Solving connectivity problems parameterized by treewidth in single exponential time

Marek Cygan[*]   Jesper Nederlof[†]   Marcin Pilipczuk[‡]   Michał Pilipczuk[§]

Johan van Rooij[¶]   Jakub Onufry Wojtaszczyk[‖]

Abstract

For the vast majority of local graph problems standard dynamic programming techniques give $c^{\mathbf{tw}}|V|^{O(1)}$ algorithms, where $\mathbf{tw}$ is the treewidth of the input graph. On the other hand, for problems with a global requirement (usually connectivity) the best–known algorithms were naive dynamic programming schemes running in $\mathbf{tw}^{O(\mathbf{tw})}|V|^{O(1)}$ time.

We breach this gap by introducing a technique we dubbed Cut&Count that allows to produce $c^{\mathbf{tw}}|V|^{O(1)}$ Monte Carlo algorithms for most connectivity-type problems, including HAMILTONIAN PATH, FEEDBACK VERTEX SET and CONNECTED DOMINATING SET, consequently answering the question raised by Lokshtanov, Marx and Saurabh [SODA'11] in a surprising way. We also show that (under reasonable complexity assumptions) the gap cannot be breached for some problems for which Cut&Count does not work, like CYCLE PACKING.

The constant c we obtain is in all cases small (at most 4 for undirected problems and at most 6 for directed ones), and in several cases we are able to show that improving those constants would cause the Strong Exponential Time Hypothesis to fail.

Our results have numerous consequences in various fields, like FPT algorithms, exact and approximate algorithms on planar and H-minor-free graphs and algorithms on graphs of bounded degree. In all these fields we are able to improve the best-known results for some problems.

1 Introduction and notation

The notion of *treewidth*, introduced in 1984 by Robertson and Seymour [56], has 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. For example, an expository algorithm to solve VERTEX COVER and INDEPENDENT SET running in time $4^{\mathbf{tw}(G)}|V|^{O(1)}$ is described in the algorithms textbook by Kleinberg and Tardos [43], while the book of Niedermeier [52] on fixed-parameter algorithms presents an algorithm with running time $2^{\mathbf{tw}(G)}|V|^{O(1)}$.

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 [19, 29] and parametrized algorithms [22] 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 applications for parametrized algorithms in general graphs [50, 60] for problems like CONNECTED VERTEX COVER and CUTWIDTH; and exact algorithms [30, 63] 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

---

[*]Institute of Informatics, University of Warsaw, Poland, `cygan@mimuw.edu.pl`

[†]Department of Informatics, University of Bergen, Norway, `Jesper.Nederlof@ii.uib.no`

[‡]Institute of Informatics, University of Warsaw, Poland, `malcin@mimuw.edu.pl`

[§]Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland, `michal.pilipczuk@students.mimuw.edu.pl`

[¶]Department of Information and Computing Sciences, Utrecht University, The Netherlands, `jmmrooij@cs.uu.nl`

[‖]Google Inc., Cracow, Poland, `onufry@google.com`

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 fails, there is no algorithm for VERTEX COVER running quicker than $(2 - \varepsilon)^{\mathbf{tw}(G)}$ for any $\varepsilon > 0$.

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 $\mathbf{tw}(G)^{O(\mathbf{tw}(G))}|V|^{O(1)}$. In these cases the typical dynamic program 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 F. Dorn, F. Fomin and D. Thilikos [22, 23] and by H. Bodlaender et al. in [24]. 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 asked by D. Lokshtanov, D. Marx and S. Saurabh [48].

To explain why the $2^{O(\mathbf{tw}(G) \log \mathbf{tw}(G))}$ dynamic programming algorithms for connectivity problems were thought to be optimal we recall the concept of a Myhill-Nerode style equivalence class. In this case two partial solutions of a subtree of the tree decomposition are said to be equivalent if they are consistent with the same set of partial solutions on the remainder of the tree decomposition [37]. The states of the naive dynamic program reflect Myhill-Nerode style equivalence classes, and it seemed to be necessary for any algorithm to memoize the information about each class, as it could be needed during further computation (see for example the recent work by Lokshtanov et al. [47, 48]). From this point of view the results of this paper come as a significant surprise.

1.1 Our results

In this paper we introduce a technique we dubbed "Cut&Count" that allows us to deal with connectivity-type problems through randomization. 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 [48], as well as for all the other sample problems mentioned in [23]: LONGEST PATH, LONGEST CYCLE, FEEDBACK VERTEX SET, HAMILTONIAN CYCLE and GRAPH METRIC TRAVELLING SALESMAN PROBLEM. Moreover, both the constant c and the exponent in $|V|^{O(1)}$ is in all cases well defined and small.

The randomization we mention comes from the usage of the Isolation Lemma [51]. 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 a graph G with n vertices, a tree decomposition of G of width t and a number k in $3^t n^{O(1)}$ 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 shall denote such an algorithm, which either confirms the existence of the object we are asking about or returns "unable to verify", and returns "unable to verify" with probability no larger than $1/2$ if the answer is positive, an algorithm with *false negatives*.

We see similar results for a plethora of other global problems. As the exact value of c in the $c^{\mathbf{tw}(G)}$ expression is often important, we gather here the results we obtain:

Theorem 1.2. *There exist Monte-Carlo algorithms that given a tree decomposition of the (underlying undirected graph of the) input graph of width t solve the following problems:*

1. STEINER TREE *in $3^t|V|^{O(1)}$ time.*

2. FEEDBACK VERTEX SET *in $3^t|V|^{O(1)}$ time.*

3. CONNECTED VERTEX COVER *in $3^t|V|^{O(1)}$ time.*

4. CONNECTED DOMINATING SET *in $4^t|V|^{O(1)}$ time.*

5. CONNECTED FEEDBACK VERTEX SET *in $4^t|V|^{O(1)}$ time.*

6. CONNECTED ODD CYCLE TRANSVERSAL *in $4^t|V|^{O(1)}$ time.*

7. MIN CYCLE COVER *in* $6^t|V|^{O(1)}$ *time for directed graphs and in* $4^t|V|^{O(1)}$ *time for undirected graphs.*

8. *Directed* LONGEST PATH *in* $6^t|V|^{O(1)}$ *time and undirected* LONGEST PATH *in* $4^t|V|^{O(1)}$ *time. This, in particular, gives the same times for solving* HAMILTONIAN PATH *for the directed and undirected cases, respectively.*

9. *Directed* LONGEST CYCLE *in* $6^t|V|^{O(1)}$ *time and undirected* LONGEST CYCLE *in* $4^t|V|^{O(1)}$ *time. This, in particular, gives the same times for solving* HAMILTONIAN CYCLE *for the directed and undirected cases, respecitvely.*

10. EXACT k-LEAF SPANNING TREE *and, in particular,* MINIMUM LEAF TREE *and* MAXIMUM LEAF TREE *in* $4^t|V|^{O(1)}$ *time for undirected graphs.*

11. EXACT k-LEAF OUTBRANCHING *and, in particular* MINIMUM LEAF OUTBRANCHING *and* MAXIMUM LEAF OUTBRANCHING *in* $6^t|V|^{O(1)}$ *time for directed graphs.*

12. MAXIMUM FULL DEGREE SPANNING TREE *in* $4^t|V|^{O(1)}$ *time.*

13. GRAPH METRIC TRAVELLING SALESMAN PROBLEM *in* $4^t|V|^{O(1)}$ *time for undirected graphs.*

The algorithms cannot give false positives and may give false negatives with probability at most $1/2$.

For a number of these results we have matching lower bounds, such as the following one:

Theorem 1.3. *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.*

We have such matching lower bounds for the following problems: CONNECTED VERTEX COVER, CONNECTED DOMINATING SET, CONNECTED FEEDBACK VERTEX SET, CONNECTED ODD CYCLE TRANSVERSAL, FEEDBACK VERTEX SET, STEINER TREE and EXACT k-LEAF SPANNING TREE. We feel that the first four results 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 $3^{\mathbf{tw}(G)}n^{O(1)}$ from the $2^{\mathbf{tw}(G)}n^{O(1)}$ algorithm of [52] 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 maximize (instead of minimizing) the number of connected components in the solution seem more difficult to solve than the problems of minimization (including problems where we demand that the solution forms a single connected component). As evidence we present lower bounds for the time complexity of solutions to such problems, proving that $c^{\mathbf{tw}(G)}$ solutions of these problems are unlikely:

Theorem 1.4. *Unless the Exponential Time Hypothesis is false, there does not exist a* $2^{o(p \log p)}|V|^{O(1)}$ *algorithm for solving* CYCLE PACKING *or* MAX CYCLE COVER. *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:

Theorem 1.5. *Unless the Exponential Time Hypothesis is false, there does not exist a* $2^{o(p \log p)}|V|^{O(1)}$ *algorithm for solving* MAXIMALLY DISCONNECTED DOMINATING SET. *The parameter p denotes the width of a given path decomposition of the input graph.*

A reader interested in just the basic workings of the Cut&Count technique will likely find the descriptions and the intuition needed in the first three sections of this work. The rest of the rather formidable volume is devoted to refined applications, in some cases needed to obtained the optimal constants, as well as arguments showing the aforementioned optimality, and can — in a sense — be considered "advanced material".

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 in a field of characteristic 2, which causes them to disappear. This line of reasoning goes back to Tutte [61], and was recently used by Björklund [6] and Björklund et. al [9].

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 problem from 1970 [34, 35] 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. [13]. 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 papers circumvent the problems stemming from the lack of singly exponential algorithms parametrized by treewidth for connectivity–type problems. For instance in the case of parametrized algorithms, sphere cuts [22, 24] (for planar and bounded genus graphs) and Catalan structures [23] (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 $2^{o(\mathbf{tw}(G) \log \mathbf{tw}(G))}$ algorithms was deemed to be more likely.

1.3 Consequences of the Cut&Count technique

As alredy mentioned, algorithms for graphs with a 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. To keep the volume of this paper manageable, we do not explore all possible applications, but only give sample applications in various directions.

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 Consequences for FPT algorithms

Let us recall the definition of the FEEDBACK VERTEX SET problem:

FEEDBACK VERTEX SET   **Parameter:** k
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 [42]. 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, 10, 16, 18, 26, 27, 33, 41, 53, 54]. The best — so far — result in this series is the $3.83^k k n^2$ result of Cao, Chen and Liu [14]. Our technique gives an improvement of their result:

Theorem 1.6. *There exists a Monte-Carlo algorithm solving the* FEEDBACK VERTEX SET *problem in 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* $1/2$.

We give similar improvements for CONNECTED VERTEX COVER (from the $2.4882^k n^{O(1)}$ of [5] to $2^k n^{O(1)}$) and CONNECTED FEEDBACK VERTEX SET (from the $46.2^k n^{O(1)}$ of [49] to $3^k n^{O(1)}$).

1.3.2 Parametrized algorithms for H-minor-free graphs

A large branch of applications of algorithms parametrized with treewidth is the bidimensionality theory, used to find subexponential algorithms for various problems in H-minor-free graphs. In this theory we use the celebrated minor theorem of Robertson and Seymour [57], which ensures that any H-minor-free graph either has treewidth bounded by $C\sqrt{k}$, or a $2\sqrt{k} \times 2\sqrt{k}$ lattice as a minor. In the latter case we are assumed to be able to answer the problem in question (for instance a $2\sqrt{k} \times 2\sqrt{k}$ lattice 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 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 algorithm 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 [23], and exploits the so called Catalan structures. The approach deals with most of the problems mentioned in our paper, 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 program on the obtained tree decomposition.

The situation is more complicated in the case of problems on directed graphs. A full equivalent of the bidimensionality theory is not developed for such problems, and only a few problems have subexponential parametrized algorithms available [2, 21]. One of the approaches is to mimic the bidimensionality approach, which again leads to solving a problem on a graph of bounded treewidth — such an approach is taken by Dorn et al. in [21] for MAXIMUM LEAF OUTBRANCHING to obtain a $2^{O(\sqrt{k}\log k)}$ algorithm. In this case, a straightforward substitution of our $6^{\mathbf{tw}(G)}|V|^{O(1)}$ algorithm for the dynamic algorithm used by Dorn et al. will give the following improvement:

Theorem 1.7. *There exists a Monte-Carlo algorithm solving the k-MAXIMUM LEAF OUTBRANCHING problem in $2^{O(\sqrt{k})}|V|^{O(1)}$ time for directed graphs for which the underlying undirected graph excludes a fixed graph H as a minor. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

1.3.3 Consequences for 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., [12]); however global problems such as the TRAVELLING SALESMAN PROBLEM and HAMILTONIAN CYCLE have also received considerable attention [8, 28, 32, 39].

Here the starting point is the following theorem by Fomin et al. [30]:

Theorem 1.8 (Fomin, Gaspers, Saurabh, Stepanov). *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 $i \in \{3, \dots, 5\}$ and $n_{\geq 6}$ is the number of vertices of degree at least 6.

This theorem is constructive, and the corresponding path decompostion (and, consequently, tree decomposition) can be found in polynomial time.

Combining this theorem with our results gives algorithms running in faster than 2^n for graphs of maximum degree 3, 4 and (in the case of the $3^{\mathbf{tw}(G)}$ and $4^{\mathbf{tw}(G)}$ algorithms) 5, as follows:

Corollary 1.9. *There exist randomized algorithms that solve the following problems:*

- STEINER TREE, FEEDBACK VERTEX SET *and* CONNECTED VERTEX COVER *in* $O(1.201^n)$ *time for cubic graphs,* $O(1.443^n)$ *time for graphs of maximum degree* 4 *and* $O(1.61^n)$ *time for graphs of maximum degree* 5*;*

- CONNECTED DOMINATING SET, CONNECTED ODD CYCLE TRANSVERSAL, CONNECTED FEEDBACK VERTEX SET, EXACT k-LEAF SPANNING TREE, MAXIMUM FULL DEGREE SPANNING TREE *and* GRAPH METRIC TRAVELLING SALESMAN PROBLEM *as well as undirected versions of* MIN CYCLE COVER, LONGEST PATH *and* LONGEST CYCLE *in* $O(1.26^n)$ *time for cubic graphs,* $O(1.588^n)$ *for graphs of maximum degree* 4 *and* $O(1.824^n)$ *for graphs of maximum degree* 5*;*

- *Directed versions of* MIN CYCLE COVER, LONGEST PATH *and* LONGEST CYCLE, *as well as for* EXACT k-LEAF OUTBRANCHING, *in* $O(1.349^n)$ *for cubic graphs and in* $O(1.818^n)$ *for graphs of maximum degree* 4.

All the aforementioned algorithms are Monte Carlo algorithms with false negatives.

The TRAVELLING SALESMAN PROBLEM in its full generality does not fall under the Cut&Count regime; however for graphs of degree four the $O(1.588^n)$ algorithm obtained for GRAPH METRIC TRAVELLING SALESMAN PROBLEM (which can easily be extended to the case of the TRAVELLING SALESMAN PROBLEM with polynomially bounded weights) is significantly faster than the best known algorithm for the case of unbounded weights of Gebauer [32], which runs in 1.733^n time. For the case of degree 5 (where we give an $O(1.824^n)$ algorithm for the TRAVELLING SALESMAN PROBLEM with polynomially bounded weights) the best known result in the general weight case is the $(2 - \varepsilon)^n$ algorithm of Björklund et al. [8]. It is worth noticing that in the case of cubic graphs we automatically obtain an algorithm for GRAPH METRIC TRAVELLING SALESMAN PROBLEM running in $2^{n/3+\varepsilon n}n^{O(1)}$ time, which coincides with the time complexity of the algorithm for TRAVELLING SALESMAN PROBLEM of Eppstein [28]. The currently fastest algorithm for TRAVELLING SALESMAN PROBLEM in cubic graphs is due to Iwama and Nakashima [39].

1.3.4 Consequences for exact algorithms on planar graphs

Here we begin with a consequence of the work of Fomin and Thilikos [31]:

Proposition 1.10. *For any planar graph G,* $\mathbf{tw}(G) + 1 \le \frac{3}{2}\sqrt{4.5n} \le 3.183\sqrt{n}$. *Moreover a tree decomposition of such width can be found in polynomial time.*

Using this we immediately obtain $c^{\sqrt{n}}$ algorithms for solving problems with a global constraint on planar graphs with good constants. For instance for the HAMILTONIAN CYCLE problem on planar graphs we obtain the following result:

Corollary 1.11. *There exists a randomized algorithm with false negatives solving* HAMILTONIAN CYCLE *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 so far was the $O(2^{6.903\sqrt{n}})$ of Bodlaender et al. [24].
Similarly, we obtain an $O(2^{6.366\sqrt{n}})$ algorithm for LONGEST CYCLE on planar graphs (compare to the $O(2^{7.223\sqrt{n}})$ of [24]), and — as in the previous subsections — well-behaved $c^{\sqrt{n}}$ algorithms for all mentioned problems.

1.4 Organization of the paper

In the introduction we present the contents of the paper. Subsection 1.1 states the main results obtained by us. After analyzing the connections to previous works in Subsection 1.2 we turn to giving sample consequences in Subsection 1.3. We finish the introduction by giving this outline.

Section 2 is devoted to presenting the background material for our algorithms. In particular in Subsection 2.2 we recall the notion of treewidth and dynamic programming on tree decompositions, while in Subsection 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. We go into all the details, as we aim to present not only the algorithms and

proofs, but also the intuition behind them. We use those intuitions in Section 4, where we give sketches of algorithms for all the other problems mentioned in Theorem 1.2.

In Section 5 we move to lower bounds. In Subsection 5.1 we present evidence that problems in which we maximize the number of connected components are unlikely to have $c^{\mathbf{tw}(G)}|V|^{O(1)}$ algorithms. In Subsection 5.2 we provide arguments that several of the algorithms provided in Section 3 have time complexity which is hard to improve. We finish the paper with a number of conclusions and open problems in Section 6

As the reader might have already noticed, there is a quite a large amount of material covered in this paper. To keep it readable a considerable number of proofs and analyses was postponed to the appendix. Thus, in Appendix A we give detailed descriptions of all the algorithms sketched in Section 4. In particular in Part A.1 we describe all the variants of the Fast Subset Convolution technique we use to decrease the constants in our algorithms. Appendix B is devoted to the proofs of the results announced in Subsection 1.3.1 — Theorem 1.6 and its analogues for CONNECTED VERTEX COVER and CONNECTED FEEDBACK VERTEX SET. In Appendix C we turn to the proofs of the lower bounds stated in Subsection 5.1, while Appendix D gives proofs for tight lower bounds stated in Subsection 5.2.

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 $X \subseteq V(G)$ by $G[X]$ we denote the subgraph induced by X. For an edge set $X \subseteq E$, we take $V(X)$ to denote the set of the endpoints of the edges of X, and by $G[X]$ — the subgraph (V, 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 $X \subseteq V(G)$ 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 $X \subseteq V$ we mean a pair (X_1, X_2), with $X_1 \cap X_2 = \emptyset$, $X_1 \cup X_2 = X$. 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 by $\deg(v)$. $\deg_H(v)$ denotes the degree of v in the subgraph H. 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 and $X \subseteq V$ or $X \subseteq E$, we denote the in- and out-degree of v in $G[X]$ by $\mathrm{indeg}_{G[X]}(v)$ and $\mathrm{outdeg}_{G[X]}(v)$ respectively.

For an edge $e = uv$ by subdividing it s times (for $s > 0$) we mean the following operation: (1) remove the edge e, (2) add s vertices $\{x_{e,1}, \ldots, x_{e,s}\}$, (3) add edges $\{ux_{e,1}, x_{e,1}x_{e,2}, \ldots, x_{e,k-1}x_{e,s}, x_{e,s}v\}$.

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 $\mathrm{cc}(G)$ denote the number of (weakly) connected components of G.

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.

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 $\omega : U \to \{1, \ldots, N\}$, 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 $s(v)$ is already defined or not.

2.2 Treewidth and pathwidth

2.2.1 Tree Decompositions

Definition 2.1 (Tree Decomposition, [56]). *A tree decomposition of a (undirected or directed) graph G is a tree $\mathbb{T}$ in which each vertex $x \in \mathbb{T}$ 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 $\mathbb{T}$.*

The *treewidth $tw(\mathbb{T})$* of a tree decomposition $\mathbb{T}$ is the size of the largest bag of $\mathbb{T}$ minus one, and the treewidth of a graph G is the minimum treewidth 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 [44]. 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 l, r, then $B_x = B_l = B_r$,*

- *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 z called the* root *with $B_z = \emptyset$ 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 l and r with $B_x = B_r = B_l$.*

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 [44] 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.

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 $v \in B_y$ (recall that x is its own descendant). We also associate with each bag x of $\mathbb{T}$ a subgraph of G as follows:

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

For an overview of tree decompositions and dynamic programming on tree decompositions see [11, 36].

2.2.2 Path Decompositions

A *path decomposition* is a tree decomposition that is a path. The pathwidth of a graph is the minimum width of all path decompositions. Path decompositions can, similarly as above, be transformed into nice path decompositions, these obviously contain no join bags.

2.3 Isolation lemma

An 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* $\omega(S') = \min_{S \in \mathcal{S}} \omega(S)$.

Recall that for $X \subseteq U$, $\omega(X)$ denotes $\sum_{u \in X} \omega(u)$.

Lemma 2.5 (Isolation Lemma, [51])**.** *Let* $\mathcal{F} \subseteq 2^U$ *be a set family over a universe* U *with* $|\mathcal{F}| > 0$. *For each* $u \in U$, *choose a weight* $\omega(u) \in \{1, 2, \ldots, N\}$ *uniformly and independently at random. Then*

$$\texttt{prob}[\omega \text{ isolates } \mathcal{F}] \geq 1 - \frac{|U|}{N}$$

It is worth mentioning that in [15], a lemma using less random bits is shown: If $|\mathcal{F}| \leq Z$, then a scheme using $O(\log |U| + \log Z)$ random bits to obtain a polynomially bounded (in unary) weight function that isolates any set system with high probability is presented.

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). This lemma has found many applications [51].

An alternative method to a similar end is obtained by using Polynomial Identity Testing [20, 58, 66] over a field of characteristic two. This second method has been already used in the field of exact and parameterized algorithms [6, 9, 45, 46, 64]. The two methods do not have differ much in their consequences: Both use the same number of random bits (the most randomness efficient algorithm are provided in [1, 15]). The challenge of giving a full derandomization seems to be equally difficult for both methods [3, 40]. The usage of the Isolation Lemma gives greater polynomial overheads, however we choose to use it because it requires less preliminary knowledge.

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 all the details in an expository manner, as we aim not only to show the solutions to these particular problems, but also to show the general workings.

The Cut&Count technique applies to problems with certain connectivity requirements. Let $\mathcal{S} \subseteq 2^U$ be a set of solutions; we aim to decide 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 connected candidate solutions. Furthermore, consider the set $\mathcal{C}$ of pairs (X, C) where $X \in \mathcal{R}$ and C is a consistent cut (to be defined later) 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 $\mathcal{C}_W = \{(X, C) \in \mathcal{C} | \omega(X) = W\}$. The general setup can thus be summarized as in Algorithm 1:

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

Corollary 3.1. *Let* $\mathcal{S} \subseteq 2^U$ *and* $\mathcal{C} \subseteq 2^{U \times (V \times V)}$. *Suppose that for every* $W \in \mathbb{Z}$:

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

2. $\texttt{CountC}(\omega, W, \mathbb{T}) \equiv |\{(X, C) \in \mathcal{C} | \omega(X) = W\}|$

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

Function `cutandcount(U, ℤ, CountC)`

Input Set U; nice tree decomposition $\mathbb{T}$; Procedure `CountC` accepting a $\omega : U \to \{1, \ldots, N\}$, $W \in \mathbb{Z}$ and $\mathbb{T}$.

1: **for** every $v \in U$ **do**
2:     Choose $\omega(v) \in \{1, \ldots, 2|U|\}$ uniformly at random.
3: **for** every $0 \le W \le 2|U|^2$ **do**
4:     **if** `CountC`$(\omega, W, \mathbb{T}) \equiv 1$ **then return yes**
5: **return no**

<center>**Algorithm 1:** `cutandcount(U, ℤ, CountC)`</center>

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 `CountC`. Usually, as we will see in the expositions of the applications, the count part requires more explanation. In the next two subsections, we illustrate both parts by giving two specific applications.

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?

The Cut part. Let us first consider the Cut part of the Cut&Count technique, and start by defining the objects we are going to count. Suppose we are given a weight function $\omega : V \to \{1, \ldots, N\}$. For any integer W, let $\mathcal{R}_W$ be the set of all such subsets X of V that $T \subseteq X$, $\omega(X) = W$ and $|X| = k$. Also, define $\mathcal{S}_W = \{X \in \mathcal{R}_W \mid G[X]$ is connected$\}$. The set $\bigcup_W \mathcal{S}_W$ is our set of solutions — if for any W this set is nonempty, our problem has a positive answer. The set $\mathcal{R}_W$ is the set of candidate solutions, where we relax the connectivity requirement. In this easy application the only requirement that remains is that the set of terminals is contained in the candidate solution.

Definition 3.2. *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 $G[X]$.*

Similarly 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 $(X, (X_1, X_2))$ such that (X_1, X_2) is a consistent cut of the underlying undirected graph of $D[X]$.

Let v_1 be an arbitrary terminal. Define $\mathcal{C}_W$ to be the set of all consistently cut subgraphs $(X, (X_1, X_2))$ such that $X \in \mathcal{R}_W$ and $v_1 \in X_1$.

Before we proceed with the Count part, let us state the following easy combinatorial identity:

Lemma 3.3. *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. $\qquad\square$

The Count part. For the Count part, the following lemma shows that the first condition of Corollary 3.1 is indeed met:

Lemma 3.4. *Let $G, \omega, \mathcal{C}_W$ and $\mathcal{S}_W$ be as defined above. Then for every W, $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$.*

Proof. Let us fix W and omit the subscripts accordingly. By Lemma 3.3, we know that $|\mathcal{C}| = \sum_{X \in \mathcal{R}} 2^{\mathtt{cc}(G[X])-1}$. Thus $|\mathcal{C}| \equiv \big|\{X \in \mathcal{R} \mid \mathtt{cc}(G[X]) = 1\}\big| = |\mathcal{S}|$. $\qquad\square$

<center>10</center>

Now the only missing ingredient left is the sub-procedure `CountC`. This sub-procedure, which counts the cardinality of $\mathcal{C}_W$ modulo 2, is a standard application of dynamic programming:

Lemma 3.5. *Given $G = (V, E)$, $T \subseteq V$, an integer k, $\omega : V \to \{1, \ldots, N\}$ and a nice tree decomposition $\mathbb{T}$, there exists an algorithm that can determine $|\mathcal{C}_W|$ modulo 2 for every $0 \leq W \leq kN$ 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 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. We now define "partial solutions": For every bag $x \in \mathbb{T}$, integers $0 \leq i \leq k$, $0 \leq w \leq kN$ and $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$ define

$$\mathcal{R}_x(i, w) = \left\{ X \subseteq V_x \big| \ (T \cap V_x) \subseteq X \ \wedge \ |X| = i \ \wedge \ \omega(X) = w \right\}$$

$$\mathcal{C}_x(i, w) = \Big\{ (X, (X_1, X_2)) \big| \ X \in \mathcal{R}_x(i, w) \ \wedge \ (X, (X_1, X_2)) \text{ is a consistently cut subgraph of } G_x$$

$$\wedge \ (v_1 \in V_x \Rightarrow v_1 \in X_1) \Big\}$$

$$A_x(i, w, s) = \left| \Big\{ (X, (X_1, X_2)) \in \mathcal{C}_x(i, w) \big| \ (s(v) = \mathbf{1}_j \Rightarrow v \in X_j) \ \wedge \ (s(v) = \mathbf{0} \Rightarrow v \notin X) \Big\} \right|$$

The intuition behind these definitions is as follows: the set $\mathcal{R}_x(i, w)$ contains all sets $X \subseteq V_x$ that could potentially be extended to a candidate solution from $\mathcal{R}$, subject to an additional restriction that the cardinality and weight of the partial solution are equal to i and w, respectively. Similarly, $\mathcal{C}_x(i, w)$ contains consistently cut subgraphs, which could potentially be extended to elements of $\mathcal{C}$, again with the cardinality and weight restrictions. The number $A_x(i, w, s)$ counts those elements of $\mathcal{C}_x(i, w)$ 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

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

— the various choices of s describe all possible intersections of an element of $\mathcal{C}$ with B_x. Observe that since we are interested in values $|\mathcal{C}_W|$ 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 $|\mathcal{C}_W| = |\mathcal{C}_r(k, W)|$.

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.

- **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:**

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

    For the first case note that by definition v can not be coloured $\mathbf{0}$ if it is a terminal. For the other cases, the accumulators 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:**

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

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

11

- **Forget vertex** v **bag** x:

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

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

- **Join bag**:

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

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. Note that as we perform all calculations modulo 2, we take only constant time to perform any arithmetic operation. $\square$

We conclude this section with the following theorem.

Theorem 3.6. *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. Run Algorithm 1 by setting $U = V$, and CountC to be the algorithm implied by Lemma 3.5. The correctness follows from Corollary 3.1 by setting $\mathcal{S} = \bigcup_W \mathcal{S}_W$ and $\mathcal{C} = \bigcup_W \mathcal{C}_W$ and Lemma 3.4. It is easy to see that the timebound follows from Lemma 3.5. $\square$

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 contains pairs (X, M), where $X \subseteq A$ is a cycle cover and $M \subseteq V, |M| = k$ is a set of *marked* vertices such that each cycle in X contains at least one marked vertex. Observe that since $|M| = k$ this ensures that in the set of solutions in each pair (X, M) the cycle cover X contains at most k cycles. Note that two different sets of marked vertices of a single cycle cover are considered to be two *different solutions*. For this reason we assign random weights *both to the arcs and vertices of D*. When we relax the requirement that in the pair (X, M) each cycle in X contains at least one vertex 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 $M \subseteq V$ is a set of k markers, (ii) a cut consistent with $D[X]$, where all the marked vertices M are on the left side of the cut. We will see that candidate solutions that contain a cycle without any marked vertex cancel modulo 2. Formal definition follows.

The Cut part. As said before, we assume that we are given a weight function $\omega : A \cup V \to \{1, \ldots, N\}$, where $N = 2|U| = 2(|A| + |V|)$.

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., $\text{outdeg}_X(v) = \text{indeg}_X(v) = 1$ for every vertex $v \in V$; $M \subseteq V, |M| = k$ and $\omega(X \cup M) = W$;*

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 vertex from the set M;

3. $\mathcal{C}_W$ as all pairs $((X, M), (V_1, V_2))$ such that:

$$(X, M) \in \mathcal{R}_W, \qquad (V_1, V_2) \text{ is a consistent cut of } D[X], \quad \text{and} \quad M \subseteq V_1.$$

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 $D, \omega, \mathcal{C}_W$ and $\mathcal{S}_W$ be defined as above. Then for every W, $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$.*

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

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

To see this, note that for any $((X, M), (V_1, V_2)) \in \mathcal{C}_W$ and any vertex set C of a cycle from X such that $M \cap C = \emptyset$, we have $((X, M), (V_1 \triangle C, V_2 \triangle C)) \in \mathcal{C}_W$ — 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 $\mathcal{C}_W$. Hence (analogously to Lemma 3.3) for any W and $(M, X) \in \mathcal{R}_W$ there are $2^{\mathtt{cc}(M, X)}$ cuts (V_1, V_2) such that $((X, M), (V_1, V_2)) \in \mathcal{C}_W$ and the lemma follows, because:

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

$\square$

Lemma 3.9. *Given $D = (V, A)$, an integer k, a weight function $\omega : A \cup V \to \{1, \ldots, N\}$ and a nice tree decomposition $\mathbb{T}$, there is an algorithm that can determine $|\mathcal{C}_W|$ modulo 2 for every $0 \leq W \leq (k + |V|)N$ in $6^t N^2 |V|^{O(1)}$ time.*

Proof sketch. We briefly sketch a $64^t N^2 |V|^{O(1)}$ time algorithm, whereas the $6^t N^2 |V|^{O(1)}$ time algorithm can be found in Appendix A.4.

Let $s \in \{00_1, 00_2, 01_1, 01_2, 10_1, 10_2, 11_1, 11_2\}^{B_x}$, and for a bag $x \in \mathbb{T}$ and integers i, w let $A_x(i, w, s)$ be the number of pairs $((X, M), (X_1, X_2))$ such that:

- $X \subseteq E_x$, i.e., X is a subset of the set of arcs introduced by x and its descendants,

- for every $v \in B_x$ and every $\mathbf{i}, \mathbf{o} \in \{0, 1\}$ we have $s(v) = \mathbf{io}_j \Rightarrow \mathrm{indeg}_X(v) = \mathbf{i} \wedge \mathrm{outdeg}_X(v) = \mathbf{o} \wedge v \in X_j$,

- for every $v \in V_x \setminus B_x$ we have $\mathrm{indeg}_X(v) = \mathrm{outdeg}_X(v) = 1$.

- (X_1, X_2) is a consistent cut of the graph (V_x, X),

- $M \subseteq (X_1 \setminus B_x)$, $\omega(X) + \omega(M) = w$ and $|M| = i$.

To obtain all values $|\mathcal{C}_W| \mod 2$ it is enough to compute $A_r(k, W, \emptyset)$ modulo two for all values of W, since $|\mathcal{C}_W| \equiv A_r(k, W, \emptyset)$.

Note that in the colouring we do not store the information whether a vertex is a marker or not. This is due to the following observation: Since the tree decomposition $\mathbb{T}$ is rooted in an empty bag, for each each vertex $v \in V$ there exists exactly one bag of the tree decomposition which forgets v. Hence if in the forget v bag we have $s(v) = 11_1$ we have an option of making v a marker and updating the accumulator i.

The running time $64^t N^2 |V|^{O(1)}$ can be obtained by using standard dynamic programming by using $A_x(i, w, s)$ as the table. The $64 = 8^2$ comes from the join bags — the naive way to calculate the values of $A_x(i, w, s)$ for a join bag x would be to iterate over all pairs choices of (i, w, s) for the child bags, and there are $|V|^{O(1)} N^2 64^t$ such choices to

consider. In order to obtain the claimed time complexity we need to reduce the number of states per vertex to six and to handle the join bags more efficiently.

To achieve these goals for vertices with colours **00** and **11** we do not specify the side of the cut and we use variants of the fast subset convolution [7] algorithm. The details can be found in Appendix A.4. □

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. Run algorithm Algorithm 1 by setting $U = A \cup V$ and CountC to be the algorithm implied by Lemma 3.9. The correctness follows from Corollary 3.1 by setting $\mathcal{S} = \bigcup_W \mathcal{S}_W$ and $\mathcal{C} = \bigcup_W \mathcal{C}_W$ and Lemma 3.8. It is easy to see that the timebound follows from Lemma 3.9. □

4 Applications of the technique for other problems

We now proceed to sketch $|V|^{O(1)}c^{\mathbf{tw}(G)}$ algorithms for other problems mentioned in the introduction. For the sake of brevity we present only quick sketches here: for each problem we define what set of solution candidates do we consider; which candidate-cut pairs we count; if necessary we argue why the non-connected candidates are counted an even number of times, while the connected candidates are counted only once; and finally we describe what states do we consider for a given bag and a given weight-sum in the dynamic programming subroutine. We also briefly mention what techniques do we use to compute the values of the dynamic programming in the join bags, as this is the most non-trivial bag to compute efficiently.

For full descriptions of the aforementioned algorithms we refer the reader to Appendix A.

4.1 FEEDBACK VERTEX SET

FEEDBACK VERTEX SET
Input: An undirected graph $G = (V, E)$ and an integer k.
Question: Does there exist a set $Y \subseteq V$ of cardinality k so that $G[V \setminus Y]$ is a forest?

Here defining the families $\mathcal{R}$ and $\mathcal{S}$ 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 4.1. *A graph with n vertices and m edges is a forest iff it has at most $n - m$ connected components.*

The simple proof is given in the appendix.

Thus, we ensure that a solution is a forest by counting the vertices and edges, and ensuring the number of connected components is bounded from above by using markers, as in Section 3.2.

Note that here we want to keep track of two distinct vertex sets — M and X. We thus set $U = V \times \{\mathbf{F}, \mathbf{M}\}$, i.e., for each vertex $v \in V$ we choose two weights: $\omega((v, \mathbf{F}))$ and $\omega((v, \mathbf{M}))$, one for $v \in X$, and the second one for $v \in M$ (the details of this are presented in the appendix).

The family of solution candidates is defined (as in the MIN CYCLE COVER case, where we also used markers) as the family of pairs (X, M) such that $M \subseteq X \subseteq V$. We use accumulators to keep track of the size of X, the number of edges in $G[X]$, the size of M and the total weight of $(X \times \{\mathbf{F}\}) \cup (M \times \{\mathbf{M}\})$. The solution is a solution candidate (X, M) with the additional properties that $G[X]$ is a forest and each connected component of $G[X]$ contains at least one vertex from M.

With a pair (X, M) we associate consistent cuts of $G[X]$ with all vertices from M being on the left side of the cut.

In this case a pair (X, M) is consistent with exactly one cut iff every connected component of $G[X]$ contains at least one marker; on the other hand if there are cc components containing no markers, we have $2^{\mathbf{cc}}$ consistent cuts.

For each bag we keep states with the following parameters:

- the number of vertices chosen to be in X ($|V| + 1$ possiblities);

- the number of already introduced edges in $G[X]$ ($|V|$ possibilities, we can discard solution candidates with at least $|V|$ edges in $G[X]$);

- the number of markers already chosen ($|V| + 1$ possibilities);

- the sum of the appropriate weights of vertices already in X and already in M;

- for each vertex in the bag one of the three possible colours: either it is not in X, or it is in X and on the left, or in X and on the right ($3^{\mathbf{tw}(G)}$ possibilities in total).

Note that (as in Section 3.2) we do not remember whether a vertex is a marker, instead simply making the choice in the appropriate forget bag.

Our algorithm answers "yes" if for some integer m and for some weight sum W there is an odd number of consistent pairs with $|V| - k$ vertices, m edges, $|V| - k - m$ markers. The join bag is trivial in this case — the choices for each vertex have to match exactly.

The details of the algorithm are given in Appendix A.2.

4.2 CONNECTED VERTEX COVER

CONNECTED VERTEX COVER
Input: An undirected graph $G = (V, E)$ and an integer k.
Question: Does there exist such a connected set $X \subseteq V$ of cardinality k that each edge is incident to at least one vertex from X?

We choose one vertex v_1 which we assume to be in the solution (we can check just two choices, using two endpoints of some edge). As we choose a vertex set, we randomly select a weight function $\omega : V \to \{1, \ldots, N\}$. The family of solution candidates $\mathcal{R}_W$ consists of vertex covers of size k with $\omega(X) = W$ and containing v_1. The family of solutions $\mathcal{S}_W$ contains elements of $\mathcal{R}_W$ that induce a connected subgraph. As in the STEINER TREE problem, we take $\mathcal{C}_W$ to be the family of pairs (X, C), with $X \in \mathcal{R}_W$ and C being a consistent cut of $G[X]$, with v_1 on the left side of the cut (to break the symmetry).

In the dynamic programming subroutine, for each bag we keep states with the following parameters:

- the number of vertices already chosen to be in X ($k + 1$ possibilities);

- the sum of the weights of those vertices;

- for each vertex one of the three states: in X and on the left, in X and on the right, or not in X ($3^{\mathbf{tw}(G)}$ possibilities in total).

The vertex cover condition is checked in the introduce edge bags. We need no tricks in the join bag, the states of all the vertices need to match.

The details of the algorithm are given in Appendix A.3.

4.3 CONNECTED DOMINATING SET

CONNECTED DOMINATING SET
Input: An undirected graph $G = (V, E)$ and an integer k.
Question: Does there exist such a connected set $X \subseteq V$ of cardinality at most k that $N[X] = V$?

The reasoning here matches the one for the CONNECTED VERTEX COVER problem almost exactly. We fix some vertex v_1, which we require to be a part of the solution (to obtain a general algorithm we iterate over all possible choices of v_1). As we choose a vertex set, we randomly select a weight function $\omega : V \to \{1, \ldots, N\}$. The family $\mathcal{R}_W$ is the family of dominating sets X in G containing v_1 of size k and weight W, while $\mathcal{S}_W$ is the family of those sets $X \in \mathcal{R}_W$ for which $G[X]$ is connected. The family $\mathcal{C}_W$ is defined as previously.

For each bag we keep states with the following parameters:

- the number of vertices already chosen to be in X ($k + 1$ possibilities);

- the sum of the weights of all vertices already in X ($Nk + 1$ possibilities);

- for each vertex in the bag one of the four possible states: in X and on the left, in X and on the right, not in X and adjacent to a vertex already in X, or not in X and not adjacent to a vertex already in X ($4^{\mathbf{tw}(G)}$ possibilities in total).

In the join bag the states of the vertices in X have to match, while the states of the vertices not in X are joined using a standard Fast Subset Convolution procedure. The details are given in Appendix A.3.

4.4 CONNECTED ODD CYCLE TRANSVERSAL

> CONNECTED ODD CYCLE TRANSVERSAL
> **Input:** An undirected graph $G = (V, E)$ and an integer k.
> **Question:** Does there exist such a connected set $X \subseteq V$ of cardinality k that $G[V \setminus X]$ is bipartite?

The reasoning here matches the one for the previous two problems almost exactly. We choose one vertex v_1 which we assume to be in the solution. The only catch is that the naive dynamic programming that solves ODD CYCLE TRANSVERSAL partitions the vertex set into three (not two) sets: the odd cycle transversal and the bipartion of the resulting bipartite graph. Thus $\mathcal{R}_W$ is the family of such partitions of V, i.e., for one odd cycle transversal X, different bipartitions of $G[V \setminus X]$ result in different solution candidates. The integer k given in the input represents the size of the odd cycle transversal, whereas the index W represents the weight of the partition. As each vertex is in one of three sets in an element of $\mathcal{R}_W$, we need to generate two weights per vertex, so that each element of $\mathcal{R}_W$ corresponds to a different subset of the domain of the weight function ω. As previously, $\mathcal{S}_W$ are the elements of $\mathcal{R}_W$ where the odd cycle transversal is connected, and in $\mathcal{C}_W$ we pair up candidate solutions with cuts consistent with the odd cycle transversal where v_1 is on a fixed side of the cut.

For each bag we keep states with the following parameters:

- the number of vertices already chosen to be in X ($k + 1$ possibilities);

- the weight of the partition;

- for each vertex one of the four states: in X and on the left, in X and on the right, or in one of two colour classes of $G[V \setminus X]$ ($4^{\mathbf{tw}(G)}$ possibilities in total).

The condition whether the chosen colour classes are correct is checked in the introduce edge bags. We need no tricks in the join bag, the states of all the vertices need to match. The details are given in Appendix A.3.

4.5 CONNECTED FEEDBACK VERTEX SET

> CONNECTED FEEDBACK VERTEX SET
> **Input:** An undirected graph $G = (V, E)$ and an integer k.
> **Question:** Does there exist a set $Y \subseteq V$ of cardinality k so that $G[Y]$ is connected and $G[V \setminus Y]$ is a forest?

Here we use the same approach as in the previous three algorithms: we take an algorithm for FEEDBACK VERTEX SET and add cuts consistent with the solution. However, now the base algorithm is not that easy, because it is the one described in Section 4.1, already using the Cut&Count technique. We thus need to apply Cut&Count twice here, but — as we will see — there are no significant difficulties.

As before, we fix one vertex v_1 to be included in the connected feedback vertex set Y.

As in the FEEDBACK VERTEX SET algorithm, we ensure that a solution induces a forest by counting vertices and edges, and ensuring the number of connected components is bounded from above by using markers, as in Section 3.2.

As we want to keep track of two distinct vertex sets — M and X, we set $U = V \times \{\mathbf{F}, \mathbf{M}\}$, i.e., for each vertex $v \in V$ choose two weights $\omega((v, \mathbf{F}))$ and $\omega((v, \mathbf{M}))$ one for $v \in X$, and the second one for $v \in M$.

The family of solution candidates is defined just as in the FEEDBACK VERTEX SET case: as the family of pairs (X, M) such that $M \subseteq X \subseteq V$, with accumulators keeping track of the size of X, the number of edges in $G[X]$, the size of M and the total weight of $X \times \{\mathbf{F}\} \cup M \times \{\mathbf{M}\}$. The solution is a solution candidate (X, M) with the additional properties that $G[X]$ is a forest, each connected component of $G[X]$ contains at least one vertex from M and, additionally to the FEEDBACK VERTEX SET case, that $G[V \setminus X]$ is connected.

With a pair (X, M) we associate two consistent cuts: one of $G[X]$ with all vertices from M being on the left side of the cut, and one of $G[V \setminus X]$, where v_1 is on the left side of the cut.

In this case a pair (X, M) is consistent with exactly one cut of $G[X]$ iff every connected component of $G[X]$ contains at least one marker; on the other hand if there are c components containing no markers, we have 2^c consistent cuts. Moreover, a pair (X, M) is consistent with $2^{\mathbf{cc}(G[V \setminus X]) - 1}$ cuts of $G[V \setminus X]$. Thus, a pair (X, M) is counted only once if it is a solution, and an even number of times otherwise.

The dynamic programming proceeds exactly as in the FEEDBACK VERTEX SET case. The details are given in Appendix A.3.

4.6 UNDIRECTED MIN CYCLE COVER

We use similar approach as for the directed case in Section 3.2. However in the undirected case the number of states is decreased because instead of keeping track of both indegree and outdegree of a vertex we handle only a single degree. Similarly as for the directed case we do not store the side of the cut for vertices of degree zero and two, which gives exactly 4 states per vertex and leads to $4^{\mathbf{tw}(G)} |V|^{O(1)}$ time complexity.

In the join bag we need to use a variant of the Fast Fourier Transform. Details can be found in Appendix A.4.

4.7 (DIRECTED) LONGEST CYCLE and (DIRECTED) LONGEST PATH

(DIRECTED) LONGEST CYCLE
Input: An undirected graph $G = (V, E)$ (or a directed graph $D = (V, A)$) and an integer k.
Question: Does there exist a (directed) simple cycle of length k in G (D)?

(DIRECTED) LONGEST PATH
Input: An undirected graph $G = (V, E)$ (or a directed graph $D = (V, A)$) and an integer k.
Question: Does there exist a (directed) simple path of length k in G (D)?

Obviously an algorithm for LONGEST CYCLE implies an algorithm of the same time complexity for the HAMILTONIAN CYCLE problem. Moreover in Appendix A.4 we show that the LONGEST PATH problem both in the directed and undirected case may be reduced to the appropriate variant of the LONGEST CYCLE problem.

Observe that for the (DIRECTED) LONGEST CYCLE problem we can mimic the algorithm for the (DIRECTED) MIN CYCLE COVER problem. It is even easier because we are looking for a connected object which means that we do not have to use markers. The only difference between (DIRECTED) LONGEST CYCLE problem and (DIRECTED) MIN CYCLE COVER is that in the (DIRECTED) LONGEST CYCLE problem we need to count the number of chosen edges, since we allow vertices of degree zero.

Details can be found in Appendix A.4.

4.8 EXACT k-LEAF SPANNING TREE and EXACT k-LEAF OUTBRANCHING

EXACT k-LEAF SPANNING TREE
Input: An undirected graph $G = (V, E)$ and an integer k.
Question: Does there exists a spanning tree of G with exactly k leaves?

EXACT k-LEAF OUTBRANCHING
Input: A directed graph $D = (V, A)$ and an integer k, and a root $r \in V$.
Question: Does there exist a spanning tree of D with all edges directed away from the root with exactly k leaves?

The above problems generalize the following problems: MAXIMUM LEAF TREE, MINIMUM LEAF TREE, MAXIMUM LEAF OUTBRANCHING and MINIMUM LEAF OUTBRANCHING, which ask for the number of leaves to be at least k or at most k.

In this subsection we only sketch a natural $8^{\mathbf{tw}(G)}|V|^{O(1)}$ solution to the EXACT k-LEAF OUTBRANCHING problem since it generalizes the EXACT k-LEAF SPANNING TREE problem (simply direct each edge in both directions and add a root r with a single outgoing arc). This can be improved to $4^{\mathbf{tw}(G)}|V|^{O(1)}$ for EXACT k-LEAF SPANNING TREE and to $6^{\mathbf{tw}(G)}|V|^{O(1)}$ for EXACT k-LEAF OUTBRANCHING using a less intuitive definition of solution candidates together with a binomial transform for join bags, which is described in Appendix A.5.

As we choose an arc set, we randomly select a weight function $\omega : A \rightarrow \{1, \ldots, N\}$. The family of solution candidates $\mathcal{R}_W$ consists of sets of exactly $|V| - 1$ arcs of total weight W, such that exactly k vertices have no outgoing arc, each vertex except the root has exactly one incoming arc and the root r has no incoming arc. $\mathcal{S}_W$ are the solution candidates $X \in \mathcal{R}_W$ such that the underlying undirected graph of $D[X]$ is connected, which is equivalent to X being an outbranching. We take $\mathcal{C}_W$ to be the family of pairs (X, C), with $X \in \mathcal{R}_W$ and C being a consistent cut of the underlying undirected graph of $D[X]$.

For each bag we keep states with the following parameters:

- the number of already chosen arcs,

- the sum of the weights of those arcs,

- the number of already forgotten vertices with no outgoing arcs,

- for each vertex one of eight states, denoting the side of the cut (two possibilities), the indegree (0 or 1, two possibilities) and whether we have already chosen some outgoing arc from this vertex (two possibilities).

For the merging of states we would have to use the Fast Subset Convolution algorithm (details in Appendix A.5).

4.9   MAXIMUM FULL DEGREE SPANNING TREE

MAXIMUM FULL DEGREE SPANNING TREE
Input: An undirected graph $G = (V, E)$ and an integer k.
Question: Does there exist a spanning tree T of G for which there are at least k vertices satisfying $\deg_G(v) = \deg_T(v)$?

A solution is any set $X \subseteq E$ with the following properties:

- $|X| = |V| - 1$;

- there are exactly k vertices v for which $\deg_G(v) = \deg_{G[X]}(v)$;

- $G[X]$ is connected (as we have $|V| - 1$ edges, this is equivalent to $G[X]$ being a tree).

We define solution candidates and the pairs to count as usual, i.e., we count consistent cuts of $G[X]$. Note that we cut the whole set V (instead of only cutting the vertices incident to some edge of X — as we want to assure that all the vertices are connected, and not only $G[V(X)]$).

In each bag we parametrize the states as follows:

- the number of edges already chosen to be in X (V possibilities);

- the sum of their weights ($N|V|$ possibilities);

- the number of already forgotten vertices satisfying $\deg_G(v) = \deg_{G[X]}(v)$ ($k + 1$ possibilities);

- for each vertex in the bag, we remember on which side of the cut it is, and whether there was an introduced edge incident to this vertex which was not chosen to be included in X ($4^{\mathbf{tw}(G)}$ possibilities in total).

In the join bag we use the standard Fast Subset Convolution algorithm. For a precise description, see Appendix A.6.

4.10 GRAPH METRIC TRAVELLING SALESMAN PROBLEM

> GRAPH METRIC TRAVELLING SALESMAN PROBLEM
> **Input:** An undirected graph $G = (V, E)$ and an integer k.
> **Question:** Does there exist a closed walk (possibly repeating edges and vertices) of length at most k that visits each vertex of the graph at least once?

Note that the existence of such a cycle is equivalent to the existence of a multisubset X of edges, which is Eulerian (that is (V, X) is connected and the degree of each vertex is even). In particular this means each edge can be assumed to occur in X at most twice (otherwise we could remove two copies of this edge).

For each edge we have to decide whether we choose it twice, once or not at all. To avoid dependency problems, we assign two different independent weights to an edge, and add the first to the some if the edge is taken once, and the second if it is taken twice. Note that we also needed two types of weights in the FEEDBACK VERTEX SET problem where have weights for chosen vertices and for markers.

A solution candidate is a multiset of edges with each edge taken at most twice, the total cardinality not exceeding k, and each vertex having an even degree. The family of solutions consists of such solution candidates X that $G[X]$ is connected. We choose the cuts we count with an edge as usual, i.e., we count consistent cuts of $G[X]$. As usual we pick one vertex to be always on the left side of the cut.

In each bag, in addition to the number of edges already chosen and the sum of their weights, we keep for each vertex the side of the cut and the parity of the number of already chosen edges incident to this vertex ($4^{\mathbf{tw}(G)}$ possibilites in total). The values in the states of the join bag are calculated using the Hadamard transform (details in Appendix A.7).

5 Lower bounds

In this section we describe a bunch of negative results concerning the possible time complexities for algorithms for connectivity problems parameterized by treewidth or pathwidth. Our goal is to complement our positive results by showing that in some situations the known algorithms (including ours) probably cannot be further improved.

First, let us introduce the complexity assumptions made in this section. Let c_k be the infimum of the set of the positive reals c that satisfy the following condition: there exists an algorithm that solves k-SAT in time $O(2^{cn})$, where n denotes the number of variables in the input formula. The Exponential Time Hypothesis (ETH for short) asserts that $c_3 > 0$, whereas the Strong Exponential Time Hypothesis (SETH) asserts that $\lim_{k \to \infty} c_k = 1$. It is well known that SETH implies ETH [38].

The lower bounds presented below are of two different types. In Section 5.1 we discuss several problems that, assuming ETH, do not admit an algorithm running in time $2^{o(p \log p)} n^{O(1)}$, where p denotes the pathwidth of the input graph. In Section 5.2 we state that, assuming SETH, the base of the exponent in our algorithms for CONNECTED VERTEX COVER, CONNECTED DOMINATING SET, CONNECTED FEEDBACK VERTEX SET, CONNECTED ODD CYCLE TRANSVERSAL, FEEDBACK VERTEX SET, STEINER TREE and EXACT k-LEAF SPANNING TREE cannot be improved further. All proofs are postponed to Appendices C and D respectively.

5.1 Lower bounds assuming ETH

In Section 4 we have shown that a lot of well-known algorithms running in $2^{O(t)} n^{O(1)}$ time can be turned into algorithms that keep track of the connectivity issues, with only small loss in the base of the exponent. The problems solved in that manner include CONNECTED VERTEX COVER, CONNECTED DOMINATING SET, CONNECTED FEEDBACK VERTEX SET and CONNECTED ODD CYCLE TRANSVERSAL. Note that using the markers technique introduced in Section 3.2 we can solve similarly the following artificial generalizations: given a graph G and an integer r, what is the minimum size of a vertex cover (dominating set, feedback vertex set, odd cycle transversal) that induces **at most** r connected components?

We provide evidence that problems in which we would ask to maximize (instead of minimizing) the number of connected components are harder: they probably do not admit algorithms running in time $2^{o(p \log p)} n^{O(1)}$, where p denotes the pathwidth of the input graph. More precisely, we show that assuming ETH there do not exist algorithms

for CYCLE PACKING, MAX CYCLE COVER and MAXIMALLY DISCONNECTED DOMINATING SET running in time $2^{o(p \log p)} n^{O(1)}$.

Let us start with formal problem definitions. The first two problems have undirected and directed versions.

CYCLE PACKING
Input: A (directed or undirected) graph $G = (V, E)$ and an integer ℓ
Question: Does G contain ℓ vertex-disjoint cycles?

MAX CYCLE COVER
Input: A (directed or undirected) graph $G = (V, E)$ and an integer ℓ
Question: Does G contain a set of at least ℓ vertex-disjoint cycles such that each vertex of G is on exactly one cycle?

The third problem is an artificial problem defined by us that should be compared to CONNECTED DOMINATING SET.

MAXIMALLY DISCONNECTED DOMINATING SET
Input: An undirected graph $G = (V, E)$ and integers ℓ and r.
Question: Does G contain a dominating set of size at most ℓ that induces **at least** r connected components?

We prove the following theorem:

Theorem 5.1. *Assuming ETH, there is no $2^{o(p \log p)} n^{O(1)}$ time algorithm for* CYCLE PACKING, MAX CYCLE COVER *(both in the directed and undirected setting) nor for* MAXIMALLY DISCONNECTED DOMINATING SET. *The parameter p denotes the width of a given path decomposition of the input graph.*

The proofs go along the framework introduced by Lokshtanov et al. [48]. We start our reduction from the $k \times k$ HITTING SET problem. By $[k]$ we denote $\{1, 2, \ldots, k\}$. In the set $[k] \times [k]$ *a row* is a set $\{i\} \times [k]$ and *a column* is a set $[k] \times \{i\}$ (for some $i \in [k]$).

$k \times k$ HITTING SET
Input: A family of sets $S_1, S_2 \ldots S_m \subseteq [k] \times [k]$, such that each set contains at most one element from each row of $[k] \times [k]$.
Question: Is there a set S containing exactly one element from each row such that $S \cap S_i \neq \emptyset$ for any $1 \leq i \leq m$?

We also consider a permutation version of this problem, $k \times k$ PERMUTATION HITTING SET, where the solution S is also required to contain exactly one vertex from each column.

Theorem 5.2 ([48], Theorem 2.4). *Assuming ETH, there is no $2^{o(k \log k)} n^{O(1)}$ time algorithm for $k \times k$ HITTING SET nor for $k \times k$ PERMUTATION HITTING SET*

Note that in [48] the statement of the above theorem only includes $k \times k$ HITTING SET. However, the proof in [48] works for the permutation variant as well without any modifications.

All proofs can be found in Section C. We first prove the bound for MAXIMALLY DISCONNECTED DOMINATING SET by a simple reduction from $k \times k$ HITTING SET. Next we provide a more involved reduction from $k \times k$ PERMUTATION HITTING SET to undirected CYCLE PACKING. Finally, using rather elementary gadgets, we reduce undirected CYCLE PACKING to the directed case and to both cases of MAX CYCLE COVER.

5.2   Lower bounds assuming SETH

Following the framework introduced by Lokshtanov et al. [47], we prove that an improvement in the base of the exponent in a number of our algorithms would contradict SETH. Formally, we prove the following theorem.

Theorem 5.3. *Unless the Strong Exponential Time Hypothesis is false, there do not exist a constant $\varepsilon > 0$ and an algorithm that given an instance $(G = (V, E), k)$ or $(G = (V, E), T, k)$ together with a path decomposition of the graph G of width p solves one of the following problems:*

1. CONNECTED VERTEX COVER *in $(3 - \varepsilon)^p |V|^{O(1)}$ time,*

2. CONNECTED DOMINATING SET *in* $(4 - \varepsilon)^p |V|^{O(1)}$ *time,*

3. CONNECTED FEEDBACK VERTEX SET *in* $(4 - \varepsilon)^p |V|^{O(1)}$ *time,*

4. CONNECTED ODD CYCLE TRANSVERSAL *in* $(4 - \varepsilon)^p |V|^{O(1)}$ *time,*

5. FEEDBACK VERTEX SET *in* $(3 - \varepsilon)^p |V|^{O(1)}$ *time,*

6. STEINER TREE *in* $(3 - \varepsilon)^p |V|^{O(1)}$ *time,*

7. EXACT k-LEAF SPANNING TREE *in* $(4 - \varepsilon)^p |V|^{O(1)}$ *time.*

Note that VERTEX COVER (without a connectivity requirement) admits a $2^t |V|^{O(1)}$ algorithm whereas DOMINATING SET, FEEDBACK VERTEX SET and ODD CYCLE TRANSVERSAL admit $3^t |V|^{O(1)}$ algorithms and those algorithms are optimal (assuming SETH) [47]. To use the Cut&Count technique for the connected versions of these problems we need to increase the base of the exponent by one to keep the side of the cut for vertices in the solution. Theorem 5.3 shows that this is not an artifact of the Cut&Count technique, but rather an intrinsic characteristic of these problems.

In Appendix D we provide reductions in spirit of [47] that prove the first five lower bounds in Theorem 5.3. The result for EXACT k-LEAF SPANNING TREE is immediate from the result for CONNECTED DOMINATING SET, as CONNECTED DOMINATING SET is equivalent to MAXIMUM LEAF TREE [25]. The result for STEINER TREE follows from the simple observation that a CONNECTED VERTEX COVER instance can be turned into a STEINER TREE instance by subdividing each edge with a terminal.

6 Concluding remarks

The main consequence of the Cut&Count technique as presented in this work could (informally) be stated as the following rule of thumb:

Rule of thumb. *Suppose we are given a graph $G = (V, E)$, a tree decomposition $\mathbb{T}$ of G, and implicitly a set family $\mathcal{F}$ of subgraphs of G. Moreover, suppose that for a bag $x \in \mathbb{T}$ the behaviour of a partial subgraph $S \cap G_x$ depends only on a (small) interface $I(S, x)$ with bag V_x. Let $\theta = \max_{x \in \mathbb{T}} |\{I(S, x) : S \in \mathcal{F}\}|$. Then we can compute $\min_{S \in \mathcal{F}} \mathrm{cc}(S)$ in $(\theta |V|)^{O(1)}$ time.*

To clarify, note that a standard dynamic programming for determining whether $\mathcal{F}$ is empty or not runs in $(\theta |V|)^{O(1)}$ time, and usually even in $\theta |V|^{O(1)}$ time. In fact, the dominant term $\theta^{O(1)}$ in the claimed running time is a rather cruel upper bound for the Cut&Count technique as well, as for many problems we can do a lot better. If $\theta = c^t |V|^{O(1)}$ with t being the treewidth of $\mathbb{T}$, then for many instances of the above we have shown solutions running in $\theta = (c+c')^t |V|^{O(1)}$ time, where c' is, intuitively, the number of states affected by the cuts of the Cut&Count technique. Moreover, many problems cannot be solved faster unless the Strong Exponential Time Hypothesis fails. We have chosen not to pursue a general theorem in the above spirit, as the techniques required to get optimal constants seem varied and depend on the particular problem.

We have also shown that several problems in which one aims to maximize the number of connected components are not solvable in $2^{o(p \log p)} |V|^{O(1)}$ unless the Exponential Time Hypothesis fails. Hence, assuming the Exponential Time Hypothesis, there is a marked difference between the minimization and maximization of the number of connected components in this context.

Finally, we leave the reader with some interesting open questions:

- Can Cut&Count be derandomized? For example, can CONNECTED VERTEX COVER be solved deterministically in $c^t |V|^{O(1)}$ on graphs of treewidth t for some constant c?

- Since general derandomization seems hard, we ask whether it is possible to derandomize the presented FPT algorithms parameterized by the solution size for FEEDBACK VERTEX SET, CONNECTED VERTEX COVER or CONNECTED FEEDBACK VERTEX SET? Note that the tree decomposition considered in these algorithms is of a very specific type, which could potentially make this problem easier than the previous one.

- Do there exist algorithms running in time $c^t|V|^{O(1)}$ on graphs of treewidth t that solve counting or weighted variants? For example can the number of Hamiltonian paths be determined, or the Traveling Salesman Problem solved in $c^t|V|^{O(1)}$ on graphs of treewidth t?

- Can exact exponential time algorithms be improved using Cut&Count (for example for CONNECTED DOMINATING SET, STEINER TREE and FEEDBACK VERTEX SET)?

- All our algorithms for directed graphs run in time $6^t|V|^{O(1)}$. Can the constant 6 be improved? Or maybe it is optimal (again, assuming SETH)?

Acknowledgements

References

[1] Manindra Agrawal and Somenath Biswas. Primality and identity testing via Chinese remaindering. *J. ACM*, 50:429–443, July 2003.

[2] Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast fast. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming*, volume 5555 of *Lecture Notes in Computer Science*, pages 49–58. Springer Berlin / Heidelberg, 2009.

[3] Vikraman Arvind and Partha Mukhopadhyay. Derandomizing the Isolation Lemma and Lower Bounds for Circuit Size. In Ashish Goel, Klaus Jansen, José D. P. Rolim, and Ronitt Rubinfeld, editors, *APPROX-RANDOM*, volume 5171 of *Lecture Notes in Computer Science*, pages 276–289. Springer, 2008.

[4] Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized Algorithms for the Loop Cutset Problem. *J. Artif. Intell. Res. (JAIR)*, 12:219–234, 2000.

[5] Daniel Binkele-Raible. *Amortized Analysis of Exponential Time and Parameterized Algorithms: Measure and Conquer and Reference Search Trees*. PhD thesis, University of Trier, 2010.

[6] Andreas Björklund. Determinant Sums for Undirected Hamiltonicity. In *FOCS*, pages 173–182, 2010.

[7] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In *Proc. of STOC'07*, pages 67–74, 2007.

[8] Andreas Bjorklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. The Travelling Salesman Problem in Bounded Degree Graphs. In Luca Aceto, Ivan Damgard, Leslie Goldberg, Magnus Halldorsson, Anna Ingolfsdottir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, volume 5125 of *Lecture Notes in Computer Science*, pages 198–209. Springer Berlin / Heidelberg, 2008.

[9] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *CoRR*, abs/1007.1161, 2010.

[10] Hans L. Bodlaender. On Disjoint Cycles. *Int. J. Found. Comput. Sci.*, 5(1):59–68, 1994.

[11] Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial Optimization on Graphs of Bounded Treewidth. *The Computer Journal*, 51(3):255–269, 2008.

[12] Nicolas Bourgeois, Bruno Escoffier, Vangelis Paschos, and Johan van Rooij. A Bottom-Up Method and Fast Algorithms for max independent set. In Haim Kaplan, editor, *SWAT 2010*, volume 6139 of *Lecture Notes in Computer Science*, pages 62–73. 2010.

[13] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved LP-based approximation for steiner tree. In *STOC*, pages 583–592, 2010.

[14] Yixin Cao, Jianer Chen, and Yang Liu. On Feedback Vertex Set New Measure and New Structures. In *Proc. of SWAT'10*, pages 93–104, 2010.

[15] Suresh Chari, Pankaj Rohatgi, and Aravind Srinivasan. Randomness-Optimal Unique Element Isolation with Applications to Perfect Matching and Related Problems. *SIAM J. Comput.*, 24(5):1036–1050, 1995.

[16] Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008.

[17] Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. *Theor. Comput. Sci.*, 411(40-42):3701–3713, 2010.

[18] Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An $O(2^{o(k)}n^3)$ FPT Algorithm for the Undirected Feedback Vertex Set Problem. In *COCOON*, pages 859–869, 2005.

[19] Erik D. Demaine and Mohammadtaghi Hajiaghayi. The bidimensionality Theory and Its Algorithmic Applications. 2005.

[20] Richard A. DeMillo and Richard J. Lipton. A Probabilistic Remark on Algebraic Program Testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

[21] Frederic Dorn, Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Beyond Bidimensionality: Parameterized Subexponential Algorithms on Directed Graphs. *CoRR*, abs/1001.0821, 2010.

[22] Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Fast Subexponential Algorithm for Non-local Problems on Graphs of Bounded Genus. In *SWAT*, pages 172–183, 2006.

[23] Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Catalan structures and dynamic programming in H-minor-free graphs. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 631–640, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[24] Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions. In *ESA*, pages 95–106. Springer, 2005.

[25] Robert James Douglas. NP-completeness and degree restricted spanning trees. *Discrete Mathematics*, 105(1-3):41–47, 1992.

[26] Rodney G. Downey and Michael R. Fellows. Fixed Parameter Tractability and Completeness. In *Complexity Theory: Current Research*, pages 191–225, 1992.

[27] Rodney G. Downey and Michael R. Fellows. In *Parameterized Complexity*, 1999.

[28] D. Eppstein. The Traveling Salesman Problem for Cubic Graphs. *J. Graph Algorithms Appl.*, 11:61–81, 2007.

[29] David Eppstein. Diameter and treewidth in minor-closed graph families. 27:275–291, 2000.

[30] Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Alexey A. Stepanov. On Two Techniques of Combining Branching and Treewidth. 54(2):181–207, 2006.

[31] Fedor V. Fomin. and Dimitrios M. Thilikos. A Simple and Fast Approach for Solving Problems on Planar Graphs. In Volker Diekert and Michel Habib, editors, *STACS*, volume 2996 of *Lecture Notes in Computer Science*, pages 56–67, 2004.

[32] H. Gebauer. On the Number of Hamilton Cycles in Bounded Degree Graphs. In *ANALCO*, 2008.

[33] Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006.

[34] Micheal Held and Richard M. Karp. The Traveling Salesman Problem and Minimum Spanning Trees. *Operations Research*, 18:1138–1162, 1970.

[35] Micheal Held and Richard M. Karp. The Traveling-Salesman Problem and Minimum Spanning Trees: Part II. *Mathematical Programming*, 1:6–25, 1971.

[36] Illya V. Hicks, Arie M. C. A. Koster, and Elif Kolotoğlu. Branch and tree decomposition techniques for discrete optimization. *Tutorials in Operations Research 2005*, pages 1–19, 2005.

[37] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - (2. ed.)*. Addison-Wesley series in computer science. Addison-Wesley-Longman, 2001.

[38] Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[39] K. Iwama and T. Nakashima. An Improved Exact Algorithm for Cubic Graph TSP. In *COCOON*, pages 108–117, 2007.

[40] Valentine Kabanets and Russell Impagliazzo. Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[41] Iyad A. Kanj, Michael J. Pelsmajer, and Marcus Schaefer. Parameterized Algorithms for Feedback Vertex Set. In *IWPEC*, pages 235–247, 2004.

[42] Richard M. Karp. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[43] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison–Wesley, 2005.

[44] Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[45] Ioannis Koutis. Faster Algebraic Algorithms for Path and Packing Problems. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2008.

[46] Ioannis Koutis and Ryan Williams. Limits and Applications of Group Algebras for Parameterized Problems. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 653–664. Springer, 2009.

[47] Daniel Lokshtanov, Daniel Marx, and Saket Saurabh. Known Algorithms on Graphs of Bounded Treewidth are Probably Optimal. In *Proc. of SODA'11 (to appear)*, 2011.

[48] Daniel Lokshtanov, Daniel Marx, and Saket Saurabh. Slightly Superexponential Parameterized Problems. In *Proc. of*

SODA'11 (to appear), 2011.

[49] Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. FPT Algorithms for Connected Feedback Vertex Set. In Md. Saidur Rahman and Satoshi Fujita, editors, *WALCOM*, volume 5942 of *Lecture Notes in Computer Science*, pages 269–280. Springer, 2010.

[50] Daniel Mölle, Stefan Richter, and Peter Rossmanith. Enumerate and Expand: Improved Algorithms for Connected Vertex Cover and Tree Cover. 43(2):234–253, 2008.

[51] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[52] Rolf Niedermeier. *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, USA, March 2006.

[53] Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster Fixed Parameter Tractable Algorithms for Undirected Feedback Vertex Set. In *ISAAC*, pages 241–248, 2002.

[54] Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Transactions on Algorithms*, 2(3):403–415, 2006.

[55] Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004.

[56] Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory*, 36(1):49–64, 1984.

[57] Neil Robertson and Paul D. Seymour. Graph Minors XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.

[58] Jack. T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27:701–717, October 1980.

[59] Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Mixed Searching and Proper-Path-Width. *Theor. Comput. Sci.*, 137(2):253–268, 1995.

[60] Dimitrios M. Thilikos, Maria Serna, and Hans L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. 56(1):1–24, 2005.

[61] W. T. Tutte. The factorization of linear graphs. *J. London Math. Soc.*, 22:107–111, 1947.

[62] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution. In *ESA*, pages 566–577, 2009.

[63] Johan M. M. van Rooij, Jesper Nederlof, and Thomas C. van Dijk. Inclusion/Exclusion Meets Measure and Conquer. In *ESA'09*, pages 554–565, 2009.

[64] Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.

[65] F. Yates. The design and analysis of factorial experiments. Technical Communication 35, Common-wealth Bureau of Soils, Harpenden, U.K., 1937.

[66] Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward Ng, editor, *Symbolic and Algebraic Computation*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer Berlin / Heidelberg, 1979.

A   Details of all algorithms

In this section we describe in detail all the algorithms sketched in Section 4. 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.

A.1   Fast subset convolution

We first recall the fast subset convolution (and its variants) [7], as it is often needed to handle join bags efficiently. We follow notation from [7]. 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 A.1. *The* subset convolution, covering product, packing product *of f and g are defined as functions $f *
g, f *_c g, f *_p g : 2^B \to R$ as follows:*

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

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

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

By computing a function $h : 2^B \to R$ we mean determining $h(T)$ for every $T \subseteq B$. Björklund et al. [7] proved that all three functions defined above can be computed efficiently.

Theorem A.2 ([7]). *The subset convolution, covering product and packing product of two given functions can be computed in $2^{|B|} |B|^{O(1)}$ ring operations.*

The following generalization of the subset convolution can be found in [17, 62].

Definition A.3. *Let $p \geq 2$ be an integer constant and let B be a finite set. For $t_1, t_2, t \in \{0, 1, \dots, p-1\}^B$ we say that $t_1 + t_2 = t$ iff $t_1(b) + t_2(b) = t(b)$ for all $b \in B$. For functions $f, g : \{0, 1, \dots, p-1\}^B \to R$ 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 $\mathbb{Z}_p$ but in $\mathbb{Z}$.

Theorem A.4 (Generalized Subset Convolution [17, 62]). *The generalized subset convolution can be computed in $p^{|B|} |B|^{O(1)}$ ring operations.*

Note that in [17] 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 timebound.

We need also the following variant of subset convolution.

Definition A.5. *Let $p \geq 2$ be an integer constant and let B be a finite set. For $t_1, t_2, t \in \mathbb{Z}_p^B$ we say that $t_1 + t_2 = t$ if $t_1(b) + t_2(b) = t(b)$ (in $\mathbb{Z}_p$) for all $b \in B$. For functions $f, g : \mathbb{Z}_p^B \to R$ define the $\mathbb{Z}_p$ product as*

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

In particular, for $p = 2$ we identify elements $\mathbb{Z}_2^B$ with subsets of B and define for $f, g : 2^B \to R$ the xor *product as:*

$$f *_x g(T) = \sum_{T_1, T_2 \subseteq B} [T_1 \triangle T_2 = T] f(T_1) g(T_2).$$

The xor product can be computed in time $2^{|B|}|B|^{O(1)}$ using the well-known Walsh-Hadamard transform. However, in our applications we need also the case $p = 4$, thus we provide a proof for both cases below. We do not state here any general theorem for arbitrary p, as in that case we would need to use fractional complex numbers during the computations, which could lead to rounding problems.

Theorem A.6. *Let $R = \mathbb{Z}$ or $R = \mathbb{Z}_q$ for some constant q. For $p = 2$ and $p = 4$ the $\mathbb{Z}_p$ product of two given functions $f, g : \mathbb{Z}_p^B \to R$ can be computed in time $p^{|B|}|B|^{O(1)}$.*

Proof. We will use a simplified version of the Fourier transform. Let us assume $R = \mathbb{Z}$, as otherwise we do calculations in $\mathbb{Z}$ and in the end we take all values modulo q. We use values of order at most $q^{O(1)}p^{O(|B|)}$, thus all arithmetical operations in $\mathbb{Z}$ take polynomial time in $|B|$ and $\log q$.

Let us introduce some definitions. Consider the ring $\mathbb{Z}[i] = \{a + bi : a, b \in \mathbb{Z}\} \subseteq \mathbb{C}$, where the addition and multiplication operators are inherited from the complex field $\mathbb{C}$. For $s, t \in \mathbb{Z}_p^B$ define $s \cdot t$ as $\sum_{b \in B} s(b)t(b) \in \mathbb{Z}$. Let ε be the degree-p root of 1 in $\mathbb{Z}[i]$, i.e., $\varepsilon = -1$ if $p = 2$ and $\varepsilon = i$ if $p = 4$. Note that $\varepsilon^p = 1$ in $\mathbb{Z}[i]$. We somewhat abuse the notation and for $c \in \mathbb{Z}_p$ use the notation ε^c (taking it to mean $\varepsilon^{c'}$, where c' is any integer congruent to c modulo p). For $f : \mathbb{Z}_p^B \to \mathbb{Z}$ define $\hat{f} : \mathbb{Z}_p^B \to \mathbb{Z}[i]$ as follows

$$\hat{f}(s) = \sum_{t \in \mathbb{Z}_p^B} f(t)\varepsilon^{s \cdot t}.$$

We first claim that $\hat{f}$ can be computed in $p^{|B|}|B|^{O(1)}$ time using an adjusted Yates algorithm [65]. We may assume $B = \{1, 2, \ldots, |B|\}$ and for $1 \le b \le |B|$ we define

$$f_b(s_1, s_2, \ldots, s_b, t_{b+1}, \ldots, t_{|B|}) = \sum_{t_1, t_2, \ldots, t_b \in \mathbb{Z}_p} f(t_1, t_2, \ldots, t_{|B|})\varepsilon^{\sum_{\beta=1}^{b} s_\beta t_\beta}.$$

Furthermore we set $f_0 = f$. Note that $f_{|B|} = \hat{f}$ and for $0 \le b < |B|$

$$f_{b+1}(s_1, s_2, \ldots, s_{b+1}, t_{b+2}, \ldots, t_{|B|}) = \sum_{t_{b+1} \in \mathbb{Z}_p} f_b(s_1, s_2, \ldots, s_b, t_{b+1}, \ldots, t_{|B|})\varepsilon^{s_{b+1}t_{b+1}}.$$

Thus, computing all f_b for $0 \le b \le |B|$ takes $p^{|B|}|B|^{O(1)}$ time and the claim is proven.

Let denote by $f \cdot g$ the pointwise multiplication of functions, i.e. $(f \cdot g)(t) = f(t) \cdot g(t)$. Observe that

$$\left(\widehat{\hat{f} \cdot \hat{g}}\right)(t) = \sum_{s \in \mathbb{Z}_p^B} (\hat{f} \cdot \hat{g})(s)\varepsilon^{s \cdot t}$$

$$= \sum_{s \in \mathbb{Z}_p^B} \left(\sum_{t_1 \in \mathbb{Z}_p^B} f(t_1)\varepsilon^{t_1 \cdot s}\right) \left(\sum_{t_2 \in \mathbb{Z}_p^B} g(t_2)\varepsilon^{t_2 \cdot s}\right) \varepsilon^{s \cdot t}$$

$$= \sum_{t_1, t_2 \in \mathbb{Z}_p^B} f(t_1)g(t_2) \sum_{s \in \mathbb{Z}_p^B} \varepsilon^{s \cdot (t_1 + t_2 + t)}$$

$$= \sum_{t_1, t_2 \in \mathbb{Z}_p^B} f(t_1)g(t_2) \prod_{b \in B} \left(\sum_{s(b) \in \mathbb{Z}_p} \left(\varepsilon^{t_1(b) + t_2(b) + t(b)}\right)^{s(b)}\right)$$

$$\left\{\text{using that} \sum_{s(b) \in \mathbb{Z}_p} (\epsilon^\tau)^{s(b)} = p \text{ if } \tau = 0 \text{ and otherwise } \frac{1 - (\epsilon^\tau)^p}{1 - \epsilon^\tau} = 0\right\}$$

$$= \sum_{t_1, t_2 \in \mathbb{Z}_p^B} f(t_1)g(t_2) \prod_{b \in B} \left(p \cdot [t_1(b) + t_2(b) + t(b) = 0]\right)$$

$$= \sum_{t_1, t_2 \in \mathbb{Z}_p^B} f(t_1)g(t_2)p^{|B|}[t_1 + t_2 + t = 0]$$

$$= p^{|B|} \left(f *_x^p g\right)(-t)$$

As $\widehat{\hat{f} \cdot \hat{g}}$ can be computed in time $p^{|B|}|B|^{O(1)}$, the theorem follows. $\qquad\square$

A.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 $Y \subseteq V$ 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 in case of a positive answer a set Y.

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 A.7. *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 connected the lemma follows. $\qquad\square$

Theorem A.8. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves the CON- STRAINED 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. Now we can use the Cut&Count technique. As the universe we take the set $U = V \times \{\mathbf{F}, \mathbf{M}\}$, where $V \times \{\mathbf{F}\}$ is used to assign weights to vertices from the chosen forest and $V \times \{\mathbf{M}\}$ for markers. As usual we assume that we are given a weight function $\omega : U \to \{1, ..., N\}$, where $N = 2|U| = 4|V|$.

The Cut part. For integers A, B, C, 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 $\omega(X \times \{\mathbf{F}\}) + \omega(M \times \{\mathbf{M}\}) = W$;

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 A.7 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 $2^{\mathrm{cc}(M,G[X])}$ cuts (X_1, X_2) such that $((X, M), (X_1, X_2)) \in \mathcal{C}_W^{A,B,C}$, where by $\mathrm{cc}(M, G[X])$ we denote the number of connected components of $G[X]$ which do not contain any marker from the set M. Hence by Lemma A.7 for every A,B,C,W satisfying $C \leq A - B$ we have $|\mathcal{S}_W^{A,B,C}| \equiv |\mathcal{C}_W^{A,B,C}|$.

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

We follow the notation from the STEINER TREE example (see Lemma 3.5). 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 the colouring) define

$$\mathcal{R}_x(a, b, c, w) = \Big\{(X, M) \mid X \subseteq V_x \setminus S \,\wedge\, |X| = a \,\wedge\, |E_x \cap E(G[X])| = b$$
$$\wedge\, M \subseteq X \setminus B_x \,\wedge\, |M| = c \,\wedge\, \omega(X \times \{\mathbf{F}\}) + \omega(M \times \{\mathbf{M}\}) = w\Big\}$$

$$\mathcal{C}_x(a, b, c, w) = \Big\{((X, M), (X_1, X_2)) \mid (X, M) \in \mathcal{R}_x(a, b, c, w)$$
$$\wedge\, M \subseteq X_1 \wedge (X, (X_1, X_2)) \text{ is a consistently cut subgraph of } G_x\Big\}$$

$$A_x(a, b, c, w, s) = \Big|\Big\{((X, M), (X_1, X_2)) \in \mathcal{C}_x(a, b, c, w)|$$
$$(s(v) = \mathbf{1}_j \Rightarrow v \in X_j) \,\wedge\, (s(v) = \mathbf{0} \Rightarrow v \notin X)\Big\}\Big|$$

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 $\mathcal{C}_x(a, b, c, w)$ 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. 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 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**:
$$A_x(0, 0, 0, 0, \emptyset) = 1$$

28

- **Introduce vertex bag**:

$$A_x(a, b, c, w, s \cup \{(v, \mathbf{0})\}) = A_y(a, b, c, w, s)$$
$$A_x(a, b, c, w, s \cup \{(v, \mathbf{1}_j)\}) = [v \notin S] A_y(a - 1, b, c, w - \omega((v, \mathbf{F})), s)$$

- **Introduce edge bag**:

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

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**:

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

If the vertex v was in X_1 then we can mark it and update the acumulator 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**:

$$A_x(a, b, c, w, s) = \sum_{a_1 + a_2 = a + |s^{-1}(\{\mathbf{1}_1, \mathbf{1}_2\})|} \sum_{b_1 + b_2 = b} \sum_{c_1 + c_2 = c}$$
$$\sum_{w_1 + w_2 = w + \omega(s^{-1}(\{\mathbf{1}_1, \mathbf{1}_2\}) \times \{\mathbf{F}\})} A_y(a_1, b_1, c_1, w_1, s) A_z(a_2, b_2, c_2, w_2, s)$$

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 $|\mathcal{C}_W^{A,B,C}| = A_r(A, B, C, W, \emptyset)$ the above recurrence leads to a dynamic programming algorithm that computes the parity of $|\mathcal{C}_W^{A,B,C}|$ for all reasonable values of W, A, B, C in $3^t |V|^{O(1)}$ time. Consequently we finish the proof of Theorem A.8. $\qquad\square$

A.3 Non-connectivity problems with an additional connectivity requirement

In this section we give details on algorithms for problems that are defined as standard "local" problems with an additional constraint that the solution needs to induce a connected subgraph. Problems described here are CONNECTED VERTEX COVER, CONNECTED DOMINATING SET, CONNECTED ODD CYCLE TRANSVERSAL and CONNECTED FEEDBACK VERTEX SET, but the approach here can be easily carried over to similar problems.

Let us start with a short informal description. Solving a problem CONNECTED X, we simply run the easy and well-known algorithm for X (or, in the case of CONNECTED FEEDBACK VERTEX SET, we run the algorithm for FEEDBACK VERTEX SET from Section A.2), but we additionally keep a cut consistent with the solution, i.e., we count the number of solution-cut pairs. Similarly as in the case of STEINER TREE, a solution to the problem X that induces c connected components is consistent with 2^{c-1} cuts, thus all the disconnected solutions cancel out modulo 2.

Similarly as in Section A.2 we solve more general versions of problems 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.

Remark A.9. *In the algorithms we assume that the set $S \subseteq V$ is nonempty, 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 iterate over all possible choices of $v_1 \in V$ and put $S = \{v_1\}$. 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-instance we only need a single run, in which a solution can be found, to give a correct answer.*

Let us now proceed with the formal arguments. For each problem, we start with a problem definition and a formal statement of a result.

A.3.1 CONNECTED VERTEX COVER

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

Theorem A.10. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves* CON-STRAINED 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 A.10 by a reduction to the STEINER TREE problem — we subdivide all edges of the graph G using terminals and add pendant terminals to S. 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 Appendix B which relies on the algorithm we describe in the proof.

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 A.9 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 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. $\mathcal{C}_W$ 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. Similarly as in the case of STEINER TREE we note that by Lemma 3.3 for each $X \in \mathcal{R}_W$ there exist $2^{\mathrm{cc}(G[X])-1}$ consistent cuts of $G[X]$, thus for any W we have $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$.

To finish the proof we need to describe a procedure $\mathtt{CountC}(\omega, W, \mathbb{T})$ that, given a nice tree decomposition $\mathbb{T}$, 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 3.5). For every bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \leq i \leq |V|$, $0 \leq w \leq N|V|$ and $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$ (called the colouring) define

$$\mathcal{R}_x(i, w) = \left\{ X \subseteq V_x \mid (S \cap V_x) \subseteq X \ \wedge \ |X| = i \ \wedge \ \omega(X) = w \ \wedge \ X \text{ is a vertex cover of } G_x \right\}$$

$$\mathcal{C}_x(i, w) = \left\{ (X, (X_1, X_2)) \mid X \in \mathcal{R}_x(i, w) \ \wedge \ (X, (X_1, X_2)) \text{ is a consistently cut subgraph of } G_x \right.$$
$$\left. \wedge \ (v_1 \in V_x \Rightarrow v_1 \in X_1) \right\}$$

$$A_x(i, w, s) = \left| \left\{ (X, (X_1, X_2)) \in \mathcal{C}_x(i, w) \mid (s(v) = \mathbf{1}_j \Rightarrow v \in X_j) \ \wedge \ (s(v) = \mathbf{0} \Rightarrow v \notin X) \right\} \right|$$

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 $\mathcal{C}$ of candidate solutions and consident 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.

30

- **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**:

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

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**:

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

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 A.10 is finished. □

A.3.2 CONNECTED DOMINATING SET

> CONSTRAINED CONNECTED DOMINATING SET
> **Input:** An undirected graph $G = (V, E)$, a subset $S \subseteq V$ and an integer k.
> **Question:** Does there exist such a connected set $S \subseteq X \subseteq V$ of cardinality at most k that $N[X] = V$?

Theorem A.11. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves CON-STRAINED CONNECTED DOMINATING SET in $4^t|V|^{O(1)}$ time. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

It is known that CONNECTED DOMINATING SET is equivalent to MAXIMUM LEAF TREE [25], hence the algorithm for EXACT k-LEAF SPANNING TREE can be used to solve CONNECTED DOMINATING SET. However, here the Cut&Count application is significantly easier and more straightforward than in the EXACT k-LEAF SPANNING TREE algorithm presented later. Thus we include the algorithm for CONNECTED DOMINATING SET below.

Proof of Theorem A.11. 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 A.9 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 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 $N[X] = V$.

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. $\mathcal{C}_W$ to be the family of pairs $(X, (X_1, X_2))$, where $X \in \mathcal{R}_W$, and (X_1, X_2) is a consistent cut of $G[X]$.

The Count part. As before we note that by Lemma 3.3 for each $X \in \mathcal{R}_W$ there exist $2^{\mathrm{cc}(G[X])-1}$ consistent cuts of $G[X]$, thus for any W we have $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$. What remains is to describe a procedure $\texttt{CountC}(\omega, W, \mathbb{T})$ that, given a nice tree decomposition $\mathbb{T}$, 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 3.5). For every bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \leq i \leq |V|, 0 \leq w \leq N|V|$ and $s \in \{\mathbf{0}_N, \mathbf{0}_Y, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$ (called the colouring) define

$$\mathcal{R}_x(i, w) = \left\{ X \subseteq V_x \mid (S \cap V_x) \subseteq X \,\wedge\, |X| = i \,\wedge\, \omega(X) = w \,\wedge\, N_{G_x}[X] = V_x \setminus s^{-1}(\mathbf{0}_N) \right\}$$

$$\mathcal{C}_x(i, w) = \Big\{ (X, (X_1, X_2)) \mid X \in \mathcal{R}_x(i, w) \,\wedge\, (X, (X_1, X_2)) \text{ is a consistently cut subgraph of } G_x$$

$$\wedge\, (v_1 \in V_x \Rightarrow v_1 \in X_1) \Big\}$$

$$A_x(i, w, s) = \Big| \Big\{ (X, (X_1, X_2)) \in \mathcal{C}_x(i, w) \mid (s(v) = \mathbf{1}_j \Rightarrow v \in X_j)$$

$$\wedge\, (s(v) = \mathbf{0}_Y \Rightarrow v \in N_{G_x}(X)) \,\wedge\, (s(v) = \mathbf{0}_N \Rightarrow v \notin N_{G_x}[X]) \Big\} \Big|$$

Here $s(v) = \mathbf{0}_Y$ means $v \notin X$ and v is dominated by X in G_x, $s(v) = \mathbf{0}_N$ means $v \notin X$ and v is not dominated by X in G_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 $\mathcal{C}$ 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. As usual, v denotes the vertex introduced and contained in an introduce bag, uv the edge introduced in an introduce edge bag, while y and z denote the left and right child of x in $\mathbb{T}$, if present.

- **Leaf bag**:
$$A_x(0, 0, \emptyset) = 1$$

- **Introduce vertex bag**:
$$A_x(i, w, s[v \to \mathbf{0}_N]) = [v \notin S]A_y(i, w, s)$$
$$A_x(i, w, s[v \to \mathbf{0}_Y]) = 0$$
$$A_x(i, w, s[v \to \mathbf{1}_1]) = A_y(i - 1, w - \omega(v), s)$$
$$A_x(i, w, s[v \to \mathbf{1}_2]) = [v \neq v_1]A_y(i - 1, w - \omega(v), s)$$

We take care of restrictions imposed by conditions $(S \cap V_x) \subseteq X$ and $v_1 \in X_1$. Note that at the moment of introducing v there are no edges incident to v in G_x, thus v cannot dominated, but not chosen.

- **Introduce edge bag**:

$$
\begin{aligned}
A_x(i,w,s) &= A_y(i,w,s) & \text{if } s(v), s(u) \in \{\mathbf{0}_N, \mathbf{0}_Y\} \\
A_x(i,w,s) &= [s(v) = s(u)]A_y(i,w,s) & \text{if } s(v), s(u) \in \{\mathbf{1}_1, \mathbf{1}_2\} \\
A_x(i,w,s) &= 0 & \text{if } s(v) = \mathbf{1}_j \wedge s(u) = \mathbf{0}_N \\
A_x(i,w,s) &= 0 & \text{if } s(v) = \mathbf{0}_N \wedge s(u) = \mathbf{1}_j \\
A_x(i,w,s) &= A_y(i,w,s) + A_y(i,w,s[u \to \mathbf{0}_N]) & \text{if } s(v) = \mathbf{1}_j \wedge s(u) = \mathbf{0}_Y \\
A_x(i,w,s) &= A_y(i,w,s) + A_y(i,w,s[v \to \mathbf{0}_N]) & \text{if } s(v) = \mathbf{0}_Y \wedge s(u) = \mathbf{1}_j
\end{aligned}
$$

Here we perform two operations. First, we filter out entries creating an inconsistent cut, i.e., ones in which the endpoints are coloured $\mathbf{1}_1$ and $\mathbf{1}_2$. Second, if one of the endpoints becomes dominated in G_x, its state could be changed from $\mathbf{0}_N$ to $\mathbf{0}_Y$

- **Forget bag**:

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

In the child bag the vertex v can have four states, but the state where v is not dominated ($s(v) = \mathbf{0}_N$) is forbidden (we will have no more chances to dominate this vertex, but all vertices need to be dominated). Thus we sum over the three remaining states.

- **Join bag**: For a colouring $s \in \{\mathbf{0}_N, \mathbf{0}_Y, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$ we define its precolouring $\hat{s} \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$ as

$$
\begin{aligned}
\hat{s}(v) &= s(v) & \text{if } s(v) \in \{\mathbf{1}_1, \mathbf{1}_2\} \\
\hat{s}(v) &= \mathbf{0} & \text{if } s(v) \in \{\mathbf{0}_Y, \mathbf{0}_N\}
\end{aligned}
$$

For a precolouring $\hat{s}$ (or a colouring s) and set $T \subseteq \hat{s}^{-1}(\mathbf{0})$ we define a colouring $s[T]$ as

$$
\begin{aligned}
s[T](v) &= \hat{s}(v) & \text{if } \hat{s}(v) \in \{\mathbf{1}_1, \mathbf{1}_2\} \\
s[T](v) &= \mathbf{0}_Y & \text{if } v \in T \\
s[T](v) &= \mathbf{0}_N & \text{if } v \in \hat{s}^{-1}(\mathbf{0}) \setminus T
\end{aligned}
$$

We can now write a recursion formula for join bags.

$$
A_x(i,w,s) = \sum_{i_1 + i_2 = i + |s^{-1}(\{\mathbf{1}_1, \mathbf{1}_2\})|} \sum_{w_1 + w_2 = w + \omega(s^{-1}(\{\mathbf{1}_1, \mathbf{1}_2\}))}
$$
$$
\sum_{T_1, T_2 \subseteq s^{-1}(\{\mathbf{0}_N, \mathbf{0}_Y\})} [T_1 \cup T_2 = s^{-1}(\mathbf{0}_Y)] A_y(i_1, w_1, s[T_1]) A_z(i_2, w_2, s[T_2])
$$

To achieve the colouring s, the precolourings of children have to be the same. Moreover, the sets of vertices coloured $\mathbf{0}_Y$ in children have to sum up to $s^{-1}(\mathbf{0}_Y)$. Since vertices coloured $\mathbf{1}_j$ in B_x are accounted for both tables of the children, we add their contribution to the accumulators.

To compute the recursion formula efficiently we need to use the fast evaluation of the covering product. For accumulators i, w and a precolouring $\hat{s}$ we define the following functions on subsets of $\hat{s}^{-1}(\mathbf{0})$:

$$
\begin{aligned}
f^{i,w,\hat{s}}(T) &= A_y(i,w,s[T]), \\
g^{i,w,\hat{s}}(T) &= A_z(i,w,s[T]).
\end{aligned}
$$

Now note that

$$
A_x(i,w,s) = \sum_{i_1 + i_2 = i + |s^{-1}(\{\mathbf{1}_1, \mathbf{1}_2\})|} \sum_{w_1 + w_2 = w + \omega(s^{-1}(\{\mathbf{1}_1, \mathbf{1}_2\}))} (f^{i_1, w_1, \hat{s}} *_c g^{i_2, w_2, \hat{s}})(s^{-1}(\mathbf{0}_Y)).
$$

By Theorem A.2, for fixed accumulators i_1, w_1, i_2, w_2 and a precolouring $\hat{s}$ the term

$$(f^{i_1, w_1, \hat{s}} *_c g^{i_2, w_2, \hat{s}})(s^{-1}(\mathbf{0}_Y))$$

can be computed in time $2^{|\hat{s}^{-1}(\mathbf{0})|}|\hat{s}^{-1}(\mathbf{0})|^{O(1)}$ at once for all colourings s with precolouring $\hat{s}$. Thus, the total time consumed by the evaluation of A_x is bounded by

$$|V|^{O(1)} \sum_{\hat{s} \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}} 2^{|\hat{s}^{-1}(\mathbf{0})|} = 4^{|B_x|}|V|^{O(1)}.$$

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 $4^t|V|^{O(1)}$ 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 (in particular, the ring operations in the convolutions used in join bags) can be done in constant time. Thus, the proof of Theorem A.11 is finished. $\square$

A.3.3 CONNECTED ODD CYCLE TRANSVERSAL

CONSTRAINED CONNECTED ODD CYCLE TRANSVERSAL
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 $G[V \setminus X]$ is bipartite?

Theorem A.12. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves* CON-STRAINED CONNECTED ODD CYCLE TRANSVERSAL *in $4^t|V|^{O(1)}$ 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 for Algorithm 1 we take $U = V \times \{\mathbf{X}, \mathbf{L}\}$, i.e., for each vertex $v \in V$ we generate two weights $\omega((v, \mathbf{X}))$ and $\omega((v, \mathbf{L}))$. Recall that we generate a random weight function $\omega : U \to \{1, 2, \ldots, N\}$, taking $N = 2|U| = 4|V|$. By Remark A.9 we may assume that $S \neq \emptyset$ and we may choose one fixed vertex $v_1 \in S$.
The Cut part. To make use of the well-known algorithm for ODD CYCLE TRANSVERSAL parameterized by treewidth, we need to define a solution not only as a set X, but we need to add a proof that $G[V \setminus X]$ is bipartite (i.e., a partition of $V \setminus X$ into two independent sets). Formally, for an integer W we define:

1. $\mathcal{R}_W$ to be the family of pairs (X, L), where $|X| = k$, $\omega(X \times \{\mathbf{X}\} \cup L \times \{\mathbf{L}\}) = W$, $S \subseteq X$, $X \cap L = \emptyset$ and L and $V \setminus (X \cup L)$ are independent sets in G;

2. $\mathcal{S}_W$ to be the family of pairs $(X, L) \in \mathcal{R}_W$ such that $G[X]$ is connected;

3. $\mathcal{C}_W$ to be the family of pairs $((X, L), (X_1, X_2))$, where $(X, L) \in \mathcal{R}_W$, $v_1 \in X_1$ and (X_1, X_2) is a consistent cut of $G[X]$.

Note that for a single set $X \subseteq V$ there may exist many proofs L that $G[V \setminus X]$ is bipartite. We consider all pairs (X, L) as *different* solutions and solution candidates. To compute weight, each pair (X, L) is represented as $X \times \{\mathbf{X}\} \cup L \times \{\mathbf{L}\} \subseteq U$, thus each pair (X, L) corresponds to a different subset of the weight domain U.
The Count part. Similarly as in the case of STEINER TREE we note that by Lemma 3.3 for each $(X, L) \in \mathcal{R}_W$ there exist $2^{\mathrm{cc}(G[X])-1}$ consistent cuts of $G[X]$, thus for any W we have $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$.

To finish the proof we need to describe a procedure $\texttt{CountC}(\omega, W, \mathbb{T})$ that, given a nice tree decomposition $\mathbb{T}$, 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 3.5). For every bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \leq i \leq |V|, 0 \leq w \leq N|V|$ and $s \in \{\mathbf{0}_L, \mathbf{0}_R, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$

(called the colouring) define

$$\mathcal{R}_x(i,w) = \Big\{(X,L) \mid X, L \subseteq V_x \;\wedge\; X \cap L = \emptyset \;\wedge\; (S \cap V_x) \subseteq X \;\wedge\; |X| = i$$
$$\wedge\; \omega(X \times \{\mathbf{X}\} \cup L \times \{\mathbf{L}\}) = w \;\wedge\; L \text{ and } V_x \setminus (X \cup L) \text{ are independent sets in } G_x\Big\}$$

$$\mathcal{C}_x(i,w) = \Big\{((X,L),(X_1,X_2)) \mid (X,L) \in \mathcal{R}_x(i,w) \;\wedge\; (X,(X_1,X_2)) \text{ is a consistently cut subgraph of } G_x$$
$$\wedge\; (v_1 \in V_x \Rightarrow v_1 \in X_1)\Big\}$$

$$A_x(i,w,s) = \Big|\Big\{((X,L),(X_1,X_2)) \in \mathcal{C}_x(i,w) \mid (s(v) = \mathbf{1}_j \Rightarrow v \in X_j)$$
$$\wedge\; (s(v) = \mathbf{0}_L \Rightarrow v \in L) \;\wedge\; (s(v) = \mathbf{0}_R \Rightarrow v \notin X \cup L)\Big\}\Big|$$

Here we plan L and $V \setminus (X \cup L)$ to be a bipartition of $G[V \setminus X]$; $s(v) = \mathbf{0}_L$ and $\mathbf{0}_R$ mean v is on the left or right side of this bipartition, respectively, while $s(v) = \mathbf{1}_j$ means v is in the odd cycle transversal, and on the appropriate side of the cut. The accumulators i and w keep track of the number of vertices in X and the weight of the pair (X,L), respectively. Hence $A_x(i,w,s)$ is the number of pairs from $\mathcal{C}$ 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. As always let v stand for the vertex introduced and contained in an introduce bag, uv for the edge introduced in an introduce edge bag, and y, z for the left and right child of x in $\mathbb{T}$ if present.

- **Leaf bag**:
$$A_x(0,0,\emptyset) = 1$$

- **Introduce vertex bag**:
$$A_x(i,w,s[v \to \mathbf{0}_L]) = [v \notin S]A_y(i, w - \omega((v,\mathbf{L})), s)$$
$$A_x(i,w,s[v \to \mathbf{0}_R]) = [v \notin S]A_y(i, w, s)$$
$$A_x(i,w,s[v \to \mathbf{1}_1]) = A_y(i - 1, w - \omega((v,\mathbf{X})), s)$$
$$A_x(i,w,s[v \to \mathbf{1}_2]) = [v \neq v_1]A_y(i - 1, w - \omega((v,\mathbf{X})), s)$$

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

- **Introduce edge bag**:

$$A_x(i,w,s) = 0 \qquad\qquad \text{if } \{s(u), s(v)\} = \{\mathbf{1}_1, \mathbf{1}_2\}$$
$$A_x(i,w,s) = 0 \qquad\qquad \text{if } s(u) = s(v) \in \{\mathbf{0}_L, \mathbf{0}_R\}$$
$$A_x(i,w,s) = A_y(i,w,s) \qquad\qquad \text{otherwise}$$

Here we remove table entries not consistent with the edge uv, i.e., table entries where the endpoints are coloured $\mathbf{1}_1$ and $\mathbf{1}_2$ (thus creating an inconsistent cut) or both coloured $\mathbf{0}_L$ or both coloured $\mathbf{0}_R$ (thus introducing an edge in $G_x[L]$ or $G_x[V_x \setminus (X \cup L)]$).

- **Forget bag**:
$$A_x(i,w,s) = \sum_{\alpha \in \{\mathbf{0}_L, \mathbf{0}_R, \mathbf{1}_1, \mathbf{1}_2\}} A_y(i, w, s[v \to \alpha])$$

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

- **Join bag**:

$$A_x(i,w,s) = \sum_{i_1+i_2=i+|s^{-1}(\{\mathbf{1_1},\mathbf{1_2}\})|} \quad \sum_{w_1+w_2=w+\omega(s^{-1}(\{\mathbf{1_1},\mathbf{1_2}\})\times\{\mathbf{X}\}\cup s^{-1}(\mathbf{0}_L)\times\{\mathbf{L}\})} A_y(i_1,w_1,s)A_z(i_2,w_2,s)$$

The only valid combinations to achieve the colouring s is to have the same colouring in both children. Since vertices coloured $\mathbf{1}_j$ and $\mathbf{0}_L$ 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 $4^t|V|^{O(1)}$ 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 A.12 is finished. $\qquad\square$

A.3.4 CONNECTED FEEDBACK VERTEX SET

CONSTRAINED CONNECTED 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 $Y \subseteq V$ of cardinality k such that $S \subseteq Y$, $G[Y]$ is connected and $G[V \setminus Y]$ is a forest?

Theorem A.13. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves the* CON- STRAINED CONNECTED FEEDBACK VERTEX SET *problem in $4^t|V|^{O(1)}$ 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. The idea is as in the previous algorithms in this subsection: we use the dynamic programming for FEEDBACK VERTEX SET, additionally keeping a cut consistent with the solution Y. However, in the previous subsections the base dynamic programming algorithms were the easy, naive ones. Here we need to use the Cut&Count based algorithm from Section A.2. Thus, we attach to a solution candidate two cuts: one of $G[Y]$, and second of $G[V \setminus Y]$.

As a universe we take the set $U = V \times \{\mathbf{F},\mathbf{M}\}$, where $V \times \{\mathbf{F}\}$ is used to assign weights to vertices from the chosen forest $G[V \setminus Y]$ and $V \times \{\mathbf{M}\}$ for markers. As usual we assume that we are given a weight function $\omega : U \to \{1,...,N\}$, where $N = 2|U| = 4|V|$. By Remark A.9 we assume $S \neq \emptyset$ and we fix one vertex $v_1 \in S$.

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

1. $\mathcal{R}_W^{A,B,C}$ to be the family of solution candidates, that is a 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 $\omega(X \times \{\mathbf{F}\}) + \omega(M \times \{\mathbf{M}\}) = W$;

2. $\mathcal{S}_W^{A,B,C}$ to be the set of solutions, that is the family of pairs (X,M), where $(X,M) \in \mathcal{R}_W^{A,B,C}$, where $G[X]$ is a forest containing at least one marker from the set M in each connected component and $G[V \setminus X]$ is connected;

3. $\mathcal{C}_W^{A,B,C}$ to be the family of triples $((X,M),(X_1,X_2),(Y_1,Y_2))$, where $(X,M) \in \mathcal{R}_W^{A,B,C}$, $M \subseteq X_1$, (X_1,X_2) is a consistent cut of $G[X]$, $v_1 \in Y_1$ and (Y_1,Y_2) is a consistent cut of $G[V \setminus X]$.

Observe that by Lemma A.7 the graph G admits a connected 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 STEINER TREE we note that by Lemma 3.3 for any $A,B,C,W,(X,M) \in \mathcal{R}_W^{A,B,C}$, there exist $2^{\mathrm{cc}(G[V \setminus X])-1}$ cuts (Y_1,Y_2) that are consistent cuts of $G[V \setminus X]$ and $v_1 \in Y_1$. Moreover, similarly as in the case of MIN CYCLE COVER (analogously to Lemma 3.8) note that there are $2^{\mathrm{cc}(M,G[X])}$ cuts (X_1,X_2) that are consistent with $G[X]$ and $M \subseteq X_1$, where by $\mathrm{cc}(M,G[X])$ we denote the number of connected components of $G[X]$ which do not contain any marker from the set M. Thus for any $A,B,C,W,(X,M) \in \mathcal{R}_W^{A,B,C}$, there are $2^{\mathrm{cc}(G[V \setminus X])-1+\mathrm{cc}(M,G[X])}$ triples $((X,M),(X_1,X_2),(Y_1,Y_2)) \in \mathcal{C}_W^{A,B,C}$. Hence by Lemma A.7 for every A,B,C,W satisfying $C \leq A - B$ we have $|\mathcal{S}_W^{A,B,C}| \equiv |\mathcal{C}_W^{A,B,C}|$.

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 $|\mathcal{C}_W^{A,B,C}|$ modulo 2 using dynamic programming.

We follow the notation from the STEINER TREE example (see Lemma 3.5). For every bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \le a \le |V|$, $0 \le b < |V|$, $0 \le c \le |V|$, $0 \le w \le 2N|V|$ and $s \in \{\mathbf{0}_1, \mathbf{0}_2, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$ (called the colouring) define

$$\mathcal{R}_x(a,b,c,w) = \Big\{ (X,M) \mid X \subseteq V_x \setminus S \;\wedge\; |X| = a \;\wedge\; |E_x \cap E(G[X])| = b$$
$$\wedge\; M \subseteq X \setminus B_x \;\wedge\; |M| = c \;\wedge\; \omega(X \times \{\mathbf{F}\}) + \omega(M \times \{\mathbf{M}\}) = w \Big\}$$

$$\mathcal{C}_x(a,b,c,w) = \Big\{ ((X,M),(X_1,X_2),(Y_1,Y_2)) \big| \, (X,M) \in \mathcal{R}_x(a,b,c,w)$$
$$\wedge\; M \subseteq X_1 \;\wedge\; (X,(X_1,X_2)) \text{ is a consistently cut subgraph of } G_x$$
$$\wedge\; (v_1 \in V_x \Rightarrow v_1 \in Y_1) \;\wedge\; (V_x \setminus X, (Y_1, Y_2)) \text{ is a consistently cut subgraph of } G_x \Big\}$$

$$A_x(a,b,c,w,s) = \Big| \Big\{ ((X,M),(X_1,X_2),(Y_1,Y_2)) \in \mathcal{C}_x(a,b,c,w) \big|$$
$$(s(v) = \mathbf{1}_j \Rightarrow v \in X_j) \;\wedge\; (s(v) = \mathbf{0}_j v \in Y_j) \Big\} \Big|$$

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, $s(v) = \mathbf{0}_j$ means $v \in Y_j$, 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 triples from $\mathcal{C}_x(a,b,c,w)$ 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.

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. We now give the recurrence for $A_x(a,b,c,w,s)$ that is used by the dynamic programming algorithm. As in the previous sections by v we denote the vertex introduced and contained in an introduce bag, by uv the edge introduced in an introduce edge bag, and by y, z 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**:
$$A_x(a,b,c,w,s[v \to \mathbf{0}_1]) = A_y(a,b,c,w,s)$$
$$A_x(a,b,c,w,s[v \to \mathbf{0}_2]) = [v \ne v_1] A_y(a,b,c,w,s)$$
$$A_x(a,b,c,w,s[v \to \mathbf{1}_j]) = [v \notin S] A_y(a-1,b,c,w-\omega((v,\mathbf{F})),s)$$

Here we take care of the constraints $S \cap X = \emptyset$ and $v_1 \in Y_1$.

- **Introduce edge bag**:

$$
\begin{aligned}
A_x(a,b,c,w,s) &= 0 & \text{if } \{s(v), s(u)\} = \{\mathbf{0}_1, \mathbf{0}_2\} \\
A_x(a,b,c,w,s) &= 0 & \text{if } \{s(v), s(u)\} = \{\mathbf{1}_1, \mathbf{1}_2\} \\
A_x(a,b,c,w,s) &= A_y(a,b-1,c,w,s) & \text{if } s(v) = s(u) \in \{\mathbf{1}_1, \mathbf{1}_2\} \\
A_x(a,b,c,w,s) &= A_y(a,b,c,w,s) & \text{otherwise}
\end{aligned}
$$

Here we remove table entries not consistent with the edge uv (i.e., creating an inconsistent cut, either (X_1, X_2) or (Y_1, Y_2)), and update the accumulator b storing the number of edges in $G[X]$.

- **Forget bag**:

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

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 four states with no additional requirements imposed.

- **Join bag**:

$$A_x(i,w,s) = \sum_{a_1+a_2=a+|s^{-1}(\{\mathbf{1_1},\mathbf{1_2}\})|}\ \sum_{b_1+b_2=b}\ \sum_{c_1+c_2=c}$$
$$\sum_{w_1+w_2=w+\omega(s^{-1}(\{\mathbf{1_1},\mathbf{1_2}\})\times\{\mathbf{F}\})} A_y(a_1,b_1,c_1,w_1,s)A_z(a_2,b_2,c_2,w_2,s)$$

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 $|\mathcal{C}_W^{A,B,C}| = A_r(A,B,C,W,\emptyset)$ the above recurrence leads to a dynamic programming algorithm that computes the parity of $|\mathcal{C}_W^{A,B,C}|$ for all reasonable values of W,A,B,C in $4^t n^{O(1)}$ time. Consequently we finish the proof of Theorem A.13. □

A.4 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: An undirected graph $G = (V,E)$ (or a directed graph $D = (V,A)$) and an integer k.
Question: Can the vertices of G (D) be covered with at most k vertex disjoint (directed) cycles?

(DIRECTED) LONGEST CYCLE
Input: An undirected graph $G = (V,E)$ (or a directed graph $D = (V,A)$) and an integer k.
Question: Does there exist a (directed) simple cycle of length k in G (D)?

(DIRECTED) LONGEST PATH
Input: An undirected graph $G = (V,E)$ (or a directed graph $D = (V,A)$) and an integer k.
Question: Does there exist a (directed) simple path of length k in G (D)?

We capture all three problems in the following artificial one.

(DIRECTED) PARTIAL CYCLE COVER
Input: An undirected graph $G = (V,E)$ (or 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 G (D) that cover exactly ℓ vertices?

Note that for $k = 1$ the above problem becomes LONGEST CYCLE, whereas for $\ell = |V|$ it becomes MIN CYCLE COVER. The LONGEST PATH problem can be easily reduced to LONGEST CYCLE, both in the directed and undirected setting. Given (DIRECTED) LONGEST PATH instance (G,k) $((D,k))$, we guess the endpoints s and t of the path in question, attach to the graph path of length $|V| + 1$ from t to s and ask for a cycle of length $|V| + 1 + k$. Moreover, given a tree decomposition $\mathbb{T}$ of G (D), a tree decomposition for the modified graph can be easily constructed by adding s and t to every bag and by covering the attached path by a sequence of additional bags of size 3. The width of the new decomposition is larger by a constant than the width of $\mathbb{T}$.

We now show how to solve PARTIAL CYCLE COVER using the Cut&Count technique, in time $4^t|V|^{O(1)}$ in the undirected case and in time $6^t|V|^{O(1)}$ in the directed case.

A.4.1 The undirected case

Theorem A.14. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves* PARTIAL CY-CLE COVER *in $4^t|V|^{O(1)}$ 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 markers. However, in this application it is more convenient to take as markers edges instead of vertices. The objects we count are subsets of edges, together with sets of marked edges, thus we take $U = E \times \{\mathbf{X}, \mathbf{M}\}$. As usual, we assume we are given a weight function $\omega : U \to \{1, 2, \ldots, N\}$, where $N = 2|U| = 4|E|$. We also assume $k \le \ell$.

The Cut part. For an integer W we define:

1. $\mathcal{R}_W$ to be the family of pairs (X, M), where $M \subseteq X \subseteq E$, $|X| = \ell$, $|M| = k$, $\omega(X \times \{\mathbf{X}\} \cup M \times \{\mathbf{M}\}) = W$ and each vertex $v \in V(X)$ has degree 2 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 edge 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 degree two, then $|V(X)| = \ell$. Thus if $(X, M) \in \mathcal{R}_W$ then X is a set of vertex disjoint cycles covering exactly ℓ vertices of G. If $(X, M) \in \mathcal{S}_W$, then the number of cycles is bounded by $|M| = k$, and if we have an X with at most k cycles, we can find an M so that $(X, M) \in \mathcal{S}_W$ for $W = \omega(X \times \{\mathbf{X}\} \cup M \times \{\mathbf{M}\})$ by taking at least one edge from each cycle. Thus, we need to check if $\mathcal{S}_W \ne \emptyset$ for some W.

The Count part. Let $((X, M), (X_1, X_2)) \in \mathcal{C}_W$. Let $\mathrm{cc}(X, M)$ denote the number of connected components of $G[X]$ that are not isolated vertices and do not contain an edge from M. If $C \subseteq X$ is the set of edges of such a 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 connected component C can be on either side of the cut (X_1, X_2). Thus there are $2^{\mathrm{cc}(M,X)}$ 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 $\mathbb{T}$, weight function ω and 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 3.5). Let $\Sigma = \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2, \mathbf{2}\}$. For every bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \le i, b \le |V|, 0 \le w \le 2N|V|$ and $s \in \Sigma^{B_x}$ (called the colouring) define

$$\mathcal{R}_x(i, b, w) = \Big\{(X, M) \mid M \subseteq X \subseteq E_x \ \wedge \ |M| = i \ \wedge \ |X| = b \ \wedge \ \omega(X \times \{\mathbf{X}\} \cup M \times \{\mathbf{M}\}) = w$$

$$\wedge \ (\forall_{v \in V(X) \setminus B_x} \deg_{G[X]}(v) = 2) \ \wedge \ (\forall_{v \in B_x} \deg_{G[X]}(v) \le 2)\Big\}$$

$$\mathcal{C}_x(i, b, w) = \Big\{((X, M), (X_1, X_2)) \big| \ (X, M) \in \mathcal{R}_x(i, b, w) \ \wedge \ V(M) \subseteq X_1$$

$$\wedge \ (X_1, X_2) \text{ is a consistent cut of the graph } (V(X), X)\Big\}$$

$$A_x(i, b, w, s) = \Big| \Big\{((X, M), (X_1, X_2)) \in \mathcal{C}_x(i, b, w) \big| \ (s(v) = \mathbf{0} \Rightarrow \deg_{G[X]}(v) = 0)$$

$$\wedge \ (s(v) = \mathbf{1}_j \Rightarrow (\deg_{G[X]}(v) = 1 \wedge v \in X_j)) \ \wedge \ (s(v) = \mathbf{2} \Rightarrow \deg_{G[X]}(v) = 2)\Big\} \Big|$$

The value of $s(v)$ denotes the degree of v in $G[X]$ and, in case of degree 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. This is a somewhat non-trivial trick — the natural implementation of dynamic programming would use 6 states for each vertex. For vertices of degree 0 this is necessary — we do not want to count isolated vertices as separate connected components, so we do not want to have a side of the cut defined for them. For vertices of degree 2 the situation is more tricky. They are cut (that is, each such vertex is on some side of the cut

in each counted object in $\mathcal{C}_x(i, b, w)$), but the information about the side of the cut will not be needed — we have a guarantee that no new edges will be added to that vertex (as 2 is the maxmimum degree). Note that the fact that we did remember the side of the cut previously ensures that when we have a path in the currently constructed solution, both endpoints of the path are remembered to be on the same side of the cut, even though we no more remember sides for the internal vertices of the path. The accumulators i, b and w keep track of the size of M, the size of X and the weight of (X, M), respectively.

Let us spend a moment discussing the choice we made to mark edges (as opposed to vertices). If we marked vertices, as we did in the previous problems, we would have a problem as to when to decide that a given vertex is a marker. The natural moment — in the forget bag — is unapplicable in this case, as (to save on space and time) we do not remember the side of the cut, so we do not know whether we can mark a vertex (remember, the whole point of marking vertices is to break the symmetry between the sides of the cut, so we have to mark only vertices that are on the left). The same problem applies to the introduce bag, moreover a vertex is introduced more than once, so we could mark it more than once (which would cause problems with the application of the Isolation Lemma). The best choice would be to mark it when we introduce an edge incident to it, but still we could mark it twice, if the introduce edge bags happen in two different branches of the tree. This can be circumvented by upgrading the nice tree decomposition definition, but the way we have chosen — to mark edges — is easier and cleaner. For edges we know that each edge is introduced exactly once, so we have a natural place to mark the edge and assure it is marked and counted exactly once.

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. 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 denote by v the vertex introduced and contained in an introduce bag, by uv the edge introduced in an introduce edge bag, and by 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**:
$$A_x(i, b, w, s[v \to \mathbf{0}]) = A_y(i, b, w, s)$$

The new vertex has degree zero and we do not impose any other constraints.

- **Introduce edge bag**: For the sake of simplicity of the recurrence formula let us define a function subs : $\Sigma \to 2^\Sigma$.

	$\mathbf{0}$	$\mathbf{1}_1$	$\mathbf{1}_2$	$\mathbf{2}$
subs	$\emptyset$	$\{\mathbf{0}\}$	$\{\mathbf{0}\}$	$\{\mathbf{1}_1, \mathbf{1}_2\}$

Intuitively, for a given state $\alpha \in \Sigma$ the value $\mathrm{subs}(\alpha)$ is the set of possible states a vertex can have before adding an incident edge.

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

$$A_x(i, b, w, s) = A_y(i, b, w, s) + \sum_{\alpha_u \in \mathrm{subs}(s(u))} \sum_{\alpha_v \in \mathrm{subs}(s(v))} \sum_{j \in \{1,2\}}$$
$$[(\alpha_u = \mathbf{1}_j \vee s(u) = \mathbf{1}_j) \wedge (\alpha_v = \mathbf{1}_j \vee s(v) = \mathbf{1}_j)]$$
$$\left(A_y(i, b - 1, w - \omega((uv, \mathbf{X})), s[u \to \alpha_u, v \to \alpha_v]) \right.$$
$$\left. + [j = 1] A_y(i - 1, b - 1, w - \omega((uv, \mathbf{X})) - \omega((uv, \mathbf{M})), s[u \to \alpha_u, v \to \alpha_v]) \right)$$

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

- **Forget bag**:
$$A_x(i, b, w, s) = A_y(i, b, w, s[v \to \mathbf{2}]) + A_y(i, b, w, s[v \to \mathbf{0}])$$
The forgotten vertex must have degree two or zero in $G[X]$.

- **Join bag**: For colourings $s_1, s_2, s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2, \mathbf{2}\}^{B_x}$ we say that $s_1 + s_2 = s$ if for each $v \in B_x$ at least one of the following holds:
$$s_1(v) = \mathbf{0} \ \wedge \ s(v) = s_2(v)$$
$$s_2(v) = \mathbf{0} \ \wedge \ s(v) = s_1(v)$$
$$s_1(v) = s_2(v) = \mathbf{1}_j \ \wedge \ s(v) = \mathbf{2}$$

We can now write the recurrence for the join bags.
$$A_x(i, b, w, s) = \sum_{i_1+i_2=i} \sum_{b_1+b_2=b} \sum_{w_1+w_2=w} \sum_{s_1+s_2=s} A_y(i_1, b_1, w_1, s_1) A_z(i_2, b_2, w_2, s_2)$$

The accumulators in the children bags need to sum up to the accumulators in the parent bag. Also the degrees need to sum up and the sides of the cut need to match, which is ensured by the constraint $s_1 + s_2 = s$.

A straightforward computation of the above recurrence leads to $16^t |V|^{O(1)}$ time. We now show how to use the $\mathbb{Z}_4$ product to obtain a better time complexity.

Let $\phi : \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2, \mathbf{2}\} \to \mathbb{Z}_4$ be defined as
$$\phi(\mathbf{0}) = 0 \qquad \phi(\mathbf{1}_1) = 1 \qquad \phi(\mathbf{1}_2) = 3 \qquad \phi(\mathbf{2}) = 2$$

Let $\phi : \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2, \mathbf{2}\}^{B_x} \to \mathbb{Z}_4^{B_x}$ be obtained by extending ϕ in the natural way. Note that ϕ is a bijection. Define $\rho : \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2, \mathbf{2}\} \to \mathbb{Z}$ as
$$\rho(\mathbf{0}) = 0 \qquad \rho(\mathbf{1}_1) = 1 \qquad \phi(\mathbf{1}_2) = 1 \qquad \phi(\mathbf{2}) = 2$$

and let $\rho(s) = \sum_{v \in B_x} \rho(s(v))$ for colouring s, i.e., $\rho(s)$ is the sum of degrees of all vertices in B_x. Let
$$f_m^{i,b,w}(\phi(s)) = [\rho(s) = m] A_y(i, b, w, s)$$
$$g_m^{i,b,w}(\phi(s)) = [\rho(s) = m] A_z(i, b, w, s)$$
$$h_m^{i,b,w}(\phi(s)) = \sum_{i_1+i_2=i} \sum_{b_1+b_2=b} \sum_{w_1+w_2=w} \sum_{m_1+m_2=m} (f_{m_1}^{i_1,b_1,w_1} *_x^4 g_{m_2}^{i_2,b_2,w_2})(\phi(s))$$

We claim that
$$A_x(i, b, w, s) = h_{\rho(s)}^{i,b,w}(\phi(s)).$$
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 values in the expansion of $h_{\rho(s)}^{i,b,w}(\phi(s))$ corresponding to a choice of s_1, s_2 and possibly having a contribution to $A_x(i, b, w, s)$ are exactly those, for which $s_1 + s_2 = s$ holds. To see this, first note that
$$\rho(\phi^{-1}(\phi(s_1) + \phi(s_2))) \leq \rho(s_1) + \rho(s_2).$$

Observe that the above inequality is an equality iff $s_1 + s_2 = s$. Thus when counting $h_m^{i,b,w}(\phi(s))$ we sum non-zero values only for such s_1, s_2 where $s = s_1 + s_2$. As the addition operator on colourings corresponds to the addition operator in $\mathbb{Z}_4$, the claim follows.

By Theorem A.6, the function $h_m^{i,b,w}$ can be computed in $4^t |V|^{O(1)}$ time and the time bound for the join bags follows.

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 $4^t |V|^{O(1)}$ 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 A.14 is finished. □

41

A.4.2 The directed case

Theorem A.15. *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 markers. As in the undirected case, in this application it is more convenient to take as markers arcs instead of vertices. The objects we count are subsets of arcs, together with sets of marked arcs, thus we take $U = A \times \{\mathbf{X}, \mathbf{M}\}$. 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 $M \subseteq X \subseteq A$, $|X| = \ell$, $|M| = k$, $\omega(X \times \{\mathbf{X}\} \cup M \times \{\mathbf{M}\}) = W$ 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 $((X, M), (X_1, X_2)) \in \mathcal{C}_W$. Let $\mathtt{cc}(X, M)$ denote the number of weakly[1] 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 $\mathbb{T}$, weight funtion 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 3.5). 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

$$\mathcal{R}_x(i, b, w) = \Big\{(X, M) \mid M \subseteq X \subseteq E_x \wedge |M| = i \wedge |X| = b \wedge \omega(X \times \{\mathbf{X}\} \cup M \times \{\mathbf{M}\}) = w$$

$$\wedge \ (\forall_{v \in V(X) \setminus B_x} \mathrm{indeg}_{G[X]}(v) = \mathrm{outdeg}_{G[X]}(v) = 1) \ \wedge \ (\forall_{v \in B_x} \mathrm{indeg}_{G[X]}(v), \mathrm{outdeg}_{G[X]}(v) \leq 1)\Big\}$$

$$\mathcal{C}_x(i, b, w) = \Big\{((X, M), (X_1, X_2)) \mid (X, M) \in \mathcal{R}_x(i, b, w) \ \wedge \ V(M) \subseteq X_1$$

$$\wedge \ (X_1, X_2) \text{ is a consistent cut of the graph}(V(X), X)\Big\}$$

$$A_x(i, b, w, s) = \left| \Big\{((X, M), (X_1, X_2)) \in \mathcal{C}_x(i, b, w) \mid (s(v) = \mathbf{io}_j \Rightarrow v \in X_j) \right.$$

$$\left. \wedge \ ((s(v) = \mathbf{io} \vee s(v) = \mathbf{io}_j) \Rightarrow (\mathrm{indeg}_{G[X]}(v) = \mathbf{i} \wedge \mathrm{outdeg}_{G[X]}(v) = \mathbf{o}))\Big\} \right|$$

The value of $s(v)$ contains an 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.

---

[1] We stress this for clarity: in $G[X]$ weakly connected components are always strongly connected components due to the requirements imposed on X.

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. 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 zero.

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

	$\mathbf{00}$	$\mathbf{01}_1$	$\mathbf{01}_2$	$\mathbf{10}_1$	$\mathbf{10}_2$	$\mathbf{11}$
insubs	$\emptyset$	$\emptyset$	$\emptyset$	$\{\mathbf{00}\}$	$\{\mathbf{00}\}$	$\{\mathbf{01}_1, \mathbf{01}_2\}$
outsubs	$\emptyset$	$\{\mathbf{00}\}$	$\{\mathbf{00}\}$	$\emptyset$	$\emptyset$	$\{\mathbf{10}_1, \mathbf{10}_2\}$

  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.

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

  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 $\alpha_u \in \text{outsubs}(s(u))$ and $\alpha_v \in \text{insubs}(s(v))$. 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 zero or two.

- **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 Generalized Subset Convolution to obtain a better time bound.

	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**

Figure 1: The join table of DIRECTED PARTIAL CYCLE COVER where it is indicated which states combine to which other states.

Let $\phi, \rho : \Sigma \to \{0, 1, 2, 3, 4, 5\}$ where

$$\phi(00) = 0 \qquad \phi(01_1) = 1 \qquad \phi(01_2) = 2 \qquad \phi(10_2) = 3 \qquad \phi(10_1) = 4 \qquad \phi(11) = 5$$
$$\rho(00) = 0 \qquad \rho(01_1) = 1 \qquad \rho(01_2) = 1 \qquad \rho(10_2) = 1 \qquad \rho(10_1) = 1 \qquad \rho(11) = 2$$

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 **1**'s in a state s, i.e., the sum of all degrees of vertices in B_x. Then, define

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

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 acccording 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 A.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 A.15 is finished. $\qquad \square$

A.5 Spanning trees with a prescribed number of leaves

In this section we provide algorithms that solve EXACT k-LEAF SPANNING TREE and EXACT k-LEAF OUTBRANCHING in time $4^t n^{O(1)}$ and $6^t n^{O(1)}$, respectively. The algorithms are very similar and use the same tricks, thus we gather them together in this subsection.

Both algorithms use almost the same Cut part that is very natural for the considered problems. However, a quite straightforward realization of the accompanying Count part would lead to running times $6^t n^{O(1)}$ and $8^t n^{O(1)}$, respectively. To obtain better time bounds we need to count objects in a more ingenious way.

> EXACT k-LEAF SPANNING TREE
> **Input:** An undirected graph $G = (V, E)$ and an integer k.
> **Question:** Does there exists a spanning tree of G with exactly k leaves?

Theorem A.16. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves* EXACT k-LEAF SPANNING TREE *in $4^t |V|^{O(1)}$ time. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

Proof. We assume that G is connected, as otherwise we can safely answer NO. We also assume $|V| \geq 3$, and therefore any spanning tree of G contains some internal bags (i.e., bags of degree at least 2). Using similar arguments as in Remark A.9, we may assume that we are given vertex $v_1 \in V$ that is required to be an internal bag of the spanning tree in question. Thus, we can look for spanning trees of G that are rooted in the given vertex v_1.

We use the Cut&Count technique. Our solutions and solution candidates are subsets of edges, thus we take $U = E$ and generate random weight function $\omega : U \to \{1, 2, \ldots, N\}$, where $N = 2|U| = 2|E|$.

The Cut part. For integers W and k we define:

1. $\mathcal{R}_W^k$ to be the family of sets $X \subseteq E$, such that $\omega(X) = W$, $|X| = |V| - 1$, $G[X]$ contains exactly k vertices of degree one, and the degree of v_1 in $G[X]$ is at least 2.

2. $\mathcal{S}_W^k$ to be the family of sets $X \in \mathcal{R}_W^k$, such that $G[X]$ is connected;

3. $\mathcal{C}_W^k$ to be the family of pairs $(X, (X_1, X_2))$, where $X \in \mathcal{R}_W^k$ and (X_1, X_2) is a consistent cut of $G[X]$ with $v_1 \in X_1$.

The condition that for $X \in \mathcal{S}_W^k$ the graph $G[X]$ is connected, together with $|X| = |V| - 1$ gives us that each $X \in \mathcal{S}_W^k$ induces a spanning tree. Thus, $\mathcal{S}_W^k$ is indeed a family of spanning trees of exactly k leaves with root v_1.

Note that, unlike in other algorithms, we use the superscript k in the definitions. To achieve claimed the running time we need to do computations for many values of k.

The Count part. To use Algorithm 1 we need to formally prove that for any W and k we have $|\mathcal{S}_W^k| \equiv |\mathcal{C}_W^k|$. Similarly as in the case of STEINER TREE we note that by Lemma 3.3 for each $X \in \mathcal{R}_W^k$ there exist $2^{\mathrm{cc}(G[X])-1}$ consistent cuts of $G[X]$, and the claim follows.

To finish the proof we need to show how to compute $|\mathcal{C}_W^k|$ modulo 2 in time $4^t n^{O(1)}$. A straightforward dynamic programming algorithm would lead to a $6^t n^{O(1)}$ time complexity (we encourage the reader to sketch this algorithm to see why the steps introduced below are needed), thus we need to be a bit more ingenious here.

Let us define the set $\overline{\mathcal{C}}_W^\ell$ to be a family of triples $(X, R, (Y_1, Y_2))$ such that

1. $X \in \bigcup_{k=0}^{|V|-1} \mathcal{R}_W^k$, (i.e., we do not impose any constraint on the number of vertices of degree one in $G[X]$),

2. $R \subseteq V \setminus \{v_1\}$ and $|R| = \ell$,

3. Each vertex $v \in R$ has degree one in $G[X]$ and the unique neighbour of v in $G[X]$ is not an element of R (i.e., $G[X]$ does not contain a connected component that consists of two vertices from R connected by an edge).

4. Let $G(V \setminus R, X)$ denote the graph with the vertex set $V \setminus R$ and the edge set consisting of those edges of X that have both endpoints in $V \setminus R$. Then we require that (Y_1, Y_2) is a consistent cut of $G(V \setminus R, X)$ with $v_1 \in Y_1$.

Informally speaking, there are two differences between $\mathcal{C}_W^k$ and $\overline{\mathcal{C}}_W^\ell$. First, instead of requiring a prescribed number of vertices of degree one, we distinguish a fixed number of vertices that have to be of degree one, and we do not care about the degrees of the other vertices. Second, we consider only consistent cuts of $G(V \setminus R, X)$, not of whole $G[X]$.

We first note that the second difference is somewhat illusory. Let $X \in \mathcal{R}_W^k$ and $R \subseteq V \setminus \{v_1\}$ be as in the definition of $\overline{\mathcal{C}}_W^\ell$, i.e., $|R| = \ell$, each $v \in R$ is of degree one in $G[X]$ and its unique neighbour in $G[X]$ is not in R. If $(X, (X_1, X_2)) \in \mathcal{C}_W^k$, then the cut (Y_1, Y_2) where $Y_j = X_j \setminus R$ is consistent with $G(V \setminus R, X)$ and

$(X, R, (Y_1, Y_2)) \in \overline{\mathcal{C}}_W^\ell$. In the other direction, observe that if $(X, R, (Y_1, Y_2)) \in \overline{\mathcal{C}}_W^\ell$, then there exists exactly one cut (X_1, X_2) of $G[X]$, such that $Y_1 \subseteq X_1$ and $Y_2 \subseteq X_2$, namely $X_j = Y_j \cup (N_{G[X]}(Y_j) \cap R)$. Thus, in the analysis that follows we can assume that (Y_1, Y_2) is in fact a consistent cut of $G[X]$ with $v_1 \in Y_1$.

Let $(X, R, (Y_1, Y_2)) \in \overline{\mathcal{C}}_W^\ell$. As there exists $2^{\mathrm{cc}(G[X])-1}$ consistent cuts of $G[X]$, for fixed X and R we have $(X, R, C) \in \overline{\mathcal{C}}_W^\ell$ for exactly $2^{\mathrm{cc}(G[X])-1}$ cuts C. Thus, if $X \notin \bigcup_{k=0}^{|V|-1} \mathcal{S}_W^k$, all elements of $\overline{\mathcal{C}}_W^\ell$ with X cancel out modulo 2.

Otherwise, if $G[X]$ induces a spanning tree of G with root v_1, there exists exactly one cut $(V, \emptyset)$ consistent with $G[X]$, such that v_1 is in the first set in the cut. Moreover, note that there are no two adjacent vertices of degree one in $G[X]$ (as $|V| \geq 3$). Thus, if $X \in \mathcal{S}_W^k$, then there exist $\binom{k}{\ell}$ choices of the set R and one choice of a cut $C = (V, \emptyset)$ such that $(X, R, C) \in \overline{\mathcal{C}}_W^\ell$ (we use the convention that $\binom{k}{\ell} = 0$ if $k < \ell$).

Summing up, we obtain that in $\mathbb{Z}_2$

$$|\overline{\mathcal{C}}_W^\ell| \equiv \sum_{k=\ell}^{|V|-1} \binom{k}{\ell} |\mathcal{S}_W^k| \equiv \sum_{k=\ell}^{|V|-1} \binom{k}{\ell} |\mathcal{C}_W^k|.$$

Note that, operating over the field $\mathbb{Z}_2$, we have obtained a linear operator that transforms a vector $(|\mathcal{C}_W^k|)_{k=0}^{|V|-1}$ into a vector $(|\overline{\mathcal{C}}_W^\ell|)_{\ell=0}^{|V|-1}$. Moreover, the matrix of this operator can be computed in polynomial time and is upper triangular with ones on the diagonal. Thus, this operator can be easily inverted, and we can compute (in $\mathbb{Z}_2$) all values $(|\mathcal{C}_W^k|)_{k=0}^{|V|-1}$ knowing all values $(|\overline{\mathcal{C}}_W^\ell|)_{\ell=0}^{|V|-1}$.

To finish the proof we need to describe a procedure $\overline{\mathtt{CountC}}(\omega, W, \ell, \mathbb{T})$ that, given a nice tree decomposition $\mathbb{T}$, weight funtion ω and integers W and ℓ computes $|\overline{\mathcal{C}}_W^\ell|$ modulo 2. Now we can use dynamic programming on the tree decomposition.

Recall that in the definition of $\overline{\mathcal{C}}_W^\ell$ the cut (Y_1, Y_2) was a consistent cut of only $G(V \setminus R, X)$. We make use of this fact to reduce the size of the table in the dynamic programming, as we do not need to remember side of the cut for vertices in R.

We follow the notation from the STEINER TREE example (see Lemma 3.5). For every bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \leq \ell \leq |V|$, $0 \leq w \leq N|E|$, $0 \leq m, d < |V|$, and $s \in \{\mathbf{1}_1, \mathbf{1}_2, \mathbf{0}_0, \mathbf{0}_1\}^{B_x}$ (called the colouring) define

$$\mathcal{R}_x(\ell, w, m, d) = \Big\{(X, R) \mid X \subseteq E_x \wedge R \subseteq V_x \wedge |X| = m \wedge |R| = \ell \wedge \omega(X) = w$$
$$\wedge \deg_{G[X]}(v_1) = d \wedge (v \in R \setminus B_x \Rightarrow \deg_{G[X]}(v) = 1) \wedge (v \in R \cap B_x \Rightarrow \deg_{G[X]}(v) \leq 1)\Big\}$$

$$\mathcal{C}_x(\ell, w, m, d) = \Big\{(X, R, (Y_1, Y_2)) \mid (X, R) \in \mathcal{R}_x(\ell, w, m, d) \wedge (v_1 \in V_x \Rightarrow v_1 \in Y_1)$$
$$\wedge \ (Y_1, Y_2) \text{ is a consistent cut of } G(V_x \setminus R, X)\Big\}$$

$$A_x(\ell, w, m, d, s) = \Big|\Big\{(X, R, (Y_1, Y_2)) \in \mathcal{C}_x(\ell, w, m, d) \Big|$$
$$(s(v) = \mathbf{1}_j \Rightarrow v \in Y_j) \wedge (s(v) = \mathbf{0}_j \Rightarrow (v \in R \wedge \deg_{G[X]}(v) = j))\Big\}\Big|$$

Here $s(v) = \mathbf{0}_j$ denotes that $v \in R$ and $\deg_{G[X]}(v) = j$, whereas $s(v) = \mathbf{1}_j$ denotes that $v \in Y_j$ (and thus $v \notin R$). The accumulators ℓ, m and w keep track of the size of R, size of X and the weight of X, respectively. The accumulator d keeps track of the degree of v_1 in $G[X]$, since we need to ensure that in the end it is at least 2. Hence $A_x(\ell, w, m, d, s)$ reflects the number of partial objects from $\overline{\mathcal{C}}$ with fixed sizes of R, X, weight of X, degree of v_1 and interface on vertices from B_x.

The algorithm computes $A_x(\ell, w, m, d, s)$ for all bags $x \in T$ in a bottom-up fashion for all reasonable values of ℓ, w, m, d and the colouring s. We now give the recurrence for $A_x(\ell, w, m, d, s)$ that is used by the dynamic programming algorithm. In order to simplify notation we denote by v the vertex introduced and contained in an introduce bag, by uv the edge introduced in an introduce edge bag, and by y, z 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**:
$$A_x(\ell, w, m, d, s[v \to \mathbf{0}_0]) = [v \neq v_1] A_y(\ell - 1, w, m, d, s)$$
$$A_x(\ell, w, m, d, s[v \to \mathbf{0}_1]) = 0$$
$$A_x(\ell, w, m, d, s[v \to \mathbf{1}_1]) = A_y(\ell, w, m, d, s)$$
$$A_x(\ell, w, m, d, s[v \to \mathbf{1}_2]) = [v \neq v_1] A_y(\ell, w, m, d, s)$$

If the new vertex is in R, it has degree zero and cannot be equal to v_1. Otherwise, we need to ensure that we do not put v_1 into Y_2.

- **Introduce edge bag**:

$$
\begin{aligned}
A_x(\ell, w, m, d, s) &= A_y(\ell, w, m, d, s) \\
&\quad + A_y(\ell, w - \omega(uv), m - 1, d - [v_1 = u \vee v_1 = v], s) && \text{if } s(u) = s(v) = \mathbf{1}_j \\
A_x(\ell, w, m, d, s) &= A_y(\ell, w, m, d, s) \\
&\quad + A_y(\ell, w - \omega(uv), m - 1, d - [v_1 = u], s[v \to \mathbf{0}_0]) && \text{if } s(v) = \mathbf{0}_1 \wedge s(u) = \mathbf{1}_j \\
A_x(\ell, w, m, d, s) &= A_y(\ell, w, m, d, s) \\
&\quad + A_y(\ell, w - \omega(uv), m - 1, d - [v_1 = v], s[u \to \mathbf{0}_0]) && \text{if } s(v) = \mathbf{1}_j \wedge s(u) = \mathbf{0}_1 \\
A_x(\ell, w, m, d, s) &= A_y(\ell, w, m, d, s) && \text{otherwise}
\end{aligned}
$$

Here we consider adding uv to X. This is possible in two cases. First, if $u, v \notin R$ and $s(u) = s(v)$. Second, if exactly one of u and v is in R (recall that we forbid edges connecting two vertices in R). In the second case we need to update the degree of the vertex in R. In both cases we need to update the degree of v_1, if needed.

- **Forget bag**:

$$
\begin{aligned}
A_x(\ell, w, m, d, s) &= [d \geq 2] A_y(\ell, w, d, s[v \to \mathbf{1}_1]) && \text{if } v = v_1 \\
A_x(\ell, w, m, d, s) &= \sum_{\alpha \in \{\mathbf{0}_1, \mathbf{1}_1, \mathbf{1}_2\}} A_y(\ell, w, m, d, s[v \to \alpha]) && \text{otherwise}
\end{aligned}
$$

If we forget $v = v_1$, we require that its degree is at least two and $v \in Y_1$. Otherwise, we require only that if $v \in R$ then $\deg_{G[X]}(v) = 1$.

- **Join bag**: We proceed similarly as in the case of join bags in the CONNECTED DOMINATING SET problem. For a colouring $s \in \{\mathbf{0}_0, \mathbf{0}_1, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$ we define its precolouring $\hat{s} \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$ as

$$
\begin{aligned}
\hat{s}(v) &= s(v) && \text{if } s(v) \in \{\mathbf{1}_1, \mathbf{1}_2\} \\
\hat{s}(v) &= \mathbf{0} && \text{if } s(v) \in \{\mathbf{0}_0, \mathbf{0}_1\}
\end{aligned}
$$

For a precolouring $\hat{s}$ (or a colouring s) and a set $T \subseteq \hat{s}^{-1}(\mathbf{0})$ we define the colouring $s[T]$

$$
\begin{aligned}
s[T](v) &= \hat{s}(v) && \text{if } \hat{s}(v) \in \{\mathbf{1}_1, \mathbf{1}_2\} \\
s[T](v) &= \mathbf{0}_1 && \text{if } v \in T \\
s[T](v) &= \mathbf{0}_0 && \text{if } v \in \hat{s}^{-1}(\mathbf{0}) \setminus T
\end{aligned}
$$

We can now write a recursion formula for the join bags.

$$A_x(\ell, w, m, d, s) = \sum_{\ell_1 + \ell_2 = \ell + |s^{-1}(\{\mathbf{0}_0, \mathbf{0}_1\})|} \quad \sum_{w_1 + w_2 = w} \quad \sum_{m_1 + m_2 = m} \quad \sum_{d_1 + d_2 = d} \quad \sum_{T_1, T_2 \subseteq s^{-1}(\{\mathbf{0}_0, \mathbf{0}_1\})}$$
$$[T_1 \cup T_2 = s^{-1}(\mathbf{0}_1)][T_1 \cap T_2 = \emptyset] A_y(\ell_1, w_1, m_1, d_1, s[T_1]) A_z(\ell_2, w_2, m_2, d_2, s[T_2])$$

To achieve the colouring s, the precolourings of the children have to be the same. Moreover, a vertex $v \in R$ has degree one only if it has degree one in exactly one of the children bags. Thus the sets of vertices coloured $\mathbf{0}_1$ in children have to be disjoint and sum up to $s^{-1}(\mathbf{0}_1)$. Since vertices coloured $\mathbf{0}_j$ in B_x are accounted for in both tables of the children, we add their contribution to the accumulator ℓ.

To compute the recursion formula efficiently we need to use the fast subset convolution. For accumulators ℓ, w, m, d and a precolouring $\hat{s}$ we define the following functions on subsets of $\hat{s}^{-1}(\mathbf{0})$:

$$f^{\ell,w,m,d,\hat{s}}(T) = A_y(\ell, w, m, d, s[T]),$$
$$g^{\ell,w,m,d,\hat{s}}(T) = A_z(\ell, w, m, d, s[T]).$$

Now note that

$$A_x(\ell, w, m, d, s) = \sum_{\ell_1+\ell_2=\ell+|s^{-1}(\{\mathbf{0}_0,\mathbf{0}_1\})|} \sum_{w_1+w_2=w} \sum_{m_1+m_2=m} \sum_{d_1+d_2=d} (f^{\ell_1,w_1,m_1,d_1,\hat{s}} * g^{\ell_2,w_2,m_2,d_2,\hat{s}})(s^{-1}(\mathbf{0}_1)).$$

By Theorem A.2, for fixed accumulators $\ell_1, w_1, m_1, d_1, \ell_2, w_2, m_2, d_2$ and a precolouring $\hat{s}$ the term

$$(f^{\ell_1,w_1,m_1,d_1,\hat{s}} * g^{\ell_2,w_2,m_2,d_2,\hat{s}})(s^{-1}(\mathbf{0}_1))$$

can be computed in time $2^{|\hat{s}^{-1}(\mathbf{0})|}|\hat{s}^{-1}(\mathbf{0})|^{O(1)}$ at once for all colourings s with precolouring $\hat{s}$. Thus, the total time consumed by the evaluation of A_x is bounded by

$$|V|^{O(1)} \sum_{\hat{s} \in \{\mathbf{0},\mathbf{1}_1,\mathbf{1}_2\}^{B_x}} 2^{|\hat{s}^{-1}(\mathbf{0})|} = 4^{|B_x|}|V|^{O(1)}.$$

It is easy to see that the above recurrence leads to a dynamic programming algorithm that computes the parity of $|\overline{\mathcal{C}}_W^\ell|$ for all values of W and ℓ in $4^t|V|^{O(1)}$ time, since $|\overline{\mathcal{C}}_W^\ell| = \sum_{d \geq 2} A_r(\ell, W, |V| - 1, d, \emptyset)$. Moreover, as we count the parities and not the numbers A_x themselves, all arithmetical operations (in particular the ring operations in the fast subset convolution) can be done in constant time. As discussed before, knowing in $\mathbb{Z}_2$ the values of $|\overline{\mathcal{C}}_W^\ell|$ for $0 \leq \ell \leq |V| - 1$ we can compute all values of $|\mathcal{C}_W^k|$ and $|\mathcal{S}_W^k|$ modulo 2. Thus, the proof of Theorem A.16 is finished. $\square$

A.5.2 EXACT k-LEAF OUTBRANCHING

EXACT k-LEAF OUTBRANCHING
Input: A directed graph $D = (V, A)$ and an integer k, and a root $r \in V$.
Question: Does there exist a spanning tree of D with all edges directed away from the root with exactly k leaves?

Theorem A.17. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves* EXACT k-LEAF OUTBRANCHING *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 in a very similar manner as for the EXACT k-LEAF SPANNING TREE problem. Our solutions and solution candidates are subsets of arcs, thus we take $U = A$ and generate random weight function $\omega : U \to \{1, 2, \ldots, N\}$, where $N = 2|U| = 2|A|$. We set $v_1 = r$.
The Cut part. For integers W and k we define:

1. $\mathcal{R}_W^k$ to be the family of sets $X \subseteq A$, such that $\omega(X) = W$, $\text{indeg}_{G[X]}(v_1) = 0$ and $\text{indeg}_{G[X]}(v) = 1$ if $v \neq v_1$, and $G[X]$ contains exactly k vertices of outdegree zero.

2. $\mathcal{S}_W^k$ to be the family of sets $X \in \mathcal{R}_W^k$, such that $G[X]$ is weakly connected;

3. $\mathcal{C}_W^k$ to be the family of pairs $(X, (X_1, X_2))$, where $X \in \mathcal{S}_W^k$ and (X_1, X_2) is a consistent cut of $G[X]$ with $v_1 \in X_1$.

The condition that for $X \in \mathcal{S}_W^k$ the graph $G[X]$ is weakly connected, together with the condition on the indegrees of vertices gives us that each $X \in \mathcal{S}_W^k$ is of size $|V| - 1$ and induces an spanning tree of G, rooted in v_1, with all edges directed away from the root.

As in the case of EXACT k-LEAF SPANNING TREE, we use the superscript k in the definitions, since we perform a similar trick in the Count part and we do computations for many values of k.

The Count part. To use Algorithm 1 we need to formally prove that for any W and k we have $|\mathcal{S}_W^k| \equiv |\mathcal{C}_W^k|$. Similarly as in the case of STEINER TREE we note that by Lemma 3.3 for each $X \in \mathcal{R}_W^k$ there exist $2^{\mathrm{cc}(G[X])-1}$ consistent cuts of $G[X]$, and the claim follows (here $\mathrm{cc}(G[X])$ denotes the number of weakly connected components of $G[X]$).

To finish the proof we need to show now to compute $|\mathcal{C}_W^k|$ modulo 2 in time $6^t n^{O(1)}$. A straightforward dynamic programming algorithm would lead to $8^t n^{O(1)}$ time complexity, thus again we need to be more careful.

Let us define the set $\overline{\mathcal{C}}_W^\ell$ to be a family of triples $(X, R, (Y_1, Y_2))$ such that

1. $X \in \bigcup_{k=0}^{|V|-1} \mathcal{R}_W^k$, (i.e., we do not impose any constraint on the number of outdegree zero vertices in $G[X]$),

2. $R \subseteq V$ and $|R| = \ell$,

3. each vertex $v \in R$ has outdegree zero in $G[X]$,

4. (Y_1, Y_2) is a consistent cut of $G(V \setminus R, X)$ (defined as in the previous subsection) with $v_1 \notin Y_2$.

Note that, unlike the EXACT k-LEAF SPANNING TREE case, we do not need to require that the vertices from R are not connected by edges from X, as this is guaranteed by the outdegree condition. Moreover, we allow $v_1 \in R$.

Informally speaking, there are two differences between $\mathcal{C}_W^k$ and $\overline{\mathcal{C}}_W^\ell$. First, instead of requiring a prescribed number of vertices of outdegree zero, we distinguish a fixed number of vertices that have to be of outdegree zero, and we do not care about the outdegrees of the other vertices. Second, we consider only consistent cuts of $G(V \setminus R, X)$, not whole $G[X]$.

We first note that (again) the second difference is only apparent. Let $X \in \mathcal{R}_W^k$ and $R \subseteq V$ be as in the definition of $\overline{\mathcal{C}}_W^\ell$, i.e., $|R| = \ell$, each $v \in R$ is of outdegree zero in $G[X]$. If $(X, (X_1, X_2)) \in \mathcal{C}_W^k$, then the cut (Y_1, Y_2) where $Y_j = X_j \setminus R$ is consistent with $G(V \setminus R, X)$ and $(X, R, (Y_1, Y_2)) \in \overline{\mathcal{C}}_W^\ell$. In the other direction, observe that if $(X, R, (Y_1, Y_2)) \in \overline{\mathcal{C}}_W^\ell$, then there exists exactly one cut (X_1, X_2) of $G[X]$, such that $v_1 \in X_1$, $Y_1 \subseteq X_1$ and $Y_2 \subseteq X_2$, namely $X_1 = Y_1 \cup (N_{G[X]}(Y_1) \cap R) \cup \{v_1\}$ and $X_2 = Y_2 \cup (N_{G[X]}(Y_2) \cap R)$ (in particular, if $v_1 \in R$, then v_1 is isolated in $G[X]$, and can be put safely to X_1). Thus, in the analysis that follows we can silently assume that (Y_1, Y_2) is in fact a consistent cut of $G[X]$ with $v_1 \in Y_1$.

Let $(X, (X_1, X_2)) \in \mathcal{C}_W^k$. Note that we have exactly $\binom{k}{\ell}$ choices of the set R, such that $(X, R, (X_1 \setminus R, X_2 \setminus R)) \in \overline{\mathcal{C}}_W^\ell$, as any choice of ℓ vertices of outdegree zero in $G[X]$ can be used as R. Thus we have that modulo 2

$$|\overline{\mathcal{C}}_W^\ell| \equiv \sum_{k=\ell}^{|V|-1} \binom{k}{\ell} |\mathcal{S}_W^k| \equiv \sum_{k=\ell}^{|V|-1} \binom{k}{\ell} |\mathcal{C}_W^k|.$$

As in the case of EXACT k-LEAF SPANNING TREE, operating over the field $\mathbb{Z}_2$, we have obtained a linear operator that transforms a vector $(|\mathcal{C}_W^k|)_{k=0}^{|V|-1}$ into a vector $(|\overline{\mathcal{C}}_W^\ell|)_{\ell=0}^{|V|-1}$. Again the matrix of that operator can be computed in polynomial time and is easily inverted (as it is upper triangular with ones on the diagonal). Thus we can compute (in $\mathbb{Z}_2$) all values $(|\mathcal{C}_W^k|)_{k=0}^{|V|-1}$ knowing all values $(|\overline{\mathcal{C}}_W^\ell|)_{\ell=0}^{|V|-1}$.

To finish the proof we need to describe a procedure $\overline{\mathtt{CountC}}(\omega, W, \ell, \mathbb{T})$ that, given a nice tree decomposition $\mathbb{T}$, weight funtion ω and integers W and ℓ computes $|\overline{\mathcal{C}}_W^\ell|$ modulo 2. Now we can use dynamic programming on the tree decomposition.

Recall that in the definition of $\overline{\mathcal{C}}_W^\ell$ the cut (Y_1, Y_2) was a consistent cut of only $G(V \setminus R, X)$. We make use of this fact to reduce the size of the table in the dynamic programming, as we do not need to remember side of the cut for vertices in R.

We follow the notation from STEINER TREE example (see Lemma 3.5). For every bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \leq \ell \leq |V|$, $0 \leq w \leq N|A|$, and $s \in \{\mathbf{1}_1, \mathbf{1}_2, \mathbf{0}\}^{B_x}$, $s_{\text{in}} \in \{\mathbf{0}, \mathbf{1}\}^{B_x}$ (called the colourings) define

$$\mathcal{R}_x(\ell, w) = \Big\{ (X, R) \mid X \subseteq E_x \wedge R \subseteq V_x \wedge |R| = \ell \wedge \omega(X) = w$$

$$\wedge \ (v \in V_x \setminus B_x \Rightarrow \text{indeg}_{G[X]}(v) = [v \neq v_1]) \wedge (v \in B_x \Rightarrow \text{indeg}_{G[X]}(v) \leq [v \neq v_1])$$

$$\wedge \ (v \in R \Rightarrow \text{outdeg}_{G[X]}(v) = 0) \Big\}$$

$$\mathcal{C}_x(\ell, w) = \Big\{ (X, R, (Y_1, Y_2)) \mid (X, R) \in \mathcal{R}_x(\ell, w) \wedge v_1 \notin Y_2$$

$$\wedge \ (Y_1, Y_2) \text{ is a consistent cut of } G(V_x \setminus R, X) \Big\}$$

$$A_x(\ell, w, s, s_{\text{in}}) = \Big| \Big\{ (X, R, (Y_1, Y_2)) \in \mathcal{C}_x(\ell, w) \mid$$

$$(s(v) = \mathbf{1}_j \Rightarrow v \in Y_j) \wedge (s(v) = \mathbf{0} \Rightarrow v \in R) \wedge (\forall_{v \in B_x} s_{\text{in}}(v) = \text{indeg}_{G[X]}(v)) \Big\} \Big|$$

Here $s(v) = \mathbf{0}$ denotes that $v \in R$, whereas $s(v) = \mathbf{1}_j$ denotes that $v \in Y_j$ (and thus $v \notin R$). The value $s_{\text{in}}(v)$ denotes the indegree of v in $G[X]$. The accumulators ℓ and w keep track of the size of R and the weight of X, respectively. Hence $A_x(\ell, w, s, s_{\text{in}})$ reflects the number of partial objects from $\overline{\mathcal{C}}$ with fixed size of R, weight of X and interface on vertices from B_x.

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

- **Leaf bag**:
$$A_x(0, 0, \emptyset, \emptyset) = 1$$

- **Introduce vertex bag**:

$$A_x(\ell, w, s[v \to \alpha], s_{\text{in}}[v \to \mathbf{1}]) = 0$$
$$A_x(\ell, w, d, s[v \to \mathbf{0}], s_{\text{in}}[v \to \mathbf{0}]) = A_y(\ell - 1, w, s, s_{\text{in}})$$
$$A_x(\ell, w, d, s[v \to \mathbf{1}_1], s_{\text{in}}[v \to \mathbf{0}]) = A_y(\ell, w, s, s_{\text{in}})$$
$$A_x(\ell, w, d, s[v \to \mathbf{1}_2], s_{\text{in}}[v \to \mathbf{0}]) = [v \neq v_1] A_y(\ell, w, s, s_{\text{in}})$$

The new vertex has indegree zero and v_1 cannot be put into Y_2.

- **Introduce edge bag**:

$$A_x(\ell, w, s, s_{\text{in}}) = A_y(\ell, w, s, s_{\text{in}}) + A_y(\ell, w - \omega((u, v)), s, s_{\text{in}}[v \to \mathbf{0}])$$
$$\text{if } (s(u) = s(v) = \mathbf{1}_j \vee (s(u) = \mathbf{1}_j \wedge s(v) = \mathbf{0})) \wedge v \neq v_1 \wedge s_{\text{in}}(v) = \mathbf{1}$$
$$A_x(\ell, w, s, s_{\text{in}}) = A_y(\ell, w, s, s_{\text{in}}) \qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise}$$

Here we consider adding the arc (u, v) to X. First, we need that $v \neq v_1$. Second, we need that $u, v \in Y_j$ or $u \in Y_j$ and $v \in R$. Moreover, we need to update the indegree of v and the accumulator keeping the weight of X.

- **Forget bag**:

$$A_x(\ell, w, s, s_{\text{in}}) = \sum_{\alpha \in \{\mathbf{0}, \mathbf{1}_1\}} A_y(\ell, w, s[v \to \alpha], s_{\text{in}}[v \to \mathbf{0}]) \qquad\qquad \text{if } v = v_1$$

$$A_x(\ell, w, s, s_{\text{in}}) = \sum_{\alpha \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}} A_y(\ell, w, s[v \to \alpha], s_{\text{in}}[v \to \mathbf{1}]) \qquad\qquad \text{otherwise}$$

If we forget $v = v_1$, we require that its indegree is zero and $v \notin Y_2$. Otherwise, we require that the indegree of the forgotten vertex is one.

- **Join bag**: Let us define for $T \subseteq B_x$ a colouring $s_{\text{in}}[T]$ as $s_{\text{in}}[T](v) = \mathbf{1}$ if $v \in T$ and $s_{\text{in}}[T](v) = \mathbf{0}$ otherwise. Then

$$A_x(\ell, w, s, s_{\text{in}}) = \sum_{\ell_1 + \ell_2 = \ell + |s^{-1}(\mathbf{0})|} \sum_{w_1 + w_2 = w} \sum_{T_1, T_2 \subseteq B_x}$$
$$[T_1 \cup T_2 = s_{\text{in}}^{-1}(\mathbf{1})][T_1 \cap T_2 = \emptyset] A_y(\ell_1, w_1, s, s_{\text{in}}[T_1]) A_z(\ell_2, w_2, s, s_{\text{in}}[T_2])$$

The colourings s in children need to be the same, whereas the colourings s_{in} in the children need to sum up to the colouring s_{in} in the bag x, i.e., a vertex v has indegree one only if it has indegree one in exactly one of the children bags. Since vertices coloured $\mathbf{0}$ in B_x are accounted for in both tables of the children, we add their contribution to the accumulator ℓ.

To compute the recursion formula efficiently we need to use fast subset convolution. For accumulators ℓ, w and a colouring s we define the following functions on subsets B_x

$$f^{\ell, w, s}(T) = A_y(\ell, w, s, s_{\text{in}}[T]),$$
$$g^{\ell, w, s}(T) = A_z(\ell, w, s, s_{\text{in}}[T]).$$

Now note that

$$A_x(\ell, w, s, s_{\text{in}}) = \sum_{\ell_1 + \ell_2 = \ell + |s^{-1}(\mathbf{0})|} \sum_{w_1 + w_2 = w} (f^{\ell_1, w_1, s} * g^{\ell_2, w_2, s})(s_{\text{in}}^{-1}(\mathbf{1}))$$

By Theorem A.2, for fixed accumulators ℓ_1, w_1, ℓ_2, w_2 and a colouring s the term

$$(f^{\ell_1, w_1, s} * g^{\ell_2, w_2, s})(s_{\text{in}}^{-1}(\mathbf{1}))$$

can be computed in time $2^t t^{O(1)}$ at once for all colourings s_{in}. Thus, the total time consumed by the evaluation of A_x is bounded by $6^t n^{O(1)}$.

It is easy to see that the above recurrence leads to a dynamic programming algorithm that computes the parity of $|\overline{\mathcal{C}}_W^\ell|$ for all values of W and ℓ in $6^t |V|^{O(1)}$ time, since $|\overline{\mathcal{C}}_W^\ell| = A_r(\ell, W, \emptyset, \emptyset)$. Moreover, as we count the parities and not the numbers A_x themselves, all arithmetical operations (in particular the ring operations in the fast subset convolution) can be done in constant time. As discussed before, knowing in $\mathbb{Z}_2$ the values of $|\overline{\mathcal{C}}_W^\ell|$ for $0 \le \ell \le |V| - 1$ we can compute all values of $|\mathcal{C}_W^k|$ and $|\mathcal{S}_W^k|$ modulo 2. Thus, the proof of Theorem A.17 is finished. $\qquad\square$

A.6 MAXIMUM FULL DEGREE SPANNING TREE

In this subsection we solve a bit more general version of MAXIMUM FULL DEGREE SPANNING TREE where the tree in question needs to contain exactly the prescribed number of vertices of full degree.

EXACT FULL DEGREE SPANNING TREE
Input: An undirected graph $G = (V, E)$ and an integer k.
Question: Does there exist a spanning tree T of G for which there are exactly k vertices satisfying $\deg_G(v) = \deg_T(v)$?

Theorem A.18. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves the* EXACT FULL DEGREE SPANNING TREE *problem in $4^t |V|^{O(1)}$ 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 a universe we take the set of edges $U = E$. As usual we assume that we are given a weight function $\omega : U \to \{1, ..., N\}$, where $N = 2|U| = 2|E|$. Let v_1 be an arbitrary vertex.

The Cut part. For an integer W we define:

1. $\mathcal{R}_W$ to be the family of solution candidates of weight W, that is subsets of exactly $|V| - 1$ edges $X \subseteq E$, $|X| = |V| - 1$, $\omega(X) = W$, such that there are exactly k vertices v satisfying $\deg_G(v) = \deg_{G[X]}(v)$,

2. $\mathcal{S}_W$ to be the set of solutions, that is solution candidates $X \in \mathcal{R}_W$ such that the graph $G[X]$ is connected;

3. $\mathcal{C}_W$ 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 the graph $G[X]$.

Observe that for $X \in \mathcal{R}_W$ the graph $G[X]$ is connected iff $G[X]$ is a tree, since $|X| = |V| - 1$. Thus $\mathcal{S}_W$ is indeed a family of solutions of weight W.

The Count part. To use Algorithm 1 we need to formally prove that for any W we have $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$. Similarly as in the case of STEINER TREE we note that by Lemma 3.3 for each $X \in \mathcal{R}_W$ there exist $2^{\mathsf{cc}(G[X])-1}$ consistent cuts of the graph $G[X]$, and the claim follows.

To finish the proof we need to show how to compute $|\mathcal{C}_W|$ modulo 2 in time $4^t n^{O(1)}$ using dynamic programming. In a state we store the number of vertices that are already forgotten and have all their incident edges chosen, the number of already chosen edges, the sum of weights of already chosen edges, and moreover for each vertex of the bag we remember the side of the cut and one bit of information whether there exists some already introduced edge incident with that vertex that was not chosen. Formal definition follows.

We follow the notation from STEINER TREE example (see Lemma 3.5). For a bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \le i \le |V|, 0 \le b < |V|, 0 \le w \le N(|V|-1)$, $s_{\mathrm{cut}} \in \{\mathbf{1}, \mathbf{2}\}^{B_x}$ and $s_{\mathrm{deg}} \in \{\mathbf{0}, \mathbf{1}\}^{B_x}$ (called the colouring) define

$$\mathcal{R}_x(i, b, w) = \Big\{ X \subseteq E_x \mid |X| = b \,\wedge\, \omega(X) = w \,\wedge\, |\{v \in V_x \setminus B_x \,:\, \deg_{G_x}(v) = \deg_X(v)\}| = i \Big\}$$

$$\mathcal{C}_x(i, b, w) = \Big\{ (X, (X_1, X_2)) \mid X \in \mathcal{R}_x(i, b, w) \,\wedge\, v_1 \in X_1 \,\wedge\, (X_1, X_2) \text{ is a consistent cut of } (V_x, X) \Big\}$$

$$A_x(i, b, w, s_{\mathrm{cut}}, s_{\mathrm{deg}}) = \Big| \Big\{ (X, (X_1, X_2)) \in \mathcal{C}_x(i, b, w) \big| (v \in X_j \cap B_x \Rightarrow s_{\mathrm{cut}}(v) = j)$$
$$\wedge\ (s_{\mathrm{deg}}(v) = \mathbf{0} \Rightarrow \deg_{G_x}(v) = \deg_X(v)) \,\wedge\, (s_{\mathrm{deg}}(v) = \mathbf{1} \Rightarrow \deg_{G_x}(v) > \deg_X(v)) \Big\} \Big|$$

By $s_{\mathrm{cut}}(v) = j$ we denote $v \in X_j$, whereas $s_{\mathrm{deg}}(v)$ is equal to one iff there exists an edge in $E_x \setminus X$ that is incident with v. Hence $A_x(i, b, w, s_{\mathrm{cut}}, s_{\mathrm{deg}})$ is the number of pairs from $\mathcal{C}_x(i, b, w)$ with a fixed interface on vertices from B_x.

The algorithm computes $A_x(i, b, w, s_{\mathrm{cut}}, s_{\mathrm{deg}})$ for all bags $x \in \mathbb{T}$ in a bottom-up fashion for all reasonable values of i, b, w, s_{cut} and s_{deg}. We now give the recurrence for $A_x(i, b, w, s_{\mathrm{cut}}, s_{\mathrm{deg}})$ that is used by the dynamic programming algorithm. As usual v denotes the vertex introduced and contained in an introduce bag, uv the edge introduced in an introduce edge bag, and y, z the left and right child of x in $\mathbb{T}$ if present.

- **Leaf bag**:
$$A_x(0, 0, 0, \emptyset, \emptyset) = 1$$

- **Introduce vertex bag**:
$$A_x(i, b, w, s_{\mathrm{cut}}[v \to \mathbf{1}], s_{\mathrm{deg}}[v \to \mathbf{0}]) = A_y(i, b, w, s_{\mathrm{cut}}, s_{\mathrm{deg}})$$
$$A_x(i, b, w, s_{\mathrm{cut}}[v \to \mathbf{2}], s_{\mathrm{deg}}[v \to \mathbf{0}]) = [v \neq v_1] A_y(i, b, w, s_{\mathrm{cut}}, s_{\mathrm{deg}})$$
$$A_x(i, b, w, s_{\mathrm{cut}}[v \to \alpha], s_{\mathrm{deg}}[v \to \mathbf{1}]) = 0$$

We make sure that v_1 belongs to X_1.

52

- **Introduce edge bag**:

$$A_x(i, b, w, s_{\text{cut}}, s_{\text{deg}}) = [s_{\text{cut}}(u) = s_{\text{cut}}(v)]A_y(i, b-1, w - \omega(uv), s_{\text{cut}}, s_{\text{deg}})$$
$$+ \sum_{\alpha_u, \alpha_v \in \{\mathbf{0},\mathbf{1}\}} A_y(i, b, w, s_{\text{cut}}, s_{\text{deg}}[u \to \alpha_u, v \to \alpha_v])$$
$$\text{if } s_{\text{deg}}(u) = s_{\text{deg}}(v) = \mathbf{1}$$
$$A_x(i, b, w, s_{\text{cut}}, s_{\text{deg}}) = [s_{\text{cut}}(u) = s_{\text{cut}}(v)]A_y(i, b-1, w - \omega(uv), s_{\text{cut}}, s_{\text{deg}})$$
$$\text{otherwise}$$

If $s_{\text{deg}}(u) = s_{\text{deg}}(v) = \mathbf{1}$ then we have an option of not taking the edge uv to the set X. In this case, the previous values of $s_{\text{deg}}(u)$ and $s_{\text{deg}}(v)$ can be arbitrary.

- **Forget bag**:

$$A_x(i, b, w, s_{\text{cut}}, s_{\text{deg}}) = \sum_{j \in \{\mathbf{1},\mathbf{2}\}} \sum_{\alpha \in \{\mathbf{0},\mathbf{1}\}} A_y(i - [\alpha = \mathbf{0}], b, w, s_{\text{cut}}[v \to j], s_{\text{deg}}[v \to \alpha])$$

If the vertex v had all incident edges chosen ($\alpha = \mathbf{0}$) then we update the accumulator i.

- **Join bag**:

The only valid combinations to achieve the colouring s_{cut} is to have the same colouring in both children. However we have $s_{\text{deg}}(v) = \mathbf{1}$ in x if and only if $s_{\text{deg}}(v) = \mathbf{1}$ in y or in z. Hence we use a covering product. We somewhat abuse the notation and identify a function s_{deg} with a subset $s_{\text{deg}}^{-1}(\mathbf{1}) \subseteq B_x$. Let us define

$$f^{i,b,w,s_{\text{cut}}}(s_{\text{deg}}) = A_y(i, b, w, s_{\text{cut}}, s_{\text{deg}})$$
$$g^{i,b,w,s_{\text{cut}}}(s_{\text{deg}}) = A_z(i, b, w, s_{\text{cut}}, s_{\text{deg}})$$
$$h^{i,b,w,s_{\text{cut}}}(s_{\text{deg}}) = \sum_{i_1+i_2=i} \sum_{b_1+b_2=b} \sum_{w_1+w_2=w} (f^{i_1,b_1,w_1,s_{\text{cut}}} *_c g^{i_2,b_2,w_2,s_{\text{cut}}})(s_{\text{deg}})$$

Consequently we have

$$A_x(i, b, w, s_{\text{cut}}, s_{\text{deg}}) = h^{i,b,w,s_{\text{cut}}}(s_{\text{deg}})$$

It is easy to see that we can combine the above recurrence with dynamic programming. For each of the $2^t |V|^{O(1)}$ argument values of i,b,w and s_{cut} the covering product $*_c$ can be computed in $2^t |V|^{O(1)}$ by Theorem A.2. Note that as we perform all calculations modulo 2, we take only constant time to perform any arithmetic operation.

Since $|\mathcal{C}_W| = A_r(k, |V| - 1, W, \emptyset, \emptyset)$ the above recurrence leads to a dynamic programming algorithm that computes the parity of $|\mathcal{C}_W|$ (and thus of $|\mathcal{S}_W|$ as well) for all reasonable values of W in $4^t |V|^{O(1)}$ time. Consequently we finish the proof of Theorem A.18. $\qquad\square$

A.7 GRAPH METRIC TRAVELLING SALESMAN PROBLEM

GRAPH METRIC TRAVELLING SALESMAN PROBLEM
Input: An undirected graph $G = (V, E)$ and an integer k.
Question: Does there exist a closed walk (possibly repeating edges and vertices) of length at most k that visits each vertex of the graph at least once?

Theorem A.19. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves the* GRAPH METRIC TRAVELLING SALESMAN PROBLEM *problem in* $4^t |V|^{O(1)}$ *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. Observe that we may assume that G is connected and $k \leq 2(|V| - 1)$ because taking twice all edges of any spanning tree gives a solution.

Since we want to distinguish the case when we take an edge once or twice to the solution, as a universe we take the set $U = E \times \{1, 2\}$, where we use $(e, 1)$ if an edge is chosen once and $(e, 2)$ in case we use e twice. As usual we assume that we are given a weight function $\omega : U \to \{1, ..., N\}$, where $N = 2|U| = 4|E|$. Let v_1 be an arbitrary vertex.

The Cut part. For integers i and W we define:

1. $\mathcal{R}_W^i$ to be the family of solution candidates of size i and weight W, that is functions $\phi \in \{0, 1, 2\}^E$ where $\sum_{e \in E} \phi(e) = i$, $\sum_{e \in E, \phi(e) > 0} \omega((e, \phi(e))) = W$, such that for each vertex $v \in V$ its degree is even, i.e., $|\{uv \in E : \phi(e) = 1\}| \equiv 0$;

2. $\mathcal{S}_W^i$ to be the set of solutions, that is solution candidates $\phi \in \mathcal{R}_W^i$ such that the graph $G[\phi^{-1}(\{1, 2\})]$ is connected;

3. $\mathcal{C}_W^i$ to be the family of pairs $(\phi, (X_1, X_2))$, where $\phi \in \mathcal{R}_W^i$, $v_1 \in X_1$, and (X_1, X_2) is a consistent cut of the graph $G[\phi^{-1}(\{1, 2\})]$.

We want to check whether there exist numbers W and $i \leq k$ such that $\mathcal{S}_W^i \neq \emptyset$.

The Count part. To use Algorithm 1 we need to formally prove that for any i and W we have $|\mathcal{S}_W^i| \equiv |\mathcal{C}_W^i|$. Similarly as in the case of STEINER TREE we note that by Lemma 3.3 for each $\phi \in \mathcal{R}_W^i$ there exist $2^{\mathsf{cc}(G') - 1}$ consistent cuts of the graph $G' = G[\phi^{-1}(\{1, 2\})]$, and the claim follows.

To finish the proof we need to show how to compute $|\mathcal{C}_W^i|$ modulo 2 in time $4^t n^{O(1)}$ using dynamic programming.

We follow the notation from STEINER TREE example (see Lemma 3.5). For a bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \leq i \leq k, 0 \leq w \leq kN$, $s_{\mathrm{cut}} \in \{\mathbf{1}, \mathbf{2}\}^{B_x}$ and $s_{\mathrm{deg}} \in \{\mathbf{0}, \mathbf{1}\}^{B_x}$ (called the colourings) define

$$\mathcal{R}_x(i, w) = \left\{ \phi \in \{0, 1, 2\}^{E_x} \,\Big|\, \sum_{e \in E_x} \phi(e) = i \;\wedge\; \sum_{e \in E_x, \phi(e) > 0} \omega((e, \phi(e))) = w \right.$$

$$\left. \wedge\; \forall_{v \in V_x \setminus B_x} |\{uv \in E_x : \phi(uv) = 1\}| \mod 2 = 0 \right\}$$

$$\mathcal{C}_x(i, w) = \left\{ (\phi, (X_1, X_2)) \,\big|\, \phi \in \mathcal{R}_x(i, w) \;\wedge\; v_1 \in X_1 \;\wedge\; (X_1, X_2) \text{ is a consistent cut of } G_x[\phi^{-1}(\{1, 2\})] \right\}$$

$$A_x(i, w, s_{\mathrm{cut}}, s_{\mathrm{deg}}) = \left| \left\{ (\phi, (X_1, X_2)) \in \mathcal{C}_x(i, w) \big| (s_{\mathrm{cut}}(v) = j \Rightarrow v \in X_j) \right. \right.$$

$$\left. \left. \wedge\; \forall_{v \in B_x} s_{\mathrm{deg}}(v) \equiv |\{uv \in E_x : \phi(uv) = 1\}| \right\} \right|$$

The accumulators i and w keep track of the number of edges chosen (with multiplicities) and the appropriate sum of weights. In the sequence s_{cut} we store the information about the side of the cut of each vertex from B_x, whereas s_{deg} is used to remember whether a vertex has an odd or even degree. Hence $A_x(i, w, s_{\mathrm{cut}}, s_{\mathrm{deg}})$ is the number of pairs from $\mathcal{C}_x(i, w)$ with a fixed interface on vertices from B_x.

The algorithm computes $A_x(i, w, s_{\mathrm{cut}}, s_{\mathrm{deg}})$ for all bags $x \in \mathbb{T}$ in a bottom-up fashion for all reasonable values of i, w, s_{cut} and s_{deg}. We now give the recurrence for $A_x(i, w, s_{\mathrm{cut}}, s_{\mathrm{deg}})$ that is used by the dynamic programming algorithm. As usual let v stand for the vertex introduced and contained in an introduce bag, uv for the edge introduced in an introduce edge bag, and y, z for the left and right child of x in $\mathbb{T}$ if present.

- **Leaf bag**:
$$A_x(0, 0, \emptyset, \emptyset) = 1$$

- **Introduce vertex bag**:
$$A_x(i, w, s_{\mathrm{cut}}[v \to \mathbf{1}], s_{\mathrm{deg}}[v \to \mathbf{0}]) = A_y(i, w, s_{\mathrm{cut}}, s_{\mathrm{deg}})$$
$$A_x(i, w, s_{\mathrm{cut}}[v \to \mathbf{2}], s_{\mathrm{deg}}[v \to \mathbf{0}]) = [v \neq v_1] A_y(i, w, s_{\mathrm{cut}}, s_{\mathrm{deg}})$$
$$A_x(i, w, s_{\mathrm{cut}}[v \to \alpha], s_{\mathrm{deg}}[v \to \mathbf{1}]) = 0$$

We make sure that v_1 belongs to X_1.

- **Introduce edge bag**:

$$A_x(i, w, s_{\text{cut}}, s_{\text{deg}}) = A_y(i, w, s_{\text{cut}}, s_{\text{deg}}) +$$
$$[s_{\text{cut}}(u) = s_{\text{cut}}(v)] A_y(i - 1, w - \omega((uv, 1)), s_{\text{cut}}, s'_{\text{deg}}) +$$
$$[s_{\text{cut}}(u) = s_{\text{cut}}(v)] A_y(i - 2, w - \omega((uv, 2)), s_{\text{cut}}, s_{\text{deg}})$$

where by s'_{deg} we denote s_{deg} with changed values for u and v, formally

$$s'_{\text{deg}} = s_{\text{deg}}[u \to 1 - s_{\text{deg}}(u), v \to 1 - s_{\text{deg}}(v)].$$

We can either not take an edge or take it once or twice.

- **Forget bag**:

$$A_x(i, w, s_{\text{cut}}, s_{\text{deg}}) = \sum_{j \in \{\mathbf{1}, \mathbf{2}\}} A_y(i, w, s_{\text{cut}}[v \to j], s_{\text{deg}}[v \to \mathbf{0}])$$

We simply check the parity of the vertex which we are about to forget, both sides of the cut are allowed.

- **Join bag**:

The only valid combinations to achieve the colouring s_{cut} is to have the same colouring in both children. However for s_{deg} we have to calculate the xor product of $s_{\text{deg}}(v)$ for y and z. We somewhat abuse the notation and identify a function s_{deg} with a subset $s_{\text{deg}}^{-1}(\mathbf{1}) \subseteq B_x$. Let us define

$$f^{i, w, s_{\text{cut}}}(s_{\text{deg}}) = A_y(i, w, s_{\text{cut}}, s_{\text{deg}})$$
$$g^{i, w, s_{\text{cut}}}(s_{\text{deg}}) = A_z(i, w, s_{\text{cut}}, s_{\text{deg}})$$
$$h^{i, w, s_{\text{cut}}}(s_{\text{deg}}) = \sum_{i_1 + i_2 = i} \sum_{w_1 + w_2 = w} (f^{i_1, w_1, s_{\text{cut}}} *_x g^{i_2, w_2, s_{\text{cut}}})(s_{\text{deg}})$$

Consequently we have

$$A_x(i, w, s_{\text{cut}}, s_{\text{deg}}) = h^{i, w, s_{\text{cut}}}(s_{\text{deg}})$$

It is easy to see that we can combine the above recurrence with dynamic programming. For each of the $2^t |V|^{O(1)}$ argument values of i, w and s_{cut} the xor product can be computed in $2^t |V|^{O(1)}$ by Theorem A.6. Note that as we perform all calculations modulo 2, we take only constant time to perform any arithmetic operation.

Since for each i we have $|\mathcal{C}_W^i| = A_r(i, W, \emptyset, \emptyset)$ the above recurrence leads to a dynamic programming algorithm that computes the parity of $|\mathcal{C}_W^i|$ (and thus of $|\mathcal{S}_W^i|$ as well) for all reasonable values of W and i in $4^t |V|^{O(1)}$ time. Consequently we finish the proof of Theorem A.19. $\qquad \square$

B Improvements in FVS, CVC and CFVS parameterized by the solution size

Our technique gives rise to an improvement of several parameterized complexity upper bounds for vertex deletion problems in which the remaining graph has to be of constant treewidth. These problems are FEEDBACK VERTEX SET, CONNECTED VERTEX COVER and CONNECTED FEEDBACK VERTEX SET. The main idea behind the new results is the combination of the *iterative compression* technique, developed by Reed et al. [55], and the Cut&Count technique.

We begin with the FEEDBACK VERTEX SET problem, as it was exhaustively studied by the parameterized complexity community. Let us recall that previously best algorithm, due Cao, Chen and Liu, runs in $3.83^k n^{O(1)}$ time [14].

Theorem B.1 (Theorem 1.6, restated). *There exists a Monte-Carlo algorithm for the* FEEDBACK VERTEX SET *problem in 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 $1 \leq i \leq n$. Observe that if G admits a feedback vertex set of size at most k, i.e. there is a set $A \subseteq V, |A| \leq k$ 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 $|A \cap \{v_1, v_2, \ldots, v_i\}| \leq |A| \leq k$.

We construct feedback vertex sets $A_1, A_2, \ldots, A_n$ of size at most k consecutively in $G_1, G_2, \ldots, G_n = G$. 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 graph G_1 (we ignore the trivial case $k = 0$). 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 $|B_{i+1}| \leq k$, then we take $A_{i+1} = B_{i+1}$. Thus we are left with the case in which, given a feedback vertex set of size $k + 1$, we need to construct a feedback vertex set of size at most k or determine that none such exists. Denote this given feedback vertex set by B.

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. To begin with, we test whether G_{i+1} admits a feedback vertex set of size at most k. We apply (using the tree decomposition obtained above as the input) the dynamic programming algorithm described in Section A.2, running in $3^k n^{O(1)}$ time, which tests whether the graph admits a feedback vertex set of size at most k. Observe that this algorithm, as described in the proof of Theorem A.8, 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 $\bar{s} : B \to \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}$ 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 $s|_B = \bar{s}$. Such a computation takes polynomial time and space. As there are 3^{k+1} such possible evaluations $\bar{s}$, 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 A.8. The algorithm considers the vertices of G_{i+1} one by one, building a set K which at the end will be the constructed A_{i+1}. 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 A.8 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 occured. 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 A.8 as a subroutine: in each of n steps of the iterative compression at most $n + 1$ times. Each of these groups of runs has a 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 large enough n. $\qquad\square$

Now we proceed to the algorithm for CONNECTED VERTEX COVER. The previously best FPT algorithm is due to Binkele-Raible [5], and runs in $2.4882^k n^{O(1)}$ time complexity. The following algorithm is also an application of 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 B.2. *There exists a Monte-Carlo algorithm for the* CONNECTED VERTEX COVER *problem in 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 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 multiedges 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.

Therefore, we can consider a sequence of graphs $G_1, G_2, \ldots, G_n = G$ (G is the connected graph given in the input), where G_i is obtained from G_{i+1} by contracting any edge and reducing possible multiedges to simple edges, and G_1 is a graph composed of a single vertex. The argument from the last paragraph ensures that we can proceed as in the proof of Theorem B.1, namely construct connected vertex covers for $G_1, G_2, \ldots, G_n$ consecutively, and the only thing we have to 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 constructed from G_{i+1} by contracting an edge uv. We construct 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 vertex cover of G_{i+1}. As $V(G_{i+1}) \setminus B$ is an independent set, then 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 A.10. As in the proof of Theorem B.1, this dynamic programming algorithm during computation of $A_x(i, w, s)$ also 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 A.10 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 $3^3 \cdot 2^{k-1}$. B induces in G_{i+1} a graph with consisting of a single large connected component (coming from A_i), and at most two additional vertices. Take any spanning tree of the large component and root it at some vertex r. We present the evaluation $\bar{s}$ in the following manner. For the root r and the two additional vertices we choose for $\bar{s}$ any value from $\{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}$, giving 3^3 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 evaluation, 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 from the starting 3, which have 3 options each. This means we only need to consider $3^3 \cdot 2^{k-1}$ 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 B.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 constructed connected vertex cover by running the algorithm from Theorem A.10 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 A.10. Therefore the probability of a false negative is bounded by $\frac{n^2+n}{2^n}$ which is less than $\frac{1}{2}$ for large enough n. $\qquad\square$

Finally, we use a similar technique to obtain an algorithm for CONNECTED FEEDBACK VERTEX SET. The previously best FPT algorithm is due to Misra et al. [49], and runs in $46.2^k n^{O(1)}$ time complexity.

Theorem B.3. *There exists a Monte-Carlo algorithm solving the* CONNECTED FEEDBACK VERTEX SET *problem in 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. Similarly as in the proof of Theorem B.2, the CONNECTED FEEDBACK VERTEX SET problem is also *contraction*–closed. Consider any graph H and obtain H' by contracting an edge uv into a vertex w. Consider a connected feedback vertex set A of a graph H of size at most k and construct a set $A' \subseteq V(H')$ as following:

- if $u, v \notin A$ then $A' = A$;

- otherwise $A' = (A \cup \{w\}) \setminus \{u, v\}$.

It can be easily seen that A' is a connected feedback vertex set of H' of size at most k.

This observation enables us to use iterative compression approach, similarly as in the proof of Theorem B.2. Namely we consider a sequence of graphs $G_1, G_2, \dots, G_n = G$ (G is the connected graph given in the input), where G_i is obtained from G_{i+1} by contracting any edge and reducing possible multiedges to simple edges. For every G_i we try to construct a connected feedback vertex set A_i in a consecutive manner and if at any step we fail, we can safely answer NO. Thus we need to show a way of constructing a connected feedback vertex set of size k in G_{i+1} given a connected feedback vertex set A_i of size k in G_i, or determining that none exists.

Let G_i be constructed from G_{i+1} by contracting an edge uv. We construct B from A_i by removing the vertex obtained in the contraction (if it was included in A_i) and inserting both u and v. Observe that B is a feedback vertex set of G_{i+1} of size at most $k+2$ containing a connected component of size at least $|B|-2$. Therefore, we can construct a tree decomposition of graph G_{i+1} of width $k+3$ by constructing the tree decomposition of width 1 of the forest $G_{i+1} \setminus B$ and including B into every bag.

Now we are going to test whether G_{i+1} admits a connected feedback vertex set of size at most k. A straightforward application of the algorithm from Theorem A.13 would yield an algorithm with running time $4^k n^{O(1)}$. This algorithm once again has the property of referring only to previously computed values with the same evaluation on the intersection of the domains, so we can also apply the method already used in proofs of Theorems B.1 and B.2 to reduce the space usage to polynomial.

Once again, using the special structure of the set B we can also reduce the time complexity down to $3^k n^{O(1)}$ by bounding the number of reasonable evaluations $\overline{s} : B \to \{\mathbf{0}_1, \mathbf{0}_2, \mathbf{1}_1, \mathbf{1}_2\}$ by $4^3 3^{k-1}$. B is a graph consisting of a large connected component and at most two additional vertices. Take any spanning tree of the connected component and root it in a vertex r. Each reasonable evaluation $\overline{s}$ can be coded in the following manner: vertex r and the two possible additional vertices have 4 possibilities of the value in $\overline{s}$, but every other vertex in the tree has only three possibilities, depending on the value $\overline{s}(u)$, where u denotes the parent of v:

- if $\overline{s}(u) = \mathbf{0}_j$, the possibilities are $\mathbf{1}_1$, $\mathbf{1}_2$ and $\mathbf{0}_j$;

- if $\overline{s}(u) = \mathbf{1}_j$, the possibilities are $\mathbf{0}_1$, $\mathbf{0}_2$ and $\mathbf{1}_j$;

as otherwise the cut could not be consistent. Thus every vertex from B has only 3 possibilities, apart from at most 3, which have 4 possibilities. So the number of reasonable evaluations $\overline{s}$ is bounded by $4^3 3^{k-1}$, thus the testing algorithm runs in time complexity $3^k n^{O(1)}$ and uses polynomial space. Again we make n independent runs of the algorithm in order to reduce the probability of a false negative to at most $\frac{1}{2^n}$.

The idea of reconstructing the solution is the same as in the proofs of Theorems B.1 and B.2. We consider vertices one by one iteratively constructing a connected feedback vertex set. At each step we determine whether the considered vertex can or cannot be taken as the next vertex of the so far built part of the solution, using the algorithm for CONSTRAINED CONNECTED FEEDBACK VERTEX SET obtained in Theorem A.13. If it can, we take it, otherwise we just proceed to the next vertex. At each step we make n independent runs to reduce the probability of a false negative to at most $\frac{1}{2^n}$. If the graph admitted a connected feedback vertex set of size at most k, we will construct it in this manner unless at least one test gives a false negative. Again, using previous observations the computation in each of the runs can be reordered so that the running time is $3^k n^{O(1)}$ and the space usage is polynomial.

Again, the union bound proves that the probability of obtaining a false negative in any of the tests is bounded by $\frac{n^2+n}{2^n}$ which for large enough n is lower than $\frac{1}{2}$, as we make at most $n^2 + n$ groups of n independent runs of the algorithm from Theorem A.13. $\qquad\square$

C Negative results under ETH

In this section we provide an evidence that the problems, where we want not to minimize, but to maximize the number of connected components, are harder, in the sense, that they probably do not admit algorithms running in time $2^{o(p \log p)}n^{O(1)}$, where p denotes the pathwidth of the input graph. More precisely, we show that assuming ETH there do not exist algorithms for CYCLE PACKING, MAX CYCLE COVER and MAXIMALLY DISCONNECTED DOMINATING SET running in time $2^{o(p \log p)}n^{O(1)}$.

Let us recall formal problem definitions. The first two problems have undirected and directed versions.

CYCLE PACKING
Input: A (directed or undirected) graph $G = (V, E)$ and an integer ℓ
Question: Does G contain ℓ vertex-disjoint cycles?

MAX CYCLE COVER
Input: A (directed or undirected) graph $G = (V, E)$ and an integer ℓ
Question: Does G contain a set of at least ℓ vertex-disjoint cycles such that each vertex of G is contained in exactly one cycle?

MAXIMALLY DISCONNECTED DOMINATING SET
Input: An undirected graph $G = (V, E)$ and integers ℓ and r
Question: Does G contain a dominating set of size at most ℓ that induces **at least** r connected components?

We prove the following theorem.

Theorem C.1 (Theorem 5.1 restated). *Assuming ETH, there is no $2^{o(p \log p)}n^{O(1)}$ time algorithm for* CYCLE PACK-ING, MAX CYCLE COVER *(both in the directed and undirected setting) nor for* MAXIMALLY DISCONNECTED DOM-INATING SET. *The parameter p denotes the width of a given path decomposition of the input graph.*

We start our reductions from $k \times k$ HITTING SET and $k \times k$ PERMUTATION HITTING SET problems. As we discussed in Section 5.1, the non-permutation version was introduced and analyzed by Lokshtanov et al. [48]. We denote $[k] = \{1, 2, \ldots, k\}$. In the set $[k] \times [k]$ *a row* is a set $\{i\} \times [k]$ and *a column* is a set $[k] \times \{i\}$ (for some $i \in [k]$). We include formal definitions for sake of completeness.

$k \times k$ HITTING SET
Input: A family of sets $S_1, S_2 \ldots S_m \subseteq [k] \times [k]$, such that each set contains at most one element from each row of $[k] \times [k]$.
Question: Is there a set S containing exactly one element from each row such that $S \cap S_i \neq \emptyset$ for any $1 \leq i \leq m$?

$k \times k$ PERMUTATION HITTING SET
Input: A family of sets $S_1, S_2 \ldots S_m \subseteq [k] \times [k]$, such that each set contains at most one element from each row of $[k] \times [k]$.
Question: Is there a set S containing exactly one element from each row and exactly one element from each column such that $S \cap S_i \neq \emptyset$ for any $1 \leq i \leq m$?

Theorem C.2 ([48], Theorem 2.4). *Assuming ETH, there is no $2^{o(k \log k)}n^{O(1)}$ time algorithm for $k \times k$ HITTING SET nor for $k \times k$ PERMUTATION HITTING SET.*

Note that in [48] the statement of the above theorem only includes $k \times k$ HITTING SET. However, the proof in [48] works for the permutation variant as well without any modifications.

We first prove the bound for MAXIMALLY DISCONNECTED DOMINATING SET by quite simple reduction from $k \times k$ HITTING SET. This is done in Section C.1. Then, in Section C.2 we prove the bound for undirected CYCLE PACKING, by quite involved reduction from $k \times k$ PERMUTATION HITTING SET. In Section C.3 we provide a reduction to directed CYCLE PACKING and in Section C.4 we provide a reduction to MAX CYCLE COVER in both variants.

C.1   MAXIMALLY DISCONNECTED DOMINATING SET

In this subsection we provide a reduction from $k \times k$ HITTING SET to MAXIMALLY DISCONNECTED DOMINATING SET. We are given an instance $(k, S_1, \ldots, S_m)$ of $k \times k$ HITTING SET, called the initial instance, and we are to construct an equivalent instance (G, ℓ, r) of MAXIMALLY DISCONNECTED DOMINATING SET.

We first set $\ell := 3k + m$ and $r := k$.

C.1.1   Gadgets

We introduce a few simple gadgets used repeatedly in the construction. In all definitions $H = (V, E)$ is an undirected graph, and the parameters ℓ and r are fixed.

Definition C.3. *By adding a* force gadget *for vertex $v \in V$ we mean the following construction: we introduce $\ell + 1$ new vertices of degree one, connected to v.*

Lemma C.4. *If graph G is constructed from graph $H = (V, E)$ by adding a force gadget to vertex $v \in V$, then v is contained in each dominating set in G of size at most ℓ.*

Proof. If D is a dominating set in G, and $v \notin D$, then all new vertices added in the force gadget need to be included in D. Thus $|D| \geq \ell + 1$. □

Definition C.5. *By adding a* one-in-many gadget *to vertex set $X \subseteq V$ we mean the following construction: we introducte $\ell + 1$ new vertices of degree $|X|$, connected to all vertices in X.*

Lemma C.6. *If graph G is constructed from graph $H = (V, E)$ by adding a one-in-many gadget to vertex set $X \subseteq V$, then each dominating set in G of size at most ℓ contains a vertex from X.*

Proof. If D is a dominating set in G, and $X \cap D = \emptyset$, then all new vertices added in the one-in-many gadget need to be included in D. Thus $|D| \geq \ell + 1$. □

We conclude with the pathwidth bound.

Lemma C.7. *Let G be a graph and let G' be a graph constructed from G by adding multiple force and one-in-many gadgets. Assume we are given a path decomposition of G of width p with the following property: for each one-in-many gadget, attached to vertex set X, there exists a bag in the path decomposition that contains X. Then, in polynomial time, we can construct a path decomposition of G' of width at most $p + 1$.*

Proof. Let w be a vertex in G', but not in G, i.e., a vertex added in one of the gadgets. By the assumptions of the lemma, there exists a bag V_w in the path decomposition of G that contains $N(w)$. For each such vertex w, we introduce a new bag $V'_w = V_w \cup \{w\}$ and we insert it into the path decomposition after the bag V_w. If V_w is multiplied for many vertices w, we insert all the new bags after V_w in an arbitrary order.

It is easy to see that the new path decomposition is a proper path decomposition of G', as V'_w covers all edges incident to w. Moreover, we increased the maximum size of bags by at most one, thus the width of the new decomposition is at most $p + 1$. □

C.1.2 Construction

Let $S_i^{\text{row}} = \{i\} \times [k]$ be a set containing all elements in the i-th row in the set $[k] \times [k]$. We denote $\mathcal{S} = \{S_s : 1 \leq s \leq m\} \cup \{S_i^{\text{row}} : 1 \leq i \leq k\}$. Note that for each $A \in \mathcal{S}$ we have $|A| \leq k$, as each set S_i contains at most one element from each row.

First let us define the graph H. We start by introducing vertices p_i^L for $1 \leq i \leq k$ and vertices p_j^R for $1 \leq j \leq k$. Then, for each set $A \in \mathcal{S}$ we introduce vertices $x_{i,j}^A$ for all $(i,j) \in A$ and edges $p_i^L x_{i,j}^A$ and $p_j^R x_{i,j}^A$. Let $X^A = \{x_{i,j}^A : (i,j) \in A\}$.

To construct graph G, we attach the following gadgets to graph H. For each $1 \leq i \leq k$ and $1 \leq j \leq k$ we attach force gadgets to vertices p_i^L and p_j^R. Moreover, for each $A \in \mathcal{S}$ we attach one-in-many gadget to the set X^A.

We now provide a pathwidth bound on the graph G.

Lemma C.8. *The pathwidth of G is at most $3k$.*

Proof. First consider the following path decomposition of H. For each $A \in \mathcal{S}$ we create a bag

$$V_A = \{p_i^L : 1 \leq i \leq k\} \cup \{p_j^R : 1 \leq j \leq k\} \cup \{x_{i,j}^A : (i,j) \in A\}.$$

The path decomposition of H consists of all bags V_A for $A \in \mathcal{S}$ in an arbitrary order. Note that the above path decomposition is a proper path decomposition of H of width at most $3k - 1$ (as $|A| \leq k$ for each $A \in \mathcal{S}$) and it satisfies conditions for Lemma C.7. $\qquad\square$

C.1.3 From hitting set to dominating set

Lemma C.9. *If the initial $k \times k$ HITTING SET instance was a YES-instance, then there exists a dominating set D in the graph G, such that $|D| = \ell$ and D induces exactly r connected components.*

Proof. Let S be a solution to the initial $k \times k$ HITTING SET instance $(k, S_1, \ldots, S_m)$. For each $A \in \mathcal{S}$ fix an element $(i_A, j_A) \in S \cap A$. Recall that S contains exactly one element from each row, thus $S \cap A \neq \emptyset$ for all sets $A \in \mathcal{S}$. Let us define:
$$D = \{p_i^L : 1 \leq i \leq k\} \cup \{p_j^R : 1 \leq j \leq k\} \cup \{x_{i_A,j_A}^A : A \in \mathcal{S}\}.$$

First note that $|D| = 3k + m$, as there are k vertices p_i^L, k vertices p_j^R, and $|\mathcal{S}| = k + m$, since $\mathcal{S}$ consists of m sets S_s and k sets S_i^{row}.

Let us now check whether D is a dominating set in G. Vertices p_i^L and p_j^R for $1 \leq i, j \leq k$ dominate all vertices of the graph H and all vertices added in the attached force gadgets. Moreover, $D \cap X^A = \{x_{i_A,j_A}^A\}$ for each $A \in \mathcal{S}$, thus D dominates all vertices added in one-in-many gadgets attached to sets X^A.

We now prove that $G[D]$ contains exactly $r = k$ connected components. Let us define for each $1 \leq j \leq k$:

$$D_j = \{p_j^R\} \cup \{p_i^L : (i,j) \in S\} \cup \{x_{i_A,j_A}^A : A \in \mathcal{S}, j_A = j\}.$$

Note that D_j is a partition of D into k pairwise disjoint sets. Moreover, observe that $G[D_j]$ is connected and, since S contains exactly one element from each row, no vertices from D_j and $D_{j'}$ are adjacent, for $j \neq j'$. This finishes the proof of the lemma. $\qquad\square$

C.1.4 From dominating set to hitting set

Lemma C.10. *If there exists a dominating set D in the graph G, such that $|D| \leq \ell$ and D induces at least r connected components, then the initial $k \times k$ HITTING SET instance was a YES-instance.*

Proof. By the properties of the force gadget, D needs to include all forced vertices, i.e., vertices p_i^L and p_j^R for $1 \leq i, j \leq k$. There are $2k$ forced vertices, thus we have $\ell - 2k = k + m$ vertices left.

By the properties of one-in-many gadgets, D needs to include at least one vertex from each set X^A, $A \in \mathcal{S}$. But $|\mathcal{S}| = k + m$ and sets X^A are pairwise disjoint. Thus, D consist of all forced vertices and exactly one vertex from each set X^A, $A \in \mathcal{S}$.

For each $1 \leq i \leq k$ let $x_{i,f(i)}^{S_i^{\mathrm{row}}}$ be the unique vertex in $D \cap X^{S_i^{\mathrm{row}}}$. Let $S = \{(i, f(i)) : 1 \leq i \leq k\}$. We claim that S is a solution to the initial $k \times k$ HITTING SET instance. It clearly contains exactly one element from each row.

Let D_j be the vertex set of the connected component of $G[D]$ that contains p_j^R. Note that $p_i^L \in D_j$ whenever $j = f(i)$, i.e., $(i, j) \in S$. This implies that $\bigcup_{j=1}^{k} D_j$ contains all vertices p_i^L. Moreover, as each vertex in X^A for $A \in \mathcal{S}$ is adjacent to some vertex p_j^R, the sets D_j are the only connected components of $G[D]$. As $G[D]$ contains at least $r = k$ connected components, $D_j \neq D_{j'}$ for $j \neq j'$.

Let $1 \leq s \leq m$ and let us focus on set $S_s \in \mathcal{S}$. Let $x_{i,j}^{S_s}$ be the unique vertex in $D \cap X^{S_s}$. Note that $x_{i,j}^{S_s}$ connects $p_i^L \in D_{f(i)}$ with $p_j^R \in D_j$. As sets D_j are pairwise distinct, this implies that $j = f(i)$ and $(i, j) \in S \cap S_s$. Thus the components of $G[D]$ are exactly the sets D_j, $1 \leq j \leq k$. $\qquad \square$

C.2 Undirected CYCLE PACKING

C.2.1 Proof overview and preliminaries

First note that for CYCLE PACKING, we can assume that the input graph may be a multigraph, i.e., it may contain multiple edges and loops. The following lemma summarizes this observation.

Lemma C.11. *Let (G, ℓ) be an instance of (directed or undirected) CYCLE PACKING, where G may contain multiple edges and loops. Then we can construct in polynomial time an equivalent (directed or undirected, respectively) instance (G', ℓ), such that G' does not contain multiple edges nor loops. Moreover, given a path decomposition of G of width p, we can construct in polynomial time a path decomposition of G' of width at most $p + 2$.*

Proof. To construct G', we replace each edge $e \in E(G)$ with a path of length three, i.e., we insert vertices u_e^1 and u_e^2 in the middle of edge e. Clearly G is a simple graph and vertex-disjoint cycle families in G and G' naturally translates into each other.

We are left with the pathwidth bound. Assume we have a path decomposition of G of width p. For each edge $e \in E(G)$ we fix a bag V_e that covers e. We introduce a new bag $V_e' = V_e \cup \{u_e^1, u_e^2\}$ and insert V_e' near the bag V_e in the path decomposition. It is easy to see that the new decomposition is a proper path decomposition of G' and its width is at most $p + 2$. $\qquad \square$

Let us introduce some extra notation. We say that a vertex is covered by a cycle (or a family of cycles) if the vertex belongs to the cycle (or belongs to at least one cycle in the family). A graph is covered by a cycle family if every its vertex is covered by the family. By $(v_1, \ldots, v_r)$ we denote a path (or a cycle) consisting of vertices $v_1, v_2, \ldots, v_r$ in this order.

We now present an overview of the proof of Theorem 5.1 for undirected CYCLE PACKING. We provide a construction that, given an instance $(k, S_1, S_2, \ldots, S_m)$ of $k \times k$ PERMUTATION HITTING SET (called an *initial instance*), produces in polynomial time an undirected graph G, an integer ℓ and a path decomposition of G with the following properties:

1. The path decomposition of G has width $O(k)$.

2. If the initial instance of $k \times k$ PERMUTATION HITTING SET is a YES-instance, then there exists a family of ℓ vertex-disjoint cycles in G. In other words, (G, ℓ) is a YES-instance of undirected CYCLE PACKING.

3. If there exist a family of ℓ vertex-disjoint cycles in G, then the initial $k \times k$ PERMUTATION HITTING SET instance is a YES-instance.

In Section C.2.2 we describe the *r-in-many gadget*, a tool used widely in the construction. In Section C.2.3 we give the construction of the graph G and show the pathwidth bound, i.e., Point 1. Points 2 and 3 are proven in Sections C.2.4 and C.2.5 respectively. Reductions to directed CYCLE PACKING and to MAX CYCLE COVER are in Sections C.3 and C.4 respectively.

C.2.2 r-in-many gadget

In this section we describe the r-*in-many gadget*, a tool used in further sections. Informally speaking, the r-in-many gadget attached to vertex set X ensures that at most r vertices from X are used in a solution (a family of cycles).

Definition C.12. *Let H be a multigraph and X be an arbitrary subset of vertices of H. By a r-in-many gadget attached to X ($1 \le r < |X|$) we mean the following construction: we introduce $(|X| - r)$ new vertices $\{u_i : 1 \le i \le |X| - r\}$ and for each $1 \le i \le |X| - r$ and $x \in X$ we add two edges xu_i. In other words, we introduce $|X| - r$ vertices connected to the set X via double edges. The set X is called an* attaching point *of the gadget. A cycle of length two, constisting of two edges xu_i for some $1 \le i \le |X| - r$ and $x \in X$, is called a* gadget short cycle.

Definition C.13. *Let H be a multigraph, let $X_1, \ldots, X_d$ be pairwise disjoint subsets of vertices of H and let $r_1, \ldots, r_d$ be integers satisfying $1 \le r_i < |X_i|$ for $1 \le i \le d$. Let G be a multigraph constructed from H by attaching to X_i a r_i-in-many gadget, for all $1 \le i \le d$. We say that G is a* gadget extension *of H. The maximum number of gadget short cycles that can be packed in G is denoted by ℓ_G, i.e., $\ell_G := \sum_{i=1}^{d} |X_i| - r_i$.*

Definition C.14. *Let G be a gadget extension of H, and let X_i and r_i be as in Definition C.13. If $\mathcal{C}_H$ is a family of vertex-disjoint cycles in H satisfying the following property: for each $1 \le i \le d$ at most r_i vertices from X_i are covered by $\mathcal{C}_H$, then we say that $\mathcal{C}_H$ is* gadget safe *in H. If $\mathcal{C}_G$ is a family of vertex-disjoint cycles in G containing ℓ_G gadget short cycles, then we say that $\mathcal{C}_G$ is* gadget safe *in G.*
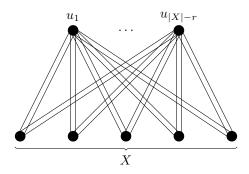
Figure 2: The r-in-many gadget attached to set X.

The following lemma shows how the r-in-many gadget is intended to be used.

Lemma C.15. *Let G be a gadget extension of H, and let X_i and r_i be as in Definition C.13. Let $\mathcal{C}_H$ be a family of cycles that is gadget safe in H. Then $\mathcal{C}_H$ can be extended to gadget safe in G family $\mathcal{C}_G$ of size $|\mathcal{C}_H| + \ell_G$*

Proof. For each $1 \le i \le d$, let $Y_i \subseteq X_i$ be a set of (arbitrarily chosen) $|X_i| - r_i$ vertices not covered by $\mathcal{C}_H$. Assign $\mathcal{C}_G := \mathcal{C}_H$. For each $1 \le i \le d$ we add to $\mathcal{C}_G$ a set of $|X_i| - r_i$ gadget short cycles, each consisting of one vertex in Y_i and one vertex u_j from the gadget attached to X_i. □

The next lemma shows that we can safetely assume that the r-in-many gadgets are used as in the proof of Lemma C.15.

Lemma C.16. *Let G be a gadget extension of H, and let X_i and r_i be as in Definition C.13. Let ℓ be the maximum possible cardinality of a family of vertex-disjoint cycles in G. Then there exists a gadget safe in G family $\mathcal{C}$ of size ℓ. Moreover, after removing from $\mathcal{C}$ all ℓ_G gadget short cycles, we obtain a gadget safe in H family of cycles.*

Proof. Let $\mathcal{C}$ be a family of vertex-disjoint cycles in G of size ℓ that maximizes the number of gadget short cycles. By contradiction, assume that $\mathcal{C}$ is not gadget safe in G. That means it contains less than $\sum_{i=1}^{d} |X_i| - r_i$ gadget short cycles, i.e., there exists $1 \le i \le d$, such that less than $|X_i| - r_i$ gadget short cycles in the gadget attached to X_i are in $\mathcal{C}$.

Let u be a vertex in the gadget attached to X_i that does not lie on a gadget short cycle in $\mathcal{C}$. If u is covered by a cycle $C \in \mathcal{C}$, then there exists $x \in X_i$ also covered by C. We can replace C with a gadget short cycle (u, x), increasing the number of gadget short cycles in $\mathcal{C}$, a contradiction.

Thus, u is not covered in $\mathcal{C}$. Let $x \in X_i$ be a vertex that is not covered by a gadget short cycle in $\mathcal{C}$ (there exists, as $r_i < |X_i|$ and sets X_i are pairwise disjoint). If x is not covered by $\mathcal{C}$, we can add gadget short cycle (u, x) to $\mathcal{C}$, increasing its size, a contradiction. Otherwise, we can replace the cycle with x with gadget short cycle (u, x), a contradiction too. Thus, $\mathcal{C}$ contains $\ell_G = \sum_{i=1}^{d} |X_i| - r_i$ gadget short cycles. It is a straightforward corollary from the definitions that after removing these ℓ_G cycles, we obtain a gadget safe in H family of cycles. $\qquad\square$

Finally, we show that attaching a r-in-many gadget may not influence much the pathwidth of the graph.

Lemma C.17. *Let G be a gadget extension of H, and let X_i and r_i be as in Definition C.13. Assume that we are given a path decomposition of H of width p, such that for each $1 \le i \le d$ there exists a bag V_i that contains the whole X_i. Then in polynomial time we can construct a path decomposition of G of width at most $p + 1$.*

Proof. Let $1 \le i \le d$ and let V_i be a bag containing X_i. We introduce bags V_i^j, $1 \le j \le |X_i| - r_i$, taking $V_i^j = V_i \cup \{u_j\}$. We insert the newly created bags V_i^j near the bag V_i in the path decomposition. As all bags V_i^j contain V_i, this modification does not spoil the properties of the path decomposition of H. Bag V_i^j covers all edges incident to u_j. Thus, the new path decomposition is a proper path decomposition of G and has width at most $p + 1$, as $|V_i^j| = |V_i| + 1$. $\qquad\square$

C.2.3 Construction

Let $(k, S_1, S_2, \ldots, S_m)$ be an instance of $k \times k$ PERMUTATION HITTING SET. W.l.o.g. we may assume that each set S_i is nonempty. We first construct a graph H as follows:

1. The vertex set $V(H)$ consists of

   (a) vertices $p_i^Z, p_i^R, q_i^Z, q_i^R$ for $1 \le i \le k$;

   (b) vertices $p_{i,j}^C, q_{i,j}^C$ for $1 \le i, j \le k$; for each $1 \le i \le k$ we denote $X_i^p = \{p_{i,j}^C : 1 \le j \le k\}$ and $X_i^q = \{q_{i,j}^C : 1 \le j \le k\}$;

   (c) vertices $x_{i,s}^L, x_{i,s}^R, y_{i,s}^L, y_{i,s}^R$ for $1 \le i \le k$ and $1 \le s \le m$;

   (d) and vertices $x_{i,s}^C, y_{i,s}^C, x_{i,s}^Z, y_{i,s}^Z, z_{i,s}^C$ for $1 \le s \le m$ and $(i, j) \in S_s$ (recall that there is at most one element in each row in S_s); we denote $X_s^x = \{x_{i,s}^C : (i, j) \in S_s\}$, $X_s^y = \{y_{i,s}^C : (i, j) \in S_s\}$ and $X_s^z = \{z_{i,s}^C : (i, j) \in S_s\}$.

   The vertex set is partitioned into four parts L, R, C and Z, according to the superscripts (the first three are acronyms for left, right and centre, the last one should be seen as an important separator between left and centre).

2. Vertices p_i^Z and p_j^R are connected into full bipartite graph with vertices $p_{i,j}^C$ inserted into the middle of each edge, i.e., for all $1 \le i, j \le k$ we add edges $p_i^Z p_{i,j}^C$ and $p_{i,j}^C p_j^R$. Similar construction is performed for vertices q_i^Z, q_j^R and $q_{i,j}^C$, i.e., for all $1 \le i, j \le k$ we add edges $q_i^Z q_{i,j}^C$ and $q_{i,j}^C q_j^R$.

3. For each $1 \le i \le k$, vertices $x_{i,s}^L$ and $y_{i,s}^L$ are arranged into path from p_i^Z to q_i^Z, i.e., $x_{i,s}^L y_{i,s}^L \in E$ for $1 \le s \le m$, $y_{i,s}^L x_{i,s+1}^L \in E$ for $1 \le s < m$ and $q_i^Z y_{i,m}^L, p_i^Z x_{i,1}^L \in E$. By $\mathcal{P}_i^L$ we denote the path from p_i^Z to q_i^Z

4. For each $1 \le i \le k$, vertices $x_{i,s}^R$ and $y_{i,s}^R$ are arranged into path from p_i^R to q_i^R, i.e., $x_{i,s}^R y_{i,s}^R$ for $1 \le s \le m$, $y_{i,s}^R x_{i,s+1}^R \in E$ for $1 \le s < m$ and $p_i^R x_{i,1}^R, q_i^R y_{i,m}^R \in E$. By $\mathcal{P}_i^R$ we denote the path from p_i^R to q_i^R.

5. For each $1 \le s \le m$, if $(i, j) \in S_s$, we add a path $(x_{i,s}^L, x_{i,s}^Z, x_{i,s}^C, x_{j,s}^R)$.

6. Similarly, for each $1 \le s \le m$, if $(i, j) \in S_s$, we add a path $(y_{i,s}^L, y_{i,s}^Z, y_{i,s}^C, y_{j,s}^R)$.

7. Moreover, for each $1 \leq s \leq m$ and $(i, j) \in S_s$ we add a cycle $(x_{i,s}^Z, z_{i,s}^C, y_{i,s}^Z)$.

The graph G is defined as a gadget extension of H by attaching to H the following r-in-many gadgets:

1. to vertex sets X_i^p and X_i^q for $1 \leq i \leq k$ we attach 1-in-many gadgets;

2. to vertex sets X_s^x and X_s^y for $1 \leq s \leq m$ we attach 1-in-many gadgets;

3. for $1 \leq s \leq m$ we attach a $(|X_s^z| - 1)$-in-many gadget to X_s^z.
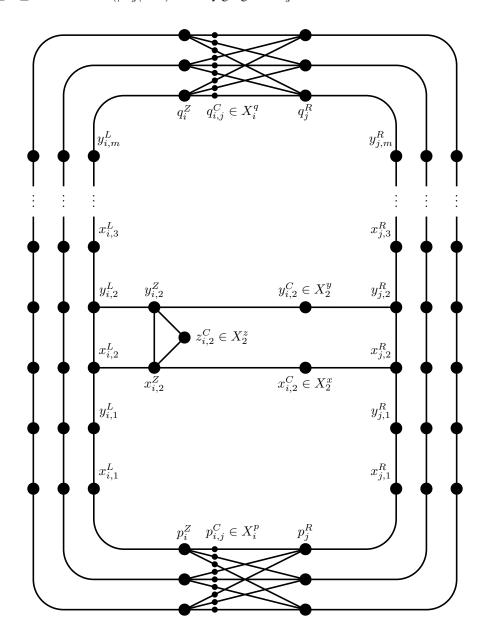
Figure 3: The part of graph H with the main frame and the part for element $(i, j) \in S_s$. Recall that the gadget safe in H family may cover at most one element of X_i^p, X_i^q, X_s^x and X_s^y and cannot cover whole X_s^z.

Clearly, the above construction can be done in polynomial time. Note that we can pack the following number of gadget short cycles in G:

$$\ell_G := \sum_{i=1}^{k}(|X_i^p| - 1 + |X_i^q| - 1) + \sum_{s=1}^{m}(|X_s^x| - 1 + |X_s^y| - 1 + |X_s^z| - (|X_s^z| - 1))$$

$$= 2k^2 - 2k - m + 2\sum_{s=1}^{m}|S_s|.$$

We take $\ell := k + \sum_{s=1}^{m}|S_s| + \ell_G$, i.e., we ask for ℓ vertex-disjoint cycles in G.

The following lemma shows the pathwidth bound of G, i.e., proves Point 1.

Lemma C.18. *In polynomial time we can construct a path decomposition of G of width $11k$.*

Proof. By Lemma C.17, it is sufficient to show a path decomposition of H of width $11k - 1$ such that each set X_i^p, $X_i^q, X_s^x, X_s^y, X_s^z$ is contained in some bag.

The path decomposition consists of bags V_i^p for $1 \leq i \leq k$, V_s for $0 \leq s \leq m$ and V_i^q for $1 \leq i \leq k$, arranged in a path in this order. We define:

1. $V_i^p = \{p_j^Z, p_j^R, p_{i,j}^C : 1 \leq j \leq k\}$ for $1 \leq i \leq k$.

2. $V_0 = \{p_i^Z, p_i^R, x_{i,1}^L, x_{i,1}^R : 1 \leq i \leq k\}$.

3. $V_s = \{x_{i,s}^L, x_{i,s}^R, y_{i,s}^L, y_{i,s}^R, x_{i,s+1}^L, x_{i,s+1}^R : 1 \leq i \leq k\} \cup \{x_{i,s}^C, y_{i,s}^C, x_{i,s}^Z, y_{i,s}^Z, z_{i,s}^C : (i,j) \in S_s\}$ for $1 \leq s < m$.

4. $V_m = \{x_{i,m}^L, x_{i,m}^R, y_{i,m}^L, y_{i,m}^R, q_i^Z, q_i^R : 1 \leq i \leq k\} \cup \{x_{i,m}^C, y_{i,m}^C, x_{i,m}^Z, y_{i,m}^Z, z_{i,m}^C : (i,j) \in S_m\}$.

5. $V_i^q = \{q_j^Z, q_j^R, q_{i,j}^C : 1 \leq j \leq k\}$ for $1 \leq i \leq k$.

It is easy to see that this is a proper path decomposition of graph H. Moreover, $X_i^p \subseteq V_i^p$, $X_i^q \subseteq V_i^q$ and $X_s^x, X_s^y, X_s^z \subseteq V_s$. As for the size bound, note that $|V_i^p| = |V_i^q| = 3k$ for $1 \leq i \leq k$, $|V_0| = 4k$ and $|V_s| \leq 11k$ for $1 \leq s \leq m$. $\qquad\square$

Let us note that the above bound is not optimal, but we need only $O(k)$ bound.

C.2.4   From hitting set to cycle cover

We prove Point 2 by the following lemma:

Lemma C.19. *If the initial $k \times k$ PERMUTATION HITTING SET instance is a YES-instance, then the graph G contains ℓ vertex-disjoint cycles.*

Proof. Let $S = \{(i, f(i)) : 1 \leq i \leq k\}$ be the solution to the $k \times k$ PERMUTATION HITTING SET instance. Recall that S contains exactly one element from each row and exactly one element from each column, thus f is a permutation of $[k]$.

By Lemma C.15, it is sufficient to show a family of cycles $\mathcal{C}$ in H that is gadget safe in H and is of size $k + \sum_{s=1}^{m}|S_s|$.

For each $1 \leq s \leq m$, fix an index $1 \leq i_s \leq k$, such that $(i_s, f(i_s)) \in S \cap S_s$. Let

$$\mathcal{C}_1 = \{(x_{i,s}^Z, y_{i,s}^Z, z_{i,s}^C) : 1 \leq s \leq m, 1 \leq i \leq k, i \neq i_s\}.$$

Note that $\mathcal{C}_1$ is a family of $\sum_{s=1}^{m}(|S_s| - 1)$ vertex-disjoint cycles, thus we need to find $k + m$ more.

Fix i, $1 \leq i \leq k$, and let $\{1 \leq s \leq m : i_s = i\} = \{s_1, s_2, \ldots, s_{h(i)}\}$ and $s_1 < s_2 < \ldots < s_{h(i)}$. Consider the following family of $h(i) + 1$ cycles $\{C(i, j) : 0 \leq j \leq h(i)\}$:

1. $C(i, 0)$ consists of the path $(p_i^Z, p_{i,f(i)}^C, p_{f(i)}^R)$, the subpath of $\mathcal{P}_{f(i)}^R$ from $p_{f(i)}^R$ to $x_{f(i),s_1}^R$, the path $(x_{f(i),s_1}^R, x_{i,s_1}^C, x_{i,s_1}^Z, x_{i,s_1}^L)$ and the subpath of $\mathcal{P}_i^L$ from x_{i,s_1}^L to p_i^Z;

2. $C(i,j)$ for $1 \leq j < h(i)$ consists of the path $(y_{i,s_j}^L, y_{i,s_j}^Z, y_{i,s_j}^C, y_{f(i),s_j}^R)$, the subpath of $\mathcal{P}_{f(i)}^R$ from $y_{f(i),s_j}^R$ to $x_{f(i),s_{j+1}}^R$, the path $(x_{f(i),s_{j+1}}^R, x_{i,s_{j+1}}^C, x_{i,s_{j+1}}^Z, x_{i,s_{j+1}}^L)$ and the subpath of $\mathcal{P}_i^L$ from $x_{i,s_{j+1}}^L$ to y_{i,s_j}^L;

3. $C(i,h(i))$ consists of the path $(y_{i,s_{h(i)}}^L, y_{i,s_{h(i)}}^Z, y_{i,s_{h(i)}}^C, y_{f(i),s_{h(i)}}^R)$, the subpath of $\mathcal{P}_{f(i)}^R$ from $y_{f(i),s_j}^R$ to $q_{f(i)}^R$, the path $(q_{f(i)}^R, q_{i,f(i)}^C, q_i^Z)$ and the subpath of $\mathcal{P}_i^L$ from q_i^Z to $y_{i,s_{h(i)}}^L$.

Note that

$$\mathcal{C}_2 = \{C(i,j) : 1 \leq i \leq k, 0 \leq j \leq h(i)\}$$

is a family of $k+m$ vertex-disjoint cycles in H and they are disjoint with $\mathcal{C}_1$. Moreover, $\mathcal{C} := \mathcal{C}_1 \cup \mathcal{C}_2$ does not cover:

1. $X_i^p \setminus \{p_{i,f(i)}^C\}$ and $X_i^q \setminus \{q_{i,f(i)}^C\}$ for $1 \leq i \leq k$;

2. $X_s^x \setminus \{x_{i_s,s}^C\}$ and $X_s^y \setminus \{y_{i_s,s}^C\}$ for $1 \leq s \leq m$;

3. $z_{i_s,s}^C \in X_s^z$ for $1 \leq s \leq m$.

Thus $\mathcal{C}$ is gadget safe in H. An example showing packing of three cycles for $(i, f(i)) \in S$ and $h(i) = 2$ can be found in Fig. 4. $\qquad \square$

C.2.5   From disjoint cycles to hitting set

In this section we prove Point 3 by the following lemma:

Lemma C.20. *If the graph G contains at least ℓ vertex-disjoint cycles, then the initial $k \times k$ PERMUTATION HITTING SET instance is a YES-instance.*

Proof. Let $\mathcal{C}_G$ be a family of vertex-disjoint cycles in G with maximum possible number of cycles. By the assumption, $|\mathcal{C}_G| \geq \ell$. By Lemma C.16 we can assume that $\mathcal{C}_G$ is gadget safe in G and let $\mathcal{C} \subseteq \mathcal{C}_G$ be the gadget safe in H family of size $|\mathcal{C}_G| - \ell_G \geq k + \sum_{s=1}^m |S_s|$, i.e., $\mathcal{C}$ consists of those cycles in $\mathcal{C}_G$ that are not gadget short cycles.

We now analyze the family $\mathcal{C}$. Informally speaking, we are going to show that $\mathcal{C}$ can be placed only in the way as in the proof of Lemma C.19.

First let us analyze subgraph $H[L \cup R \cup C]$. Note that this subgraph is a forest containing:

1. k paths $\mathcal{P}_i^L$ for $1 \leq i \leq k$ without the endpoints, i.e., without p_i^Z and q_i^Z;

2. k trees consisting of paths $\mathcal{P}_j^R$ ($1 \leq j \leq k$) with attached leaves $p_{i,j}^C$, $q_{i,j}^C$ for $1 \leq i \leq k$ and $x_{i,s}^C$, $y_{i,s}^C$ for $1 \leq s \leq m$, $(i,j) \in S_s$.

3. $\sum_{s=1}^m |S_s|$ isolated vertices $z_{i,s}^C$, $1 \leq s \leq m$, $(i,j) \in S_s$.

Consider now a subgraph of H induced by $L \cup R \cup C \cup \{a\}$, where a is an arbitrary vertex in Z. Note that this graph is a forest. Indeed, each vertex in Z has at most one edge incident to each connected component of $H[L \cup R \cup C]$:

1. for $1 \leq i \leq k$ the vertex p_i^Z is adjacent to $x_{i,1}^L$ on path $\mathcal{P}_i^L$ and vertices $p_{i,j}^C$ for $1 \leq j \leq k$;

2. similarly for $1 \leq i \leq k$ the vertex q_i^Z is adjacent to $y_{i,m}^L$ on path $\mathcal{P}_i^L$ and vertices $q_{i,j}^C$ for $1 \leq j \leq k$;

3. for $1 \leq s \leq m$, $(i,j) \in S_s$ the vertex $x_{i,s}^Z$ is adjacent to $z_{i,s}^C$, $x_{i,s}^L$ and $x_{i,s}^C$;

4. similarly for $1 \leq s \leq m$, $(i,j) \in S_s$ the vertex $y_{i,s}^Z$ is adjacent to $z_{i,s}^C$, $y_{i,s}^L$ and $y_{i,s}^C$.
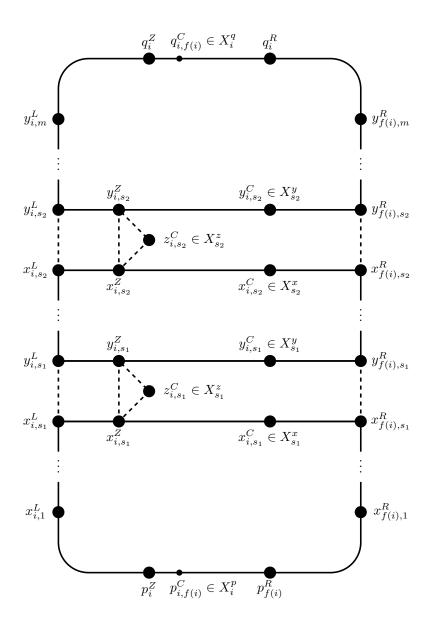
Figure 4: An example how to pack three cycles for $(i, f(i)) \in S$ and $h(i) = 2$.

Thus, each cycle from $\mathcal{C}$ contains at least two vertices from Z. But, $|Z| = 2k + 2\sum_{s=1}^{m} |S_s| = 2|\mathcal{C}|$. Thus, each cycle in $\mathcal{C}$ contains exactly two vertices from Z and $\mathcal{C}$ covers Z.

Let $C_i \in \mathcal{C}$ be the cycle that covers p_i^Z. The vertex p_i^Z has neighbours $x_{i,1}^L$ and X_i^p. As we are allowed to choose only one vertex from X_i^p, the cycle C_i contains a path $(x_{i,1}^L, p_i^Z, p_{i,f(i)}^C, p_{f(i)}^R)$ for some $1 \leq f(i) \leq k$. If the cycle C_i contains the edge $p_{f(i)}^R p_{j,f(i)}^C$ for $j \neq i$, it contains also p_j^Z and $x_{j,1}^L$. But $x_{j,1}^L$ and $x_{i,1}^L$ are in different connected components of $H[L \cup R \cup C]$, thus C_i needs to contain a third vertex in Z, a contradiction. Thus, C_i contains the path $(x_{i,1}^L, p_i^Z, p_{i,f(i)}^C, p_{f(i)}^R, x_{f(i),1}^R)$. Note that this in particular implies that f is a permutation of $[k]$.

We claim that $S = \{(i, f(i)) : 1 \leq i \leq k\}$ is a hitting set in the initial $k \times k$ PERMUTATION HITTING SET instance. It clearly contains exactly one element from each row and from each column. We now show that $S \cap S_s \neq \emptyset$ for each $1 \leq s \leq m$.

Let
$$Z_s = \{p_i^Z : 1 \leq i \leq k\} \cup \{x_{i,t}^Z, y_{i,t}^Z : 1 \leq t \leq s, (i,j) \in S_t\} \subseteq Z$$

for $0 \leq s \leq m$ and let

$$E_s = \{y_{i,s}^L x_{i,s+1}^L : 1 \leq i \leq k\} \cup \{y_{j,s}^R x_{j,s+1}^R : 1 \leq j \leq k\} \subseteq E(H)$$

for $1 \leq s < m$ and let

$$E_0 = \{p_i^Z x_{i,1}^L : 1 \leq i \leq k\} \cup \{p_j^R x_{j,1}^R : 1 \leq i \leq k\} \subseteq E(H).$$

Note that for $0 \leq s < m$ the set E_s is a set of $2k$ edges that separate Z_s from $Z \setminus Z_s$.

We now select cycles $C(i,s) \in \mathcal{C}$ for $1 \leq i \leq k$ and $0 \leq s \leq m$ with the following property: for $1 \leq i \leq k$ and $1 \leq s < m$ the edges $y_{i,s}^L x_{i,s+1}^L$ and $y_{f(i),s}^R x_{f(i),s+1}^R$ lie on $C(i,s)$ and for $1 \leq i \leq k$ the edges $p_i^L x_{i,1}^L$ and $p_{f(i)}^R x_{f(i),1}^R$ lie on $C(i,0)$. Note that cycles $C(i,0) = C_i$ satisfy the above property. We select cycles $C(i,s)$ by an induction on s and in the s-th step of the induction we prove that there exists $1 \leq i \leq k$ such that $(i, f(i)) \in S_s$.

Let us fix s, $0 \leq s < m$. We show some more properties of cycles $\{C(i,s) : 1 \leq i \leq k\}$ that we use in the induction step. Note that each cycle $C(i,s)$ contains two edges from E_s and each edge from E_s is contained in some $C(i,s)$. Moreover, each edge in E_s is in different connected component of $H[L \cup R \cup C]$. Thus, each cycle $C(i,s)$ contains: a subpath of $\mathcal{P}_i^L$, a subpath of the connected component of $H[L \cup R \cup C]$ containing $\mathcal{P}_{f(i)}^R$, a vertex in Z_s and a vertex in $Z \setminus Z_s$. This in particular implies that cycles $\{C(i,s) : 1 \leq i \leq k\}$ are pairwise different. As E_s separates Z_s from $Z \setminus Z_s$, each cycle in $\mathcal{C} \setminus \{C(i,s) : 1 \leq i \leq k\}$ contains either two vertices in Z_s, or two vertices in $Z \setminus Z_s$.

We now perform an induction step. Let $1 \leq s \leq m$ and assume that we have selected cycles $C(i, s-1)$ for $1 \leq i \leq k$.

Let $z_{i,s}^C$ be a (arbitrarily chosen) vertex not covered by $\mathcal{C}$, where $(i,j) \in S_s$. Let us focus on the vertex $x_{i,s}^Z$. It has three neighbours apart from $z_{i,s}^C$: vertices $x_{i,s}^L$, $x_{i,s}^C$ and $y_{i,s}^Z$. The vertex $x_{i,s}^C$ has degree two, and the other neighbour is $x_{j,s}^R$. As $x_{i,s}^L \in C(i, s-1)$ and $x_{j,s}^R \in C(f^{-1}(j), s-1)$, the vertex $x_{i,s}^Z$ lies on $C(i, s-1)$ or $C(f^{-1}(j), s-1)$. Both $C(i, s-1)$ and $C(f^{-1}(j), s-1)$ are not allowed to cover two vertices from $Z \setminus Z_{s-1}$, thus $x_{i,s}^Z$ and $y_{i,s}^Z$ lie on different cycles in $\mathcal{C}$. But this means that $C(i, s-1)$ or $C(f^{-1}(j), s-1)$ contains the path $(x_{i,s}^L, x_{i,s}^Z, x_{i,s}^C, x_{j,s}^R)$, thus $C(i, s-1) = C(f^{-1}(j), s-1)$. Since $\{C(i, s-1) : 1 \leq i \leq k\}$ are pairwise different, $j = f(i)$ and $(i, f(i)) \in S_s$.

Now focus on vertex $y_{i,s}^Z$. The vertex $x_{i,s}^Z$ is used on cycle $C(i, s-1)$, $y_{i,s}^Z \notin C(i, s-1)$ and we assumed the vertex $z_{i,s}^C$ is not covered by $\mathcal{C}$. Thus, $y_{i,s}^Z$ lies on a cycle C with path $(y_{i,s}^L, y_{i,s}^Z, y_{i,s}^C, y_{f(i),s}^R)$. As $x_{i,s}^L, x_{f(i),s}^R \in C(i, s-1)$, and we are allowed to cover only one vertex from X_s^y, C contains a path $(x_{i,s+1}^L, y_{i,s}^L, y_{i,s}^Z, y_{i,s}^C, y_{f(i),s}^R, y_{f(i),s+1}^R)$ (if $s < m$) or $(q_i^Z, y_{i,s}^L, y_{i,s}^Z, y_{i,s}^C, y_{f(i),s}^R, q_{f(i)}^R)$ (if $s = m$).

Now focus on vertices $x_{i',s}^Z$ for $i' \neq i$. As $x_{i,s}^C$ is covered by $\mathcal{C}$, the vertex $x_{i',s}^C$ cannot be covered too. Thus, $x_{i',s}^Z$ and $y_{i',s}^Z$ lie on the same cycle, say $C^{i'}$. As $C(i', s-1)$ is not allowed to cover two vertices from $Z \setminus Z_{s-1}$, the vertex $x_{i',s}^L$ does not lie on $C^{i'}$, thus $C^{i'} = (x_{i',s}^Z, y_{i',s}^Z, z_{i',s}^C)$.

As vertices $x_{i',s}^Z$ and $y_{i',s}^Z$ are covered by the cycle $(x_{i',s}^Z, z_{i',s}^C, y_{i',s}^Z)$, the cycle $C(i', s-1)$ contains the path $(x_{i',s}^L, y_{i',s}^L, x_{i',s+1}^L)$ (if $s < m$) or $(x_{i',s}^L, y_{i',s}^L, q_{i'}^Z)$ (if $s = m$). As vertices $x_{i,s}^C$ and $y_{i,s}^C$ are covered by cycles $C(i, s-1)$ and C, the cycle $C(i', s-1)$ contains path $(x_{f(i'),s}^R, y_{f(i'),s}^R, x_{f(i'),s+1}^R)$ (if $s < m$) or $(x_{f(i'),s}^R, y_{f(i'),s}^R, q_{f(i')}^R)$ (if $s = m$).

Thus we can put $C(i, s) = C$ and $C(i', s) = C(i', s-1)$ for $i' \neq i$ and the induction step is performed. As we maintain the induction step up to $s = m$, for each $1 \leq s \leq m$ we prove that $(i, f(i)) \in S_s$, thus the initial $k \times k$ PERMUTATION HITTING SET instance is a YES-instance. $\qquad\square$

C.3 From undirected to directed CYCLE PACKING

In this section we provide a reduction from undirected to directed CYCLE PACKING, proving Theorem 5.1 for directed CYCLE PACKING.

Lemma C.21. *Let (G, ℓ) be an instance of undirected* CYCLE PACKING. *Then we can construct in polynomial time an equivalent instance (G', ℓ') of directed* CYCLE PACKING. *Moreover, given a path decomposition of G of width p, in polynomial time we can construct a path decomposition of G' of width at most $p + 3$.*

Proof. In many problems, a reduction from an undirected version to a directed one is performed by simply changing each edge uv into pair of arcs (u, v) and (v, u). However, in the case of CYCLE PACKING, such a reduction introduces many 2-cycles (u, v) that do not have a counterpart in the original undirected graph. We circumvent this problem by adding a directed version of 1-in-many gadget.

To construct graph G', for each edge $e = uv \in E(G)$ we introduce three extra vertices x_e^{uv}, x_e^{vu} and z_e, and we replace the edge e with three cycles: $(u, x_e^{uv}, v, x_e^{vu})$, (x_e^{uv}, z_e) and (x_e^{vu}, z_e). In graph G' we ask for $\ell' := \ell + |E(G)|$ vertex-disjoint cycles. The cycles (x_e^{uv}, z_e) and (x_e^{vu}, z_e) are called *short cycles*.
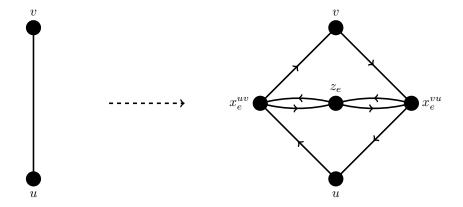
Figure 5: The construction of the gadget replacing edge uv.

First assume that we have a family $\mathcal{C}$ of ℓ vertex-disjoint cycles in G. For each cycle $C \in \mathcal{C}$, we orient it in an arbitrary way, and translate it into a cycle C' in G' (i.e., if C goes from u to v via edge $e = uv$, then in G' the cycle C' uses arcs (u, x_e^{uv}) and (x_e^{uv}, v)). In this way we create a family $\mathcal{C}'$ of ℓ vertex-disjoint cycles in G'. This family has a property that for each $e \in E(G)$ at least one vertex x_e^{uv} and x_e^{vu} is not covered. Thus we can add a cycle (x_e^{uv}, z_e) or (x_e^{vu}, z_e) to $\mathcal{C}'$, obtaining a family of ℓ' cycles.

In the other direction, let $\mathcal{C}'$ be a family of vertex-disjoint cycles in G' that contains maximum possible number of short cycles among families of vertex-disjoint cycles of maximum possible size. Assume $|\mathcal{C}'| \geq \ell'$.

We claim that $\mathcal{C}'$ contains $|E(G)|$ short cycles. As there are $|E(G)|$ vertices z_e, it may not contain more. Assume that it contains less than $|E(G)|$ short cycles. Let $e = uv$ be an edge, such that z_e is not covered by a short cycle. If z_e is not covered by $\mathcal{C}'$, we can add the short cycle (x_e^{uv}, z_e) to $\mathcal{C}'$, possibly deleting a cycle covering x_e^{uv}. Otherwise, if z_e is covered by a cycle C, then x_e^{uv} or x_e^{vu} (say x_e^{uv}) also belongs to C. But then we can replace C with the cycle (z_e, x_e^{uv}). In both cases, we increase the number of short cycles in $\mathcal{C}'$ while not decreasing its size, a contradiction.

Let $\mathcal{C}$ be the other ℓ cycles in $\mathcal{C}'$ that are not short cycles. Each such cycle C' does not cover vertices z_e, thus if it covers x_e^{uv}, it contains a subpath (u, x_e^{uv}, v). Moreover, either x_e^{uv} or x_e^{vu} is covered by a short cycle in $\mathcal{C}'$. Thus C' translates into a cycle C in G, by taking an edge e for each vertex x_e^{uv} visited by C'. In this way we obtain ℓ vertex-disjoint cycles in G.

We are left with the pathwidth bound. Assume we have a path decomposition of G of width p. We construct a path decomposition of G' in the following way. For each $e \in E(G)$, we pick a bag V_e that covers e. We create a new bag $V'_e := V_e \cup \{x_e^{uv}, x_e^{vu}, z_e\}$ and insert it into the path decomposition near V_e. It is easy to see that this is a proper path decomposition of G' and its width is at most $p + 3$. $\square$

C.4 From CYCLE PACKING to MAX CYCLE COVER

In this section we provide a reduction from CYCLE PACKING to MAX CYCLE COVER that proves Theorem 5.1 for MAX CYCLE COVER, both in directed and undirected setting. Formally, we prove the following lemma

Lemma C.22. *Let (G, ℓ) be an instance of (directed or undirected) CYCLE PACKING. Then we can construct in polynomial time an equivalent instance (G', ℓ') of (directed or undirected, respectively) MAX CYCLE COVER. Moreover, given a path decomposition of G of width p, in polynomial time we can construct a path decomposition of G of width at most $p + 3$.*

Proof. To construct G', we take G and for each $v \in V(G)$ we introduce three new vertices a_v, b_v and c_v and five new arcs (a_v, b_v), (b_v, c_v), (c_v, a_v), (c_v, v) and (v, a_v) (in the undirected setting, these arcs are edges without direction). We ask for a cycle cover with at least $\ell' := \ell + |V(G)|$ cycles.
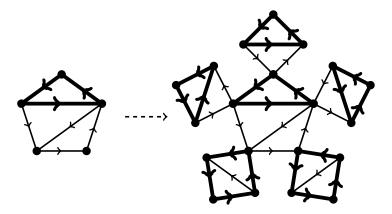
Figure 6: An example of the reduction from CYCLE PACKING to MAX CYCLE COVER in the directed setting, together with the conversion of cycle families.

Let $\mathcal{C}$ be a set of ℓ vertex-disjoint cycles in G. To construct a cycle cover of size $\ell' = \ell + |V(G)|$ in G', we take the cycles in $\mathcal{C}$ and for each vertex $v \in V(G)$: if v is covered by $\mathcal{C}$, we take the cycle (a_v, b_v, c_v), and otherwise we take the cycle (a_v, b_v, c_v, v).

In the other direction, let $\mathcal{C}'$ be a cycle cover of G' with at least ℓ' cycles. For each $v \in V(G)$ let C_v be the cycle that covers b_v. Note that $a_v, c_v \in C_v$. Thus $\mathcal{C}' \setminus \{C_v : v \in V(G)\}$ is a family of at least $\ell' - |V(G)| = \ell$ vertex-disjoint cycles in G.

We are left with the pathwidth bound. Assume we have a path decomposition of graph G of width p. For each vertex $v \in V(G)$, we pick a single bag V_v that contains v. We introduce a new bag $V_v' = V_v \cup \{a_v, b_v, c_v\}$ and insert it near V_v in the path decomposition. It is easy to see that this is a proper path decomposition of G' and its width is at most $p + 3$. □

D Negative results under SETH

In this section we provide an evidence that our algorithms for CONNECTED VERTEX COVER, CONNECTED DOMINATING SET, CONNECTED FEEDBACK VERTEX SET, CONNECTED ODD CYCLE TRANSVERSAL, FEEDBACK VERTEX SET, STEINER TREE and EXACT k-LEAF SPANNING TREE are probably optimal, i.e., the exponential in treewith term has an optimal base of the exponent. In other words, we prove here Theorem 5.3. The lower bounds are proven under the assumption of the Strong Exponential Time Hypothesis, that is, we prove that a faster algorithm solving one of the considered problems would imply a faster than exhaustive search algorithm for SAT.

We first note that the result for EXACT k-LEAF SPANNING TREE follows from the result for CONNECTED DOMINATING SET, as EXACT k-LEAF SPANNING TREE is not easier than MAXIMUM LEAF TREE, which is equivalent to

CONNECTED DOMINATING SET [25]. Second, note that the lower bound for STEINER TREE follows from the lower bound for CONNECTED VERTEX COVER by the following simple reduction.

Lemma D.1. *Given a* CONNECTED VERTEX COVER *instance* (G, k) *together with a tree decomposition of G of width t (or path decomposition of G of width p), one can construct in polynomial time an equivalent instance (G', T, k') of* STEINER TREE *together with a tree decomposition of G' of width t (or path decomposition of G of width at most $p + 1$, respectively).*

Proof. Given the instance (G, k), we construct the instance (G', T, k') by subdividing each edge of G with a terminal and setting $k' = |T| + k$ (i.e., we replace each edge $e = uv$ in G with a path u, w_e, v and we take $T = \{w_e : e \in E(G)\}$). It is easy to see that G contains a connected vertex cover of size at most k if and only if (G', T) contains a Steiner tree of cardinality at most $k + |T| = k + |E(G)|$.

Moreover, given a tree decomposition $\mathbb{T}$ of G of width t, we can construct a tree decomposition $\mathbb{T}'$ of G' of width at most t in the following manner. If G is a forest, G' is a forest too and we construct the optimal tree decomposition of G' in polynomial time. Otherwise, $t \geq 2$ and for each $e \in E(G)$ we pick one fixed node $x(e)$ whose bag covers e. We first set $\mathbb{T}' := \mathbb{T}$ and then, for each $uv = e \in E(G)$, we attach a leaf node $y(e)$ to the node $x(e)$ with the bag $B_{y(e)} = \{u, v, w_e\}$.

In the case of a path decomposition $\mathbb{T}$ of G of width p, we can construct a path decomposition $\mathbb{T}'$ of G' of width at most $p + 1$ in the following manner. For each $e \in E(G)$ we pick one fixed node $x(e)$ whose bag covers e. We first set $\mathbb{T}' := \mathbb{T}$ and then, sequentially for each $uv = e \in E(G)$, we create a new node $y(e)$ with the bag $B_{y(e)} = B_{x(e)} \cup \{w_e\}$ and insert it immediately after the node $x(e)$ on the path decomposition. □

Thus we are left with the first five problems of Theorem 5.3. In the proofs we follow the same approach as Lokshtanov et al [47] in the lower bounds for problems without the connectivity requirement. We mostly base on the VERTEX COVER and DOMINATING SET lower bounds of [47], however our gadgets are adjusted to the considered problems. For each problem we show a polynomial-time construction that, given a SAT instance with n variables, constructs an equivalent instance of the considered problem, together with a path decomposition of the underlying graph of width roughly $\log_3(2^n) = n/\log 3$ in the case of CONNECTED VERTEX COVER and FEEDBACK VERTEX SET and of width roughly $\log_4(2^n) = n/2$ in the case of the other problems. Thus, each of the following subsections consists of three parts. First, we give a construction procedure that, given a SAT formula Φ, produces an instance of the considered problem. Second, we prove that the constructed instance is equivalent to the formula Φ. Finally, we show the claimed pathwidth bound.

Similarly as in [47], we prove pathwidth bounds for the constructed graphs using mixed search game. Let us recall the definition from [47].

Definition D.2 ([59, 47])**.** *In a* mixed search game, *a graph G is considered as a system of tunnels. Initially, all edges are contaminated by a gas. An edge is cleared by placing searchers at both its end-points simultaneously or by sliding a searcher along the edge. A cleared edge is re-contaminated if there is a path from an uncleared edge to the cleared edge without any searchers on its vertices or edges. A search is a sequence of operations that can be of the following types: (a) placement of a new searcher on a vertex; (b) removal of a searcher from a vertex; (c) sliding a searcher on a vertex along an incident edge and placing the searcher on the other end. A* search strategy *is winning if after its termination all edges are cleared. The* mixed search number *of a graph G, denoted $ms(G)$, is the minimum number of searchers required for a winning strategy of mixed searching on G.*

Proposition D.3 ([59])**.** *For a graph G, $pw(G) \leq ms(G) \leq pw(G) + 1$.*

Moreover, in each case considered by us, the presented cleaning strategy easily yield a polynomial time algorithm that constructs a path decomposition of G of width not greater than the number of searchers used.

D.1 CONNECTED VERTEX COVER

Theorem D.4. *Assuming SETH, there cannot exist a constant $\varepsilon > 0$ and an algorithm that given an instance $(G = (V, E), k)$ together with a path decomposition of the graph G of width p solves the* CONNECTED VERTEX COVER *problem in $(3 - \varepsilon)^p |V|^{O(1)}$ time.*

Construction Given $\varepsilon > 0$ and an instance Φ of SAT with n variables and m clauses we construct a graph G as follows. We first choose a constant integer η, which value depends on ε only. The exact formula for η is presented later. We partition variables of Φ into groups $F_1, \ldots, F_{n'}$, each of size at most $\beta = \lfloor \log 3^\eta \rfloor$, hence $n' = \lceil n/\beta \rceil$. Note that now $\eta n' \sim n/\log 3$, the pathwidth of G will be roughly $\eta n'$.

First, we add to the graph G two vertices r and r^*, connected by an edge. In the graph G the vertex r^* will be of degree one, thus any connected vertex cover of G needs to include r. The vertex r is called a *root*.

Second, we take $a = m(2\eta n' + 1)$ and for each $1 \le t \le n'$ and $1 \le \ell \le \eta$ we create a path $\mathcal{P}_{t,\ell}$ consisting of $2a$ vertices $v_{t,\ell,k}^\alpha$, $0 \le k < a$ and $1 \le \alpha \le 2$, arranged in the following order:

$$v_{t,\ell,0}^1, v_{t,\ell,0}^2, v_{t,\ell,1}^1, \ldots, v_{t,\ell,a-1}^1, v_{t,\ell,a-1}^2.$$

Furthermore we connect all vertices $v_{t,\ell,k}^2$ ($0 \le k < a$) and $v_{t,\ell,0}^1$ to the root r. To simplify further notation we denote $v_{t,\ell,a}^1 = r$. Let $\mathcal{V}$ be the set of all vertices on all paths $\mathcal{P}_{t,\ell}$.

We now provide a description of a group gadget $\mathbf{B}_{t,k}$, which will enable us to encode 2^β possible assignments of one group of β variables. Fix a block F_t, $1 \le t \le n'$, and a position k, $0 \le k < a$. For each $1 \le \ell \le \eta$ we create three vertices $h_{t,\ell,k}^\alpha$, $1 \le \alpha \le 3$ that are pairwise adjacent and all are adjacent to the root r. Moreover, we add edges $h_{t,\ell,k}^1 v_{t,\ell,k}^1$, $h_{t,\ell,k}^2 v_{t,\ell,k}^2$ and $h_{t,\ell,k}^3 v_{t,\ell,k+1}^1$. Let $\mathcal{H}_{t,\ell,k} = \{h_{t,\ell,k}^\alpha : 1 \le \alpha \le 3\}$, $\mathcal{H}_{t,k} = \bigcup_{\ell=1}^\eta \mathcal{H}_{t,\ell,k}$. and $\mathcal{H} = \bigcup_{t=1}^{n'} \bigcup_{k=0}^{a-1} \mathcal{H}_{t,k}$. Note that each (connected) vertex cover in G needs to include at least two out of three vertices from each set $\mathcal{H}_{t,\ell,k}$.

In order to encode 2^β assignments we consider subsets of $\mathcal{H}_{t,k}$ that contain *exactly one* vertex out of each set $\mathcal{H}_{t,\ell,k}$. For a sequence $S = (s_1, \ldots, s_\eta) \in \{1,2,3\}^\eta$ by $S(\mathcal{H}_{t,k})$ we denote the set $\{h_{t,\ell,k}^{s_\ell} : 1 \le \ell \le \eta\}$. For each sequence $S \in \{1,2,3\}^\eta$ we add three vertices $x_{t,k}^S$, $x_{t,k}^{S*}$ and $y_{t,k}^S$, where $x_{t,k}^S$ is also adjacent to all the vertices of $S(\mathcal{H}_{t,k})$ (recall that η and β are constants depending only on ε). We add edges $x_{t,k}^S x_{t,k}^{S*}$, $x_{t,k}^S y_{t,k}^S$ and $y_{t,k}^S r$. In the graph G the vertices $x_{t,k}^{S*}$ are of degree one, thus any connected vertex cover in G needs to include all vertices $x_{t,k}^S$. Let $\mathcal{Y}_{t,k} = \{y_{t,k}^S : S \in \{1,2,3\}^\eta\}$ for $1 \le t \le n'$, $0 \le k < a$ and $\mathcal{Y} = \bigcup_{t=1}^{n'} \bigcup_{k=0}^{a-1} \mathcal{Y}_{t,k}$.

Additionally we add two adjacent vertices $z_{t,k}$ and $z_{t,k}^*$ and connect $z_{t,k}$ to vertices $y_{t,k}^S$ for all $S \in \{1,2,3\}^\eta$. Again, the vertex $z_{t,k}^*$ is of degree one in G and forces $z_{t,k}$ to be included in any connected vertex cover of G.

The above step finishes the construction of the group gadgets needed to encode an assignment and now we add vertices used to check the satisfiability of the formula Φ. Observe that for a group of variables F_t there are at most 2^β possible assignments and there are $3^\eta \ge 2^\beta$ vertices $x_{t,k}^S$ for sequences S from the set $\{1,2,3\}^\eta$ in each group gadget $\mathbf{B}_{t,k}$, hence we can assign a *unique* sequence S to each assignment. Let $C_0, \ldots, C_{m-1}$ be the clauses of the formula Φ. For each clause C_i we create $(2\eta n' + 1)$ pairs of adjacent vertices $c_{i,j}$ and $c_{i,j}^*$, one for each $0 \le j < (2\eta n' + 1)$. The vertex $c_{i,j}^*$ is of degree one in G and therefore forces any connected vertex cover of G to include $c_{i,j}$. The vertex $c_{i,j}$ can only be connected to gadgets $\mathbf{B}_{t,mj+i}$ for $1 \le t \le n'$. For each group of variables F_t we consider all sequences $S \in \{1,2,3\}^\eta$ that correspond to an assignment of F_t satisfying the clause C_i (i.e., one of the variables of F_t is assigned a value such that C_i is already satisfied). For each such sequence S and for each $0 \le j < (2\eta n' + 1)$ we add an edge $y_{t,mj+i}^S c_{i,j}$.

We can view the whole construction as a matrix of group gadgets, where each row corresponds to some group of variables F_t and each column is devoted to some clause in such a way that each clause gets $(2\eta n' + 1)$ private columns (but not consecutive) of the group gadget matrix, as in Figure 8.

Finally, let $K = \eta n' \cdot 3a + (3^\eta + 2)n'a + a + 1$ be the size of the vertex cover we ask for.

Correctness

Lemma D.5. *If Φ has a satisfying assignment, then there exists a connected vertex cover in G of size K.*

Proof. Given a satisfying assignment ϕ of the formula Φ we construct a connected vertex cover $X \subseteq V$ as follows. Let X_{force} be the set of vertices that are forced to be in any connected vertex cover of G, that is r, $c_{i,j}$ for $0 \le i < m$, $0 \le j < (2\eta n' + 1)$, $x_{t,k}^S$ for $1 \le t \le n'$, $0 \le k < m(2\eta n' + 1)$, $S \in \{1,2,3\}^\eta$ and $z_{t,k}$ for $1 \le t \le n'$, $0 \le k < m(2\eta n' + 1)$. Note that $|X_{\text{force}}| = 1 + a + (3^\eta + 1)n'a$.
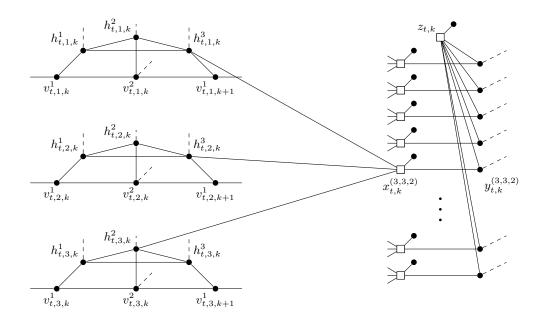
Figure 7: Group gadget $\mathbf{B}_{t,k}$ for $\eta = 3$. Dashed edges are connecting a vertex with the root r. Vertices that have a pendant neighbour and thus need to be included in any connected vertex cover of G are presented as squares.

For each group of variables F_t we consider the sequence $S_t \in \{1, 2, 3\}^\eta$ which corresponds to the restriction of the assignment ϕ to the variables of F_t. Let

$$X_h = \bigcup_{t=1}^{n'} \bigcup_{k=0}^{a-1} \bigcup_{\ell=1}^{\eta} \{h_{t,\ell,k}^\alpha : \alpha \neq S_t(\ell)\}.$$

The set X_h includes exactly two vertices out of each set $\mathcal{H}_{t,\ell,k}$, thus $|X_h| = \eta n' \cdot 2a$. Moreover, we define the set X_v to contain all vertices $v_{t,\ell,k}^2$ if $S_t(\ell) = 2$, and all vertices $v_{t,\ell,k}^1$ otherwise ($1 \leq t \leq n'$, $1 \leq \ell \leq \eta$, $0 \leq k < a$). The set X_v includes every other vertex on each path $\mathcal{P}_{t,\ell}$, thus $|X_v| = \eta n' a$.

Finally, let us define the set X_y to be the set of all vertices $y_{t,k}^{S_t}$ for all $1 \leq t \leq n'$ and $0 \leq k < a$. Let $X = X_{\text{force}} \cup X_h \cup X_v \cup X_y$. Note that as $|X_y| = n'a$ we have $|X| = K$. We now verify that X is a connected vertex cover of G.

First, we verify that $G \setminus X$ is an edgeless graph.

1. The vertices r^*, $c_{i,j}^*$, $x_{t,k}^{S*}$ and $z_{t,k}^*$ are isolated in $G \setminus X$, as their single neighbours in G are included in X_{force}.

2. The vertices $v_{t,\ell,k}^\alpha$ that are not in X are isolated in $G \setminus X$, as X_v contains every other vertex on each path $\mathcal{P}_{t,\ell}$ and we chose α in such a manner that the neighbours of $v_{t,\ell,k}^\alpha$ from $\mathcal{H}$ are in X_h.

3. The vertices $h_{t,\ell,k}^{S_t(\ell)}$ are isolated in $G \setminus X$, since their single neighours on paths $\mathcal{P}_{t,\ell}$ are in X_v and all other neighbours of $h_{t,\ell,k}^{S_t(\ell)}$ are in X_{force} and in X_h.

4. Finally, the vertices $y_{t,k}^S$ are isolated in $G \setminus X$ since their neighbourhoods are contained in X_{force}.

To finish the proof we need to verify that $G[X]$ is connected. We ensure it by showing that in $G[X]$ each vertex in X is connected to the root r.

1. The claim is obvious for X_h and X_y, as they are contained in the neighbourhood of r.
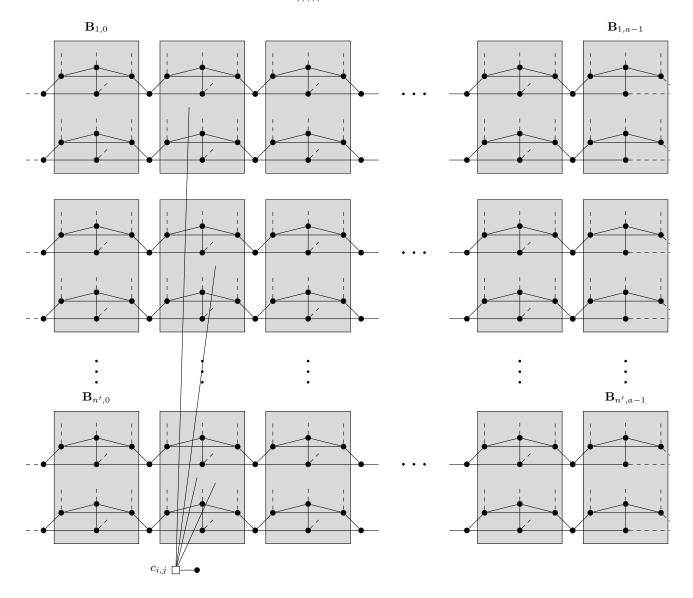
Figure 8: Gray rectangles represent group gadgets. Dashed edges connect have the root vertex r as one of the endpoints. Vertices that have a pendant neighbour and thus need to be included in any connected vertex cover of G are presented as squares.

2. If vertices $v_{t,\ell,k}^2$ belong to X_v, they are connected to r by direct edges. Otherwise, the vertices $v_{t,\ell,k}^1$ are connected via vertices $h_{t,\ell,k}^1$ or $h_{t,\ell,k-1}^3$ (with the exception of vertices $v_{t,\ell,0}^1$ that are connected directly).

3. Each vertex $x_{t,k}^S$ for $S \neq S_t$ is connected to r via any vertex $h_{t,\ell,k}^{S(\ell)}$ for which $S(\ell) \neq S_t(\ell)$.

4. Vertices $x_{t,k}^{S_t}$ and $z_{t,k}$ are connected to r via $y_{t,k}^{S_t}$.

5. Finally, each vertex $c_{i,k}$ is connected to r via any vertex $y_{t,k}^{S_t}$ for which the assignment ϕ on variables from F_t satisfies the clause C_i.

$\square$

Lemma D.6. *If there exists a connected vertex cover X of size at most K in the graph G, then Φ has a satisfying assignment.*

Proof. As in the previous lemma, let X_{force} be the set of vertices that are forced to be in any connected vertex cover of G, that is r, $c_{i,j}$ for $1 \leq i \leq m$, $0 \leq j < (2\eta n' + 1)$, $x_{t,k}^{S}$ for $1 \leq t \leq n'$, $0 \leq k < m(2\eta n' + 1)$, $S \in \{1, 2, 3\}^\eta$ and $z_{t,k}$ for $1 \leq t \leq n'$, $0 \leq k < m(2\eta n' + 1)$. Note that $|X_{\text{force}}| = 1 + a + (3^\eta + 1)n'a$ and $X_{\text{force}} \subseteq X$.

Let $X_v = X \cap \mathcal{V}$, $X_h = X \cap \mathcal{H}$ and $X_y = X \cap \mathcal{Y}$. Note that

1. X_v needs to include at least a vertices from each path $\mathcal{P}_{t,\ell}$, thus $|X_v| \geq \eta n'a$.

2. X_h needs to include at least two vertices out of each set $\mathcal{H}_{t,\ell,k}$, thus $|X_h| \geq \eta n' \cdot 2a$.

3. X_y needs to include at least one vertex $y_{t,k}^S$ for each $1 \leq t \leq n'$ and $0 \leq k < a$ to ensure that the vertex $z_{t,k}$ is connected to the root r in $G[X]$. Thus $|X_y| \geq n'a$.

As $|X| \leq K$, we have $|X_v| = \eta n'a$, $|X_h| = \eta n' \cdot 2a$, $|X_y| = n'a$ and $|X| = K$.

As $|X_h| = \eta n' \cdot 2a$, for each $1 \leq t \leq n'$, $1 \leq \ell \leq \eta$ and $0 \leq k < a$ we have $|X_h \cap \mathcal{H}_{t,\ell,k}| = 2$. This allows us to define a sequence $S_{t,k} \in \{1, 2, 3\}^\eta$ satisfying $h_{t,\ell,k}^{S_{t,k}(\ell)} \notin X_h$ for $1 \leq \ell \leq \eta$.

Note that the vertex $x_{t,k}^{S_{t,k}} \in X_{\text{force}}$ does not have a neighbour in X_h, thus, to connect it to the root r, we need to have $y_{t,k}^{S_{t,k}} \in X_y$. As $|X_y| = n'a$, we infer that $|\mathcal{Y}_{t,k} \cap X| = 1$ for all $1 \leq t \leq n'$ and $0 \leq k < a$, i.e., $y_{t,k}^S \in X$ if and only if $S = S_{t,k}$.

We now show that for fixed t the sequences $S_{t,k}$ cannot differ much for $0 \leq k < a$. As $|X_v| = \eta n'a$, for each $1 \leq t \leq n'$, $1 \leq \ell \leq \eta$ and $0 \leq k \leq a$ we have that $|X_v \cap \{v_{t,\ell,k}^\alpha : 1 \leq \alpha \leq 2\}| = 1$. Let $\alpha(t, \ell, k)$ be such that $v_{t,\ell,k}^{\alpha(t,\ell,k)} \in X_v$. Now note that for $1 \leq t \leq n'$, $1 \leq \ell \leq \eta$ and $0 \leq k < a - 1$:

1. If $S_{t,k}(\ell) = 3$, then $\alpha(t, \ell, k + 1) = 1$, as otherwise the edge $v_{t,\ell,k+1}^1 h_{t,\ell,k}^3$ is not covered by X. Moreover, $h_{t,\ell,k+1}^1 \in X_h$, as otherwise $v_{t,\ell,k+1}^1$ is isolated in $G[X]$, and $h_{t,\ell,k+1}^2 \in X_h$, as otherwise the edge $v_{t,\ell,k+1}^2 h_{t,\ell,k+1}^2$ is not covered by X. Thus $S_{t,k+1}(\ell) = 3$ as well.

2. If $S_{t,k}(\ell) = 1$ then $\alpha(t, \ell, k) = 1$, as otherwise the edge $v_{t,\ell,k}^1 h_{t,\ell,k}^1$ is not covered by X. Thus $\alpha(t, \ell, k + 1) = 1$, as otherwise the edge $v_{t,\ell,k}^2 v_{t,\ell,k+1}^1$ is not covered by X, and $S_{t,k+1}(\ell) \neq 2$, as otherwise the edge $v_{t,\ell,k+1}^2 h_{t,\ell,k+1}^2$ is not covered by X.

For fixed $1 \leq t \leq n'$ and $1 \leq \ell \leq \eta$ define the sequence $\hat{S}_{t,\ell}(k) = S_{t,k}(\ell)$. From the above arguments we infer that the sequence $\hat{S}_{t,\ell}(k)$ cannot change more than twice. As $a = m(2\eta n' + 1)$, we conclude that there exists an index $0 \leq j < (2\eta n' + 1)$ such that for all $1 \leq t \leq n'$, $1 \leq \ell \leq \eta$ the sequence $\hat{S}_{t,\ell}(mj + i)$ is constant for $0 \leq i < m$.

We create now an assignment ϕ by taking, for each group of variables F_t, an assignment corresponding to the sequence $S_{t,mj}$. Now we prove that ϕ satisfies Φ. Take any clause C_i, $0 \leq i < m$, and focus on the vertex $c_{i,j} \in X_{\text{force}}$. The vertex $c_{i,j}$ needs to be connected to the root r in $G[X]$, thus for some $1 \leq t \leq n'$ and $S \in \{1, 2, 3\}^\eta$ we have $y_{t,mj+i}^S \in X$ and the assignment of F_t that corresponds to S satisfies C_i. However, we know that $y_{t,mj+i}^S \in X$ implies that $S = S_{t,mj+i}$, and thus ϕ satisfies C_i. $\square$

Pathwidth bound

Lemma D.7. *Pathwidth of the graph G is at most $\eta n' + O(3^\eta)$. Moreover a path decomposition of such width can be found in polynomial time.*

Proof. We give a mixed search strategy to clean the graph with $\eta n' + O(3^\eta)$ searchers. First we put a searcher in the vertex r^* and slide it to the root r. This searcher remains there till the end of the cleaning process.

For a gadget $\mathbf{B}_{t,k}$ we call the vertices $v^1_{t,\ell,k}$ and $v^1_{t,\ell,k+1}$, $1 \leq \ell \leq \eta$, as *entry vertices* and *exit vertices* respectively. We search the graph in $a = m(2\eta n' + 1)$ rounds. At the beginning of round k ($0 \leq k < a$) there are searchers on the entry vertices of the gadget $\mathbf{B}_{t,k}$ for every $1 \leq t \leq n'$. Let $0 \leq i < m$ and $0 \leq j < (2\eta n' + 1)$ be integers such that $k = i + mj$. We place a searcher on $c^*_{i,j}$ and slide it to $c_{i,j}$. Then, for each $1 \leq t \leq n'$ in turn we:

- put $O(3^\eta)$ searchers on all vertices of the group gadget $\mathbf{B}_{t,k}$,

- put 2η searchers on all vertices $v^2_{t,\ell,k}$ and $v^1_{t,\ell,k+1}$, $1 \leq \ell \leq \eta$,

- remove searchers from all vertices of the group gadget $\mathbf{B}_{t,k}$ and $v^\alpha_{t,\ell,k}$ for $1 \leq \ell \leq \eta$ and $1 \leq \alpha \leq 2$.

The last step of the round is removing a searcher from the vertex $c_{i,j}$. After the last round the whole graph G is cleaned. Since we reuse $O(3^\eta)$ searchers for cleaning group gadgets, $\eta n' + O(3^\eta)$ searchers suffice to clean the graph.

Using the above graph cleaning process a path decomposition of width $\eta n' + O(3^\eta)$ can be constructed in polynomial time. $\qquad\square$

Proof of Theorem D.4. Suppose CONNECTED VERTEX COVER can be solved in $(3 - \varepsilon)^p |V|^{O(1)}$ time provided that we are given a path decomposition of G of width p. Let $\lambda = \log_3(3 - \varepsilon) < 1$. We choose η large enough such that $\frac{\log 3^\eta}{\lfloor \log 3^\eta \rfloor} < \frac{1}{\lambda}$. Given an instance of SAT we construct an instance of CONNECTED VERTEX COVER using the above construction and the chosen value of η. Next we solve CONNECTED VERTEX COVER using the $3^{\lambda p}|V|^{O(1)}$ time algorithm. Lemmata D.5, D.6 ensure correctness, whereas Lemma D.7 implies that running time of our algorithm is $3^{\lambda \eta n'}|V|^{O(1)}$, however we have

$$3^{\lambda \eta n'} = 2^{\lambda \eta n' \log 3} = 2^{\lambda n' \log 3^\eta} \leq 2^C \cdot 2^{\lambda n \log 3^\eta / \lfloor \log 3^\eta \rfloor} = 2^C \cdot 2^{\lambda' n}$$

for some $\lambda' < 1$ and $C = \lambda \log 3^\eta$. This concludes the proof. $\qquad\square$

D.2 CONNECTED DOMINATING SET

Theorem D.8. *Assuming SETH, there cannot exist a constant $\varepsilon > 0$ and an algorithm that given an instance $(G = (V, E), k)$ together with a path decomposition of the graph G of width p solves the CONNECTED DOMINATING SET problem in $(4 - \varepsilon)^p |V|^{O(1)}$ time.*

Construction Given $\varepsilon > 0$ and an instance Φ of SAT with n variables and m clauses we construct a graph G as follows. We assume that the number of variables n is even, otherwise we add a single dummy variable. We partition variables of Φ into groups $F_1, \dots, F_{n'}$, each of size two, hence $n' = n/2$. The pathwidth of G will be roughly n'.

First, we add to the graph G two vertices r and r^*, connected by an edge. In the graph G the vertex r^* is of degree one, thus any connected dominating set of G needs to include r. The vertex r is called a *root*.

Second, we take $a = m(n + 1)$ and for each $1 \leq t \leq n'$ we create a path $\mathcal{P}_t$ consisting of $4a$ vertices $v^\alpha_{t,k}$ and $h^\alpha_{t,k}$, $0 \leq k < a$ and $1 \leq \alpha \leq 2$. On the path $\mathcal{P}_t$ the vertices are arranged in the following order:

$$v^1_{t,0}, h^1_{t,0}, v^2_{t,0}, h^2_{t,0}, v^1_{t,1}, \dots, h^2_{t,a-1}.$$

Let $\mathcal{V}$ and $\mathcal{H}$ be the sets of all vertices $v^\alpha_{t,k}$ and $h^\alpha_{t,k}$ ($1 \leq t \leq n'$, $0 \leq k < a$, $1 \leq \alpha \leq 2$), respectively. We connect vertices $v^1_{t,0}$ and all vertices in $\mathcal{H}$ to the root r. To simplify further notation we denote $v^1_{t,a} = r$, note that $h^2_{t,a-1}v^1_{t,a} \in E$.

Third, for each $1 \leq t \leq n'$ and $0 \leq k < a$ we introduce guard vertices $p^1_{t,k}$, $p^2_{t,k}$ and $q_{t,k}$. Each guard vertex is of degree two in G, namely $p^1_{t,k}$ is adjacent to $v^1_{t,k}$ and $v^2_{t,k}$, $p^2_{t,k}$ is adjacent to $v^2_{t,k}$ and $v^1_{t,k+1}$ and $q_{t,k}$ is adjacent to $h^1_{t,k}$ and $h^2_{t,k}$. Thus, each guard vertex ensures that at least one of its neighbours is contained in any connected dominating set in G.
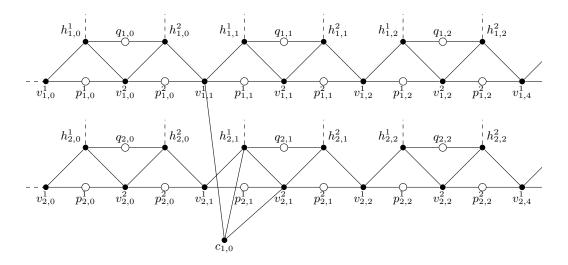
Figure 9: Part of the construction for CONNECTED DOMINATING SET. Dashed edges are connecting a vertex with the root r. Empty circles represent guard vertices.

The intuition of the construction made so far is as follows. For each two-variable block F_t we encode any assignment of the variables in F_t as a choice whether to take $v_{t,k}^1$ or $v_{t,k}^2$ and $h_{t,k}^1$ or $h_{t,k}^2$ to the connected dominating set in G.

We have finished the part of the construction needed to encode an assignment and now we add vertices used to check the satisfiability of the formula Φ. Let $C_0, \ldots, C_{m-1}$ be the clauses of the formula Φ. For each clause C_i we create $(n+1)$ vertices $c_{i,j}$, one for each $0 \le j < n+1$. Consider a clause C_i and a group of variables $F_t = \{x_t^1, x_t^2\}$. If x_t^1 occurs positively in C_i then we connect $c_{i,j}$ with $v_{t,mj+i}^1$ and if x_t^1 occurs negatively in C_i then we connect $c_{i,j}$ with $v_{t,mj+i}^2$. Similarly if x_t^2 occurs positively in C_i then we connect $c_{i,j}$ with $h_{t,mj+i}^1$ and if x_t^2 occurs negatively in C_i then we connect $c_{i,j}$ with $h_{t,mj+i}^2$. Intuitively taking the vertex $v_{t,mj+i}^1$ into a connected dominating set corresponds to setting x_t^1 to true, whereas taking the vertex $h_{t,mj+i}^1$ into a connected dominating set corresponds to setting x_t^2 to true.

We can view the whole construction as a matrix, where each row corresponds to some group of variables F_t and each column is devoted to some clause in such a way that each clause gets $(n+1)$ private columns (but not consecutive) of the matrix.

Finally, let $K = 1 + n' \cdot 2a$ be the size of the connected dominating set we ask for.

Correctness

Lemma D.9. *If Φ has a satisfying assignment, then there exists a connected dominating set X in the graph G of size K.*

Proof. Given a satisfying assignment ϕ of the formula Φ we construct a connected dominating set X as follows. For each block $F_t = \{x_t^1, x_t^2\}$ and for each $0 \le k < a$ we include into X:

1. the vertex $v_{t,k}^1$ if $\phi(x_t^1)$ is true, and $v_{t,k}^2$ otherwise;

2. the vertex $h_{t,k}^1$ if $\phi(x_t^2)$ is true, and $h_{t,k}^2$ otherwise.

Finally, we put r into X. Note that $|X| = 1 + n' \cdot 2a = K$. We now verify that X is a connected dominating set in G. First, we verify that X dominates all vertices in G.

1. r^* and all vertices in $\mathcal{H}$ are dominated by the root r.

2. All guards $p_{t,k}^1$, $p_{t,k}^2$ and $q_{t,k}^1$ are dominated by $X \cap (\mathcal{H} \cup \mathcal{V})$ (with the possible exception of $p_{t,a-1}^2$ that is dominated by r).

3. All vertices in $\mathcal{V}$ are dominated by $X \cap \mathcal{H}$ (with the possible exception of $v_{t,0}^1$ that is dominated by r).

4. Finally, each clause vertex $c_{i,j}$ is dominated by any vertex $v_{t,mj+i}^\alpha$ or $h_{t,mj+i}^\alpha$ that corresponds to a variable that satisfies C_i in the assignment ϕ.

To finish the proof we need to ensure that $G[X]$ is connected. We prove this by showing that each vertex in X is connected to the root r in $G[X]$. This is obvious for vertices in $X \cap \mathcal{H}$, as $\mathcal{H} \subseteq N_G(r)$. Moreover, for each $1 \leq t \leq n'$ and $0 \leq k < a$:

1. if $v_{t,k}^1 \in X$, then $v_{t,k}^1$ is connected to the root via $h_{t,k-1}^2$ or $h_{t,k}^1$, with the exception of $v_{t,0}^1$, that is connected to r directly;

2. if $v_{t,k}^2 \in X$, then $v_{t,k}^2$ is connected to the root via $h_{t,k}^1$ or $h_{t,k}^2$.

$\square$

Lemma D.10. *If there exists a connected dominating set X of size at most K in the graph G, then Φ has a satisfying assignment.*

Proof. First note that the vertex r^* ensures that $r \in X$. Moreover, the guard vertices $p_{t,k}^1$ and $q_{t,k}$ ensure that for each $1 \leq t \leq n'$ and $0 \leq k < a$ at least one vertex $v_{t,k}^\alpha$ and at least one vertex $h_{t,k}^\alpha$ ($1 \leq \alpha \leq 2$) belongs to X. As $|X| \leq 1 + n' \cdot 2a$ and we have $n' \cdot 2a$ aforementioned guards with disjoint neighbourhoods, for each $1 \leq t \leq n'$ and $0 \leq k < a$ exactly one vertex $v_{t,k}^\alpha$ and exactly one vertex $h_{t,k}^\alpha$ belongs to X. Moreover, $X \subseteq \{r\} \cup \mathcal{V} \cup \mathcal{H}$.

For each $0 \leq k < a$ we construct an assignment ϕ_k as follows. For each block $F_t = \{x_t^1, x_t^2\}$ we define:

1. $\phi_k(x_t^1)$ to be true if $v_{t,k}^1 \in X$ and false if $v_{t,k}^2 \in X$;

2. $\phi_k(x_t^2)$ to be true if $h_{t,k}^1 \in X$ and false if $h_{t,k}^2 \in X$.

We now show that the assignments ϕ_k cannot differ much for all indices $0 \leq k < a$. Note that for each block $F_t = \{x_t^1, x_t^2\}$ and $0 \leq k < a - 1$:

1. if $\phi_k(x_t^1)$ is true, then $\phi_{k+1}(x_t^1)$ is also true, as otherwise $v_{t,k}^2, v_{t,k+1}^1 \notin X$ and the guard $p_{t,k}^2$ is not dominated by X;

2. if $\phi_k(x_t^2)$ is true, then $\phi_{k+1}(x_t^2)$ is also true, as otherwise $h_{t,k}^2, h_{t,k+1}^1 \notin X$ and the vertex $v_{t,k}^2$ is either not dominated by X (if $v_{t,k}^2 \notin X$) or isolated in $G[X]$ (if $v_{t,k}^2 \in X$).

For each variable x we define a sequence $\widehat{\phi}_x(k) = \phi_k(x)$, $0 \leq k < a$. From the reasoning above we infer that for each variable x the sequence $\widehat{\phi}_x(k)$ can change its value at most once, from false to true. Thus, as $a = m(n+1)$, we conclude that there exists $0 \leq j < n+1$ such that for all $0 \leq i < m$ the assignments ϕ_{mj+i} are equal.

We claim that the assigment $\phi = \phi_{mj}$ satisfies Φ. Consider a clause C_i and focus on the vertex $c_{i,j}$. It is not contained in X, thus one of its neighbour is contained in X. As this neighbour corresponds to an assignment of one variable that both satisfies C_i (by the construction process) and is consistent with $\phi_{mj+i} = \phi$ (by the definition of ϕ_{mj+i}), the assignment ϕ satisfies C_i and the proof is finished. $\square$

Pathwidth bound

Lemma D.11. *Pathwidth of the graph G is at most $n' + O(1)$. Moreover a path decomposition of such width can found in polynomial time.*

Proof. We give a mixed search strategy to clean the graph with $n' + 9$ searchers. First we put a searcher in the vertex r^* and slide it to the root r. This searcher remains there till the end of the cleaning process.

We search the graph in $a = m(n+1)$ rounds. At the beginning of round k ($0 \le k < a$) there are searchers on all vertices $v_{t,k}^1$ for $1 \le t \le n'$. Let $0 \le i < m$ and $0 \le j < n+1$ be integers such that $k = i + mj$. We place a searcher on $c_{i,j}$. Then, for each $1 \le t \le n'$ in turn we put 7 searchers on vertices $p_{t,k}^1, v_{t,k}^2, p_{t,k}^2, v_{t,k+1}^1, h_{t,k}^1, h_{t,k}^2$ and $q_{t,k}$, and then remove 7 searchers from vertices $v_{t,k}^1, p_{t,k}^1, v_{t,k}^2, p_{t,k}^2, h_{t,k}^1, h_{t,k}^2$ and $q_{t,k}$. The last step of the round is removing a searcher from the vertex $c_{i,j}$. After the last round the whole graph G is cleaned. Since we reuse 8 searchers in the cleaning process, $n' + 9$ searchers suffice to clean the graph.

Using the above graph cleaning process a path decomposition of width $n' + O(1)$ can be constructed in polynomial time. $\qquad\square$

Proof of Theorem D.8. Suppose CONNECTED DOMINATING SET can be solved in $(4-\varepsilon)^p |V|^{O(1)}$ time provided that we are given a path decomposition of G of width p. Given an instance of SAT we construct an instance of CONNECTED DOMINATING SET using the above construction and solve it using the $(4-\varepsilon)^p |V|^{O(1)}$ time algorithm. Lemmata D.9, D.10 ensure correctness, whereas Lemma D.11 implies that running time of our algorithm is $(4-\varepsilon)^{n/2} |V|^{O(1)}$, however we have $(4-\varepsilon)^{n/2} = (\sqrt{4-\varepsilon})^n$ and $\sqrt{4-\varepsilon} < 2$. This concludes the proof. $\qquad\square$

D.3   CONNECTED FEEDBACK VERTEX SET **and** CONNECTED ODD CYCLE TRANSVERSAL

Theorem D.12. *Assuming SETH, there cannot exist a constant $\varepsilon > 0$ and an algorithm that given an instance $(G = (V, E), k)$ together with a path decomposition of the graph G of width p solves the* CONNECTED FEEDBACK VERTEX SET *problem in $(4 - \varepsilon)^p |V|^{O(1)}$ time.*

Theorem D.13. *Assuming SETH, there cannot exist a constant $\varepsilon > 0$ and an algorithm that given an instance $(G = (V, E), k)$ together with a path decomposition of the graph G of width p solves the* CONNECTED ODD CYCLE TRANSVERSAL *problem in $(4 - \varepsilon)^p |V|^{O(1)}$ time.*

In this section we prove Theorems D.12 and D.13 at once. That is, we provide a single reduction that, given an instance Φ of SAT with n variables and m clauses, produces a graph G together with path decomposition of width roughly $n/2$ and integer K, such that (1) if Φ is satisfiable then G admits a connected feedback vertex set of size at most K (2) if G admits a connected odd cycle transversal of size at most K, then Φ is satisfiable. As any connected feedback vertex set is a connected odd cycle transversal as well, this is sufficient to prove lower bounds for CONNECTED FEEDBACK VERTEX SET and CONNECTED ODD CYCLE TRANSVERSAL.

Construction   Before we start, let us introduce one small gadget. By *introducing a pentagon edge* vw we mean the following construction: we add three new vertices $u_{vw}^1, u_{vw}^2, u_{vw}^3$ and edges $vu_{vw}^1, u_{vw}^1 w, vu_{vw}^2, u_{vw}^2 u_{vw}^3, u_{vw}^3 w$. In the graph G the vertices u_{vw}^α are of degree two and thus the created pentagon ensures that any connected feedback vertex set or connected odd cycle transversal of G includes v or w. We call u_{vw}^α *guard vertices*.

Given $\varepsilon > 0$ and an instance Φ of SAT with n variables and m clauses we construct a graph G as follows. We assume that the number of variables n is even, otherwise we add a single dummy variable. We partition variables of Φ into groups $F_1, \ldots, F_{n'}$, each of size two, hence $n' = n/2$. The pathwidth of G will be roughly n'.

First, we add to the graph G five vertices r^1, r^2, r, r^* and r^{**} and edges $r^1 r^2, rr^*, r^* r^{**}$ and $r^{**}r$. In the graph G the vertices r^* and r^{**} are of degree two, thus any connected feedback vertex set or connected odd cycle transversal of G needs to include r. The vertex r is called a *root*.

Second, we take $a = m(n+1)$ and for each $1 \le t \le n'$ we create a path $\mathcal{P}_t$ consisting of $4a$ vertices $v_{t,k}^\alpha$ and $h_{t,k}^\alpha$, $0 \le k < a$ and $1 \le \alpha \le 2$. On the path $\mathcal{P}_t$ the vertices are arranged in the following order:

$$v_{t,0}^1, h_{t,0}^1, v_{t,0}^2, h_{t,0}^2, v_{t,1}^1, \ldots, h_{t,a-1}^2.$$

Let $\mathcal{V}$ and $\mathcal{H}$ be the sets of all vertices $v_{t,k}^\alpha$ and $h_{t,k}^\alpha$ ($1 \le t \le n'$, $0 \le k < a$, $1 \le \alpha \le 2$), respectively. We connect vertices $v_{t,0}^1$ and all vertices in $\mathcal{H}$ to the root r. Moreover, we connect all vertices $h_{t,k}^\alpha$ ($1 \le t \le n'$, $0 \le k < a$, $1 \le \alpha \le 2$) to the vertex r^α. To simplify further notation we denote $v_{t,a}^1 = r$, note that $h_{t,a-1}^2 v_{t,a}^1 \in E$.

Third, for each $1 \le t \le n'$ and $0 \le k < a$ we introduce pentagon edges $v_{t,k}^1 v_{t,k}^2$, $v_{t,k}^2 v_{t,k+1}^1$ and $h_{t,k}^1 h_{t,k}^2$.
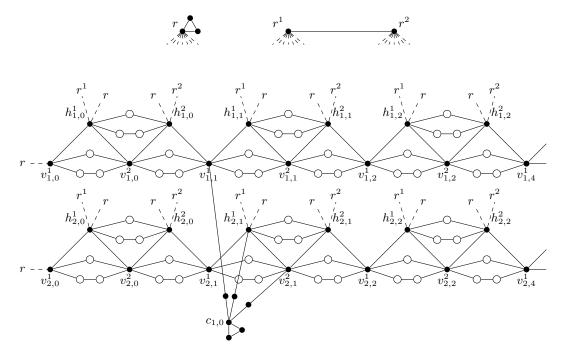
Figure 10: Part of the construction for CONNECTED FEEDBACK VERTEX SET and CONNECTED ODD CYCLE TRANSVERSAL. Dashed edges have one endpoint r, r^1 or r^2. Empty circles represent guard vertices in the pentagon edges.

The intuition of the construction made so far is as follows. For each two-variable block F_t we encode any assignment of the variables in F_t as a choice whether to take $v^1_{t,k}$ or $v^2_{t,k}$ and $h^1_{t,k}$ or $h^2_{t,k}$ to the connected feedback vertex set or connected odd cycle transversal in G.

We have finished the part of the construction needed to encode an assignment and now we add vertices used to check the satisfiability of the formula Φ. Let $C_0, \ldots, C_{m-1}$ be the clauses of the formula Φ. For each clause C_i we create $(n+1)$ triples of vertices $c_{i,j}, c^*_{i,j}, c^{**}_{i,j}$, one for each $0 \le j < n+1$. Each such triple is connected into a triangle. The vertices $c^*_{i,j}$ and $c^{**}_{i,j}$ are of degree two in G, thus they ensure that each vertex $c_{i,j}$ is contained in any connected feedback vertex set or connected odd cycle transversal in G. Let $\mathcal{C}$ be the set of all vertices $c_{i,j}$, $|\mathcal{C}| = a$.

Consider a clause C_i and a group of variables $F_t = \{x^1_t, x^2_t\}$. If x^1_t occurs positively in C_i then we connect $c_{i,j}$ with $v^1_{t,mj+i}$ via a path of length two, that is we add a vertex $c^v_{i,j,t,1}$ and edges $v^1_{t,mj+i}c^v_{i,j,t,1}$,$c^v_{i,j,t,1}c_{i,j}$. If x^1_t occurs negatively in C_i then we connect $c_{i,j}$ with $v^2_{t,mj+i}$ via a path of length two, that is we add a vertex $c^v_{i,j,t,2}$ and edges $v^2_{t,mj+i}c^v_{i,j,t,2}$,$c^v_{i,j,t,2}c_{i,j}$. Similarly if x^2_t occurs positively in C_i then we connect $c_{i,j}$ with $h^1_{t,mj+i}$ via a path of length two, that is we add a vertex $c^h_{i,j,t,1}$ and edges $h^1_{t,mj+i}c^h_{i,j,t,1}$,$c^h_{i,j,t,1}c_{i,j}$. If x^2_t occurs negatively in C_i then we connect $c_{i,j}$ with $h^2_{t,mj+i}$ via a path of length two, that is we add a vertex $c^h_{i,j,t,2}$ and edges $h^2_{t,mj+i}c^h_{i,j,t,2}$,$c^h_{i,j,t,2}c_{i,j}$.

Intuitively, taking the vertex $v^1_{t,mj+i}$ into a connected feedback vertex set or connected odd cycle transversal corresponds to setting x^1_t to true, whereas taking the vertex $h^1_{t,mj+i}$ corresponds to setting x^2_t to true.

We can view the whole construction as a matrix, where each row corresponds to some group of variables F_t and each column is devoted to some clause in such a way that each clause gets $(n+1)$ private columns (but not consecutive) of the matrix.

Finally, let $K = 1 + 2a + n' \cdot 2a$ be the size of the connected dominating set we ask for.

Correctness

81

Lemma D.14. *If Φ has a satisfying assignment, then there exists a connected feedback vertex set X in the graph G of size K.*

Proof. Given a satisfying assignment ϕ of the formula Φ we construct a connected feedback vertex set X as follows. For each block $F_t = \{x_t^1, x_t^2\}$ and for each $0 \le k < a$ we include into X:

1. the vertex $v_{t,k}^1$ if $\phi(x_t^1)$ is true, and $v_{t,k}^2$ otherwise;

2. the vertex $h_{t,k}^1$ if $\phi(x_t^2)$ is true, and $h_{t,k}^2$ otherwise.

Moreover, we put r and all vertices in $\mathcal{C}$ into X. Finally, for each clause C_i let $x_t^\alpha \in F_t$ be any fixed variable satisfying C_i in the assignment ϕ. For each $0 \le j < n+1$ we add to the set X exactly one neighbour of $c_{i,j}$, namely we add the vertex $c_{i,j,t,\beta}^\gamma$, where $\gamma = v$ if $\alpha = 1$ and $\gamma = h$ otherwise, whereas $\beta = 1$ if $\phi(x_t^\alpha)$ is true and $\beta = 2$ otherwise.

Note that $|X| = 1 + 2a + n' \cdot 2a = K$. We now verify that X is a connected feedback vertex set in G. First, we verify that $G \setminus X$ is a forest.

1. $r^*, r^{**}, c_{i,j}^*, c_{i,j}^{**}, c_{i,j,t,\beta}^\gamma$ $(0 \le i < m, 0 \le j < n+1, 1 \le t \le n', 1 \le \beta \le 2, \gamma \in \{v, h\})$ are either contained in X or of degree one in $G \setminus X$.

2. Each guard vertex in $G \setminus X$ is either of degree at most one or is of degree two and has a leaf as a neighbour, as X includes at least one endpoint of each pentagon edge.

3. In $G \setminus X$ the vertices from $\mathcal{V} \setminus X$ are connected to guard vertices, vertices $c_{i,j,t,\beta}^\gamma$, and at most one vertex from $\mathcal{H} \setminus X$.

4. In $G \setminus X$ the vertices from $\mathcal{H} \setminus X$ are connected to guard vertices, vertices $c_{i,j,t,\beta}^\gamma$, at most one vertex from $\mathcal{V} \setminus X$, and exactly one vertex from the set $\{r^1, r^2\}$.

5. In $G \setminus X$ the vertices r^1 and r^2 are connected to each other and to some vertices in $\mathcal{H} \setminus X$, but no vertex in $\mathcal{H} \setminus X$ can reach both r^1 and r^2 in $G \setminus X$ without using the edge $r^1 r^2$.

To finish the proof we need to ensure that $G[X]$ is connected. We prove this by showing that each vertex in X is connected to the root r in $G[X]$. This is obvious for vertices in $X \cap \mathcal{H}$, as $\mathcal{H} \subseteq N_G(r)$. For each $1 \le t \le n'$ and $0 \le k < a$:

1. if $v_{t,k}^1 \in X$, then $v_{t,k}^1$ is connected to root via $h_{t,k-1}^2$ or $h_{t,k}^1$, with the exception of $v_{t,0}^1$, that is connected to r directly;

2. if $v_{t,k}^2 \in X$, then $v_{t,k}^2$ is connected to root via $h_{t,k}^1$ or $h_{t,k}^2$.

We are left with the vertices $c_{i,j}$ and their neighbours chosen to X for $0 \le i < m, 0 \le j < n+1$. However, by the definition of X the only neighbour of $c_{i,j}$ chosen to X connects it to a vertex $w \in \mathcal{V} \cup \mathcal{H}$ corresponding to a choice of the value $\phi(x)$ for some variable x. Therefore $w \in X$, so $c_{i,j}$ along with its only neighbour from X are also connected to the root. $\square$

Lemma D.15. *If there exists a connected odd cycle transversal X of size at most K in the graph G, then Φ has a satisfying assignment.*

Proof. First note that $r \in X$ and $\mathcal{C} \subseteq X$. Moreover, X needs to contain at least one endpoint of each pentagon edge $h_{t,k}^1 h_{t,k}^2$ and $v_{t,k}^1 v_{t,k}^2$ $(1 \le t \le n', 0 \le k < a)$ and these pentagon edges are pairwise disjoint. Furthermore, each vertex in $\mathcal{C}$ needs to have a neighbour in X, but $\mathcal{C}$ is an independent set and the neighbourhoods of vertices from $\mathcal{C}$ are pairwise disjoint and disjoint from $\mathcal{H} \cup \mathcal{V} \cup \{r\}$. So far we have one vertex r, a vertices in $\mathcal{C}$, $n' \cdot 2a$ endpoints of pentagon edges and a neighbours of vertices from $\mathcal{C}$, thus, as $|X| \le K = 1 + 2a + n' \cdot 2a$, X contains r, $\mathcal{C}$, exactly one endpoint of each pentagon edge, exactly one neighbour of each vertex from $\mathcal{C}$ and nothing more. In particular, $r^1, r^2 \notin X$.

For each $0 \le k < a$ we construct an assignment ϕ_k as follows. For each block $F_t = \{x_t^1, x_t^2\}$ we define:

1. $\phi_k(x_t^1)$ to be true if $v_{t,k}^1 \in X$ and false if $v_{t,k}^2 \in X$;

2. $\phi_k(x_t^2)$ to be true if $h_{t,k}^1 \in X$ and false if $h_{t,k}^2 \in X$.

We now show that the assignments ϕ_k cannot differ much for all indices $0 \le k < a$. Note that for each block $F_t = \{x_t^1, x_t^2\}$ and $0 \le k < a - 1$:

1. if $\phi_k(x_t^1)$ is true, then $\phi_{k+1}(x_t^1)$ is also true, as otherwise X contains no endpoint of the pentagon edge $v_{t,k}^2 v_{t,k+1}^1$.

2. if $\phi_k(x_t^2)$ is true, then $\phi_{k+1}(x_t^2)$ is also true, as otherwise $h_{t,k}^2, h_{t,k+1}^1 \notin X$ and either the vertex $v_{t,k}^2$ is not connected to the root in $G[X]$ (if $v_{t,k}^2 \in X$, since vertices from $\mathcal{C}$ are leaves in $G[X]$) or $G \setminus X$ contains a cycle of length five consisting of vertices $v_{t,k}^2, h_{t,k}^2, r^2, r^1$ and $h_{t,k+1}^1$ (if $v_{t,k}^2 \notin X$).

For each variable x we define a sequence $\widehat{\phi}_x(k) = \phi_k(x)$, $0 \le k < a$. From the reasoning above we infer that for each variable x the sequence $\widehat{\phi}_x(k)$ can change its value at most once, from false to true. Thus, as $a = m(n+1)$, we conclude that there exists $0 \le j < n+1$ such that for all $0 \le i < m$ the assignments ϕ_{mj+i} are equal.

We claim that the assigment $\phi = \phi_{mj}$ satisfies Φ. Consider a clause C_i and focus on the vertex $c_{i,j} \in X$. As $G[X]$ is connected, there exists a vertex x in the set $N(N(c_{i,j}))$ that belongs to $X \cap (\mathcal{V} \cup \mathcal{H})$. This vertex x corresponds to an assignment of one variable that both satisfies C_i (by the construction process) and is consistent with $\phi_{mj+i} = \phi$ (by the definition of ϕ_{mj+i}). Thus the assignment ϕ satisfies C_i and the proof is finished. $\square$

Pathwidth bound

Lemma D.16. *Pathwidth of the graph G is at most $n' + O(1)$. Moreover, a path decomposition of such width can found in polynomial time.*

Proof. We give a mixed search strategy to clean the graph with $n' + O(1)$ searchers. First we put five searchers on the vertices r^1, r^2, r, r^* and r^{**} and then remove the searchers from the vertices r^* and r^{**}. The searchers on the vertices r^1, r^2 and r remain till the end of the cleaning process.

We search the graph in $a = m(n+1)$ rounds. At the beginning of round k ($0 \le k < a$) there are searchers on all vertices $v_{t,k}^1$ for $1 \le t \le n'$. Let $0 \le i < m$ and $0 \le j < n+1$ be integers such that $k = i + mj$. We first place three searchers on $c_{i,j}$, $c_{i,j}^*$ and $c_{i,j}^{**}$ and afterwards we remove the searchers from $c_{i,j}^*$ and $c_{i,j}^{**}$.

Then, for each $1 \le t \le n'$ in turn we put $O(1)$ searchers on vertices $v_{t,k}^2$, $v_{t,k+1}^1$, $h_{t,k}^1$, $h_{t,k}^2$, the guard vertices of pentagon edges $v_{t,k}^1 v_{t,k}^2$, $v_{t,k}^2 v_{t,k+1}^1$, $h_{t,k}^1 h_{t,k}^2$, and all vertices $c_{i,j,t,\beta}^\gamma$, and then remove searchers from the vertices $v_{t,k}^1$, $v_{t,k}^2$, $h_{t,k}^1$, $h_{t,k}^2$, all aforementioned guard vertices and vertices $c_{i,j,t,\beta}^\gamma$. The last step of the round is removing a searcher from the vertex $c_{i,j}$. After the last round the whole graph G is cleaned. Since we reuse searchers in the cleaning process, $n' + O(1)$ searchers suffice to clean the graph.

Using the above graph cleaning process a path decomposition of width $n' + O(1)$ can be constructed in polynomial time. $\square$

Proof of Theorems D.12 and D.13. Suppose CONNECTED FEEDBACK VERTEX SET or CONNECTED ODD CYCLE TRANSVERSAL can be solved in $(4 - \varepsilon)^p |V|^{O(1)}$ provided that we are given a path decomposition of G of width p. Given an instance of SAT we construct an instance of CONNECTED FEEDBACK VERTEX SET or CONNECTED ODD CYCLE TRANSVERSAL using the above construction and solve it using the $(4 - \varepsilon)^p |V|^{O(1)}$ time algorithm. Lemmata D.14, D.15, together with an observation that any connected feedback vertex set is also a connected odd cycle transversal in G, ensure correctness, whereas Lemma D.16 implies that running time of our algorithm is $(4 - \varepsilon)^{n/2} |V|^{O(1)}$, however we have $(4 - \varepsilon)^{n/2} = (\sqrt{4 - \varepsilon})^n$ and $\sqrt{4 - \varepsilon} < 2$. This concludes the proof. $\square$

D.4 FEEDBACK VERTEX SET

Theorem D.17. *Assuming SETH, there cannot exist a constant $\varepsilon > 0$ and an algorithm that given an instance $(G = (V, E), k)$ together with a path decomposition of the graph G of width p solves the* FEEDBACK VERTEX SET *problem in $(3 - \varepsilon)^p |V|^{O(1)}$ time.*

Construction The construction here is a bit different than in the previous subsections, as we do not have a constraint that the solution needs to induce a connected subgraph. As one may notice, this connectivity constraint is used intensively in the previous subsections, in particular, it gives quite an easy way to verify the correctness of the assignment encoded in the solution. In the case of FEEDBACK VERTEX SET we need to do it in a different and more complicated way. Parts of the construction here resembles the lower bound proof for ODD CYCLE TRANSVERSAL by Lokshtanov, Marx and Saurabh [47].

Before we start, let us introduce one small gadget, used already in the proof of the lower bounds for CONNECTED FEEDBACK VERTEX SET and CONNECTED ODD CYCLE TRANSVERSAL. By *introducing a triangle edge vw* we mean the following construction: we add a new vertex u_{vw} and edges vw, vu_{vw}, wu_{vw}. We call u_{vw} *a guard vertex* and in the graph G its degree equals two. Note that any feedback vertex set X in G needs to intersect the triangle composed of vertices $\{v, w, u_{vw}\}$ and, moreover, if $u_{vw} \in X$ then $X \setminus \{u_{vw}\} \cup \{v\}$ is also a feedback vertex set in G of not greater size. Thus we may focus only on feedback vertex sets in G that do not contain guard vertices. Each such feedback vertex set needs to include at least one endpoint of each triangle edge.

Given $\varepsilon > 0$ and an instance Φ of SAT with n variables and m clauses we construct a graph G as follows. We first choose a constant integer η, which value depends on ε only. The exact formula for η is presented later. We partition variables of Φ into groups $F_1, \ldots, F_{n'}$, each of size at most $\beta = \lfloor \log 3^\eta \rfloor$, hence $n' = \lceil n/\beta \rceil$. Note that now $\eta n' \sim n/\log 3$, the pathwidth of G will be roughly $\eta n'$.

First, we add to the graph G a vertex r, called a *root*.

Second, we take $a = m(2\eta n' + 1)$ and for each $1 \leq t \leq n'$ and $1 \leq \ell \leq \eta$ we create a path $\mathcal{P}_{t,\ell}$ consisting of $3a$ vertices $v_{t,\ell,k}^\alpha$, $0 \leq k < a$ and $1 \leq \alpha \leq 3$, arranged in the following order:

$$v_{t,\ell,0}^1, v_{t,\ell,0}^2, v_{t,\ell,0}^3, v_{t,\ell,1}^1, \ldots, v_{t,\ell,a-1}^1, v_{t,\ell,a-1}^2, v_{t,\ell,a-1}^3.$$

Let $\mathcal{V}_{t,\ell,k} = \{v_{t,\ell,k}^\alpha : 1 \leq \alpha \leq 3\}$, $\mathcal{V}_{t,k} = \bigcup_{\ell=1}^\eta \mathcal{V}_{t,\ell,k}$ and $\mathcal{V} = \bigcup_{t=1}^{n'} \bigcup_{k=0}^{a-1} V_{t,k}$.

Third, for each two consecutive vertices $v_{t,\ell,k}^\alpha$, $v_{t,\ell,k'}^{\alpha'}$ on the path $\mathcal{P}_{t,\ell}$ we introduce vertices $h_{t,\ell,k}^{\alpha,1}$ and $h_{t,\ell,k}^{\alpha,2}$ connected by a triangle edge. Furthermore, we add edges $h_{t,\ell,k}^{\alpha,1} v_{t,\ell,k}^\alpha$, $h_{t,\ell,k}^{\alpha,1} r$, $h_{t,\ell,k}^{\alpha,2} v_{t,\ell,k'}^{\alpha'}$, $h_{t,\ell,k}^{\alpha,2} r$. Let $\mathcal{H}$ be the set of all vertices $h_{t,\ell,k}^{\alpha,\gamma}$.

We now provide a description of a group gadget $\mathbf{B}_{t,k}$, which will enable us to encode 2^β possible assignments of one group of β variables. Fix a block F_t, $1 \leq t \leq n'$, and a position k, $0 \leq k < a$. The group gadget $\mathbf{B}_{t,k}$ includes (already created) vertices $v_{t,\ell,k}^\alpha$, $h_{t,\ell,k}^{\alpha,\gamma}$ ($1 \leq \ell \leq \eta$, $1 \leq \alpha \leq 3$, $1 \leq \gamma \leq 2$) and all guard vertices in the triangle edges between them. Moreover, we perform the following construction.

For each $1 \leq \ell \leq \eta$ we introduce three vertices $p_{t,\ell,k}^\alpha$ ($1 \leq \alpha \leq 3$), pairwise connected by triangle edges. Moreover, for each $1 \leq \alpha \leq 3$ we connect $p_{t,\ell,k}^\alpha$ and $v_{t,\ell,k}^\alpha$ by a triangle edge. Let $\mathcal{P}$ be the set of all vertices $p_{t,\ell,k}^\alpha$ in the whole graph G.

In order to encode 2^β assignments we consider subsets of $\mathcal{V}_{t,k}$ that contain *exactly one* vertex out of each set $\mathcal{V}_{t,\ell,k}$. For each sequence $S = (s_1, \ldots, s_\eta) \in \{1, 2, 3\}^\eta$ we perform the following construction. First, for each $1 \leq \ell \leq \eta$ we introduce three vertices $q_{t,\ell,k}^{S,\alpha}$ ($1 \leq \alpha \leq 3$), pairwise connected by triangle edges. Second, we connect $q_{t,\ell,k}^{S,\alpha}$ with $v_{t,\ell,k}^\alpha$ ($1 \leq \alpha \leq 3$) with a triangle edge. Third, we introduce a vertex $x_{t,k}^S$ and connect all vertices $q_{t,\ell,k}^{S,s_\ell}$ ($1 \leq \ell \leq \eta$) and the vertex $x_{t,k}^S$ into a cycle $\mathcal{Q}_{t,k}^S$. Fourth, we connect all vertices $x_{t,k}^S$ for $S \in \{1, 2, 3\}^\eta$ into a cycle $\mathcal{X}_{t,k}$. Finally, for each $S \in \{1, 2, 3\}^\eta$ we introduce two new vertices $y_{t,k}^S$ and $z_{t,k}^S$ and triangle edges $x_{t,k}^S y_{t,k}^S$ and $y_{t,k}^S z_{t,k}^S$. Let $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ be the sets of all vertices $x_{t,k}^S$, $y_{t,k}^S$ and $z_{t,k}^S$, respectively. This finishes the construction of the group gadget $\mathbf{B}_{t,k}$.

We add vertices used to check the satisfiability of the formula Φ. Observe that for a group of variables F_t there are at most 2^β possible assignments and there are $3^\eta \geq 2^\beta$ vertices $x_{t,k}^S$ for sequences S from the set $\{1, 2, 3\}^\eta$ in each group gadget $\mathbf{B}_{t,k}$, hence we can assign a *unique* sequence S to each assignment. Let $C_0, \ldots, C_{m-1}$ be the clauses of the formula Φ. For each clause C_i and for each $0 \leq j < 2\eta n' + 1$ we perform the following construction that uses gadgets $\mathbf{B}_{t,mj+i}$ for $1 \leq t \leq n'$. For each group of variables F_t we consider the set $\mathcal{S}_{t,i}$ of all sequences $S \in \{1, 2, 3\}^\eta$ that correspond to an assignment of F_t satisfying the clause C_i (i.e., one of the variables of F_t is assigned a value such that C_i is already satisfied). We connect all vertices $z_{t,mj+i}^S$ for $1 \leq t \leq n'$, $S \in \mathcal{S}_{t,i}$ into a cycle $\mathcal{C}_{i,j}$. The vertices on the cycle $\mathcal{C}_{i,j}$ are sorted in the order of increasing value of t.
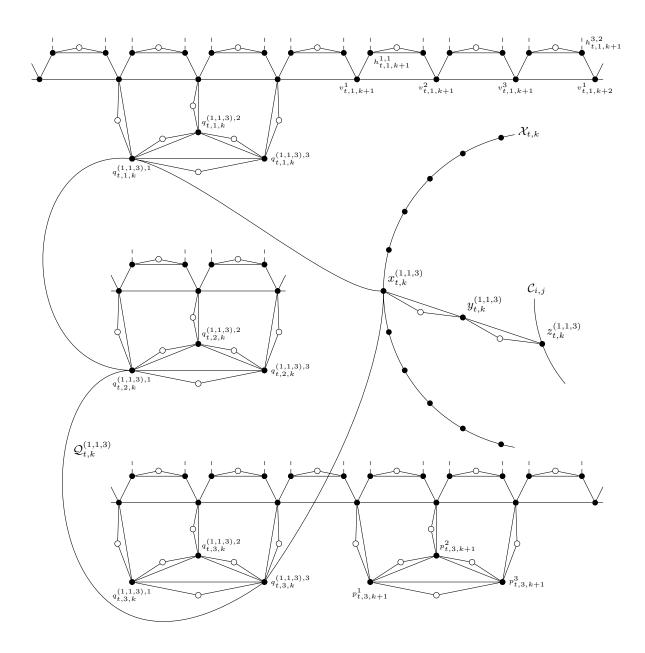
Figure 11: Part of the construction around group gadget $\mathbf{B}_{t,k}$ for $\eta = 3$. Dashed edges are connecting a vertex with the root r. Empty circles represent guard vertices.

We can view the whole construction as a matrix of group gadgets, where each row corresponds to some group of variables F_t and each column is devoted to some clause in such a way that each clause gets $(2\eta n' + 1)$ private columns (but not consecutive) of the group gadget matrix, as in Figure 12.
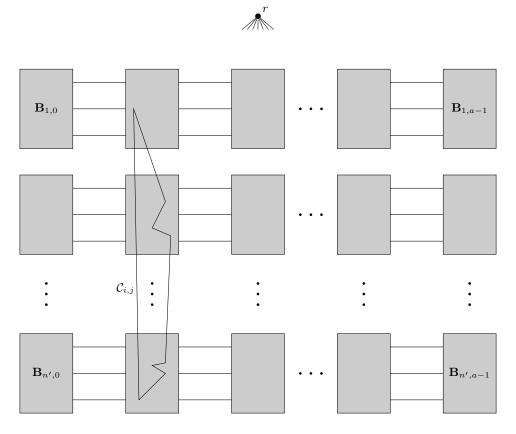

Figure 12: The arrangement of the group gadgets in G.

Finally, we define the size of the feedback vertex set we are looking for as $K = K_h + K_v + K_p + K_q + K_x + K_y$, where

$$K_h = \eta n'(3a - 1) \qquad K_v = \eta n' a \qquad K_p = \eta n' \cdot 2a$$
$$K_q = \eta n' \cdot 2a3^\eta \qquad K_x = n'a \qquad K_y = n'a3^\eta.$$

Correctness

Lemma D.18. *If Φ has a satisfying assignment, then there exists a feedback vertex set in G of size K.*

Proof. Given a satisfying assignment ϕ of the formula Φ we construct a feedback vertex set $X \subseteq V$ as follows.

For each group of variables F_t we consider the sequence $S_t \in \{1, 2, 3\}^\eta$ which corresponds to the restriction of

the assignment ϕ to the variables of F_t. Let

$$X_v = \{v_{t,\ell,k}^{S_t(\ell)} : 1 \le t \le n', 1 \le \ell \le \eta, 0 \le k < a\}$$

$$X_p = \{p_{t,\ell,k}^{\alpha} : 1 \le t \le n', 1 \le \ell \le \eta, 0 \le k < a, 1 \le \alpha \le 3, \alpha \ne S_t(\ell)\}$$

$$X_q = \{q_{t,\ell,k}^{S,\alpha} : 1 \le t \le n', 1 \le \ell \le \eta, 0 \le k < a, S \in \{1,2,3\}^\eta, 1 \le \alpha \le 3, \alpha \ne S_t(\ell)\}$$

$$X_x = \{x_{t,k}^{S_t} : 1 \le t \le n', 0 \le k < a\}$$

$$X_y = \{y_{t,k}^{S} : 1 \le t \le n', 0 \le k < a, S \in \{1,2,3\}^\eta, S \ne S_t\}$$

$$X_z = \{z_{t,k}^{S_t} : 1 \le t \le n', 0 \le k < a\}$$

Moreover, we define the set $X_h \subseteq \mathcal{H}$ to contain, for each $1 \le t \le n'$, $1 \le \ell \le \eta$, $0 \le k < a$, $1 \le \alpha \le 3$ and $(k, \alpha) \ne (a-1, 3)$, the vertex $h_{t,\ell,k}^{\alpha,1}$ if $v_{t,\ell,k}^{\alpha} \notin X_v$ and the vertex $h_{t,\ell,k}^{\alpha,2}$ otherwise. Note that $|X_v| = K_v$, $|X_p| = K_p$, $|X_q| = K_q$, $|X_x| = K_x$, $|X_y| + |X_z| = K_y$ and $|X_h| = K_h$. Thus a set $X = X_v \cup X_p \cup X_q \cup X_x \cup X_y \cup X_z \cup X_h$ is of size K.

To finish the proof we need to verify that X is a feedback vertex set of G. First, we ensure that X includes at least one endpoint of every triangle edge in G.

1. X includes one vertex from each pair $h_{t,\ell,k}^{\alpha,1}$ and $h_{t,\ell,k}^{\alpha,2}$.

2. X includes two out of three vertices in each triple $p_{t,\ell,k}^{\alpha}$, $1 \le \alpha \le 3$, and in each triple $q_{t,\ell,k}^{S,\alpha}$, $1 \le \alpha \le 3$.

3. If $p_{t,\ell,k}^{\alpha} \notin X$ or $q_{t,\ell,k}^{S,\alpha} \notin X$, then $v_{t,\ell,k}^{\alpha} \in X$.

4. If $y_{t,k}^{S} \notin X$ then both $x_{t,k}^{S}$ and $z_{t,k}^{S}$ are in X.

Let G_0 be the graph G with deleted guard vertices. We have just shown that each guard vertex in $G \setminus X$ is of degree zero or one. Thus if $G_0 \setminus X$ is a forest, then $G \setminus X$ is a forest too.

Now note that $X_p \cup X_q \cup X_v$ separates $(\{r\} \cup \mathcal{V} \cup \mathcal{H}) \setminus X$ from the rest of the graph G_0. To see this recall that if $p_{t,\ell,k}^{\alpha} \notin X$ or $q_{t,\ell,k}^{S,\alpha} \notin X$, then $v_{t,\ell,k}^{\alpha} \in X$. Let $G_1 = G_0[\{r\} \cup \mathcal{V} \cup \mathcal{H}]$. We now show that $G_1 \setminus X$ is a forest. Note that in $G_1 \setminus X_h$ each cycle contains at least one vertex from $\mathcal{V}$. Recall that the set X_v contains every third vertex on each path $\mathcal{V}_{t,\ell}$. Let $v_{t,\ell,k}^{\alpha}$ and $v_{t,\ell,k'}^{\alpha'}$ be any two consecutive vertices on $\mathcal{V}_{t,\ell}$ that are not in X_v. It is straightforward to check that that in the set of at most 4 neighbours of these two vertices in $\mathcal{H}$, at most one is not in X_h. Thus, the vertices in $\mathcal{V} \setminus X_v$ do not take part in any cycle in $G_1 \setminus X$, and $G_1 \setminus X$ is a forest.

Now note that $X_v \cup X_q \cup X_x \cup X_y$ separates $(\mathcal{Q} \cup \mathcal{X}) \setminus X$ from the rest of the graph G_0. To see this recall that if $p_{t,\ell,k}^{\alpha} \notin X$ or $q_{t,\ell,k}^{S,\alpha} \notin X$, then $v_{t,\ell,k} \in X$ and if $x_{t,k}^{S} \notin X$ then $y_{t,k}^{S} \in X$. Let $G_2 = G_0[\mathcal{Q} \cup \mathcal{X}]$. We now show that $G_2 \setminus X$ is a forest. The vertices in $\mathcal{Q} \cup \mathcal{X}$ from different group gadgets are not adjacent, thus we focus on a single group gadget $\mathbf{B}_{t,k}$. Let $S \in \{1,2,3\}^\eta$. Note that in $G_2 \setminus X$ the vertices from $\mathcal{Q}_{t,k}^{S} \setminus X$ are of degree at most two, except for the vertex $x_{t,k}^{S}$. If $S \ne S_t$, then for any $1 \le \ell \le \eta$ such that $S(\ell) \ne S_t(\ell)$ we have $q_{t,\ell,k}^{S,S(\ell)} \in X_q$ and the cycle $\mathcal{Q}_{t,k}^{S}$ is intersected by X. On the other hand, if $S = S_t$, then $x_{t,k}^{S} \in X_x$. Thus, the vertices $\mathcal{Q} \setminus X$ are not contained in any cycle in $G_2 \setminus X$. Moreover, on each cycle $\mathcal{X}_{t,k}$ we have $x_{t,k}^{S_t} \in X_x$ and we infer that $G_2 \setminus X$ is a forest.

As for $\mathcal{Y}$, note that if $y_{t,k}^{S} \notin X$ then $S = S_t$ and $x_{t,k}^{S}, z_{t,k}^{S} \in X$, and $y_{t,k}^{S}$ is isolated in $G_0 \setminus X$.

We are left with $\mathcal{Z}$. The graph $G_3 = G_0[\mathcal{Z}]$ consists of a cycles $\mathcal{C}_{i,j}$, $0 \le i < m$, $0 \le j < 2\eta n' + 1$. Consider a clause C_i and an index $0 \le j < 2\eta n' + 1$. As ϕ satisfies C_i, there exists a block F_t, such that a variable from this block satisfies C_i. Then $z_{t,mi+j}^{S_t}$ is both on the cycle $\mathcal{C}_{i,j}$ and in X, and $G_3 \setminus X$ is a forest, too. □

Lemma D.19. *If there exists a feedback vertex set X of size at most K in the graph G, then Φ has a satisfying assignment.*

Proof. As it is discussed at the begining of the construction process, we may assume that no guard vertex is in X. Let $X_h = X \cap \mathcal{H}$, similarly we define X_v, X_p, X_q, X_x, X_y and X_z. Furthermore, we assume that, among all feedback vertex sets of size at most K in G that do not contain any guard vertex, the set X is such a one that $|X_p|$ is the smallest possible.

We now lower bound the sizes of the sets X_h, X_v, X_p, X_q, X_x, X_y and X_z.

1. For each $1 \leq t \leq n'$, $0 \leq k < a$, $S \in \{1,2,3\}^\eta$, at least one of the vertices $y_{t,k}^S$ and $z_{t,k}^S$ is in X (as they are connected by a triangle edge), thus $|X_y| + |X_z| \geq n'a3^\eta = K_y$.

2. For each $1 \leq t \leq n'$ and $0 \leq k < a$ at least one vertex of the set X needs to hit the cycle $\mathfrak{X}_{t,k}$, to $|X_x| \geq n'a = K_x$.

3. For each $1 \leq t \leq n'$, $0 \leq k < a$, $1 \leq \ell \leq \eta$ and $S \in \{1,2,3\}^\eta$ at least two vertices out of the triple $\{q_{t,\ell,k}^{S,\alpha} : 1 \leq \alpha \leq 3\}$ need to be included in X, thus $|X_q| \geq \eta n' \cdot 2a3^\eta = K_q$.

4. For each $1 \leq t \leq n'$, $0 \leq k < a$, $1 \leq \ell \leq \eta$ at least two vertices out of the triple $\{p_{t,\ell,k}^\alpha : 1 \leq \alpha \leq 3\}$ need to be included in X. Moreover, if $p_{t,\ell,k}^\alpha \notin X$ for some $1 \leq \alpha \leq 3$, then $v_{t,\ell,k}^\alpha \in X$, as otherwise the triangle edge $v_{t,\ell,k}^\alpha p_{t,\ell,k}^\alpha$ is not covered by X. Thus $|X_p| + |X_v| \geq \eta n' \cdot 3a = K_p + K_v$.

5. For each $1 \leq t \leq n'$, $0 \leq k < a$, $1 \leq \ell \leq \eta$, $1 \leq \alpha \leq 3$, such that $(k,\alpha) \neq (a-1,3)$, at least one endpoint of the triangle edge $h_{t,\ell,k}^{\alpha,1} h_{t,\ell,k}^{\alpha,2}$ needs to be included in X. Thus $|X_h| \geq \eta n'(3a-1) = K_h$.

As $|X| \leq K = K_h + K_v + K_p + K_q + K_x + K_y$, we infer that in all aforementioned inequalities we have equalities, and $r \notin X$.

Recall that we have assumed that $|X_p|$ is the smallest possible. Let $1 \leq t \leq n'$, $0 \leq k < a$, $1 \leq \ell \leq \eta$ and focus on the triple $\{p_{t,\ell,k}^\alpha : 1 \leq \alpha \leq 3\}$. If it is wholly contained in X, then $X \setminus \{p_{t,\ell,k}^1\} \cup \{v_{t,\ell,k}^1\}$ is also a feedback vertex set of G, of not greater size, not containing any guard vertex, and with smaller size of $|X_p|$. Thus X contains exactly two vertices out of each such triple, $|X_p| = K_p$, $|X_v| = K_v$ and X_v contains exactly one vertex out of each triple $\{v_{t,\ell,k}^\alpha : 1 \leq \alpha \leq 3\}$.

We strengthen the above observation by showing the following claim: for any $1 \leq t \leq n'$, $1 \leq \ell \leq \eta$ and any three consecutive vertices v_A, v_B, v_C on the path $\bigcup_{k=0}^{a-1} \mathcal{V}_{t,\ell,k}$, at least one of these vertices is in X_v. By contradiction, assume that $v_A, v_B, v_C \notin X$. Recall that the graph G contains vertices h_A^1, h_A^2, h_B^1, h_B^2, edges $r h_A^1$, $r h_A^2$, $r h_B^1$, $r h_B^2$, $h_A^1 v_A$, $h_A^2 v_B$, $h_B^1 v_B$, $h_B^2 v_C$ and triangle edges $h_A^1 h_A^2$, $h_B^1 h_B^2$. Note that $|\{h_A^1, h_A^2, h_B^1, h_B^2\} \setminus X| = 2$, as $|X_h| = K_h$ and X_h contains exactly one vertex from each pair connected by a triangle edge. These two vertices in $\mathcal{H} \setminus X$, together with v_A, v_B, v_C and the root r induce a subgraph of $G \setminus X$ with 6 vertices and 6 edges, a contradiction.

For $1 \leq t \leq n'$ and $1 \leq \ell \leq \eta$ let us define a sequence $s_{t,\ell}(k)$, $0 \leq k < a$, such that $v_{t,\ell,k}^\alpha \in X_v$ iff $\alpha = s_{t,\ell}(k)$. By the observation made in the previous paragraph we infer that the sequence $s_{t,\ell}$ cannot increase, thus its value can change at most twice. As $a = m(2\eta n' + 1)$, we infer that there exists an index $0 \leq j < 2\eta n' + 1$ such that for all $1 \leq t \leq n'$, $1 \leq \ell \leq \eta$ we have $s_{t,\ell}(mj) = s_{t,\ell}(mj + i)$ for all $0 \leq i < m$. For each block F_t, let $S_t = (s_{t,\ell}(mj))_{\ell=1}^\eta \in \{1,2,3\}^\eta$ and let ϕ be an assignment that corresponds to the sequence S_t for each block F_t. We claim that ϕ satisfies Φ.

Take any clause C_i, $0 \leq i < m$. Take any block F_t. As $|X_q| = K_q$, the set X_q includes exactly two vertices out of each triple $\{q_{t,\ell,mj+i}^{S,\alpha} : 1 \leq \alpha \leq 3\}$ for $1 \leq \ell \leq \eta$, $S \in \{1,2,3\}^\eta$. As $q_{t,\ell,mj+i}^{S,\alpha}$ is connected to $v_{t,\ell,mj+i}^\alpha$ by a triangle edge, we infer that $q_{t,\ell,mj+i}^{S,\alpha} \notin X$ iff $v_{t,\ell,mj+i}^\alpha \in X$, which is equivalent to $S_t(\ell) = \alpha$. Thus $x_{t,mj+i}^{S_t} \in X$, as otherwise the cycle $\mathcal{Q}_{t,mj+i}^{S_t}$ is disjoint with X. As $|X_x| = K_x$, the set X_x contains exactly one vertex out of each cycle $\mathfrak{X}_{t,k}$, and we infer that $x_{t,mj+i}^S \notin X$ for $S \neq S_t$. Recall that $x_{t,mj+i}^S$ and $y_{t,mj+i}^S$ are connected by a triangle edge, thus $y_{t,mj+i}^S \in X$ for $S \neq S_t$. As $|X_y| + |X_z| = K_y$, we know that the set X contains exactly one endpoint out of each triangle edge $y_{t,mj+i}^S z_{t,mj+i}^S$, and we infer that if $z_{t,mj+i}^S \in X$ then $S = S_t$. Finally, if X is a feedback vertex set in G, X hits the cycle $\mathcal{C}_{i,j}$, thus there exists a block F_t and a sequence $S \in \{1,2,3,\}^\eta$ such that $z_{t,mj+i}^S \in X$ and the assignment of the variables of the block F_t that corresponds to S satisfies C_i. However, we have proven that $z_{t,mj+i}^S \in X$ implies $S = S_t$, thus ϕ satisfies C_i and the proof is finished. $\qquad \square$

Pathwidth bound

Lemma D.20. *Pathwidth of the graph G is at most $\eta n' + O(\eta 3^\eta)$. Moreover a path decomposition of such width can found in polynomial time.*

Proof. We give a mixed search strategy to clean the graph with $\eta n' + O(\eta 3^\eta)$ searchers. First we put a searcher in the root r. This searcher remains there till the end of the cleaning process.

For a gadget $\mathbf{B}_{t,k}$ we call the vertices $v^1_{t,\ell,k}$ for $1 \leq \ell \leq \eta$, as *entry vertices*. We search the graph in $a = m(2\eta n' + 1)$ rounds. In the beginning of round k ($0 \leq k < a$) there are searchers on the entry vertices of the gadget $\mathbf{B}_{t,k}$ for every $1 \leq t \leq n'$. Let $0 \leq i < m$ and $0 \leq j < 2\eta n' + 1$ be integers such that $k = i + mj$. We place a searcher on the last vertex of the cycle $\mathcal{C}_{i,j}$ (recall that the vertices on $\mathcal{C}_{i,j}$ are sorted by the block number). Then, for each $1 \leq t \leq n'$ in turn we:

- put $O(\eta 3^\eta)$ searchers on all vertices of the group gadget $\mathbf{B}_{t,k}$,

- put η searchers on entry vertices of the group gadget $\mathbf{B}_{t,k+1}$ (except for the last round),

- put a searcher in the first vertex after the vertices of $\mathbf{B}_{t,k} \cap \mathcal{Z}$ on the cycle $\mathcal{C}_{i,j}$,

- remove searchers from all vertices of the group gadget $\mathbf{B}_{t,k}$.

The last step of the round is removing the remaining searcher on the cycle $\mathcal{C}_{i,j}$. After the last round the whole graph G is cleaned. Since we reuse $O(\eta 3^\eta)$ searchers for cleaning group gadgets, $\eta n' + O(\eta 3^\eta)$ searchers suffice to clean the graph.

Using the above graph cleaning process a path decomposition of width $\eta n' + O(\eta 3^\eta)$ can be constructed in polynomial time. □

Proof of Theorem D.17. Suppose FEEDBACK VERTEX SET can be solved in $(3-\varepsilon)^p |V|^{O(1)}$ time provided that we are given a path decomposition of G of width p. Let $\lambda = \log_3(3 - \varepsilon) < 1$. We choose η large enough such that $\frac{\log 3^\eta}{\lfloor \log 3^\eta \rfloor} < \frac{1}{\lambda}$. Given an instance of SAT we construct an instance of FEEDBACK VERTEX SET using the above construction and the chosen value of η. Next we solve FEEDBACK VERTEX SET using the $3^{\lambda p}|V|^{O(1)}$ time algorithm. Lemmata D.18, D.19 ensure correctness, whereas Lemma D.20 implies that running time of our algorithm is $3^{\lambda \eta n'}|V|^{O(1)}$, however we have

$$3^{\lambda \eta n'} = 2^{\lambda \eta n' \log 3} = 2^{\lambda n' \log 3^\eta} \leq 2^C \cdot 2^{\lambda n \log 3^\eta / \lfloor \log 3^\eta \rfloor} = 2^C \cdot 2^{\lambda' n}$$

for some $\lambda' < 1$ and $C = \lambda \log 3^\eta$. This concludes the proof. □