of the first author [24].

Section 2 is devoted to presenting the background material for our algorithms: In Section 2.2, we recall the notion of treewidth, and in Section 2.3, we introduce the Isolation Lemma. In Section 3, we present the Cut&Count technique on two examples: the Steiner Tree problem and the Directed Min Cycle Cover problem. Moreover, Section 3.3 contains a general overview of how our framework can be applied to connectivity problems. Section 4 contains the details of the dynamic programming for Min Cycle Cover, Feedback Vertex Set, and Connected Vertex Cover exhibiting some non-trivial tricks required to complete the proofs. Finally, in Section 5, we consider the solution size parametrizations and present $3^k|V|^{O(1)}$ and $2^k|V|^{O(1)}$ time algorithms for Feedback Vertex Set and Connected Vertex Cover, respectively.

2 PRELIMINARIES AND NOTATION

2.1 Notation

Let $G=(V,E)$ be a graph (possibly directed). By $V(G)$ and $E(G)$, we denote the sets of vertices and edges of G, respectively. For a vertex set $X \subset V(G)$ by $G[X]$, we denote the subgraph induced by X. For an edge set $X \subset E$, we take $V(X)$ to denote the set of the endpoints of the edges of X. Note that in the graph $G[X]$ for an edge set X the set of vertices remains the same as in the graph G.

For an undirected graph $G = (V, E)$, the *open neighbourhood* of a vertex v, denoted $N(v)$, stands for $\{u \in V : uv \in E\}$, while the *closed* neighbourhood $N[v]$ is $N(v) \cup \{v\}$. Similarly, for a set $X \subset V(G)$ by $N[X]$, we mean $\bigcup_{v \in X} N[v]$ and by $N(X)$, we mean $N[X] \setminus X$.

By a *cut* of a set $X \subset V$, we mean a pair (X_1, X_2), with $X_1 \cap X_2 = \emptyset$, $X_1 \cup X_2 = X$ (note that one of the sides of a cut might be empty). We refer to X_1 and X_2 as to the (left and right) *sides* of the cut.

We denote the degree of a vertex v in a graph H by $\deg_H(v)$, or shortly $\deg(v)$ when it is clear which graph it refers to. For $X \subseteq V$ or $X \subseteq E$, $\deg_X(v)$ is a short for $\deg_{G[X]}(v)$. If G is a directed graph, we denote the in- and out-degree of v in G by $\mathrm{indeg}_G(v)$ and $\mathrm{outdeg}_G(v)$, respectively. By a degree of a vertex in a directed graph we denote the sum of its indegree and outdegree.

In a directed graph G by *weakly connected components,* we mean the connected components of the underlying undirected graph. For a (directed) graph G, we let $\mathtt{cc}(G)$ denote the number of (weakly) connected components of G.

We denote the symmetric difference of two sets A and B by $A \triangle B$. For two integers a, b, we use $a \equiv b$ to indicate that a is even if and only if b is even. We use Iverson's bracket notation: If p is a predicate, we let $[p]$ be 1 if p if true and 0 otherwise. If $\omega : U \to \mathbb{Z}$, we shorthand $\omega(S) = \sum_{e \in S} \omega(e)$ for $S \subseteq U$.

For a function s by $s[v \to \alpha]$, we denote the function $s \setminus \{(v, s(v))\} \cup \{(v, \alpha)\}$. Note that this definition works regardless of whether v belongs to the domain of s or not (in the latter case we extend the domain).

2.2 Treewidth

Definition 2.1 (Tree Decomposition, [72]). A *tree decomposition* of a (undirected or directed) graph $G = (V, E)$ is a tree $\mathbb{T}$ in which each node $x \in \mathbb{T}$ has an assigned set of vertices $B_x \subseteq V$ (called a *bag*) such that $\bigcup_{x \in \mathbb{T}} B_x = V$ with the following properties:

— for any $uv \in E$, there exists an $x \in \mathbb{T}$ such that $u, v \in B_x$;
— if $v \in B_x$ and $v \in B_y$, then $v \in B_z$ for all z on the path from x to y in $\mathbb{T}$.

In what follows, we identify nodes of $\mathbb{T}$ and the bags assigned to them. The *width* of a tree decomposition $\mathbb{T}$ (denoted as $\mathbf{tw}(\mathbb{T})$) is the size of the largest bag of $\mathbb{T}$ minus one, and the treewidth of a graph G is the minimum width over all possible tree decompositions of G.

Dynamic programming algorithms on tree decompositions are often presented on nice tree decompositions which were introduced by Kloks [57]. We refer to the tree decomposition definition given by Kloks as to a *standard nice tree decomposition.*

Definition 2.2. A *standard nice tree decomposition* is a tree decomposition where:

— Every bag has at most two children.
— If a bag x has two children y, z, then $B_x = B_y = B_z$.
— If a bag x has one child y, then either $|B_x| = |B_y| + 1$ and $B_y \subseteq B_x$ or $|B_x| + 1 = |B_y|$ and $B_x \subseteq B_y$.

We present a slightly different definition of a nice tree decomposition.

Definition 2.3 (Nice Tree Decomposition). A *nice tree decomposition* is a tree decomposition with one special bag r called the *root* with $B_r = \emptyset$ and in which each bag is one of the following types:

— **Leaf bag:** a leaf x of $\mathbb{T}$ with $B_x = \emptyset$.
— **Introduce vertex bag:** an internal vertex x of $\mathbb{T}$ with one child vertex y for which $B_x = B_y \cup \{v\}$ for some $v \notin B_y$. This bag is said to *introduce* v.

— **Introduce edge bag:** an internal vertex x of $\mathbb{T}$ labeled with an edge $uv \in E$ with one child bag y for which $u, v \in B_x = B_y$. This bag is said to *introduce* uv.
— **Forget bag:** an internal vertex x of $\mathbb{T}$ with one child bag y for which $B_x = B_y \setminus \{v\}$ for some $v \in B_y$. This bag is said to *forget* v.
— **Join bag:** an internal vertex x with two child vertices y and z with $B_x = B_y = B_z$.

We additionally require that every edge in E is introduced exactly once.

We note that this definition is slightly different than usual. In our definition we have the extra requirements that bags associated with the leafs and the root are empty. Moreover, we added the introduce edge bags.

Given a tree decomposition, a standard nice tree decomposition of equal width can be found in polynomial time [57] and in the same running time, it can easily be modified to meet our extra requirements, as follows: add a series of forget bags to the old root, and add a series of introduce vertex bags below old leaf bags that are nonempty; Finally, for every edge $uv \in E$ add an introduce edge bag above the first bag with respect to the in-order traversal of $\mathbb{T}$ that contains u and v.

For two bags x, y of a rooted tree we say that y is a descendant of x if it is possible to reach x when starting at y and going only up the tree. In particular, x is its own descendant. By fixing the root of $\mathbb{T}$, we associate with each bag x in a tree decomposition $\mathbb{T}$ a vertex set $V_x \subseteq V$ where a vertex v belongs to V_x if and only if there is a bag y which is a descendant of x in $\mathbb{T}$ with $v \in B_y$. We also associate with each bag x of $\mathbb{T}$ a subgraph of G as follows:

$$G_x = (V_x, E_x = \{e : e \text{ is introduced in a descendant of } x\}).$$

For an overview of tree decompositions and dynamic programming on tree decompositions see [16, 51].

2.3 Isolation Lemma

An important ingredient of our algorithms is the Isolation Lemma:

Definition 2.4. A function $\omega : U \to \mathbb{Z}$ *isolates* a set family $\mathcal{F} \subseteq 2^U$ if there is a unique $S' \in \mathcal{F}$ with $\omega(S') = \min_{S \in \mathcal{F}} \omega(S)$.

Lemma 2.5 (Isolation Lemma, [65]). *Let $\mathcal{F} \subseteq 2^U$ be a set family over a universe U with $|\mathcal{F}| > 0$, and let $N > |U|$ be an integer. For each $u \in U$, choose a weight $\omega(u) \in \{1, 2, \ldots, N\}$ uniformly and independently at random. Then* $\texttt{prob}[\omega \text{ isolates } \mathcal{F}] \geq 1 - |U|/N$.

The Isolation Lemma allows us to count objects modulo 2, since with a large probability it reduces a possibly large number of solutions to some problem to a unique one (with an additional weight constraint imposed).

An alternative method to a similar end is obtained by using Polynomial Identity Testing [34, 74, 80] over a field of characteristic two. This second method has been already used in the field of exact and parameterized algorithms [10, 59]. The two methods do not differ much in their consequences: Both use the same number of random bits, and the challenge of giving a full derandomization seems to be equally difficult for both methods [2, 54]. The usage of the Isolation Lemma gives greater polynomial overheads; however, we choose to use it because it requires less preliminary knowledge and it simplifies the presentation.

3 CUT&COUNT: ILLUSTRATION OF THE TECHNIQUE

In this section, we present the Cut&Count technique by demonstrating how it applies to the Steiner Tree and Directed Min Cycle Cover problems. We go through the details in an expository manner, as we aim not only at showing the solutions to these particular problems but also to show the general workings.

We have chosen Steiner Tree and Directed Min Cycle Cover problems to show that our technique can be applied both to vertex and edge selection problems, and both to undirected and directed graphs and also that not only it allows for ensuring connectivity but more generally it allows to minimize the number of connected components.

In the last section of this chapter, we give an overview of the Cut&Count technique.

3.1 Steiner Tree

Steiner Tree
Input: An undirected graph $G = (V, E)$, a set of terminals $T \subseteq V$ and an integer k.
Question: Is there a set $X \subseteq V$ of cardinality k such that $T \subseteq X$ and $G[X]$ is connected?

In what follows, we will call the set X asked for in the question of the Steiner Tree problem a *solution*. Let $N = 2|V|$ and for each $v \in V$ choose a weight $\omega(v) \in \{1, \ldots, N\}$ uniformly and independently at random. For each $W \in \{1, \ldots, kN\}$, let $\mathcal{S}_W$ be the set of solutions of weight W. Clearly, if there is no solution, then for every weight W we have $\mathcal{S}_W = \emptyset$. However, if there is a solution, then by the Isolation Lemma, for some $W \in \{1, \ldots, kN\}$, with probability at least 1/2 (at least $1 - \frac{|U|}{N} = 1 - \frac{|V|}{2|V|} = 1/2$), we have $|\mathcal{S}_W| = 1$; in particular, we have that $|\mathcal{S}_W|$ is odd. Hence, we reduced the decision problem to the problem of counting the number of weight W solutions modulo 2. A method to perform this counting efficiently is described in two parts: the Cut part and the Count part.

The main goal of the Cut part is to define a set $\mathcal{C}_W$ such that (1) $|\mathcal{C}_W| \equiv |\mathcal{S}_W| \pmod 2$ and (2) $|\mathcal{C}_W|$ can be computed using $2^{O(\mathbf{tw}(G))}|V|^{O(1)}$ arithmetic operations. In the Count part we prove that the set $|\mathcal{C}_W|$ indeed has the desired properties.

From here on, we will simply write $a \equiv b$ instead of $a \equiv b \pmod 2$ for equivalence modulo two.

The Cut part. In the Cut part we always start with defining a set of *candidate solutions* which is a superset of all solutions to the problem we are solving. Those candidate solutions are local, in the sense that they are easy to control using standard dynamic programming techniques on tree decompositions. In the Steiner Tree problem, we look for a set X of size k containing all the terminals such that $G[X]$ is connected. The set of candidate solutions $\mathcal{R}_W$ is obtained by relaxing the connectivity constraint:

$$\mathcal{R}_W = \{X \subseteq V : T \subseteq X \wedge \omega(X) = W \wedge |X| = k\}.$$

In this easy application of the Cut&Count method, the only requirement that remains is that the set of terminals is contained in the candidate solution, i.e, it need not be connected. Although the cardinality of $\mathcal{R}_W$ can be easily computed within the desired time bound, the parity of $|\mathcal{R}_W|$ does not need to match the parity of $|\mathcal{S}_W|$.

In order to describe the required set $\mathcal{C}_W$, we define a set of *consistent* cuts of induced subgraphs of G. Recall, a cut of a graph $G = (V, E)$ is a partition of the set V into two sets $(V_1, V \setminus V_1)$.

Definition 3.1. A cut (V_1, V_2) of an undirected graph $G = (V, E)$ is *consistent* if $u \in V_1$ and $v \in V_2$ implies $uv \notin E$. A *consistently cut subgraph* of G is a pair $(X, (X_1, X_2))$ such that (X_1, X_2) is a consistent cut of $G[X]$.

To break the symmetry, instead of considering all 2^k cuts of $G[X]$, we consider only 2^{k-1} of them by selecting some vertex v_1 (in most applications arbitrarily chosen) and assuring that $v_1 \in X_1$. In the current Steiner Tree problem setting, let v_1 be an arbitrary terminal from T (w.l.o.g. $T \neq \emptyset$).

Define C_W as

$$C_W = \{(X, (X_1, X_2)) : X \in \mathcal{R}_W \wedge (X_1, X_2) \text{ is a consistent cut of } G[X] \wedge v_1 \in X_1\}.$$

The Count part. The crucial part follows, which is to prove that in the set of candidate solutions each solution of the problem is consistent with **exactly one** cut, whereas all other candidate solutions are consistent with an **even number** of cuts.

Lemma 3.2. *Let $G = (V, E)$ be a graph and let X be a subset of vertices such that $v_1 \in X \subseteq V$. The number of consistently cut subgraphs $(X, (X_1, X_2))$ such that $v_1 \in X_1$ is equal to $2^{\mathtt{cc}(G[X])-1}$.*

Proof. By definition, we know for every consistently cut subgraph $(X, (X_1, X_2))$ and connected component C of $G[X]$ that either $C \subseteq X_1$ or $C \subseteq X_2$. For the connected component containing v_1, the choice is fixed, and for all $\mathtt{cc}(G[X])-1$ other connected components we are free to choose a side of a cut, which gives $2^{\mathtt{cc}(G[X])-1}$ possibilities leading to different consistently cut subgraphs. □

Now it is easy to see that instead of calculating $|\mathcal{S}_W| \bmod 2$ directly, we can calculate $|C_W| \bmod 2$ instead.

Lemma 3.3. *Let G, ω, C_W, and $\mathcal{S}_W$ be as defined above. Then for every W, $|\mathcal{S}_W| \equiv |C_W|$.*

Proof. By Lemma 3.2, we know that $|C_W| = \sum_{X \in \mathcal{R}_W} 2^{\mathtt{cc}(G[X])-1}$. Therefore, $|C_W| \equiv |\{X \in \mathcal{R}_W | \mathtt{cc}(G[X]) = 1\}| = |\mathcal{S}_W|$. □

Now the only missing ingredient left is a sub-procedure `CountC`, which computes the cardinality of C_W modulo 2. It is a standard application of dynamic programming.

Lemma 3.4. *Given $G = (V, E)$, $T \subseteq V$, an integer k, $\omega : V \to \{1, \ldots, N\}$ and a nice tree decomposition $\mathbb{T}$ of width t, there exists an algorithm that can determine $|C_W|$ modulo 2 for every $0 \le W \le kN$ in $3^t N^2 |V|^{O(1)}$ time.*

Proof. We use dynamic programming, but we first need some preliminary definitions. Recall that for a bag $x \in \mathbb{T}$ we denoted by V_x the set of vertices in bags of all descendants of x, while by G_x we denoted the graph composed of vertices V_x and the edges E_x introduced by the descendants of x. Let $v_1 \in T$ an arbitrary terminal from T. We now define "partial solutions": for every bag $x \in \mathbb{T}$, for integers $i = 0, \ldots, k$, and $w = 0, \ldots, kN$ and for every $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$, define

$$\begin{aligned}
\mathcal{R}_x(i, w) &= \{X \subseteq V_x :\ (T \cap V_x) \subseteq X \ \wedge\ |X| = i \ \wedge\ \omega(X) = w\}, \\
C_x(i, w) &= \{(X, (X_1, X_2)) :\ X \in \mathcal{R}_x(i, w) \ \wedge\ (X, (X_1, X_2)) \text{ is a consistently} \\
&\qquad \text{cut subgraph of } G_x \wedge\ (v_1 \in V_x \Rightarrow v_1 \in X_1)\}, \\
A_x(i, w, s) &= |\{(X, (X_1, X_2)) \in C_x(i, w) :\ (s(v) = \mathbf{1}_j \Rightarrow v \in X_j) \\
&\qquad \wedge\ (s(v) = \mathbf{0} \Rightarrow v \notin X)\}|.
\end{aligned}$$

The intuition behind these definitions is as follows: The set $\mathcal{R}_x(i, w)$ contains all sets $X \subset V_x$ that could potentially be extended to a candidate solution from $\mathcal{R} = \bigcup \mathcal{R}_W$, subject to an additional restriction that the cardinality and weight of the partial solution are equal to i and w, respectively. Similarly, $C_x(i, w)$ contains consistently cut subgraphs, which could potentially be extended to elements of $C = \bigcup C_W$, again with the cardinality and weight restrictions. The integer $A_x(i, w, s)$ counts those elements of $C_x(i, w)$ which additionally behave on vertices of B_x in a fashion prescribed by the sequence s. $\mathbf{0}, \mathbf{1}_1$, and $\mathbf{1}_2$ (we refer to them as colours) describe the position of any particular vertex with respect to a set X with a consistent cut (X_1, X_2) of $G[X]$—the vertex can either be outside X, in X_1 or in X_2. In particular, note that

$$\sum_{s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}} A_x(i, w, s) = |C_x(i, w)|$$

—the various choices of s describe all possible intersections of an element of C with B_x. Observe that since we are interested in values $|C_W|$ modulo 2 it suffices to compute values $A_r(k, W, \emptyset)$ for all W (recall that r is the root of the tree decomposition), because $|C_W| = |C_r(k, W)|$.

We now give the recurrence for $A_x(i, w, s)$ which is used by the dynamic programming algorithm. In order to simplify the notation, let v denote the vertex introduced and contained in an introduce bag, and let y, z denote the left and right children of x in $\mathbb{T}$, if present (if there is only one child, we denote it by y).

— **Leaf bag** x**:**

$$A_x(0, 0, \emptyset) = 1.$$

All other values of $A_x(i, w, s)$ are zeroes.

— **Introduce vertex** v **bag** x**:** for $i = 0, \ldots, k$, for $w = 0, \ldots, kN$, for $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_y}$

$$\begin{aligned} A_x(i, w, s[v \to \mathbf{0}]) &= [v \notin T] A_y(i, w, s), \\ A_x(i, w, s[v \to \mathbf{1}_1]) &= A_y(i - 1, w - \omega(v), s), \\ A_x(i, w, s[v \to \mathbf{1}_2]) &= [v \neq v_1] A_y(i - 1, w - \omega(v), s). \end{aligned}$$

where $s[v \to \mathbf{0}]$ is the sequence s with the element representing v replaced by $\mathbf{0}$, and $[v \notin T]$ is Iverson's bracket notation which equals 1 if $v \notin T$ and 0 otherwise. For the first case above note that by definition v can not be coloured $\mathbf{0}$ if it is a terminal. For the other cases, the accumulators i, w have to be updated and we have to make sure we do not put $s(v_1) = \mathbf{1}_2$.

— **Introduce edge** uv **bag** x**:** for $i = 0, \ldots, k$, for $w = 0, \ldots, kN$, for $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$

$$A_x(i, w, s) = [s(u) = \mathbf{0} \ \vee \ s(v) = \mathbf{0} \ \vee \ s(u) = s(v)] A_y(i, w, s).$$

Here, we filter table entries inconsistent with the edge (u, v), i.e., table entries where the endpoints are coloured $\mathbf{1}_1$ and $\mathbf{1}_2$.

— **Forget vertex** v **bag** x**:** for $i = 0, \ldots, k$, for $w = 0, \ldots, kN$, for $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$

$$A_x(i, w, s) = \sum_{\alpha \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}} A_y(i, w, s[v \to \alpha]).$$

In the child bag, the vertex v can have three states so we sum over all of them.

— **Join bag:** for $i = 0, \ldots, k$, for $w = 0, \ldots, kN$, for $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$

$$A_x(i, w, s) = \sum_{i_1 + i_2 = i + |s^{-1}(\{\mathbf{1}_1, \mathbf{1}_2\})|} \ \sum_{w_1 + w_2 = w + \omega(s^{-1}(\{\mathbf{1}_1, \mathbf{1}_2\}))} A_y(i_1, w_1, s) A_z(i_2, w_2, s).$$

The only valid combinations to achieve the colouring s is to have the same colouring in both children. Since vertices coloured $\mathbf{1}_j$ in B_x are accounted for in the accumulated weights of both of the children, we add their contribution to the accumulators.

It is easy to see that the Lemma can now be obtained by combining the above recurrence with dynamic programming. The running time follows because there are $3^t N|V|^{O(1)}$ values $A_x(i, w, s)$ to compute, which can take $O(N|V|)$ time each in a join bag. Note that as we perform all calculations modulo 2, we take only constant time to perform any arithmetic operation. □

We conclude this section with the following theorem.

THEOREM 3.5. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves* STEINER TREE *in* $3^t|V|^{O(1)}$ *time. The algorithm cannot give false positives and may give false negatives with probability at most* $1/2$.

PROOF. Our algorithm is as follows. Set $N = 2|V|$, obtain a nice tree decomposition $\mathbb{T}$, and choose the weights $\omega : V \to \{1, \ldots, N\}$ uniformly and independently at random. Using Lemma 3.4, calculate $|\mathcal{C}_W|$ modulo 2, for every $W = 0, \ldots, kN$ in $3^t|V|^{O(1)}$ time. If for some W we have $|\mathcal{C}_W| \equiv 1$, then return **yes**. Otherwise return **no**.

To prove correctness use the Isolation Lemma (Lemma 2.5), where we substitute U with V and $\mathcal{F}$ with $\mathcal{S}$. We infer that if $\mathcal{S} \neq \emptyset$, then with probability at least $1/2$ there exists an index W, for which $|\mathcal{S}_W| = 1$ and consequently by Lemma 3.3 we have $|\mathcal{C}_W| \equiv 1$. □

3.2 Directed Cycle Cover

DIRECTED MIN CYCLE COVER
Input: A directed graph $D = (V, A)$, an integer k.
Question: Can the vertices of D be covered with at most k vertex disjoint directed cycles?

This problem is significantly different from the one considered in the previous section, since the aim is to maximize connectivity in a more flexible way: in the previous section the solution induced one connected component, while it may induce at most k weakly connected components in the context of the current section. Note that with the Cut&Count technique as introduced above, the solutions we are looking for cancel modulo 2.

We introduce a concept called *markers*. A set of solutions consists of pairs (X, M), where $X \subseteq A$ is a cycle cover and $M \subseteq X$, $|M| = k$ is a set of *marked* arcs, such that each cycle in X contains at least one marked arc. Since $|M| = k$, this ensures that for every solution (X, M) the cycle cover X consists of at most k cycles. Note that distinguishing two different sets of marked arcs of a single cycle cover is considered to induce two *different solutions*. For this reason, with each arc of the graph we associate *two* random weights: the first contributes to the weight of a solution, when an arc belongs to X, while the second contributes additionally, when it belongs to M as well. When we relax the requirement that in the pair (X, M) each cycle in X contains at least one arc from M, we obtain a set of candidate solutions. The objects we count are pairs consisting of (i) a pair (X, M), where $X \subseteq A$ is a cycle cover and $M \subseteq X$ is a set of k markers, (ii) a cut consistent with $D[X]$, where all the marked arcs from M have both endpoints on the left side of the cut. We will see that candidate solutions that contain a cycle without any marked arc cancel modulo 2. Formal definitions follow.

The Cut part. Let $\cdot\mathbf{X}\cdot$, $\cdot\mathbf{M}\cdot$ be symbols. As said before, we assume that we are given a weight function $\omega : A \times \{\cdot\mathbf{X}\cdot\} \cup A \times \{\cdot\mathbf{M}\cdot\} \to \{1, \ldots, N\}$. The arguments $A \times \{\cdot\mathbf{X}\cdot\}$ correspond to the contribution of choosing an arc to belong to X, while $A \times \{\cdot\mathbf{M}\cdot\}$ correspond to additional contribution of choosing it to M as well.

Definition 3.6. For a directed graph $D = (V, A)$ a cut (V_1, V_2) is consistent if (V_1, V_2) is a consistent cut in the underlying undirected graph. A consistently cut subgraph of D is a pair $(X, (V_1, V_2))$ where $X \subseteq A$ such that (V_1, V_2) is a consistent cut of the underlying undirected graph of $D[X]$.

Definition 3.7. For an integer W we define

(1) $\mathcal{R}_W$ to be the family of candidate solutions, that is, $\mathcal{R}_W$ is the family of all pairs (X, M), such that $X \subseteq A$ is a cycle cover, i.e., $\mathrm{outdeg}_{D[X]}(v) = \mathrm{indeg}_{D[X]}(v) = 1$ for every vertex $v \in V$; $M \subseteq X$, $|M| = k$ and $\omega(X \times \{\cdot\mathbf{X}\cdot\} \cup M \times \{\cdot\mathbf{M}\cdot\}) = W$;

(2) $\mathcal{S}_W$ to be the family of solutions, that is, $\mathcal{S}_W$ is the family of all pairs (X, M), where $(X, M) \in \mathcal{R}_W$ and every cycle in X contains at least one arc from the set M;
(3) C_W as all pairs $((X, M), (V_1, V_2))$ such that $(X, M) \in \mathcal{R}_W$, (V_1, V_2) is a consistent cut of $D[X]$ and $V(M) \subseteq V_1$.

Observe that the graph D admits a cycle cover with at most k cycles if and only if there exists W such that $\mathcal{S}_W$ is nonempty.

The Count part. We proceed to the Count part by showing that candidate solutions that contain an unmarked cycle cancel modulo 2.

Lemma 3.8. *Let D, ω, C_W, and $\mathcal{S}_W$ be defined as above. Then, for every W, $|\mathcal{S}_W| \equiv |C_W|$.*

Proof. For subsets $M \subseteq X \subseteq A$, let $\mathtt{cc}(M, X)$ denote the number of weakly connected components of $D[X]$ not containing any arc from M. Then,

$$|C_W| = \sum_{(X,M)\in\mathcal{R}_W} 2^{\mathtt{cc}(M,X)}.$$

To see this, note that for any $((X, M), (V_1, V_2)) \in C_W$ and any vertex set C of a cycle from X not containing arcs from M, we have $((X, M), (V_1 \triangle C, V_2 \triangle C)) \in C_W$—we can move all the vertices of C to the other side of the cut, also obtaining a consistent cut. Thus, for any set of choices of a side of the cut for every cycle not containing a marker, there is an object in C_W. Hence, (analogously to Lemma 3.2) for any W and $(M, X) \in \mathcal{R}_W$, there are $2^{\mathtt{cc}(M,X)}$ cuts (V_1, V_2) such that $((X, M), (V_1, V_2)) \in C_W$ and the lemma follows, because

$$|C_W| \equiv |\{((X, M), (V_1, V_2)) \in C_W : \mathtt{cc}(M, X) = 0\}| = |\mathcal{S}_W|. \qquad \square$$

Now, it suffices to present a dynamic programming routine counting $|C_W|$ modulo 2 in a bottom-up fashion. This procedure is technical, because we optimize the base of the exponential function. However, the ideas of the proof of the following lemma are not directly related to the Cut&Count technique itself, we postpone the proof to Section 4.1, where we also show how to solve the Longest Path problem in a similar manner.

Lemma 3.9. *Given $D = (V, A)$, an integer k, a weight function $\omega : A\times\{\cdot\mathbf{X}\cdot\}\cup A\times\{\cdot\mathbf{M}\cdot\} \to \{1, \ldots, N\}$ and a tree decomposition $\mathbb{T}$ of width t, there is an algorithm that can determine $|C_W|$ modulo 2 for every $0 \le W \le (k + |V|)N$ in $6^t N^2 |V|^{O(1)}$ time.*

Combining all the observations, we can conclude the following:

Theorem 3.10. *There exists a Monte-Carlo algorithm that, given a tree decomposition of width t, solves* Directed Min Cycle Cover *in $6^t |V|^{O(1)}$ time. The algorithm cannot give false positives and may give false negatives with probability at most* $1/2$.

Proof. The algorithm is as follows. Set $U = A \times \{\cdot\mathbf{X}\cdot\} \cup A \times \{\cdot\mathbf{M}\cdot\}$, $N = 2|U|$, and choose the weights $\omega : A \cup V \to \{1, \ldots, N\}$ uniformly and independently at random. Using Lemma 3.9 calculate $|C_W|$ modulo 2, for every $W = 0, \ldots, (k + |V|)N$ in $6^t |V|^{O(1)}$ time. If for some W we have $|C_W| \equiv 1$, then return **yes**. Otherwise return **no**.

The correctness follows from Lemma 3.8 and Isolation Lemma (Lemma 2.5). $\square$

3.3 General Idea Overview

The Cut&Count technique applies to problems with certain connectivity requirements. Let $\mathcal{S} \subseteq 2^U$ be a set of solutions (usually the universe U is the set of vertices or edges/arcs of the input graph); we aim at deciding whether it is empty. Conceptually, Cut&Count can naturally be split in two parts:

- **The Cut part:** Relax the connectivity requirement by considering the set $\mathcal{R} \supseteq \mathcal{S}$ of possibly disconnected candidate solutions. Furthermore, consider the set $\mathcal{C}$ of pairs (X, C) where $X \in \mathcal{R}$ and C is a consistent cut of X.
- **The Count part:** Compute $|\mathcal{C}|$ modulo 2 using a sub-procedure. Non-connected candidate solutions $X \in \mathcal{R} \setminus \mathcal{S}$ cancel since they are consistent with an even number of cuts. Connected candidates $x \in \mathcal{S}$ remain.

Note that we need the number of solutions to be odd in order to make the counting part work. For this we use the Isolation Lemma (Lemma 2.5): We introduce uniformly and independently chosen weights $\omega(v)$ for every $v \in U$ and compute $|\mathcal{C}_W|$ modulo 2 for every W, where $\mathcal{C}_W = \{(X, C) \in \mathcal{C} : \omega(X) = W\}$. If for some W we have $|\mathcal{C}_W| \equiv 1$, then return **yes**. Otherwise return **no**. The general setup can thus be summarized as in Algorithm 1:

ALGORITHM 1: Cut&Count general schema.

```
for every v ∈ U do
    Choose ω(v) ∈ {1, . . . , 2|U|} uniformly at random.
for every W ∈ {0, . . . , 2|U|^2} do
    if |{(X, C) ∈ C : ω(X) = W}| ≡ 1 then return yes
return no
```

The following corollary that we use throughout the article follows from Lemma 2.5 by setting $\mathcal{F} = \mathcal{S}$ and $N = 2|U|$:

Corollary 3.11. *Let $\mathcal{S} \subseteq 2^U$ and $\mathcal{C} \subseteq 2^U \times (2^V \times 2^V)$. Suppose that for every $W \in \mathbb{Z}$:*

$$|\{(X, C) \in \mathcal{C} : \omega(X) = W\}| \equiv |\{X \in \mathcal{S} : \omega(X) = W\}|.$$

*Then, Algorithm 1 returns **no** if $\mathcal{S}$ is empty and **yes** with probability at least $\frac{1}{2}$ otherwise.*

When applying the technique, both the Cut and the Count part are non-trivial: In the Cut part, one has to find the proper relaxation of the solution set, and in the Count part one has to show that the number of non-solutions is even for each W and provide an algorithm which computes $|\mathcal{C}_W|$ mod 2. Usually, the count part requires more explanation.

4 CUT&COUNT APPLIED TO SEVERAL PROBLEMS

In this section, we give full details of dynamic programming routines on graphs of bounded treewidth for several problems. First, in Section 4.1, we present the full desciption of Min Cycle Cover (ommited in Section 3.2) and Longest Path. Next, in Section 4.2, we consider Feedback Vertex Set, while in Section 4.3 the Connected Vertex Cover problem is studied.

While obtaining the 6^t dependency in Section 4.1 is non-trivial, the content of Sections 4.2 and 4.3 is not very deep on its own. However, in Section 5, we use the algorithms from this section for Feedback Vertex Set and Connected Vertex Cover to obtain the best known solution size parametrizations for those problems, which is the reason why we give the details of the application of Cut&Count here.

In all algorithms we assume that we are given a tree decomposition of the input graph G of width t. The algorithms all start with constructing a nice tree decomposition, as in Definition 2.3. In the dynamic programming descriptions, we follow the notation from the Steiner Tree example (see Lemma 3.4). Moreover, we solve unweighted versions of all the problems; however, the algorithms can be easily extended to the weighted case when weights are bounded by a polynomial in $|V|$.

4.1 Longest Cycles, Paths, and Cycle Covers

In this section, we consider the following three problems, both in the directed and undirected setting.

Directed Min Cycle Cover **Input:** A directed graph $D = (V, A)$ and an integer k. **Question:** Can the vertices of D be covered with at most k vertex disjoint directed cycles?

Directed Longest Cycle **Input:** A directed graph $D = (V, A)$ and an integer k. **Question:** Does there exist a directed simple cycle of length k in D?

Directed Longest Path **Input:** A directed graph $D = (V, A)$ and an integer k. **Question:** Does there exist a directed simple path of length k in D?

We capture all three problems in the following artificial one.

Directed Partial Cycle Cover **Input:** A directed graph $D = (V, A)$ and integers k and ℓ. **Question:** Does there exist a family of at most k vertex disjoint directed cycles in D that cover exactly ℓ vertices?

Note that for $k = 1$ the above problem becomes Directed Longest Cycle, whereas for $\ell = |V|$ it becomes Directed Min Cycle Cover. The Directed Longest Path problem can be easily reduced to Directed Longest Cycle: given a (Directed) Longest Path instance (D, k), we add one additional vertex v to D and connect all vertices $d \in V$ to v by arcs in both directions (d, v) and (v, d). Moreover, given a tree decomposition $\mathbb{T}$ of D, a tree decomposition for the modified graph can be easily constructed by adding v to every bag. This increases the width of the new decomposition by one compared to the width of $\mathbb{T}$.

We now show how to solve Directed Partial Cycle Cover using the Cut&Count technique, in time $6^t|V|^{O(1)}$. Observe that the undirected cases of all the problems considered in this subsection can be reduced to the directed ones by bidirecting edges. Moreover, if we want to optimize the constant in the undirected case, it is possible to design $4^{\mathbf{tw}(G)}|V|^{O(1)}$ time algorithms, as shown in [24, 28].

Theorem 4.1. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves* Directed Partial Cycle Cover *in $6^t|V|^{O(1)}$ time. The algorithm cannot give false positives and may give false negatives with probability at most* 1/2.

Proof. We use the Cut&Count technique. To count the number of cycles we use marked arcs. The objects we count are subsets of arcs, together with sets of marked arcs, thus we take $U = A \times \{\cdot\mathbf{X}\cdot, \cdot\mathbf{M}\cdot\}$. As usual, we assume we are given a weight function $\omega : U \to \{1, 2, \ldots, N\}$, where $N = 2|U| = 4|A|$. We also assume $k \leq \ell$.

The Cut part. For an integer W we define

(1) $\mathcal{R}_W$ to be the family of pairs (X, M), where $M \subseteq X \subseteq A$, $|X| = \ell$, $|M| = k$, $\omega(X \times \{\cdot\mathbf{X}\cdot\} \cup M \times \{\cdot\mathbf{M}\cdot\}) = W$, and each vertex $v \in V(X)$ has indegree and outdegree 1 in $G[X]$.
(2) $\mathcal{S}_W$ to be the family of pairs $(X, M) \in \mathcal{R}_W$, such that each connected component of $G[X]$ is either an isolated vertex or contains an arc from M.

(3) $\mathcal{C}_W$ to be the family of pairs $((X, M), (X_1, X_2))$, where $(X, M) \in \mathcal{R}_W$ and (X_1, X_2) is a consistent cut of the graph $(V(X), X)$ with $V(M) \subseteq X_1$.

Note that if $|X| = \ell$ and each vertex in $V(X)$ has indegree and outdegree one, then $|V(X)| = \ell$. Thus, similarly as before we need to check if $\mathcal{S}_W \neq \emptyset$ for some W.

The Count part. Let $((X, M), (X_1, X_2)) \in \mathcal{C}_W$ be a set of arcs X with markers M and (X_1, X_2) a consistent cut of $(V(X), X)$. Let $\mathtt{cc}(X, M)$ denote the number of weakly[2] connected components of $G[X]$ that are not isolated vertices and do not contain an arc from M. If $C \subseteq X$ is the set of arcs of such a weakly connected component of $G[X]$, then $((X, M), (X_1 \triangle V(C), X_2 \triangle V(C))) \in \mathcal{C}_W$, i.e., the weakly connected component C can be on either side of the cut (X_1, X_2). Thus, there are $2^{\mathtt{cc}(M,X)}$ elements in $\mathcal{C}_W$ that correspond to any pair $(X, M) \in \mathcal{R}_W$, and we infer that $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$.

To finish the proof we need to describe a procedure $\mathtt{CountC}(\omega, W, \mathbb{T})$ that, given a nice tree decomposition $\mathbb{T}$, a weight function ω and an integer W, computes $|\mathcal{C}_W|$ modulo 2.

As usual we use dynamic programming. We follow the notation from the Steiner Tree example (see Lemma 3.4). Let $\Sigma = \{\mathbf{00}, \mathbf{01}_1, \mathbf{01}_2, \mathbf{10}_1, \mathbf{10}_2, \mathbf{11}\}$. For every bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \leq i, b \leq |V|$, $0 \leq w \leq 2N|V|$ and $s \in \Sigma^{B_x}$ (called the colouring) define

$$\begin{aligned}
\mathcal{R}_x(i, b, w) = \{(X, M) \ :\ & M \subseteq X \subseteq E_x \ \wedge\ |M| = i \ \wedge\ |X| = b \ \wedge\ \omega(X \times \{\cdot\mathbf{X}\cdot\} \cup M \times \{\cdot\mathbf{M}\cdot\}) = w \\
& \wedge\ (\forall_{v \in V(X)\setminus B_x} \mathrm{indeg}_{G[X]}(v) = \mathrm{outdeg}_{G[X]}(v) = 1) \\
& \wedge\ (\forall_{v \in B_x} \max\{\mathrm{indeg}_{G[X]}(v), \mathrm{outdeg}_{G[X]}(v)\} \leq 1)\}, \\
\mathcal{C}_x(i, b, w) = \{((X, M), (X_1, X_2)) \ :\ & (X, M) \in \mathcal{R}_x(i, b, w) \ \wedge\ V(M) \subseteq X_1 \\
& \wedge\ (X_1, X_2) \text{ is a consistent cut of the graph } (V(X), X), \} \\
A_x(i, b, w, s) = |\{((X, M), (X_1, X_2)) \in \mathcal{C}_x(i, b, w) \ :\ & (s(v) = \mathbf{io}_j \Rightarrow v \in X_j) \\
& \wedge\ ((s(v) = \mathbf{io} \vee s(v) = \mathbf{io}_j) \Rightarrow (\mathrm{indeg}_{G[X]}(v) = \mathbf{i} \wedge \mathrm{outdeg}_{G[X]}(v) = \mathbf{o}))\}|,
\end{aligned}$$

here $\mathbf{io}$ and $\mathbf{io}_j$ are the decompositions of the symbols from $\Sigma = \{\mathbf{00}, \mathbf{01}_1, \mathbf{01}_2, \mathbf{10}_1, \mathbf{10}_2, \mathbf{11}\}$ into integer variables $\mathbf{i}$, $\mathbf{o}$, and j.

The value of $s(v)$ contains information about the indegree and outdegree of v and, in case when the degree of v is one, $s(v)$ also stores information about the side of the cut v belongs to. We note that we do not need to store the side of the cut for v if its degree is 0 and 2, since it is not yet or no more needed. The accumulators i, b, and w keep track of the size of M, the size of X and the weight of (X, M), respectively.

The algorithm computes $A_x(i, b, w, s)$ for all bags $x \in T$ in a bottom-up fashion for all reasonable values of i, b, w, and s. Before we present the routine computing the values $A_x(i, b, w, s)$, we first recall a variant of fast subset convolution [11], as it is needed to handle join bags efficiently. We follow notation from [11]. Let $f, g : 2^B \to R$ for some finite set B and ring R. In all our applications the ring R is $\mathbb{Z}_2$, thus the ring operations take constant time.

Definition 4.2. The *subset convolution* of f and g is defined as a function $f * g : 2^B \to R$ as follows:

$$(f * g)(T) = \sum_{T_1, T_2 \subseteq T} [T_1 \cup T_2 = T][T_1 \cap T_2 = \emptyset] f(T_1) g(T_2).$$

By computing a function $h : 2^B \to R$, we mean determining $h(T)$ for every $T \subseteq B$. Björklund et al. [11] proved that subset convolution can be computed efficiently. The following generalization of the subset convolution can be found in [30, 79].

[2] We stress this for clarity: in $G[X]$ weakly connected components are always strongly connected components due to the requirements imposed on X. That is, every vertex $v \in V(X)$ has indegree and outdegree 1 in $G[X]$.

Definition 4.3. Let $p \geq 2$ be an integer constant and let B be a finite set. For $t_1, t_2, t \in \{0, 1, \ldots, p-1\}^B$, we say that $t_1 + t_2 = t$ iff $t_1(b) + t_2(b) = t(b)$ for all $b \in B$. For functions $f, g : \{0, 1, \ldots, p-1\}^B \to R$ define

$$(f *^p g)(t) = \sum_{t_1+t_2=t} f(t_1)g(t_2).$$

Note that here the addition **is not** evaluated in $\mathbb{Z}_p^B$ but in $\mathbb{Z}^B$, i.e., not modulo p.

Theorem 4.4 (Generalized Subset Convolution [30, 78, 79]). *The generalized subset convolution can be computed in $p^{|B|}|B|^{O(1)}$ ring operations.*

Note that in [30] only the case $R = \mathbb{Z}$ is considered. However, in our applications ($R = \mathbb{Z}_2$), we can perform calculations in $\mathbb{Z}$ and at the end take all computed values modulo 2 within the claimed time bound.

We now give the recurrence for $A_x(i, b, w, s)$ that is used by the dynamic programming algorithm. In order to simplify notation, let v be the vertex introduced and contained in an introduce bag, (u, v) the arc introduced in an introduce edge (arc) bag, and y, z the left and right child of x in $\mathbb{T}$ if present.

— **Leaf bag:**

$$A_x(0, 0, 0, \emptyset) = 1.$$

— **Introduce vertex bag:**

$$A_x(i, b, w, s[v \to \mathbf{00}]) = A_y(i, b, w, s).$$

The new vertex has indegree and outdegree 0.

— **Introduce edge (arc) bag:** For the sake of simplicity of the recurrence formula let us define functions insubs, outsubs : $\Sigma \to 2^\Sigma$.

$\alpha \in \Sigma$	$\mathbf{00}$	$\mathbf{01}_1$	$\mathbf{01}_2$	$\mathbf{10}_1$	$\mathbf{10}_2$	$\mathbf{11}$
insubs(α)	$\emptyset$	$\emptyset$	$\emptyset$	$\{\mathbf{00}\}$	$\{\mathbf{00}\}$	$\{\mathbf{01}_1, \mathbf{01}_2\}$
outsubs(α)	$\emptyset$	$\{\mathbf{00}\}$	$\{\mathbf{00}\}$	$\emptyset$	$\emptyset$	$\{\mathbf{10}_1, \mathbf{10}_2\}$

Intuitively, for a given state $\alpha \in \Sigma$ the values insubs(α) and outsubs(α) are the sets of possible states a vertex can have before adding an incoming and respectively outgoing arc.

We can now write the recurrence for the introduce arc bag.

$$\begin{aligned} A_x(i, b, w, s) = &A_y(i, b, w, s) + \sum_{\alpha_u \in \text{outsubs}(s(u))} \sum_{\alpha_v \in \text{insubs}(s(v))} \sum_{j \in \{1,2\}} \\ &[(\alpha_u = \mathbf{10}_j \vee s(u) = \mathbf{01}_j) \wedge (\alpha_v = \mathbf{01}_j \vee s(v) = \mathbf{10}_j)] \\ &(A_y(i, b-1, w - \omega(((u, v), \cdot\mathbf{X}\cdot)), s[u \to \alpha_u, v \to \alpha_v]) \\ &+ [j = 1]A_y(i-1, b-1, w - \omega((u, v), \cdot\mathbf{X}\cdot) - \omega((u, v), \cdot\mathbf{M}\cdot), s[u \to \alpha_u, v \to \alpha_v])). \end{aligned}$$

To see that all cases are handled correctly, first notice that we can always choose not to use the introduced arc. Observe that in order to add the arc (u, v) by the definition of insubs and outsubs we need to have $\alpha_u \in$ outsubs$(s(u))$ and $\alpha_v \in$ insubs$(s(v))$. We use the integer j to iterate over two sides of the cut the arc (u, v) can be contained in. Finally we check whether $j = 1$ before we make (u, v) a marker.

— **Forget vertex v bag x:**

$$A_x(i, b, w, s) = A_y(i, b, w, s[v \to \mathbf{11}]) + A_y(i, b, w, s[v \to \mathbf{00}]).$$

The forgotten vertex must have degree 0 or 2.

	00	**01**$_1$	**01**$_2$	**10**$_2$	**10**$_1$	**11**
00	**00**	**01**$_1$	**01**$_2$	**10**$_2$	**10**$_1$	**11**
01$_1$	**01**$_1$	XX	XX	XX	**11**	XX
01$_2$	**01**$_2$	XX	XX	**11**	XX	XX
10$_2$	**10**$_2$	XX	**11**	XX	XX	XX
10$_1$	**10**$_1$	**11**	XX	XX	XX	XX
11	**11**	XX	XX	XX	XX	XX

Fig. 1. The join table of Directed Partial Cycle Cover where it is indicated which states combine to which other states.

— **Join bag:** We have two children y and z. Figure 1 shows how two individual states of a vertex in y and z combine to a state of x. XX indicates that two states do not combine. The correctness of the table is easy to check.

For colourings $s_1, s_2, s \in \Sigma^{B_x}$, we say that $s_1 + s_2 = s$ if for each vertex $v \in B_x$ the values of $s_1(v)$ and $s_2(v)$ combine into $s(v)$ as in Figure 1. We can now write the recurrence formula for join bags.

$$A_x(i,b,w,s) = \sum_{i_1+i_2=i} \sum_{b_1+b_2=b} \sum_{w_1+w_2=w} \sum_{s_1+s_2=s} A_y(i_1,b_1,w_1,s_1)A_z(i_2,b_2,w_2,s_2).$$

A straightforward computation of the above formula leads to $36^t|V|^{O(1)}$ time complexity. We now show how to use the Generalized Subset Convolution to obtain a better time bound. Let $\phi, \rho : \Sigma \to \{0,1,2,3,4,5\}$ where

$$\begin{array}{llllll} \phi(\mathbf{00}) = 0 & \phi(\mathbf{01}_1) = 1 & \phi(\mathbf{01}_2) = 2 & \phi(\mathbf{10}_2) = 3 & \phi(\mathbf{10}_1) = 4 & \phi(\mathbf{11}) = 5 \\ \rho(\mathbf{00}) = 0 & \rho(\mathbf{01}_1) = 1 & \rho(\mathbf{01}_2) = 1 & \rho(\mathbf{10}_2) = 1 & \rho(\mathbf{10}_1) = 1 & \rho(\mathbf{11}) = 2. \end{array}$$

Let $\phi : \Sigma^{B_x} \to \{0,1,2,3,4,5\}^{B_x}$ be obtained by extending ϕ in the natural way. Define $\rho : \Sigma^{B_x} \to \mathbb{Z}$ as $\rho(s) = \sum_{e \in B_x} \rho(e)$. Hence, ρ reflects the total number of $\mathbf{1}$'s in a state s, i.e., the sum of all degrees of vertices in B_x. Then, define

$$\begin{aligned} f_m^{i,b,w}(\phi(s)) &= [\rho(s) = m]A_y(i,b,w,s), \\ g_m^{i,b,w}(\phi(s)) &= [\rho(s) = m]A_z(i,b,w,s), \\ h_m^{i,b,w}(\phi(s)) &= \sum_{i_1+i_2=i} \sum_{b_1+b_2=b} \sum_{w_1+w_2=w} \sum_{m_1+m_2=m} \left(f_{m_1}^{i_1,b_1,w_1} *^6 g_{m_2}^{i_2,b_2,w_2}\right)(\phi(s)). \end{aligned}$$

We claim that

$$A_x(i,b,w,s) = h_{\rho(s)}^{i,b,w}(\phi(s)).$$

To see this, first notice that the values of accumulators are divided among the children, and that no vertex or edge is accounted for twice by the definition of A_x. Hence, it suffices to prove that exactly all combinations of table entries from A_y and A_z that combine to state s according to Table 1 contribute to $A_x(i,b,w,s)$. Notice that if $\alpha, \beta \in \Sigma$, and $\gamma = \phi^{-1}(\phi(\alpha) + \phi(\beta))$, then $\rho(\gamma) \leq \rho(\alpha) + \rho(\beta)$. This implies that the only pairs that contribute to $h_m^{i,b,w}(\phi(s))$ are the pairs not leading to crosses in Table 1 since for the other pairs we have $\rho(\gamma) < \rho(\alpha) + \rho(\beta)$. Finally, notice that for every such pair, we have that γ is the correct state, and hence correctness follows.

Finally we obtain that, by Theorem 4.4, the values $A_x(i,b,w,s)$ for a join bag x can be computed in time $6^t|V|^{O(1)}$.

It is easy to see that the above recurrence leads to a dynamic programming algorithm that computes the parity of $|\mathcal{S}_W|$ for all values of W in $6^t|V|^{O(1)}$ time, since $|\mathcal{C}_W| = A_r(k, \ell, W, \emptyset)$ and $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$. Moreover, as we count the parities and not the numbers A_x themselves, all arithmetical operations can be done in constant time. Thus, the proof of Theorem 4.1 is finished. □

4.2 Feedback Vertex Set

In this section, we show an algorithm for a more general version of the Feedback Vertex Set problem, where we are additionally given a set of vertices that have to belong to the solution.

Constrained Feedback Vertex Set **Input:** An undirected graph $G = (V, E)$, a subset $S \subseteq V$, and an integer k. **Question:** Does there exist a set $Y \subset V$ of cardinality k such that $S \subseteq Y$ and $G[V \setminus Y]$ is a forest?

This constrained version of the problem is useful when we want to obtain not only binary output, but also use self-reducibility to find the actual set of vertices—the solution Y. We take advantage of this generalized problem definition in Section 5.1.

Here, defining a solution candidate with a relaxed connectivity condition to work with our technique is somewhat more tricky, as there is no explicit connectivity requirement in the problem to begin with. We proceed by choosing the (presumed) forest left after removing the candidate solution and using the following simple lemma:

Lemma 4.5. *A graph $G = (V, E)$ with n vertices and m edges is a forest iff it has at most $n - m$ connected components.*

Proof. Let $E = \{e_1, \ldots, e_m\}$. Consider a graph $G_0 = (V, \emptyset)$ with the same set of vertices and an empty set of edges. We add edges from the set E to the graph G_0 one by one. Observe that G is a forest iff after adding each edge from E to the graph G_0 the number of connected components of G_0 decreases. Since initially G_0 has n connected components the lemma follows. □

Theorem 4.6. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves the* Constrained Feedback Vertex Set *problem in $3^t|V|^{O(1)}$ time. The algorithm cannot give false positives and may give false negatives with probability at most* 1/2.

Proof. We use the Cut&Count technique. As the universe we take the set $U = V \times \{\cdot\mathbf{F}\cdot, \cdot\mathbf{M}\cdot\}$, where $V \times \{\cdot\mathbf{F}\cdot\}$ is used to assign weights to vertices from the chosen forest and $V \times \{\cdot\mathbf{M}\cdot\}$ for markers. As usual we assume that we are given a weight function $\omega : U \to \{1, \ldots, N\}$, where $N = 2|U| = 4|V|$.

The Cut part. For integers A, B, C, and W we define

(1) $\mathcal{R}_W^{A,B,C}$ to be the family of solution candidates: marked subgraphs excluding S of size and weight prescribed by super-/sub-scripts, i.e., $\mathcal{R}_W^{A,B,C}$ is the family of pairs (X, M), where $X \subseteq V \setminus S$, $|X| = A$, $G[X]$ contains exactly B edges, $M \subseteq X$, $|M| = C$ and $\omega(X \times \{\cdot\mathbf{F}\cdot\}) + \omega(M \times \{\cdot\mathbf{M}\cdot\}) = W$;
(2) $\mathcal{S}_W^{A,B,C}$ to be the set of solutions: the family of pairs (X, M), where $(X, M) \in \mathcal{R}_W^{A,B,C}$ and $G[X]$ is a forest containing at least one marker from the set M in each connected component;
(3) $\mathcal{C}_W^{A,B,C}$ to be the family of pairs $((X, M), (X_1, X_2))$, where $(X, M) \in \mathcal{R}_W^{A,B,C}$, $M \subseteq X_1$, and (X_1, X_2) is a consistent cut of $G[X]$.

Observe that by Lemma 4.5, the graph G admits a feedback vertex set of size k containing S if and only if there exist integers B,W such that the set $\mathcal{S}_W^{n-k,B,n-k-B}$ is nonempty.

The Count part. Similarly as in the case of Min Cycle Cover (analogously to Lemma 3.8) note that for any A,B,C,W,$(X,M) \in \mathcal{R}_W^{A,B,C}$, there are $2^{\mathtt{cc}(M,G[X])}$ cuts (X_1,X_2) such that $((X,M),(X_1,X_2)) \in C_W^{A,B,C}$, where by $\mathtt{cc}(M,G[X])$, we denote the number of connected components of $G[X]$, which do not contain any marker from the set M. Hence, by Lemma 4.5 for every A,B,C,W satisfying $C \leq A - B$ we have $|\mathcal{S}_W^{A,B,C}| \equiv |C_W^{A,B,C}|$.

Now we describe a procedure $\mathtt{CountC}(\omega, A, B, C, W, \mathbb{T})$ that, given a nice tree decomposition $\mathbb{T}$, weight function ω and integers A,B,C,W, computes $|C_W^{A,B,C}|$ modulo 2 using dynamic programming.

For every bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \leq a \leq |V|$, $0 \leq b < |V|$, $0 \leq c, \leq |V|$, $0 \leq w \leq 2N|V|$ and $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$ (called a colouring) define

$$\begin{aligned}
\mathcal{R}_x(a,b,c,,w) = \{(X,M) \ :\ & X \subseteq V_x \setminus S \ \wedge\ |X| = a \ \wedge\ |E_x \cap E(G[X])| = b \\
& \wedge\ M \subseteq X \setminus B_x \ \wedge\ |M| = c, \ \wedge\ \omega(X \times \{\cdot\mathbf{F}\cdot\}) + \omega(M \times \{\cdot\mathbf{M}\cdot\}) = w\}, \\
C_x(a,b,c,,w) = \{((X,M),(X_1,X_2)) \ :\ & (X,M) \in \mathcal{R}_x(a,b,c,,w) \\
& \wedge\ M \subseteq X_1 \wedge (X,(X_1,X_2)) \text{ is a consistently cut subgraph of } G_x\}, \\
A_x(a,b,c,,w,s) = |\{((X,M),(X_1,X_2)) & \in C_x(a,b,c,,w) : \\
& (s(v) = \mathbf{1}_j \Rightarrow v \in X_j) \ \wedge\ (s(v) = \mathbf{0} \Rightarrow v \notin X)\}|.
\end{aligned}$$

Note that we assume $b < |V|$ because otherwise an induced subgraph containing b edges is definitely not a forest.

Similarly as in the case of Steiner Tree, $s(v) = \mathbf{0}$ means $v \notin X$, whereas $s(v) = \mathbf{1}_j$ corresponds to $v \in X_j$. The accumulators a,b,c, and w keep track of the number of vertices and edges in the subgraph induced by vertices from X, number of markers already used and the sum of weights of chosen vertices and markers. Hence $A_x(a,b,c,,w,s)$ is the number of pairs from $C_x(a,b,c,,w)$ with a fixed interface on vertices from B_x. Note that we ensure that no vertex from B_x is yet marked, because we decide whether to mark a vertex or not in its forget bag. Recall that the tree decomposition is rooted in an empty bag; hence, for every vertex, there exists exactly one forget bag forgetting it.

The algorithm computes $A_x(a,b,c,,w,s)$ for all bags $x \in \mathbb{T}$ in a bottom-up fashion for all reasonable values of a, b, c,, w, and s (defined above). We now give the recurrence for $A_x(a,b,c,,w,s)$ that is used by the dynamic programming algorithm. In order to simplify notation let v be the vertex introduced and contained in an introduce bag, uv the edge introduced in an introduce edge bag, and let y,z stand for the left and right child of x in $\mathbb{T}$ if present.

— **Leaf bag:**

$$A_x(0,0,0,0,\emptyset) = 1.$$

— **Introduce vertex bag:**

$$\begin{aligned}
A_x(a,b,c,,w,s[v \to \mathbf{0}]) &= A_y(a,b,c,,w,s), \\
A_x(a,b,c,,w,s[v \to \mathbf{1}_j]) &= [v \notin S] A_y(a-1,b,c,,w-\omega((v,\cdot\mathbf{F}\cdot)),s).
\end{aligned}$$

— **Introduce edge bag:**

$$\begin{aligned}
A_x(a,b,c,,w,s) = {} & [s(u) = \mathbf{0} \vee s(v) = \mathbf{0} \vee s(u) = s(v)] \\
& \cdot A_y(a, b - [s(u) = s(v) \neq \mathbf{0}], c,,w,s).
\end{aligned}$$

Here, we remove table entries not consistent with the edge uv, and update the accumulator b storing the number of edges in the induced subgraph.

- **Forget bag:**

$$A_x(a, b, c, , w, s) = A_y(a, b, c, -1, w - \omega((v, \cdot\mathbf{M}\cdot)), s[v \to \mathbf{1}_1]\}) + \sum_{\alpha \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}} A_y(a, b, c, , w, s[v \to \alpha]\})).$$

  If the vertex v was in X_1, then we can mark it and update the accumulator $c,$. If we do not mark the vertex v then it can have any of the three states with no additional requirements imposed.

- **Join bag:**

$$A_x(a, b, c, , w, s) = \sum_{a_1+a_2=a+|s^{-1}(\{\mathbf{1}_1,\mathbf{1}_2\})|} \sum_{b_1+b_2=b} \sum_{c,_1+c,_2=c,} \sum_{w_1+w_2=w+\omega(s^{-1}(\{\mathbf{1}_1,\mathbf{1}_2\})\times\{\cdot\mathbf{F}\cdot\})} A_y(a_1, b_1, c,_1, w_1, s) A_z(a_2, b_2, c,_2, w_2, s).$$

  The only valid combinations to achieve the colouring s is to have the same colouring in both children. Since vertices coloured $\mathbf{1}_j$ in B_x are accounted for in both tables of the children, we add their contribution to the accumulators a and w.

Since $|C_W^{A,B,C}| = A_r(A, B, C, W, \emptyset)$ the above recurrence leads to a dynamic programming algorithm that computes the parity of $|C_W^{A,B,C}|$ for all reasonable values of W, A, B, C in $3^t|V|^{O(1)}$ time. Consequently we finish the proof of Theorem 4.6. □

4.3 Connected Vertex Cover

In this section, we show a $3^t|V|^{O(1)}$ time algorithm for Connected Vertex Cover. Similarly as in Section 4.2, we solve a more general version of the problem where additionally as a part of the input we are given a set $S \subseteq V$ which contains vertices that must belong to a solution.

> Constrained Connected Vertex Cover
> **Input:** An undirected graph $G = (V, E)$, a subset $S \subseteq V$ and an integer k
> **Question:** Does there exist a subset $X \subseteq V$ of cardinality k such that $S \subseteq X$, $G[X]$ is connected and each edge $e \in E$ is incident with at least one vertex from X?

Remark 4.7. In the algorithms, we assume that the set $S \subseteq V$ is non-empty, so we can choose one fixed vertex $v_1 \in S$ that needs to be included in a fixed side of all considered cuts (cf. algorithm for Steiner Tree in Section 3.1). To solve the problem where $S = \emptyset$, we simply take an edge uv can call the algorithm with $S = \{u\}$ and $S = \{v\}$. Note that this does not increase the probability that the (Monte-Carlo) algorithm gives a wrong answer. Our algorithms can only give false negatives, so in the case of a YES answer in the first run, we do not need the second run to give a correct answer.

Theorem 4.8. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves* Constrained Connected Vertex Cover *in $3^t|V|^{O(1)}$ time. The algorithm cannot give false positives and may give false negatives with probability at most* $1/2$.

There exists an easy proof of Theorem 4.8 by a reduction to the Steiner Tree problem—subdivide all edges of the graph G and consider the vertices from S and those created from the subdivisions as terminals. Such a transformation does not change the treewidth of the graph by more than one. Nonetheless, we prove the theorem by a direct application of the Cut&Count technique, in a similar manner as for the Steiner Tree problem in Section 3.1. Our motivation for choosing the second approach is that we need it to develop an algorithm for Connected Vertex Cover parameterized by the solution size in Section 5.2, which relies on the algorithm we describe here.

Proof. We use the Cut&Count technique. As the universe for Algorithm 1, we take the vertex set $U = V$. Recall that we generate a random weight function $\omega : U \to \{1, 2, \ldots, N\}$, taking $N = 2|U| = 2|V|$. By Remark 4.7, we may assume that $S \neq \emptyset$ and we may choose one fixed vertex $v_1 \in S$.

The Cut part. For an integer W we define

(1) $\mathcal{R}_W$ to be the family of solution candidates (vertex covers) of size k and weight W: $\mathcal{R}_W$ is the family of sets $X \subseteq V$ such that $S \subseteq X$, $|X| = k$, $\omega(X) = W$ and X is a vertex cover of G;
(2) $\mathcal{S}_W$ to be the family of solutions of size k and weight W, that is sets $X \in \mathcal{R}_W$ such that $G[X]$ is connected;
(3) C_W to be the family of pairs $(X, (X_1, X_2))$, where $X \in \mathcal{R}_W$, $v_1 \in X_1$ and (X_1, X_2) is a consistent cut of $G[X]$.

The Count part. By a similar argument as in Lemma 3.2 for each $X \in \mathcal{R}_W$ there exist $2^{\mathrm{cc}(G[X])-1}$ consistent cuts of $G[X]$ where $v_1 \in X_1$, thus for any W we have $|\mathcal{S}_W| \equiv |C_W|$.

To finish the proof we need to describe a procedure $\texttt{CountC}(\omega, W, \mathbb{T})$ that, given a nice tree decomposition $\mathbb{T}$, a weight function ω and an integer W, computes $|C_W|$ modulo 2.

For every bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \leq i \leq |V|$, $0 \leq w \leq N|V|$ and $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$ define:

$$\begin{aligned} \mathcal{R}_x(i, w) &= \{X \subseteq V_x \ : \ (S \cap V_x) \subseteq X \ \wedge \ |X| = i \ \wedge \ \omega(X) = w \\ &\qquad \wedge \ X \text{ is a vertex cover of } G_x\}, \\ C_x(i, w) &= \{(X, (X_1, X_2)) \ : \ X \in \mathcal{R}_x(i, w) \ \wedge \ (X, (X_1, X_2)) \text{ is a consistently} \\ &\qquad \text{cut subgraph of } G_x \ \wedge \ (v_1 \in V_x \Rightarrow v_1 \in X_1)\}, \\ A_x(i, w, s) &= |\{(X, (X_1, X_2)) \in C_x(i, w) \ : \ (s(v) = \mathbf{1}_j \Rightarrow v \in X_j) \ \wedge \ (s(v) = \mathbf{0} \Rightarrow v \notin X)\}|. \end{aligned}$$

Similarly as in the case of Steiner Tree, $s(v) = \mathbf{0}$ means $v \notin X$, whereas $s(v) = \mathbf{1}_j$ corresponds to $v \in X_j$. The accumulators i and w keep track of the number of vertices in the solution and their weights, respectively. Hence, $A_x(i, w, s)$ is the number of pairs from C of candidate solutions and consistent cuts on G_x, with fixed size, weight, and interface on vertices from B_x.

The algorithm computes $A_x(i, w, s)$ for all bags $x \in T$ in a bottom-up fashion for all reasonable values of i, w, and s. We now give the recurrence for $A_x(i, w, s)$ that is used by the dynamic programming algorithm. In order to simplify notation denote by v the vertex introduced and contained in an introduce bag, by uv the edge introduced in an introduce edge bag, and let y, z be the left and right child of x in $\mathbb{T}$ if present.

— **Leaf bag:**

$$A_x(0, 0, \emptyset) = 1.$$

— **Introduce vertex bag:**

$$\begin{aligned} A_x(i, w, s[v \to \mathbf{0}]) &= [v \notin S] A_y(i, w, s), \\ A_x(i, w, s[v \to \mathbf{1}_1]) &= A_y(i - 1, w - \omega(v), s), \\ A_x(i, w, s[v \to \mathbf{1}_2]) &= [v \neq v_1] A_y(i - 1, w - \omega(v), s). \end{aligned}$$

We take care of the restrictions imposed by the conditions $(S \cap V_x) \subseteq X$ and $v_1 \in X_1$.

— **Introduce edge bag:**

$$A_x(i, w, s) = [s(u) = s(v) \neq \mathbf{0} \ \vee \ (s(u) = \mathbf{0} \wedge s(v) \neq \mathbf{0}) \ \vee \ (s(u) \neq \mathbf{0} \wedge s(v) = \mathbf{0})] A_y(i, w, s).$$

Here, we remove table entries not consistent with the edge uv, i.e., table entries where the endpoints are colored $\mathbf{1}_1$ and $\mathbf{1}_2$ (thus creating an inconsistent cut) or $\mathbf{0}$ and $\mathbf{0}$ (thus leaving an edge that is not covered).

— **Forget bag:**

$$A_x(i, w, s) = \sum_{\alpha \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}} A_y(i, w, s[v \to \alpha]).$$

In the child bag, the vertex v can have three states, and no additional requirements are imposed, so we sum over all the three states.

— **Join bag:**

$$A_x(i, w, s) = \sum_{i_1 + i_2 = i + |s^{-1}(\{\mathbf{1}_1, \mathbf{1}_2\})|} \quad \sum_{w_1 + w_2 = w + \omega(s^{-1}(\{\mathbf{1}_1, \mathbf{1}_2\}))} A_y(i_1, w_1, s) A_z(i_2, w_2, s).$$

The only valid combination to achieve the colouring s is to have the same colouring in both children. Since vertices coloured $\mathbf{1}_j$ in B_x are accounted for in both tables of the children, we add their contribution to the accumulators.

It is easy to see that the above recurrence leads to a dynamic programming algorithm that computes the parity of $|\mathcal{S}_W|$ for all values of W in $3^t |V|^{O(1)}$ time, since $|\mathcal{C}_W| = A_r(k, W, \emptyset)$ and $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$. Moreover, as we count the parities and not the numbers A_x themselves, all arithmetical operations can be done in constant time. Thus, the proof of Theorem 4.8 is finished. □

5 SOLUTION SIZE PARAMETERIZATION

In this section, we show how the Cut&Count technique may be used to obtain FPT algorithms when parameterized by the solution size. We study vertex deletion problems in which the remaining graph has to be of constant treewidth, i.e., Feedback Vertex Set and Connected Vertex Cover, and improve the best known FPT algorithms for those problems. The main idea behind the new results is the combination of the *iterative compression* technique, developed by Reed et al. [71], and the Cut&Count technique.

5.1 Feedback Vertex Set

Theorem 5.1 (Theorem 1.4, Restated). *There exists a Monte-Carlo algorithm, which solves the* Feedback Vertex Set *problem for a graph with n vertices in* $3^k n^{O(1)}$ *time and polynomial space. The algorithm cannot give false positives and may give false negatives with probability at most* 1/2.

Proof. Let $v_1, v_2, \ldots, v_n$ be an arbitrary ordering of the vertices of the given graph $G = (V, E)$. Let us denote $G_i = G[\{v_1, v_2, \ldots, v_i\}]$ for all $1 \le i \le n$. Observe that if G admits a feedback vertex set of size at most k, i.e., there is a set $A \subseteq V$, $|A| \le k$ such that $G[V \setminus A]$ is a forest, then so do all the graphs G_i, because $G_i[\{v_1, v_2, \ldots, v_i\} \setminus A]$ is a forest as well and $|A \cap \{v_1, v_2, \ldots, v_i\}| \le |A| \le k$.

We construct feedback vertex sets $A_1, A_2, \ldots, A_n$ of size at most k consecutively in the graphs $G_1, G_2, \ldots, G_n = G$. If at any step the algorithm finds out that the set we seek does not exist (with high probability), we answer NO. We begin with $A_1 = \emptyset$, which is a feasible solution in the graph G_1. The idea of iterative compression is that when we are to construct the set A_{i+1}, we can use the previously constructed set A_i. Let $B_{i+1} = A_i \cup \{v_{i+1}\}$. Observe that B_{i+1} is a feedback vertex set in G_{i+1}. If $|B_{i+1}| \le k$, then we take $A_{i+1} = B_{i+1}$. Thus, we are left with the case in which, given a feedback vertex set, call it B, of size $k + 1$, we need to construct a feedback vertex set of size at most k or determine that none such exists.

As B is a feedback vertex set, the graph induced by the rest of the vertices is a forest. Thus, we can construct a tree decomposition of the graph G_{i+1} of width at most $k + 2$ by creating a tree decomposition of the forest of width 1 and adding the whole set B to each bag. We apply (using the tree decomposition obtained above as the input) the dynamic programming algorithm described in Section 4.2, running in $3^k n^{O(1)}$ time, which tests whether the graph G_{i+1} admits a feedback vertex set of size at most k. Observe that this algorithm, as described in the proof of Theorem 4.6, uses exponential space. However, in each step when computing $A_x(a,b,c,w,s)$ the algorithm refers only to values $A_y(a',b',c',w',s')$, where $s' = s$ on the intersection of the domains of s and s'. In our case the intersection of every two bags of the tree decomposition contains B. Therefore, we can reorder the computation in the following manner: for every evaluation $\overline{s} : B \to \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}$, we fix it as the "core" evaluation for every bag in the decomposition and run the algorithm to compute all the values $A_x(a,b,c,w,s)$, where $s|_B = \overline{s}$. Such a computation takes polynomial time and space. As there are 3^{k+1} such possible evaluations $\overline{s}$, the algorithm runs in $3^k n^{O(1)}$ time and in polynomial space. We make n independent runs of the algorithm in order to assure that the probability of a false negative is at most $\frac{1}{2^n}$.

Once, we have done this, we already tested with high probability whether the desired feedback vertex set exists or not. If the answer is negative, we answer NO. Otherwise we need to explicitly construct the set A_{i+1} in order to use it in the next step of the iterative compression. We make use of the algorithm for Constrained Feedback Vertex Set, given by Theorem 4.6. The algorithm considers the vertices of G_{i+1} one by one, building a set K which at the end will be the set A_{i+1} we want to construct. We begin with $K = \emptyset$ and preserve an invariant that at each step there is a feedback vertex set of size at most k containing the set K. When considering the vertex v, we test in $3^k n^{O(1)}$ time whether the graph admits a constrained feedback vertex set of size at most k with $S = K \cup \{v\}$, making n independent runs of the algorithm given by Theorem 4.6 in order to reduce the probability of a false negative to at most $\frac{1}{2^n}$. If the answer is positive, we can safely add v to K as we know that there is a feedback vertex set of size at most k containing $K \cup \{v\}$ (recall our algorithms do not return false positives). Otherwise we simply proceed to the next vertex. The computation terminates when K is already a feedback vertex set or when we have exhausted all vertices. Observe that if G_{i+1} admits a feedback vertex set of size at most k, this construction will terminate building a feedback vertex set A_{i+1} of size at most k unless there was an error in at least one of the tests. If we exhaust all vertices, we answer NO, as an error has occurred. Note that in each run of the algorithm for Constrained Feedback Vertex Set, we can reorder the computation in the same way as in the previous paragraph to reduce space usage to polynomial.

Observe that the described algorithm at most $n^2 + n$ times makes n independent runs of the algorithm from Theorem 4.6 as a subroutine: at most $n + 1$ times in each of the n steps of the iterative compression. Each of these groups of runs has the probability of a false negative bounded by $\frac{1}{2^n}$, thus the probability of a false negative is bounded by $\frac{n^2+n}{2^n}$, which is lower than $\frac{1}{2}$ for n large enough. □

5.2 Connected Vertex Cover

Now we proceed to the algorithm for Connected Vertex Cover. The previously best FPT algorithm is due to Binkele-Raible [8], and runs in $2.4882^k n^{O(1)}$ time complexity. As in Section 5.1 our algorithm uses iterative compression; however, we make use of the connectivity requirement in order to reduce the complexity from $3^k n^{O(1)}$ down to $2^k n^{O(1)}$.

Theorem 5.2. *There exists a Monte-Carlo algorithm which solves the* Connected Vertex Cover *problem for a graph with n vertices in $2^k n^{O(1)}$ time and polynomial space. The algorithm cannot give false positives and may give false negatives with probability at most* 1/2.

Proof. Firstly observe that the Connected Vertex Cover problem is *contraction*–closed. This means that if a graph H admits a connected vertex cover A of size at most k, then H' obtained from H by contracting an edge of H (and reducing possible multi-edges to simple edges) also admits a connected vertex cover A' of size at most k. Indeed, the contracted edge uv needs to be covered by A, so $u \in A$ or $v \in A$. Thus, we can construct A' by removing u and v from A and adding the vertex obtained from the contracted edge. It can be easily seen that A' is a connected vertex cover of H' of size at most k.

Moreover, without loss of generality, we can assume that the given graph G is connected. Therefore, we can consider a sequence of graphs $G_1, G_2, \ldots, G_n = G$, where G_i is obtained from G_{i+1} by contracting any edge and reducing possible multi-edges to simple edges, and G_1 is a graph composed of a single vertex. The argument from the previous paragraph ensures that we can proceed as in the proof of Theorem 5.1, namely, construct connected vertex covers for $G_1, G_2, \ldots, G_n$ consecutively, and the only thing we have to show is how to construct a connected vertex cover of size k in G_{i+1} given a connected vertex cover A_i of size k in G_i, or determine that none exists.

Let G_i be the graph constructed from G_{i+1} by contracting an edge uv. We construct a set B from A_i by removing the vertex obtained in the contraction (if it was contained in A_i) and inserting both u and v. Observe that B is of size at most $k + 2$ and it is a connected vertex cover of G_{i+1}. As $V(G_{i+1}) \setminus B$ is an independent set, we can construct a path decomposition of G_{i+1} of width at most $k + 2$: for every vertex from $V(G_{i+1}) \setminus B$, we introduce a bag, connect the bags in any order, and then add the set B to every bag.

Now we are going to test whether G_{i+1} admits a connected vertex cover of size at most k. We could apply the algorithm from Theorem 4.8. As in the proof of Theorem 5.1, also this dynamic programming algorithm during the computation of $A_x(i, w, s)$ refers only to values $A_y(i', w', s')$ for s' such that $s = s'$ on the intersection of domains of s and s'. Therefore, similarly as before, we would iterate through all possible evaluations $\bar{s} : B \to \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}$, each time computing all the values $A_x(i, w, s)$ such that $s|_B = \bar{s}$ in polynomial time, thus using only polynomial space in the whole algorithm. Unfortunately, the algorithm given by Theorem 4.8 runs in $3^k n^{O(1)}$ time.

We can, however, reduce the complexity by bounding the number of reasonable evaluations $\bar{s} : B \to \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}$ by $3 \cdot 2^{k+1}$. Since B is a connected vertex cover in G_{i+1}, it induces a graph consisting of a single large connected component. Take any spanning tree of the single, large component in $G_{i+1}[B]$ and root it at some vertex r. We present the evaluation $\bar{s}$ in the following manner. For the root r, we choose any value from $\{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}$ for $\bar{s}$ giving 3 choices in total. Now consider the rest of the tree (containing all the remaining vertices from B) in a top–down manner. Observe that every vertex v from the tree has only two possible evaluations, depending on the evaluation of its parent u:

— if $\bar{s}(u) = \mathbf{0}$, the two possible options are $\mathbf{1}_1, \mathbf{1}_2$, as otherwise the edge connecting v with its parent would not be covered;
— if $\bar{s}(u) = \mathbf{1}_j$, the two possible options are $\mathbf{0}$ and $\mathbf{1}_j$, as otherwise the evaluation $\bar{s}$ would not describe any consistent cut.

Thus, each of $k + 2$ elements of B has only two options, except for r, which has 3 options. This means we only need to consider $3 \cdot 2^{k+1}$ possible "core" evaluations $\bar{s}$, which yields an algorithm with running time $2^k n^{O(1)}$, using polynomial space. As previously, we make n independent runs of the algorithm in order to reduce the probability of a false negative to at most $\frac{1}{2^n}$.

Once, we have tested whether G_{i+1} admits a connected vertex cover of size at most k, we can construct it explicitly similarly as in the proof of Theorem 5.1 using the algorithm for Constrained Connected Vertex Cover. We consider vertices one by one, each time determining whether the vertex can be inserted into the connected vertex cover we are constructing by running the algorithm from Theorem 4.8 n times. Observe that all these runs can be done in $2^k n^{O(1)}$ time and polynomial space complexity using the same technique as in the testing. Thus, we succeed in constructing A_{i+1} unless at least one of the tests returns a false negative.

The algorithm makes at most n^2+n groups of n independent runs of algorithm from Theorem 4.8. Therefore, the probability of a false negative is bounded by $\frac{n^2+n}{2^n}$, which is less than $\frac{1}{2}$ for n large enough. □

We would like to note that after the extended abstract of this article was published, Cygan [25] has obtained a deterministic $2^k n^{O(1)}$ time algorithm for Connected Vertex Cover. Moreover, Cygan et al. [26] have shown that unless the Strong Exponential Time Hypothesis fails there does not exist an algorithm with $(2-\varepsilon)^k n^{O(1)}$ running time, which computes the parity of the number of connected vertex covers of size k in a given graph. Moreover, improving over the 2^k dependency for the decision variant of Connected Vertex Cover would lead to a refutation of the Set Cover Conjecture [26]. To the best of our knowledge this is the first example of a problem parameterized by the solution size where there exists some evidence showing that the best known dependency $f(k)$ might be optimal.

6 OPEN PROBLEMS

As we have already mentioned in Section 1.4, since the extended version of this article was published the research on algorithms for connectivity problems on bounded treewidth graphs has been very active. Still, however, there are unresolved questions, which we believe are worth investigating.

Perhaps the most natural question to ask is whether the base of the exponential function given by the Cut&Count technique, which is optimal for several problems under the Strong Exponential Time Hypothesis (see Table 1), can be obtained by a deterministic algorithm. Secondly, despite we know that weighted [15, 43] and (some) counting problem variants [15] can be solved in single-exponential time, we do not know whether the same complexity can be obtained as for the regular versions of these problems. It would be very interesting to see a separation between the best possible dependency on treewidth (or pathwidth) between a weighted problem variant and its regular unweighted counterpart.

Finally, for the concrete problem of Hamiltonicity, we do not have matching upper and lower bounds for the case of bounded treewidth. For graphs of bounded pathwidth, we know that a $(2+\sqrt{2})^{\mathbf{pw}(G)}|V(G)|^{O(1)}$ time algorithm exists [27] and there is matching $(2+\sqrt{2}-\epsilon)^{\mathbf{pw}(G)}|V(G)|^{O(1)}$ lower bound under the Strong Exponential Time Hypothesis [27]. For the more general case of bounded treewidth graphs this article gives a $4^{\mathbf{tw}G}|V(G)|^{O(1)}$ time algorithm, so there is a gap between $2+\sqrt{2}$ and 4 in the base of the exponential function.

REFERENCES

[1] Jochen Alber, Hans L. Bodlaender, Henning Fernau, Ton Kloks, and Rolf Niedermeier. 2002. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica* 33, 4 (2002), 461–493.

[2] Vikraman Arvind and Partha Mukhopadhyay. 2008. Derandomizing the isolation lemma and lower bounds for circuit size. In *Proceedings of the 11th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems/12th International Workshop on Randomization and Computation, APPROX/RANDOM 2008 (LNCS).* A. Goel, K. Jansen, J. D. P. Rolim, and R. Rubinfeld (Eds.), Vol. 5171, Springer, 276–289.

[3] Mohammad Hossein Bateni, Mohammad Taghi Hajiaghayi, and Dániel Marx. 2011. Approximation schemes for steiner forest on planar graphs and graphs of bounded treewidth. *Journal of the ACM* 58, 5 (2011), 21:1–21:37.

[4] Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. 2000. Randomized algorithms for the loop cutset problem. *Journal of Artificial Intelligence Research* 12, 1 (2000), 219–234.

[5] Benjamin Bergougnoux. 2019. *Matrix Decompositions and Algorithmic Applications to (Hyper)Graphs.* Ph.D. Dissertation. Université Clermont Auvergne.

[6] Benjamin Bergougnoux and Mamadou Moustapha Kanté. 2017. Fast exact algorithms for some connectivity problems parameterized by clique-width. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC'16)*, Vol. 63, Jiong Guo and Danny Hermelin (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[7] Benjamin Bergougnoux and Mamadou Moustapha Kanté. 2019. More applications of the d-neighbor equivalence: Connectivity and acyclicity constraints. In *Proceedings of the 27th Annual European Symposium on Algorithms, ESA 201 (Leibniz International Proceedings in Informatics)*. M. A. Bender, O. Svensson, and G. Herman (Eds.), Vol. 144, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 17:1–17:14.
[8] Daniel Binkele-Raible. 2010. *Amortized Analysis of Exponential Time and Parameterized Algorithms: Measure and Conquer and Reference Search Trees*. Ph.D. Dissertation. University of Trier, Trier, Germany.
[9] Andreas Björklund. 2011. Private communication. (2011).
[10] Andreas Björklund. 2014. Determinant sums for undirected hamiltonicity. *SIAM Journal on Computing* 43, 1 (2014), 280–299.
[11] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. 2007. Fourier meets Möbius: Fast subset convolution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, STOC 2007*. D. S. Johnson and U. Feige (Eds.), ACM Press, 67–74.
[12] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. 2017. Narrow sieves for parameterized paths and packings. *J. Comput. Syst. Sci.* 87 (2010), 119–139.
[13] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. 2012. The traveling salesman problem in bounded degree graphs. *ACM Transactions on Algorithms* 8, 2 (2012), 18:1–18:13.
[14] Hans L. Bodlaender. 1994. On disjoint cycles. *International Journal of Foundations of Computer Science* 5, 1 (1994), 59–68.
[15] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. 2015. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation* 243 (2015), 86–111.
[16] Hans L. Bodlaender and Arie M. C. A. Koster. 2008. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal* 51, 3 (2008), 255–269.
[17] Glencora Borradaile, Philip N. Klein, and Claire Mathieu. 2009. An $O(n \log n)$ approximation scheme for steiner tree in planar graphs. *ACM Transactions on Algorithms* 5, 3 (2009), 31:1–31:31.
[18] Glencora Borradaile and Hung Le. Optimal dynamic program for r-domination problems over tree decompositions. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation.* 8:1–8:23.
[19] Nicolas Bourgeois, Bruno Escoffier, Vangelis Th. Paschos, and Johan M. M. van Rooij. 2012. Fast algorithms for max independent set. *Algorithmica* 62, 1–2 (2012), 382–415.
[20] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. 2013. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM* 60, 1 (2013), 6:1–6:33.
[21] Yixin Cao, Jianer Chen, and Yang Liu. 2015. On feedback vertex set: New measure and new structures. *Algorithmica* 73, 1 (2015), 63–86.
[22] Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. 2008. Improved algorithms for feedback vertex set problems. *Journal of Computer and System Sciences* 74, 7 (2008), 1188–1198.
[23] Radu Curticapean, Nathan Lindzey, and Jesper Nederlof. 2018. A tight lower bound for counting hamiltonian cycles via matrix rank. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*. A. Czumaj (Ed.). Society for Industrial and Applied Mathematics, 1080–1099.
[24] Marek Cygan. 2012. *Cut&Count Technique for Graph Connectivity Problems Parameterized by Treewidth*. Ph.D. Dissertation. University of Warsaw, Warsaw, Poland.
[25] Marek Cygan. 2012. Deterministic parameterized connected vertex cover. In *Proceedings of the 13th Scandinavian Symposium and Workshops on Algorithm Theory* . F. V. Fomin and P. Kaski (Eds.), Vol. 7357. Springer, 95–106.
[26] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. 2012. On problems as hard as CNF-SAT. In *Proceedings of the 27th Conference on Computational Complexity*. IEEE Computer Society, 74–84.
[27] Marek Cygan, Stefan Kratsch, and Jesper Nederlof. 2013. Fast Hamiltonicity checking via bases of perfect matchings. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing*. D. Boneh, T. Roughgarden, and J. Feigenbaum (Eds.), ACM Press, 301–310.
[28] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. 2011. Solving connectivity problems parameterized by treewidth in single exponential time. arXiv:1103.0534. Retrieved from https://arxiv.org/abs/1103.0534.
[29] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. 2011. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*. Rafail Ostrovsky (Ed.), IEEE Computer Society, 150–159.
[30] Marek Cygan and Marcin Pilipczuk. 2010. Exact and approximate bandwidth. *Theoretical Computer Science* 411, 40–42 (2010), 3701–3713.

[31] Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. 2007. An $O(2^{O(k)} n^3)$ FPT algorithm for the undirected feedback vertex set problem. *Theory of Computing Systems* 41, 3 (2007), 479–492.

[32] Erik D. Demaine and Mohammad T. Hajiaghayi. 2008. The bidimensionality theory and its algorithmic applications. *The Computer Journal* 51, 3 (2008), 292–302.

[33] Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Ken ichi Kawarabayashi. 2005. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, 637–646.

[34] Richard A. DeMillo and Richard J. Lipton. 1978. A probabilistic remark on algebraic program testing. *Information Processing Letters* 7, 4 (1978), 193–195.

[35] Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. 2006. Fast subexponential algorithm for non-local problems on graphs of bounded genus. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory*. L. Arge and R. Freivalds (Eds.), Vol. 4059. Springer, 172–183.

[36] Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. 2012. Catalan structures and dynamic programming in H-Minor-Free graphs. *Journal of Computer and System Sciences* 78, 5 (2012), 1606–1622.

[37] Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. 2010. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica* 58, 3 (2010), 790–810.

[38] Rodney G. Downey and Michael R. Fellows. 1993. Fixed parameter tractability and completeness. In *Proceedings of the Complexity Theory: Current Research*. K. Ambos-Spies, S. Homer, and U. Schöning (Eds.), Cambridge University Press, 191–225.

[39] Rodney G. Downey and Michael R. Fellows. 1999. *Parameterized Complexity*. Springer.

[40] David Eppstein. 2000. Diameter and treewidth in minor-closed graph families. *Algorithmica* 27, 3 (2000), 275–291.

[41] David Eppstein. 2007. The traveling salesman problem for cubic graphs. *Journal of Graph Algorithms and Applications* 11, 1 (2007), 61–81.

[42] Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Alexey A. Stepanov. 2009. On two techniques of combining branching and treewidth. *Algorithmica* 54, 2 (2009), 181–207.

[43] Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. 2014. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *25st Annual ACM-SIAM Symposium on Discrete Algorithms*. C. Chekuri (Ed.), Society for Industrial and Applied Mathematics, 142–151.

[44] Fedor V. Fomin and Dimitrios M. Thilikos. 2004. A simple and fast approach for solving problems on planar graphs. In *Proceedings of the 21st International Symposium on Theoretical Aspects of Computer Science*. V. Diekert and M. Habib (Eds.), Vol. 2996. Springer, 56–67.

[45] Heidi Gebauer. 2011. Enumerating all Hamilton cycles and bounding the number of Hamilton cycles in 3-regular graphs. *The Electronic Journal of Combinatorics* 18, 1 (2011), 132.

[46] Heidi Gebauer. 2011. Finding and enumerating Hamilton cycles in 4-regular graphs. *Theoretical Computer Science* 412, 35 (2011), 4579–4591.

[47] Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. 2006. Compression-Based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences* 72, 8 (2006), 1386–1396.

[48] Falko Hegerfeld and Stefan Kratsch. 2020. Solving connectivity problems parameterized by treedepth in single-exponential time and polynomial space. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science*. C. Paul and M. Bläser (Eds.), Vol. 154. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 29:1–29:16.

[49] Michael Held and Richard M. Karp. 1970. The traveling-salesman problem and minimum spanning trees. *Operations Research* 18, 6 (1970), 1138–1162.

[50] Micheal Held and Richard M. Karp. 1971. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming* 1, 1 (1971), 6–25.

[51] Illya V. Hicks, Arie M. C. A. Koster, and Elif Kolotoğlu. 2005. Branch and tree decomposition techniques for discrete optimization. In *Emerging Theory, Methods, and Applications, Chapter 1*. 1–29. DOI: 10.1287/educ.1053.0017

[52] Russell Impagliazzo and Ramamohan Paturi. 2001. On the complexity of k-SAT. *Journal of Computer and System Sciences* 62, 2 (2001), 367–375.

[53] Kazuo Iwama and Takuya Nakashima. 2007. An improved exact algorithm for cubic graph TSP. In *Proceedings of the 13th Annual International Conference on Computing and Combinatorics*. G. Lin (Ed.), Vol. 4598. Springer, 108–117.

[54] Valentine Kabanets and Russell Impagliazzo. 2004. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity* 13, 1–2 (2004), 1–46.

[55] Iyad A. Kanj, Michael J. Pelsmajer, and Marcus Schaefer. 2004. Parameterized algorithms for feedback vertex set. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation*. R. G. Downey, M. R. Fellows, and F. K. H. A. Dehne (Eds.), Vol. 3162. Springer, 235–247.

[56] Richard M. Karp. 1972. Reducibility among combinatorial problems. In *Proceedings of the symposium on the Complexity of Computer Computations*. Raymond E. Miller and James W. Thatcher (Eds.), Plenum Press, 85–103.
[57] Ton Kloks. 1994. *Treewidth, Computations and Approximations*. Springer.
[58] Tomasz Kociumaka and Marcin Pilipczuk. 2014. Faster deterministic feedback vertex set. *Information Processing Letters* 114, 10 (2014), 556–560.
[59] Ioannis Koutis. 2008. Faster algebraic algorithms for path and packing problems. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*. L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz (Eds.), Vol. 5125. Springer, 575–586.
[60] Jason Li and Jesper Nederlof. 2020. Detecting feedback vertex sets of size k in $O^*(2.7^k)$ time. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*. S. Chawla (Ed.), Society for Industrial and Applied Mathematics, 971–989.
[61] Daniel Lokshtanov, Daniel Marx, and Saket Saurabh. 2011. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*. D. Randall (Ed.), Society for Industrial and Applied Mathematics, 777–789.
[62] Daniel Lokshtanov, Daniel Marx, and Saket Saurabh. 2011. Slightly superexponential parameterized problems. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*. D. Randall (Ed.), Society for Industrial and Applied Mathematics, 760–776.
[63] Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. 2012. FPT algorithms for connected feedback vertex set. *Journal of Combinatorial Optimization* 24, 2 (2012), 131–146.
[64] Daniel Mölle, Stefan Richter, and Peter Rossmanith. 2008. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. *Theory of Computing Systems* 43, 2 (2008), 234–253.
[65] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. 1987. Matching is as easy as matrix inversion. *Combinatorica* 7, 1 (1987), 105–113.
[66] Jesper Nederlof, Johan M. M. van Rooij, and Thomas C. van Dijk. 2014. Inclusion/Exclusion meets measure and conquer. *Algorithmica* 69, 3 (2014), 685–740.
[67] Rolf Niedermeier. 2006. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and its Applications, Vol. 31. Oxford University Press.
[68] Willem J. A. Pino, Hans L. Bodlaender, and Johan M. M. van Rooij. 2017. Cut and count and representative sets on branch decompositions. In *Proceedings of the11th International Symposium on Parameterized and Exact Computation*. Jiong Guo and Danny Hermelin (Eds.), Vol. 63. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 27:1–27:12.
[69] Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. 2005. Faster algorithms for feedback vertex set. *Electronic Notes in Discrete Mathematics* 19 (2005), 273–279.
[70] Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. 2006. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Transactions on Algorithms* 2, 3 (2006), 403–415.
[71] Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. 2004. Finding odd cycle transversals. *Operations Research Letters* 32, 4 (2004), 299–301.
[72] Neil Robertson and Paul D. Seymour. 1984. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B* 36, 1 (1984), 49–64.
[73] Donald J. Rose. 1974. On simple characterizations of k-trees. *Discrete Mathematics* 7, 3–4 (1974), 317–322.
[74] Jacob T. Schwartz. 1980. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM* 27, 4 (1980), 701–717.
[75] Jan Arne Telle and Andrzej Proskurowski. 1997. Algorithms for vertex partitioning problems on partial k-Trees. *SIAM Journal on Discrete Mathematics* 10, 4 (1997), 529–550.
[76] Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. 2005. Cutwidth I: A linear time fixed parameter algorithm. *Journal of Algorithms* 56, 1 (2005), 1–24.
[77] William T. Tutte. 1947. The factorization of linear graphs. *Journal of the London Mathematical Society* 22, 2 (1947), 107–111.
[78] Johan M. M. van Rooij. 2021. A generic convolution algorithm for join operations on tree decompositions. In *Proceedings of the16th International Computer Science Symposium in Russia*. Rahul Santhanam and Daniil Musatov (Eds.), Vol. 12730. Springer, 435–459.
[79] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. 2009. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Proceedings of the 17th Annual European Symposium on Algorithms*. A. Fiat and P. Sanders (Eds.), Vol. 5757. Springer, 566–577.
[80] Richard Zippel. 1979. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*. E. W. Ng (Ed.), Vol. 72. Springer, 216–226.

Received November 2020; revised December 2021; accepted December 2021