University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Marek Cygan

Cut&Count technique for graph connectivity problems parameterized by treewidth

PhD dissertation

Supervisor

dr hab. Łukasz Kowalik

Institute of Informatics
University of Warsaw

September 2011

Author's declaration:
aware of legal responsibility I hereby declare that I have written this dissertation myself and all the contents of the dissertation have been obtained by legal means.

September 21, 2011 . . . . . .
date *Marek Cygan*

Supervisor's declaration:
the dissertation is ready to be reviewed

September 21, 2011 . . . . . .
date *dr hab. Łukasz Kowalik*

Abstract

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

In this dissertation we breach this gap by introducing a novel technique we named Cut&Count that allows to produce $c^{\mathbf{tw}}|V|^{O(1)}$ time Monte Carlo algorithms for most connectivity-type problems, including HAMILTONIAN CYCLE, STEINER TREE, FEEDBACK VERTEX SET and CONNECTED VERTEX COVER.

These results have numerous consequences in various fields, like parameterized complexity, exact and approximate algorithms on planar and H-minor-free graphs and exact algorithms on graphs of bounded degree. In all these fields we are able to improve the best-known results for some problems. Also, looking from a more theoretical perspective, our results are surprising since the equivalence relation that partitions all partial solutions with respect to extendability to global solutions consists of at least $\mathbf{tw}^{\mathbf{tw}}$ equivalence classes for all these problems.

In contrast to the problems aiming to *minimize* the number of connected components that we solve using Cut&Count as mentioned above, we show that, assuming the Exponential Time Hypothesis, the aforementioned gap cannot be breached for some problems that aim to *maximize* the number of connected components like CYCLE PACKING.

The constant c in our algorithms 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.

Keywords: treewidth, randomization, isolation lemma, parameterized algorithms, exact exponential algorithms.

ACM Classification: F.2.2, G.2.1, G.2.2.

Streszczenie

Dla większości algorytmicznych problemów grafowych o charakterze lokalnym (w których świadectwo rozwiązania może zostać zweryfikowane przez sprawdzenie sąsiedztwa każdego z wierzchołków osobno) klasyczne techniki programowania dynamicznego prowadzą do algorytmów o złożoności $c^{\mathbf{tw}}|V|^{O(1)}$, gdzie $\mathbf{tw}$ oznacza szerokość drzewiastą danego grafu $G = (V, E)$ a c jest pewną stałą. Natomiast dla problemów z globalnym wymogiem dotyczącym całej struktury rozwiązania (takim jak spójność) najlepsze znane do tej pory algorytmy mają w złożoności czynnik $\mathbf{tw}^{\mathbf{tw}}$.

W rozprawie prezentujemy nową technikę, którą nazwaliśmy „tnij i zliczaj", która pozwala na uzyskanie algorytmów Monte Carlo o złożoności $c^{\mathbf{tw}}|V|^{O(1)}$ dla wielu problemów z warunkiem spójności, takich jak cykl Hamiltona, drzewo Steinera, zbiór rozcyklający czy też spójne pokrycie wierzchołkowe.

Przedstawione wyniki mają zastosowania w wielu poddziedzinach takich jak złożoność parametryzowana, dokładne i aproksymacyjne algorytmy dla grafów planarnych oraz grafów z zabronionym minorem jak również w dokładnych algorytmach wykładniczych dla grafów o ograniczonym stopniu. We wszystkich wymienionych przykładach technika tnij i zliczaj pozwala na poprawienie najlepszych istniejących wyników dla pewnych problemów. Ponadto, z bardziej teoretycznego punktu widzenia, nasze wyniki są zaskakujące gdyż relacja równoważności w zbiorze częściowych rozwiązań przechodzących przez separator względem ich rozszerzalności do pełnego rozwiązania składa się z co najmniej $\mathbf{tw}^{\mathbf{tw}}$ klas równoważności dla wszystkich wymienionych problemów.

W przeciwieństwie do problemów, gdzie celem jest zminimalizowanie liczby spójnych składowych rozwiązania, a które rozwiązujemy w czasie $c^{\mathbf{tw}}|V|^{O(1)}$ za pomocą techniki tnij i zliczaj, pokazaliśmy że algorytmy o takiej złożoności nie istnieją dla problemów, gdzie należy zmaksymalizować liczbę spójnych składowych rozwiązania (takich jak problem pakowania cykli), przy założeniu że nie istnieje algorytm podwykładniczy dla problemu spełnialności formuł.

Stała c we wszystkich naszych algorytmach jest niewielka, gdyż wynosi co najwyżej 4 dla grafów nieskierowanych oraz co najwyżej 6 dla grafów skierowanych. Co więcej dla kilku z rozważanych problemów udowodniliśmy że poprawienie stałych wynikających z użycia techniki tnij i zliczaj nie jest możliwe, przy bardzo silnych założeniach teoriozłożonościowych.

Słowa kluczowe: szerokość drzewiasta, randomizatcja, lemat o izolacji, algorytmy parametryzowane, dokładne algorytmy wykładnicze.

Klasyfikacja tematyczna ACM: F.2.2, G.2.1, G.2.2.

Dedicated to my Wonderful Wife

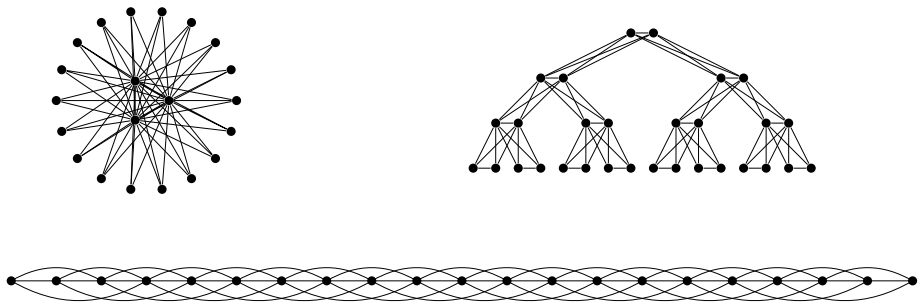
Contents

Chapter 1

Introduction

It is commonly believed that no NP-hard problem is solvable in polynomial time. However it often happens that real-life instances have much more structure comparing to a general instance specification from the mathematical problem definition. For this reason a concept of *parameterized complexity* arose, where hardness of an instance does not depend solely on its size. In the parameterized setting we assume that each instance is equipped with an additional value k — a parameter which aims to reflect the instance complexity. The goal is to provide an algorithm with $f(k)n^{O(1)}$ time complexity, where n is the instance size and f is a function independent of n. Observe that such an algorithm is polynomial for any constant value of k and moreover the degree of the polynomial is independent of the parameter value. The main motivation of research in the parameterized complexity is that practical instances happen to have small parameter values. As addressed in the textbook of Downey and Fellows [36], it appears that many graphs occurring in real-life applications have small *treewidth*, thus allowing efficient algorithms.

The notion of *treewidth* was introduced independently by Rose in 1974 [69] (under the name of partial k-tree) and in 1984 by Robertson and Seymour [68] as a parameter which is to estimate hardness of an instance by reflecting its resemblance to a tree. As an example we present three graphs of treewidth at most 4 which illustrate the structure of bounded treewidth graphs. The exact definition of treewidth is given in Section 2.2.



In many cases treewidth 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 [54], while the book of Niedermeier [64] 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 [4, 13, 27, 37] and parametrized algorithms [31, 39] 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 [61, 73] for problems like CONNECTED VERTEX COVER[1] and CUTWIDTH; and exact algorithms [39, 76] such as MINIMUM MAXIMAL MATCHING and DOMINATING SET.

In many cases, where the problem to be solved is "local" (loosely speaking this means that the property of the object to be found can be verified by checking separately the neighbourhood of each vertex), matching upper and lower bounds for the running time of the optimal solution are known. Of course, these are not absolute lower bounds (as this would imply P≠NP), but they hold under the *Strong Exponential Time Hypothesis* (defined in Section 2.4). For instance for the aforementioned $2^{\mathbf{tw}(G)}|V|^{O(1)}$ time 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 faster than in $O((2-\varepsilon)^{\mathbf{tw}(G)}|V|^{O(1)})$ time for any $\varepsilon > 0$. It is worth noting that it is not a universal opinion that SETH holds, but nevertheless reducing a problem to the classic SAT problem studied for several decades may be used as an evidence of hardness.

On the other hand, when the problem involves some sort of a "global" constraint — e.g., connectivity — the best known algorithms usually have a running time on the order of $2^{O(\mathbf{tw}(G)\log \mathbf{tw}(G))}|V|^{O(1)}$. In these cases the typical dynamic programming routine has to keep track of all the ways in which the solution can traverse the corresponding separator of the tree decomposition, that is $\Omega(l^l)$, where l is the size of the separator, i.e. treewidth. This obviously implies weaker results in the applications mentioned above. This problem was observed, for instance, by Dorn, Fomin and Thilikos [31, 32] and by Dorn et al. in [33], and the question whether the known $2^{O(\mathbf{tw}(G)\log \mathbf{tw}(G))}|V|^{O(1)}$ parametrized algorithms for HAMILTONIAN PATH, CONNECTED VERTEX COVER and CONNECTED DOMINATING SET are optimal was explicitly asked by Lokshtanov, Marx and Saurabh [59].

1.1 Our results

We introduce a technique we named "Cut&Count". For most problems involving a global constraint our technique gives a randomized algorithm with running time $c^{\mathbf{tw}(G)}|V|^{O(1)}$. In particular we are able to give such algorithms for HAMILTONIAN PATH, CONNECTED VERTEX COVER, CONNECTED DOMINATING SET (i.e. the three problems mentioned in [59]), as well as for all the other sample problems mentioned in [32]: LONGEST PATH, LONGEST CYCLE, FEEDBACK VERTEX SET, HAMILTONIAN CYCLE and GRAPH METRIC TRAVELLING SALESMAN PROBLEM. Moreover, the constant c is in all cases well

[1] Definitions of all the problems mentioned in this dissertation are gathered in Appendix A

defined and small. The randomization we mention comes from the usage of the Isolation Lemma [63]. 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 $G = (V, E)$, a tree decomposition of G of width t and a number k, in $3^t|V|^{O(1)}$ time either states that there exists a connected vertex cover of size at most k in G, or that it could not verify this hypothesis. If there indeed exists such a cover, the algorithm will return "unable to verify" with probability at most $1/2$.*

If the algorithm from Theorem 1.1 returns the "unable to verify" answer for a YES-instance, we say it returned a *false negative*. Note that in order to decrease the probability of a false negative to any small $\varepsilon > 0$ it suffices to repeat the algorithm $\log_2(\frac{1}{\varepsilon})$ times. We show similar results for a number of other global problems. As the exact value of c in the $c^{\mathbf{tw}(G)}$ expression is often important and highly non-trivial to obtain, we gather the results in column A of Table 1.1.

The previously known $2^{O(\mathbf{tw}(G) \log \mathbf{tw}(G))}$ dynamic programming routines for connectivity problems were thought to be optimal, because in these routines the dynamic programming table reflects the whole information that needs to be memoized in order to continue the computation. For every two distinct tables at some bag of the tree decomposition there exists a possible future on which the algorithm should behave differently. This resembles the notion of Myhill-Nerode equivalence classes [47], which, in a variety of settings, define the minimal automaton for a given problem. Hence, shrinking the size of the dynamic programming table would be, in some sense, trying to reduce the size of the minimal automaton. From this point of view our results come as a significant surprise.

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

Theorem 1.2. *Unless the Strong Exponential Time Hypothesis is false, for every constant $\varepsilon > 0$ there is no algorithm that given an instance $(G = (V, E), T, k)$ together with a path decomposition of the graph G of width p solves the* STEINER TREE *problem in $(3 - \varepsilon)^p|V|^{O(1)}$ time.*

Since each path decomposition is also a tree decomposition a lower bound for pathwidth is at least as strong as for treewidth. We have such matching lower bounds for several other problems presented in Column C of Table 1.1. We feel that the results for CONNECTED VERTEX COVER, CONNECTED DOMINATING SET, CONNECTED FEEDBACK VERTEX SET and CONNECTED ODD CYCLE TRANSVERSAL are of particular interest here and should be compared to the algorithms and lower bounds for the analogous problems without the connectivity requirement. For instance in the case of CONNECTED VERTEX COVER the results show that the increase in running time to $3^{\mathbf{tw}(G)}|V|^{O(1)}$ from the $2^{\mathbf{tw}(G)}|V|^{O(1)}$ algorithm of [64] 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

Problem name	A	B	C
STEINER TREE	3^t		$(3-\varepsilon)^p$
FEEDBACK VERTEX SET	3^t		$(3-\varepsilon)^p$
CONNECTED VERTEX COVER	3^t		$(3-\varepsilon)^p$
CONNECTED DOMINATING SET	4^t		$(4-\varepsilon)^p$
CONNECTED FEEDBACK VERTEX SET	4^t		$(4-\varepsilon)^p$
CONNECTED ODD CYCLE TRANSVERSAL	4^t		$(4-\varepsilon)^p$
UNDIRECTED MIN CYCLE COVER	4^t		
DIRECTED MIN CYCLE COVER	6^t		
UNDIRECTED LONGEST PATH (CYCLE)	4^t		
DIRECTED LONGEST PATH (CYCLE)	6^t		
EXACT k-LEAF SPANNING TREE	4^t		$(4-\varepsilon)^p$
EXACT k-LEAF OUTBRANCHING	6^t		
MAXIMUM FULL DEGREE SPANNING TREE	4^t		
GRAPH METRIC TRAVELLING SALESMAN PROBLEM	4^t		
(DIRECTED) CYCLE PACKING		$2^{\Omega(p \log p)}$	
(DIRECTED) MAX CYCLE COVER		$2^{\Omega(p \log p)}$	
MAXIMALLY DISCONNECTED DOMINATING SET		$2^{\Omega(p \log p)}$	

Table 1.1: Summary of our results. For the sake of presentation in each entry we skip the $|V|^{O(1)}$ multiplicative term. Column A is devoted to time complexities of our algorithms for treewidth (denoted by t) parametrization. For example, the entry at the third row represents Theorem 1.1. Column B contains lower bounds showing problems that are not solvable in $c^t|V|^{O(1)}$ time for any constant c unless the Exponential Time Hypothesis fails. Column C presents lower bounds under the Strong Exponential Time Hypothesis for pathwidth parametrization, which immediately imply lower bounds with the same constant for the treewidth parametrization. All problems statements can be found in Appendix A.

— problems that ask to maximize (instead of minimizing) the number of connected components in the solution seem more difficult to solve than the problems that ask to minimize (including problems where we demand that the solution forms a single connected component). As an evidence we prove that $c^{\mathbf{tw}(G)}|V|^{O(1)}$ solutions of the two problems mentioned above are unlikely (for Exponential Time Hypothesis definition see Section 2.4):

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

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

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* MAXIMALLY DISCONNECTED DOMINATING SET. *The parameter p denotes the width of a given path decomposition of the input graph.*

1.2 Consequences of the Cut&Count technique

As already 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. We do not explore all possible applications, but only give sample applications in various directions since a reader can easily obtain omitted by-products in an analogous manner.

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 number of techniques and approaches, but also in the ease in which new strong results can be obtained.

1.2.1 Parameterized algorithms on general graphs

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 [53]. 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 $f(k)|V|^{O(1)}$, 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 [5, 11, 20, 26, 35, 36, 43, 52, 65, 66]. The best — so far — result in this series is the $3.83^k k|V|^2$ result of Cao, Chen and Liu [18]. Our technique gives an improvement of their result:

Theorem 1.5. *There exists a Monte Carlo algorithm with constant one-sided error probability that solves the* FEEDBACK VERTEX SET *problem in a graph $G = (V, E)$ in $3^k|V|^{O(1)}$ time and polynomial space.*

We give similar improvements for CONNECTED VERTEX COVER (from $2.4882^k|V|^{O(1)}$ of [6] to $2^k|V|^{O(1)}$) and CONNECTED FEEDBACK VERTEX SET (from $46.2^k|V|^{O(1)}$ of [60] to $3^k|V|^{O(1)}$).

Furthermore we show that under Strong Exponential Time Hypothesis one can not count the parity of the number of connected vertex covers of size k in $(2-\varepsilon)^k|V|^{O(1)}$ time and our algorithm presented in Chapter 5 counts the parity of the number of connected vertex covers in $2^k|V|^{O(1)}$ time. To the best of our knowledge this is the first algorithm parameterized by the solution size for which an evidence of optimality is shown.

Details of the algorithms and lower bound for problems parameterized by the solution size can be found in Chapter 5.

1.2.2 Parameterized algorithms on H-minor-free graphs

A large branch of applications of algorithms parametrized by treewidth is the bidimensionality theory, used to find subexponential algorithms for various problems in H-minor-free graphs. In this theory we use the theorem of Demaine et al. [28], 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 programming algorithm suffices, thus giving us a $2^{O(\sqrt{k})}$ algorithm to check whether a graph has a vertex cover no larger than k. In the case of CONNECTED VERTEX COVER, however, the standard dynamic programming routine gives a $2^{O(\sqrt{k}\log k)}$ complexity — thus, we lose a logarithmic factor in the exponent.

There were a number of attempts to deal with this problem, taking into account the structure of the graph, and using it to deduce some properties of the tree decomposition under consideration. The latest and most efficient of those approaches is due to Dorn, Fomin and Thilikos [32], 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 which are rarely considered to be important in the bidimensionality theory), but rather in simplifying the proof — instead of delving into the combinatorial structure of each particular problem, we are back to a simple framework of applying the Robertson-Seymour theorem and then following up with a dynamic programming algorithm on the obtained tree decomposition.

The situation is more complicated in the case of problems on directed graphs. 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 [30] 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 1.6. *There exists a Monte Carlo algorithm with constant one-sided error probability that solves the* 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.*

1.2.3 Exact Algorithms on 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., [14]); however global problems such as the TRAVELLING SALESMAN PROBLEM (TSP) and HAMILTONIAN CYCLE have also received considerable attention [9, 38, 41, 50].

In what follows we let n denote the number of vertices of the given graph. The starting point is the following theorem by Fomin et al. [39]:

Theorem 1.7 ([39]). *For any $\varepsilon > 0$ there exists an integer n_ε such that for any graph G with $n > n_\varepsilon$ vertices,*

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

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

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

Corollary 1.8. *There exists a Monte Carlo algorithm with constant one-sided error probability that solves the* CONNECTED VERTEX COVER *problem in $O(1.201^n)$ time for cubic graphs, $O(1.443^n)$ for graphs of maximum degree 4 and $O(1.61^n)$ for graphs of maximum degree 5.*

Furthermore in Section 4.3.1 we improve the general $4^{\mathbf{tw}(G)}|V|^{O(1)}$ algorithm for the HAMILTONIAN CYCLE problem to $3^{\mathbf{pw}(G)}|V|^{O(1)}$ in case of a path decomposition of cubic graphs. Consequently we prove the following theorem which improves over previously best results for the maximum degree three $O(1.251^n)$ algorithm of Iwama and Nakashima [50] and for the degree four $O(1.657^n)$ algorithm of Björklund [7].

Corollary 1.9. *There exists a Monte Carlo algorithm with constant one-sided error probability that solves the* HAMILTONIAN CYCLE *problem in $O(1.201^n)$ time for cubic graphs and $O(1.588^n)$ for graphs of maximum degree 4.*

1.2.4 Consequences for exact algorithms on planar graphs

Recall from the previous section that n denotes the number of vertices of the given graph. Here we begin with a consequence of the work of Fomin and Thilikos [40]:

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

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

Corollary 1.11. *There exists a Monte Carlo algorithm with constant one-sided error probability that solves the* HAMILTONIAN CYCLE *problem on planar graphs in $O(4^{3.183\sqrt{n}}) = O(2^{6.366\sqrt{n}})$ time.*

To the best of our knowledge the best algorithm known so far was the $O(2^{6.903\sqrt{n}})$ of Bodlaender et al. [33].

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 [33]), and — as in the previous sections — well-behaved $c^{\sqrt{n}}$ algorithms for all mentioned problems.

1.3 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 any other field of characteristic 2), which causes them to disappear. This line of reasoning goes back to Tutte [74], and was recently used by Björklund [7] and Björklund et. al [10].

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 [44, 45] 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. [15]. 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 single exponential algorithms parametrized by treewidth for connectivity–type problems. For instance in the case of parametrized algorithms, sphere cuts [31, 33] (for planar and bounded genus graphs) and Catalan structures [32] (for H-minor-free graphs) were used to obtain $2^{O(\sqrt{k})}|V|^{O(1)}$ algorithms for a number of problems with connectivity requirements. To the best of our knowledge, however, no attempt to attack the problem directly was published before; indeed the non-existence of $2^{o(\mathbf{tw}(G)\log \mathbf{tw}(G))}|V|^{O(1)}$ algorithms was deemed to be more likely.

For classical graph problems the base of the exponent for treewidth parametrization was improved a few times. For example Alber et al. [2] gave a $4^{\mathbf{tw}(G)}|V|^{O(1)}$-time algorithm for DOMINATING SET, improving over the natural $9^{\mathbf{tw}(G)}|V|^{O(1)}$ algorithm of Telle and Proskurowski [72]. Recently, van Rooij et al. [75] observed that one could use fast subset convolution [8] to improve the running time of algorithms on graphs of bounded treewidth. Their results include a $3^{\mathbf{tw}(G)}|V|^{O(1)}$ algorithm for DOMINATING SET. However to the best of our knowledge our work is the first one where the time complexity is improved upon the space bound of the naive approach.

1.4 Organization of the dissertation

In Chapter 2 we introduce notation and basic definitions, including treewidth, pathwidth, the Isolation Lemma and the fast subset convolution together with related operators (some of them are new) which are later used to obtain small constant in the time complexity of our algorithms.

Chapter 3 is to present and demonstrate the Cut&Count technique by solving STEINER TREE and DIRECTED MIN CYCLE COVER problems. Applications of the technique to a number of different problems is described in Chapter 4.

In Chapter 5 we show a combination of Cut&Count technique and iterative compression which enables us to obtain faster than previously known algorithms for problems parameterized by the solution size, i.e., FEEDBACK VERTEX SET,CONNECTED VERTEX COVER,CONNECTED FEEDBACK VERTEX SET. Furthermore in Section 5.4 we prove that

under SETH there is no algorithm running in $(2-\varepsilon)^k|V|^{O(1)}$ which determines the parity of the number of connected vertex covers of size k, which matches the time complexity of the algorithm presented in the same chapter.

In Chapters 6 and 7 we present lower bounds of two different types. In Chapter 6 we prove that several problems, assuming the Exponential Time Hypothesis, do not admit an algorithm running in $2^{o(p \log p)}n^{O(1)}$ time, where p denotes the pathwidth of the input graph. The aim of this chapter is to show a substantial difference between problems maximizing and minimizing connectivity of a solution. In Chapter 7 we prove 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. Consequently we show that the Cut&Count technique not only allows us to obtain $c^{\mathbf{tw}(G)}|V|^{O(1)}$ time algorithms, breaking the $\mathbf{tw}(G)^{\Omega(\mathbf{tw}(G))}|V|^{O(1)}$ barrier, but for at least some problems leads to optimum values of the constant c.

We finish the dissertation with conclusions and open problems in Chapter 8. Finally Appendix A contains statements of problems enumerated in Table 1.1 and it is followed by Index.

Articles

Most of this dissertation comes from the article *Solving connectivity problems parameterized by treewidth in single exponential time*, which is a joint work with J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij and J. O. Wojtaszczyk. The extended abstract of this publication was accepted to the *52th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2011* [23].

The generalized fast subset convolution algorithm from Section 2.5 used as a tool in some applications of the Cut&Count technique was published in the article *Exact and approximate bandwidth*, co-authored by M. Pilipczuk, in proceedings of the *36th International Colloquium on Automata, Languages and Programming, ICALP 2009*, LNCS 5555, pages 304-315 [24] and later in *Theoretical Computer Science*, 411(40-42), pages 3701-3713, 2010 [25].

The lower bounds for CONNECTED VERTEX COVER parameterized by the solution size from Section 5.4 is a part the manuscript *On problems as hard as CNF-SAT*, co-authored by H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, M. Wahlström, which is currently under review [22].

Acknowledgements

I would like to thank my advisors: Krzysztof Diks and Łukasz Kowalik for encouraging me to pursue the scientific career and guiding through the research world.

Moreover I would like to thank all my co-authors: Daniel Binkele-Raible, Ljiljana Brankovic, Maxime Crochemore, Holger Dell, Henning Fernau, Fedor Fomin, Fabrizio

Grandoni, Costas S. Iliopoulos, Joachim Kneis, Łukasz Kowalik, Dieter Kratsch, Marcin Kubica, Alexander Langer, Stefano Leonardi, Mathieu Liedloff, Daniel Lokshtanov, Borut Luzar, Dániel Marx, Marcin Mucha, Yoshio Okamoto, Ramamohan Paturi, Geevarghese Philip, Jakub Radoszewski, Erik Jan van Leeuwen, Peter Rossmanith, Wojciech Rytter, Piotr Sankowski, Saket Saurabh, Ildikó Schlotter, Riste Škrekovski, Tomasz Walen, Magnus Wahlström, Mateusz Wykurz and especially Johan M. M. van Rooij, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Jakub Onufry Wojtaszczyk, who are the co-authors of the core publication my dissertation is based on.

Most importantly I would like to thank my Lovely Wife, for her support in all my efforts and understanding during conference trips and research visits.

Chapter 2

Notation and Preliminaries

In this chapter we introduce notation and definitions used throughout the dissertation, in particular we define the treewidth and the pathwidth of graphs. Furthermore in Section 2.3 we state the Isolation Lemma and present its proof for the sake of completeness. In Section 2.4 we present the statement of the (Strong) Exponential Time Hypothesis.

In Section 2.5 we recall the fast subset convolution and related operators. We also introduce some new operators: the generalized fast subset convolution and the $\mathbb{Z}_p$ product. These tools are used to speed-up the dynamic programming routines discussed in Chapter 4. However, we believe that the new operators are of independent interest and will become useful in other applications.

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

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

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

For two nodes 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 v belongs to the domain of s or not (in the latter case we extend the domain).

2.2 Treewidth and pathwidth

2.2.1 Tree Decompositions

Definition 2.1 (Tree Decomposition, [68])**.** *A* tree decomposition *of a (undirected or directed) graph G is a tree $\mathbb{T}$ in which each node $x \in \mathbb{T}$ has an assigned set of vertices $B_x \subseteq V$ (called a* bag*) such that $\bigcup_{x \in \mathbb{T}} B_x = V$ with the following properties:*

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

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

$$\begin{aligned} \mathrm{width}(\mathbb{T}) &= \max\{|X_x| - 1 \mid x \in \mathbb{T}\} \\ \mathbf{tw}(G) &= \min\{\mathrm{width}(\mathbb{T}) \mid \mathbb{T} \text{ a tree decomposition of } G\} \end{aligned}$$

We note that the minus one in the definition exists to set the treewidth of trees to one.

Dynamic programming algorithms on tree decompositions are often presented on nice tree decompositions which were introduced by Kloks [55]. 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 node 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 node 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 node 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 node 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 [55] 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 [12, 46].

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

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

Lemma 2.5 (Isolation Lemma, [63])**.** *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*

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

Proof from [63]. Consider an arbitrary element $u \in U$ of the universe which appears in at least one set of $\mathcal{F}$ and does not appear in at least one set of $\mathcal{F}$ (if such u does not exist then $|\mathcal{F}| = 1$ and the lemma obviously holds). Fix the weights of all elements except u. Define the threshold for element u, to be an integer α_u; such that if $\omega(u) \leq \alpha_u$ then u is contained in some minimum weight subset, and if $\omega(u) > \alpha_u$ then u is in no minimum weight subset.

Clearly, if $\omega(u) < \alpha_u$, then the element u must be in every minimum weight subset of $\mathcal{F}$. Thus ambiguity about the element u occurs iff $\omega(u) = \alpha_u$, since in this case there is a minimum weight subset of $\mathcal{F}$ that contains u and another which does not. In this case we shall say that the element u is *singular*.

We now make the crucial observation that the threshold, α_u, was defined without reference to the weight, $\omega(u)$. It follows that α_u is independent of $\omega(u)$. Since $\omega(u)$ is a uniformly distributed integer in $[1, N]$, we have:

$$\Pr[u \text{ is singular }] \leq \frac{1}{N}\,.$$

Since if no element of the universe U is singular then ω isolates $\mathcal{F}$ the lemma follows directly from the above formula and the union bound. □

It is worth mentioning that in [19], 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.

Consider a problem where the set of solutions is some family $\mathcal{F}$ of subsets of a universe U (usually U is the set of vertices or edges of the input graph). Suppose that we know how to obtain the parity of the number (that is $|\mathcal{F}| \bmod 2$) of solutions for this problem and we want to solve the decision variant, that is to verify whether there exists a solution or not. It might be the case that the number of solutions is positive and even, hence we can not assume that there is no solution solely based on the fact that the number of solutions is even. However if we use the Isolation Lemma and there exists at least one solution, then with high probability there exists some weight value w, for which the number of solutions of weight w is equal to one, which is an odd number. Hence it is enough to count the parity of the number of solutions of a prescribed weight. For this reason the Isolation Lemma has found many applications [63].

An alternative method to a similar end is obtained by using Polynomial Identity Testing [29, 70, 79] over a field of characteristic two. This second method has been already used in the field of exact and parameterized algorithms [7, 10, 56, 57, 77]. 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, 19]). The challenge of giving a full derandomization seems to be equally difficult for both methods [3, 51]. The usage of the Isolation Lemma gives greater polynomial overheads, however we choose to use it because in our opinion it makes the results less technical and hence easier to follow.

2.4 (Strong) Exponential Time Hypothesis

In this section we introduce the complexity assumptions defined by Impagliazzo and Paturi [48].

The k-SAT problem is a restriction of SAT, where each clause in the input formula has at most k literals. 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), conjectured by Impagliazzo, Paturi and Zane [49], asserts that $c_3 > 0$, whereas the Strong Exponential Time Hypothesis defined by Impagliazzo and Paturi [48] (SETH), asserts that $\lim_{k\to\infty} c_k = 1$. It is well known that SETH implies ETH [48].

We would like to mention that SETH is not a commonly believed assumption, however since SAT is a central problem of computational complexity and there was no algorithm that would refute SETH for several decades, it makes sense to use it as an evidence that a problem is hard.

2.5 Fast subset convolution

In this section we recall and generalize operators that we will use to efficiently handle join bags of a nice tree decomposition in dynamic programming routines in Chapter 4. All operators work on functions from a subset lattice $(2^B, \subseteq)$ for a finite set B where domain of each function is some ring R. We start with a *ζ-transform* and *μ-transform* definition of a function $f : 2^B \to R$.

Definition 2.6. *The* ζ-transform *of a function f is defined as a function $(\zeta f) : 2^B \to R$ as follows:*

$$(\zeta f)(T) = \sum_{T_1 \subseteq T} f(T_1).$$

The μ-transform *of a function f is defined as a function $(\mu f) : 2^B \to R$ as follows:*

$$(\mu f)(T) = \sum_{T_1 \subseteq T} (-1)^{|T \setminus T_1|} f(T_1).$$

As observed in [8] we can efficiently compute both ζ-transform and μ-transform using Yates's method [78]. By computing a function $h : 2^B \to R$ we mean determining $h(T)$ for every $T \subseteq B$. For completeness we give a proof below.

Lemma 2.7. *For a function $f : 2^B \to R$ we can compute ζf and μf in $O(|B|2^{|B|})$ ring operations.*

Proof. We present a proof for the ζ-transform since the proof for the μ-transform is analogous. We may assume $B = \{1, 2, \ldots, |B|\}$ and for $0 \leq b < |B|$ and for every sequence $(s_1, \ldots, s_b, t_{b+1}, \ldots, t_{|B|}) \in \{0,1\}^B$ we define

$$f_b(s_1, s_2, \ldots, s_b, t_{b+1}, \ldots, t_{|B|}) = \sum_{t'_i \in \{0,1\}, t'_i \leq t_i \text{ for } i=b+1,\ldots,|B|} f(s_1, s_2, \ldots, s_b, t'_{b+1}, \ldots, t'_{|B|})$$

Observe that $f_0 = \hat{f}$. We also set $f_{|B|} = f$. Moreover for $0 < b \leq |B|$ and for every sequence $(s_1, \ldots, s_{b-1}, t_b, \ldots, t_{|B|}) \in \{0,1\}^B$

$$f_{b-1}(s_1, s_2, \ldots, s_{b-1}, t_b, \ldots, t_{|B|}) = \sum_{t'_b \in \{0,1\}, t'_b \leq t_b} f_b(s_1, s_2, \ldots, s_{b-1}, t'_b, t_{b+1}, \ldots, t_{|B|}).$$

It follows that we can use dynamic programming to compute all functions f_b (for decreasing values of $b \in \{0, \ldots, |B|\}$). □

Now we recall the fast subset convolution (and its variants) [8].

Definition 2.8. *For two functions $f, g : 2^B \to R$ the* subset convolution *and* covering product *of f and g are defined as functions $f * g, f *_c g : 2^B \to R$ as follows:*

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

As observed in [8] we can compute the covering product by using ζ-transform, μ-transform and the subset convolution. Again we present the proof for completeness.

Lemma 2.9. *For two functions $f, g : 2^B \to R$ we have*

$$f *_c g = \mu((\zeta f) \cdot (\zeta g)),$$

where $\cdot$ is the elementwise product $((\zeta f) \cdot (\zeta g))(T) = ((\zeta f)(T))((\zeta g)(T))$.

Proof. Observe that

$$(\mu((\zeta f) \cdot (\zeta g)))(T) = \sum_{T_1 \subseteq T} (-1)^{|T \setminus T_1|} \sum_{A, B \subseteq T_1} f(A) g(B).$$

Moreover for each ordered pair (A, B) of subsets of T_1 the coefficient of the term $f(A)g(B)$ is equal to $\sum_{(A \cup B) \subseteq T_1 \subseteq T} (-1)^{|T \setminus T_1|}$ which is equal to one if $A \cup B = T$ and zero otherwise. □

Björklund et al. [8] proved that the *subset convolution* can be computed efficiently, which combined with Lemmas 2.7, 2.9 gives the following theorem.

Theorem 2.10 ([8])**.** *The subset convolution and the covering product of two given functions can be computed in* $2^{|B|}|B|^{O(1)}$ *ring operations.*

Instead of presenting the proof of the above theorem we prove its slight generalization as shown by Cygan and Pilipczuk in [25].

Definition 2.11. *Let* $p \geq 2$ *be an integer constant and let* B *be a finite set. For three functions* $t_1, t_2, t \in \{0, 1, \ldots, p-1\}^B$ *we say that* $t_1 + t_2 = t$ *iff* $t_1(b) + t_2(b) = t(b)$ *for all* $b \in B$*. For functions* $f, g : \{0, 1, \ldots, p-1\}^B \to R$ *define the generalized subset convolution as a function* $f *^p g : \{0, 1, \ldots, p-1\}^B \to R$ *such that for every* $t \in \{0, 1, \ldots, p-1\}^B$

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

Note that here the addition **is not** evaluated in $\mathbb{Z}_p$ but in $\mathbb{Z}$ and for $p = 2$ the operators $*^p$ and $*$ are equal. We show that when the ring R is equal to $\mathbb{Z}_2$ we can compute the generalized subset convolution efficiently using the Fast Fourier Transform. In fact one can prove the theorem for any ring R but then a more detailed analysis of the number of precision bits needed by FFT is necessary and since we only use functions with values from $\mathbb{Z}_2$ we focus on this particular ring.

Theorem 2.12. *Let* $p \geq 2$ *be an integer constant. For* $f, g : \{0, 1, \ldots, p-1\}^B \to \mathbb{Z}_2$ *their generalized subset convolution can be computed in* $p^{|B|}|B|^{O(1)}$ *time.*

Proof. We group tuples in $\{0, 1, \ldots, p-1\}^B$ according to the sum of their coordinates, that is let $S_k = \{(a_i)_{i=1}^{|B|} \in \{0, 1, \ldots, p-1\}^B : \sum_{i=1,\ldots,|B|} a_i = k\}$ for $k = 0, \ldots, (p-1)|B|$.

Furthermore we define a function $f_k : \{0, 1, \ldots, p-1\}^B \to \mathbb{Z}_2$ such that $f_k(t) = f(t)$ if $t \in S_k$ and $f_k(t) = 0$ otherwise (similarly we define g_k for each $k = 0, \ldots, (p-1)|B|$). Observe that

$$(f *^p g)(t) = \sum_{k_f = 0, \ldots, (p-1)|B|} \sum_{k_g = 0, \ldots, (p-1)|B|} (f_{k_f} *^p g_{k_g})(t)\,.$$

Thus by a cost of an $O(|B|^2)$ overhead it is sufficient to compute $f_{k_f} *^p g_{k_g}$ for fixed values of k_f and k_g.

The Fast Fourier Transform is an efficient algorithm which computes the discrete Fourier transform and its inverse [21]. To use FFT we need to look at our problem from a different angle. We treat each tuple $t \in \{0, 1, \ldots, p-1\}^B$ as a monomial $x^{\mathrm{val}(t)}$, where $\mathrm{val} : \{0, 1, \ldots, p-1\}^B \to [0, p^{|B|} - 1]$ is a function such that $\mathrm{val}(t)$ is the value of t when treated as a number written in base p.

Observation 2.13. *Let* $t_1 \in S_{k_f}, t_2 \in S_{k_g}$ *be two tuples of the form* $t_1 = (a_1, \ldots, a_{|B|}), t_2 = (b_1, \ldots, b_{|B|})$*. Then for each* $i = 1, \ldots, |B|$ *we have* $a_i + b_i < p$ *iff the sum of digits in the* p*-ary representation of* $\mathrm{val}(t_1) + \mathrm{val}(t_2)$ *is equal to* $k_f + k_g$*.*

Now we represent f_{k_f} and g_{k_g} as polynomials, that is

$$\mathrm{poly}(f_{k_f}) = \sum_{t \in S_{k_f}} f_{k_f}(t) x^{\mathrm{val}(t)},$$
$$\mathrm{poly}(g_{k_g}) = \sum_{t \in S_{k_g}} g_{k_g}(t) x^{\mathrm{val}(t)}.$$

Let P be a polynomial over $\mathbb{Z}_2$ defined as the product of $\mathrm{poly}(f_{k_f})$ and $\mathrm{poly}(g_{k_g})$. We can compute P using the standard Fast Fourier Transform in $O(p^{|B|}|B|^{O(1)})$ time (see e.g. [17]), since operations in $\mathbb{Z}_2$ take constant time.

Finally, for each $t \in \{0, 1, \ldots, p-1\}^B$, to obtain the value of $(f_{k_f} *^p g_{k_g})(t)$ we consider two cases. If $t \in S_{k_f+k_g}$ then we set $(f_{k_f} *^p g_{k_g})(t)$ as the coefficient of the monomial $x^{\mathrm{val}(t)}$ in P, otherwise we set $(f_{k_f} *^p g_{k_g})(t) = 0$. The correctness follows from Observation 2.13. □

The last needed variant of the subset convolution follows.

Definition 2.14. *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 $t_1(b) + t_2(b) \equiv t(b) \pmod p$ for all $b \in B$. For functions $f, g : \mathbb{Z}_p^B \to R$ define the* $\mathbb{Z}_p$ product *as a function $f *_x^p g : \mathbb{Z}_p^B \to R$ such that for every $t \in \mathbb{Z}_p^B$*

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

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 2.15. *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 abuse the notation

somewhat 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 [78]. We may assume $B = \{1, 2, \ldots, |B|\}$ and for $1 \leq b \leq |B|$ 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 $0 \leq b < |B|$

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

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

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

$$\begin{aligned}
\left(\widehat{\hat{f} \cdot \hat{g}}\right)(t) &= \sum_{s \in \mathbb{Z}_p^B} (\hat{f} \cdot \hat{g})(s)\varepsilon^{s \cdot t} \\
&= \sum_{s \in \mathbb{Z}_p^B} \left(\sum_{t_1 \in \mathbb{Z}_p^B} f(t_1)\varepsilon^{t_1 \cdot s} \right) \left(\sum_{t_2 \in \mathbb{Z}_p^B} g(t_2)\varepsilon^{t_2 \cdot s} \right) \varepsilon^{s \cdot t} \\
&= \sum_{t_1, t_2 \in \mathbb{Z}_p^B} f(t_1)g(t_2) \sum_{s \in \mathbb{Z}_p^B} \varepsilon^{s \cdot (t_1 + t_2 + t)} \\
&= \sum_{t_1, t_2 \in \mathbb{Z}_p^B} f(t_1)g(t_2) \sum_{s \in \mathbb{Z}_p^B} \varepsilon^{\sum_{b \in B} s(b)(t_1(b) + t_2(b) + t(b))} \\
&= \sum_{t_1, t_2 \in \mathbb{Z}_p^B} f(t_1)g(t_2) \sum_{s \in \mathbb{Z}_p^B} \prod_{b \in B} (\varepsilon^{t_1(b) + t_2(b) + t(b)})^{s(b)} \\
&= \sum_{t_1, t_2 \in \mathbb{Z}_p^B} f(t_1)g(t_2) \prod_{b \in B} \left(\sum_{s(b) \in \mathbb{Z}_p} \left(\varepsilon^{t_1(b) + t_2(b) + t(b)}\right)^{s(b)} \right) \\
&\left\{ \text{by the fact that } \sum_{s(b) \in \mathbb{Z}_p} (\varepsilon^\tau)^{s(b)} = p \text{ if } \tau = 0 \text{ and otherwise } \frac{1 - (\varepsilon^\tau)^p}{1 - \varepsilon^\tau} = 0 \right\} \\
&= \sum_{t_1, t_2 \in \mathbb{Z}_p^B} f(t_1)g(t_2) \prod_{b \in B} (p \cdot [t_1(b) + t_2(b) + t(b) = 0]) \\
&= \sum_{t_1, t_2 \in \mathbb{Z}_p^B} f(t_1)g(t_2)p^{|B|}[t_1 + t_2 + t = 0] \\
&= p^{|B|} \left(f *_x^p g\right)(-t)
\end{aligned}$$

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

Chapter 3

Cut&Count: Illustration of the technique

In this chapter we present the Cut&Count technique by demonstrating how it applies to the STEINER TREE and DIRECTED MIN CYCLE COVER problems. We go through the details in an expository manner, as we aim not only to show the solutions to these particular problems, but also to show the general workings.

We have chosen STEINER TREE and DIRECTED MIN CYCLE COVER problems to show that our technique can be applied both to vertex and edge selection problems, and both to undirected and directed graphs and also that not only it allows for ensuring connectivity but more generally it allows to minimize the number of connected components.

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

3.1 Steiner Tree

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

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

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

The Cut part. In the Cut part we always start with defining a set of *candidate solutions*

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

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

In this easy application of the Cut&Count method the only requirement that remains is that the set of terminals is contained in the candidate solution.

Although the cardinality of $\mathcal{R}_W$ cab be easily computed within the desired time bound, the parity of $|\mathcal{R}_W|$ needs not match the parity of $|\mathcal{S}_W|$. In order to describe the required set $\mathcal{C}_W$, we define a set of cuts, which are partitions of the set V into two sets $(V_1, V \setminus V_1)$. However to break the symmetry instead of considering all 2^n cuts we consider only 2^{n-1} of them by selecting some vertex v_1 (in most applications arbitrarily chosen) and assuring that $v_1 \in V_1$. Then we define when a subgraph is *consistent* with a cut.

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

Let v_1 be an arbitrary terminal. Define $\mathcal{C}_W$ as

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

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

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

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

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

Lemma 3.3. *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 3.2, we know that $|\mathcal{C}| = \sum_{X \in \mathcal{R}} 2^{\mathtt{cc}(G[X])-1}$. Thus $|\mathcal{C}| \equiv \left|\{X \in \mathcal{R} | \mathtt{cc}(G[X]) = 1\}\right| = |\mathcal{S}|$. □

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

Lemma 3.4. *Given $G = (V, E)$, $T \subseteq V$, an integer k, $\omega : V \to \{1, \ldots, N\}$ and a nice tree decomposition $\mathbb{T}$ of width t, there exists an algorithm that can determine $|\mathcal{C}_W|$ 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 in bags of all descendants of x, while by G_x we denoted the graph composed of vertices V_x and the edges E_x introduced by the descendants of x. We now define "partial solutions": For every bag $x \in \mathbb{T}$, integers $i = 0, \ldots, k$, $w = 0, \ldots, kN$ and $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$ define

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

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

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

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 (if there is only one child, we denote it by y).

- **Leaf bag** x:

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

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

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

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

  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 i,w have to be updated and we have to make sure we do not put $s(v_1) = \mathbf{1}_2$.

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

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

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

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

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

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

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

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

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

It is easy to see that the Lemma can now be obtained by combining the above recurrence with dynamic programming. Note that as we perform all calculations modulo 2, we take only constant time to perform any arithmetic operation. □

We conclude this section with the following theorem.

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

Proof. Our algorithm is as follows. Set $N = 2|V|$. Using Lemma 3.4 calculate $|\mathcal{C}_W|$ modulo 2, for every $W = 0, \ldots, kN$ in $3^t|V|^{O(1)}$ time. If for some W we have $|\mathcal{C}_W| \equiv 1$, then return **yes**. Otherwise return **no**.

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

3.2 Directed Cycle Cover

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

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

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

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

Definition 3.6. *For a directed graph $D = (V, A)$ a cut (V_1, V_2) is consistent if (V_1, V_2) is a consistent cut in the underlying undirected graph. A consistently cut subgraph of D is a pair $(X, (X_1, X_2))$ such that (X_1, X_2) is a consistent cut of the underlying undirected graph of $D[X]$.*

Definition 3.7. *For an integer W we define:*

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

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 arc from the set M;*

3. *$\mathcal{C}_W$ as all pairs $((X, M), (V_1, V_2))$ such that $(X, M) \in \mathcal{R}_W$, (V_1, V_2) is a consistent cut of $D[X]$ and $V(M) \subseteq V_1$.*

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

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

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

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

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

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 not containing arcs from M, 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 3.2) for any W and $(M, X) \in \mathcal{R}_W$ there are $2^{\mathtt{cc}(M,X)}$ cuts (V_1, V_2) such that $((X, M), (V_1, V_2)) \in \mathcal{C}_W$ and the lemma follows, because:

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

□

Now, it suffices to present a dynamic programming routine counting $|\mathcal{C}_W|$ modulo 2 in a bottom-up fashion. Since the proof for the LONGEST PATH problem is almost the same we present an algorithm for those two problems at once in the next chapter.

Lemma 3.9. *Given $D = (V, A)$, an integer k, a weight function $\omega : A \cup V \to \{1, \ldots, N\}$ and a tree decomposition $\mathbb{T}$ of width 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.*

Combining all the observations, we can conclude the following:

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

Proof. The algorithm is as follows. Set $U = A \times \{\mathbf{X}\} \cup A \times \{\mathbf{M}\}$ and $N = 2|U|$. Using Lemma 3.9 calculate $|\mathcal{C}_W|$ modulo 2, for every $W = 0, \ldots, (k + |V|)N$ in $6^t |V|^{O(1)}$ time. If for some W we have $|\mathcal{C}_W| \equiv 1$, then return **yes**. Otherwise return **no**.

The correctness follows from Lemma 3.8 and Isolation Lemma (Lemma 2.5). □

3.3 General idea overview

The Cut&Count technique applies to problems with certain connectivity requirements. Let $\mathcal{S} \subseteq 2^U$ be a set of solutions (usually the universe U is the set of vertices or edges/arcs of the input graph); we aim 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 disconnected candidate solutions. Furthermore, consider the set $\mathcal{C}$ of pairs (X, C) where $X \in \mathcal{R}$ and C is a consistent cut of X.
- **The Count part**: Compute $|\mathcal{C}|$ modulo 2 using a sub-procedure. Non-connected candidate solutions $X \in \mathcal{R} \setminus \mathcal{S}$ cancel since they are consistent with an even number of cuts. Connected candidates $x \in \mathcal{S}$ remain.

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

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

Algorithm 1: Cut&Count general schema.

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

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

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

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

Chapter 4

Cut&Count applied to several problems

In this chapter we describe in detail all the algorithms that appear in Column A of Table 1.1 except STEINER TREE which was already studied in the previous chapter. In particular we present missing details for the DIRECTED MIN CYCLE COVER problem. Furthermore we show that when we are given a path decomposition of a graph of maximum degree three then for some problems we may improve the time complexity and consequently we prove Corollary 1.9.

Subsequent sections in this chapter are independent hence a reader interested in a solution for a single problem may jump to the appropriate section directly. However in all sections we use notation and arguments described in details in the previous chapter hence we assume that when reading a section from this chapter the reader is already familiar with the previous chapter and tools from Section 2.5.

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

4.1 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 the solution 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 4.1. *A graph $G = (V, E)$ with n vertices and m edges is a forest iff it has at most $n - m$ connected components.*

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

Theorem 4.2. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves the* CONSTRAINED FEEDBACK VERTEX SET *problem in $3^t|V|^{O(1)}$ time. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

Proof. We use the Cut&Count technique. As the universe we take the set $U = V \times \{\mathbf{F}, \mathbf{M}\}$, 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 4.1 the graph G admits a feedback vertex set of size k containing S if and only if there exist integers B,W such that the set $\mathcal{S}_W^{n-k,B,n-k-B}$ is nonempty.

The Count part. Similarly as in the case of MIN CYCLE COVER (analogously to Lemma 3.8) note that for any A,B,C,W,$(X, M) \in \mathcal{R}_W^{A,B,C}$, there are $2^{\mathtt{cc}(M,G[X])}$ cuts (X_1, X_2) such that $((X, M), (X_1, X_2)) \in \mathcal{C}_W^{A,B,C}$, where by $\mathtt{cc}(M, G[X])$ we denote the number of connected components of $G[X]$ which do not contain any marker from the set M. Hence by Lemma 4.1 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.

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

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

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

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

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

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

- **Introduce vertex bag**:
$$A_x(a,b,c,w,s[v \to \mathbf{0}]) = A_y(a,b,c,w,s)$$
$$A_x(a,b,c,w,s[v \to \mathbf{1}_j]) = [v \notin S] A_y(a-1,b,c,w-\omega((v,\mathbf{F})),s)$$

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

  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**:

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

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

- **Join bag**:

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

  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 4.2. □

4.2 Non-connectivity problems with 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 well-known algorithm for X (or, in the case of CONNECTED FEEDBACK VERTEX SET, we run the algorithm for FEEDBACK VERTEX SET from Section 4.1), 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 4.1 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 4.3. *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.

4.2.1 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 a subset $X \subseteq V$ of cardinality k such that $S \subseteq X$, $G[X]$ is connected and each edge $e \in E$ is incident with at least one vertex from X?

Theorem 4.4. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves* CONSTRAINED CONNECTED VERTEX COVER *in $3^t|V|^{O(1)}$ time. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

There exists an easy proof of Theorem 4.4 by a reduction to the STEINER TREE problem — we subdivide all edges of the graph G using terminals and add the 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 Chapter 5 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 4.3 we may assume that $S \neq \emptyset$ and we may choose one fixed vertex $v_1 \in S$.

The Cut part. For an integer W we define:

1. $\mathcal{R}_W$ to be the family of solution candidates (vertex covers) of size k and weight W: $\mathcal{R}_W$ is the family of sets $X \subseteq V$ such that $S \subseteq X$, $|X| = k$, $\omega(X) = W$ and X is a vertex cover of G;

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

3. $\mathcal{C}_W$ 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. By a similar argument as in Lemma 3.2 for each $X \in \mathcal{R}_W$ there exist $2^{\mathtt{cc}(G[X])-1}$ 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.

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

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

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

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

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 consistent cuts on G_x, with fixed size, weight and interface on vertices from B_x.

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

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

- **Introduce vertex bag**:
$$\begin{aligned} A_x(i,w,s[v \to \mathbf{0}]) &= [v \notin S] A_y(i,w,s) \\ A_x(i,w,s[v \to \mathbf{1}_1]) &= A_y(i-1, w-\omega(v), s) \\ A_x(i,w,s[v \to \mathbf{1}_2]) &= [v \neq v_1] A_y(i-1, w-\omega(v), s) \end{aligned}$$
We take care of the restrictions imposed by the conditions $(S \cap V_x) \subseteq X$ and $v_1 \in X_1$.

- **Introduce edge bag**:
$$A_x(i,w,s) = [s(u) = s(v) \neq \mathbf{0} \vee (s(u) = \mathbf{0} \wedge s(v) \neq \mathbf{0}) \vee (s(u) \neq \mathbf{0} \wedge s(v) = \mathbf{0})] A_y(i,w,s)$$
Here we remove table entries not consistent with the edge uv, i.e., table entries where the endpoints are colored $\mathbf{1}_1$ and $\mathbf{1}_2$ (thus creating an inconsistent cut) or $\mathbf{0}$ and $\mathbf{0}$ (thus leaving an edge that is not covered).

- **Forget bag**:

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

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

- **Join bag**:

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

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

It is easy to see that the above recurrence leads to a dynamic programming algorithm that computes the parity of $|\mathcal{S}_W|$ for all values of W in $3^t|V|^{O(1)}$ time, since $|\mathcal{C}_W| = A_r(k, W, \emptyset)$ and $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$. Moreover, as we count the parities and not the numbers A_x themselves, all arithmetical operations can be done in constant time. Thus, the proof of Theorem 4.4 is finished. □

4.2.2 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 4.5. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves* CONSTRAINED 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 [34], 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 4.5. We use the Cut&Count technique. As the universe for Algorithm 1 we take the vertex set $U = V$. Recall that we generate a random weight function $\omega : U \to \{1, 2, \ldots, N\}$, taking $N = 2|U| = 2|V|$. By Remark 4.3 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 (dominating sets) 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 a similar argument as in Lemma 3.2 for each $X \in \mathcal{R}_W$ there exist $2^{\mathtt{cc}(G[X])-1}$ 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 $\mathtt{CountC}(\omega, W, \mathbb{T})$ that, given a nice tree decomposition $\mathbb{T}$, weight function ω and an integer W, computes $|\mathcal{C}_W|$ modulo 2.

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

$$\begin{aligned}
\mathcal{R}_x(i,w) &= \Big\{ X \subseteq V_x \;\Big|\; (S \cap V_x) \subseteq X \;\wedge\; |X| = i \;\wedge\; \omega(X) = w \\
&\qquad \wedge\; N_{G_x}[X] = V_x \setminus s^{-1}(\mathbf{0}_N) \Big\} \\
\mathcal{C}_x(i,w) &= \Big\{ (X,(X_1,X_2)) \Big|\; X \in \mathcal{R}_x(i,w) \;\wedge\; (X,(X_1,X_2)) \text{ is a consistently} \\
&\qquad \text{cut subgraph of } G_x \;\wedge\; (v_1 \in V_x \Rightarrow v_1 \in X_1) \Big\} \\
A_x(i,w,s) &= \Big| \Big\{ (X,(X_1,X_2)) \in \mathcal{C}_x(i,w) \Big|\; (s(v) = \mathbf{1}_j \Rightarrow v \in X_j) \\
&\qquad \wedge\; (s(v) = \mathbf{0}_Y \Rightarrow v \in N_{G_x}(X)) \;\wedge\; (s(v) = \mathbf{0}_N \Rightarrow v \notin N_{G_x}[X]) \Big\} \Big|
\end{aligned}$$

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

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 recursive formula for join bags.

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

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 recursive 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 $\hat{s}^{-1}(\mathbf{0})$:

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

Now note that

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

By Theorem 2.10, 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 4.5 is finished. □

4.2.3 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 4.6. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves* CONSTRAINED 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 4.3 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. By a similar argument as in Lemma 3.2 for each $(X, L) \in \mathcal{R}_W$ there exist $2^{\mathtt{cc}(G[X])-1}$ 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.

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

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

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

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

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**:
$$\begin{aligned} A_x(i, w, s[v \to \mathbf{0}_L]) &= [v \notin S]A_y(i, w - \omega((v, \mathbf{L})), s) \\ A_x(i, w, s[v \to \mathbf{0}_R]) &= [v \notin S]A_y(i, w, s) \\ A_x(i, w, s[v \to \mathbf{1}_1]) &= A_y(i - 1, w - \omega((v, \mathbf{X})), s) \\ A_x(i, w, s[v \to \mathbf{1}_2]) &= [v \neq v_1]A_y(i - 1, w - \omega((v, \mathbf{X})), s) \end{aligned}$$
We take care of restrictions imposed by conditions $(S \cap V_x) \subseteq X$ and $v_1 \in X_1$.
- **Introduce edge bag**:
$$\begin{aligned} A_x(i, w, s) &= 0 && \text{if } \{s(u), s(v)\} = \{\mathbf{1}_1, \mathbf{1}_2\} \\ A_x(i, w, s) &= 0 && \text{if } s(u) = s(v) \in \{\mathbf{0}_L, \mathbf{0}_R\} \\ A_x(i, w, s) &= A_y(i, w, s) && \text{otherwise} \end{aligned}$$
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**:
$$A_x(i, w, s) = \sum_{i_1 + i_2 = i + |s^{-1}(\{\mathbf{1_1}, \mathbf{1_2}\})|} \sum_{\substack{w_1 + w_2 = w + \omega(Z) \\ Z = s^{-1}(\{\mathbf{1_1}, \mathbf{1_2}\}) \times \{\mathbf{X}\} \cup s^{-1}(\mathbf{0}_L) \times \{\mathbf{L}\}}} A_y(i_1, w_1, s)A_z(i_2, w_2, s)$$
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 4.6 is finished. □

4.2.4 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 4.7. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves the* CONSTRAINED 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 4.1. Thus, we attach two cuts to a solution candidate: 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 4.3 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 4.1 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. By a similar argument as in Lemma 3.2 for any A,B,C,W,$(X, M) \in \mathcal{R}_W^{A,B,C}$, there exist $2^{\mathtt{cc}(G[V\setminus X])-1}$ 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 $2^{\mathtt{cc}(M,G[X])}$ cuts (X_1, X_2) that are consistent with $G[X]$ and $M \subseteq X_1$, where by $\mathtt{cc}(M, G[X])$ we denote the number of connected components of $G[X]$ which do not contain any marker from the set M. Thus for any A,B,C,W,$(X, M) \in \mathcal{R}_W^{A,B,C}$,

there are $2^{\mathtt{cc}(G[V\setminus X])-1+\mathtt{cc}(M,G[X])}$ triples $((X,M),(X_1,X_2),(Y_1,Y_2)) \in \mathcal{C}_W^{A,B,C}$. Hence by Lemma 4.1 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.

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 $s \in \{\mathbf{0}_1, \mathbf{0}_2, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$ define

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

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

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

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**:
$$A_x(a,b,c,w,s[v \to \mathbf{0}_1]) = A_y(a,b,c,w,s)$$
$$A_x(a,b,c,w,s[v \to \mathbf{0}_2]) = [v \neq v_1]A_y(a,b,c,w,s)$$
$$A_x(a,b,c,w,s[v \to \mathbf{1}_j]) = [v \notin S]A_y(a-1,b,c,w-\omega((v,\mathbf{F})),s)$$
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**:

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

  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**:

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

  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 4.7. □

4.3 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 a path of length $|V| + 1$ from t to s to the graph 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.

4.3.1 The undirected case

Theorem 4.8. *There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves* PARTIAL CYCLE 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 $k \leq \ell$.
The Cut part. For an integer W we define:

1. $\mathcal{R}_W$ to be the family of pairs (X, M), where $M \subseteq X \subseteq E$, $|X| = \ell$, $|M| = k$, $\omega(X \times \{\mathbf{X}\} \cup M \times \{\mathbf{M}\}) = W$ 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 a set of markers 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 $\mathcal{S}_W \neq \emptyset$ 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 connected components of $G[X]$ that are neither isolated vertices nor 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 $2^{\mathtt{cc}(M,X)}$ 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 function ω and and an integer W, computes $|\mathcal{C}_W|$ modulo 2.

Let $\Sigma = \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2, \mathbf{2}\}$. For every bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \leq i, b \leq |V|$, $0 \leq w \leq 2N|V|$ and $s \in \Sigma^{B_x}$ define

$$\begin{aligned}
\mathcal{R}_x(i,b,w) = \Big\{ & (X,M) \;\Big|\; M \subseteq X \subseteq E_x \;\wedge\; |M| = i \;\wedge\; |X| = b \\
& \wedge\; \omega(X \times \{\mathbf{X}\} \cup M \times \{\mathbf{M}\}) = w \;\wedge\; (\forall_{v \in V(X) \setminus B_x} \deg_{G[X]}(v) = 2) \\
& \wedge\; (\forall_{v \in B_x} \deg_{G[X]}(v) \leq 2) \Big\} \\
\mathcal{C}_x(i,b,w) = \Big\{ & ((X,M),(X_1,X_2)) \big|\; (X,M) \in \mathcal{R}_x(i,b,w) \;\wedge\; V(M) \subseteq X_1 \\
& \wedge\; (X_1,X_2) \text{ is a consistent cut of the graph } (V(X),X) \Big\} \\
A_x(i,b,w,s) = \Big| \Big\{ & ((X,M),(X_1,X_2)) \in \mathcal{C}_x(i,b,w) \big|\; (s(v) = \mathbf{0} \Rightarrow \deg_{G[X]}(v) = 0) \\
& \wedge\; (s(v) = \mathbf{1}_j \Rightarrow (\deg_{G[X]}(v) = 1 \wedge v \in X_j)) \\
& \wedge\; (s(v) = \mathbf{2} \Rightarrow \deg_{G[X]}(v) = 2) \Big\} \Big|
\end{aligned}$$

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 maximum 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 inapplicable 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 extending 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 $\mathrm{subs} : \Sigma \to 2^{\Sigma}$.

$\alpha \in \Sigma$	$\mathbf{0}$	$\mathbf{1}_1$	$\mathbf{1}_2$	$\mathbf{2}$
$\mathrm{subs}(\alpha)$	$\emptyset$	$\{\mathbf{0}\}$	$\{\mathbf{0}\}$	$\{\mathbf{1}_1, \mathbf{1}_2\}$

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

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

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

In the above formula by s' we denote $s[u \to \alpha_u, v \to \alpha_v]$. 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:

$$\begin{aligned} s_1(v) = \mathbf{0} \ \wedge \ s(v) &= s_2(v) \\ s_2(v) = \mathbf{0} \ \wedge \ s(v) &= s_1(v) \\ s_1(v) = s_2(v) = \mathbf{1}_j \ \wedge \ s(v) &= \mathbf{2} \end{aligned}$$

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

$$\rho(\mathbf{0}) = 0 \qquad \rho(\mathbf{1}_1) = 1 \qquad \rho(\mathbf{1}_2) = 1 \qquad \rho(\mathbf{2}) = 2$$

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 for any s_1, s_2 such that $\phi(s_1) + \phi(s_2) = \phi(s)$ 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 2.15, 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 4.8 is finished. □

If we restrict the class of graphs that we consider to graphs of maximum degree three which we call *subcubic* and assume that a **path** decomposition is given we may obtain a better time complexity for the general PARTIAL CYCLE COVER problem. As discussed in Section 1.2.3 and as a consequence obtain an $O(1.201^n)$ time algorithm for the HAMILTONIAN PATH problem in subcubic graphs.

Theorem 4.9. *There exists a Monte-Carlo algorithm that given a path decomposition of width p of a subcubic graph solves* PARTIAL CYCLE COVER *in $3^p|V|^{O(1)}$ time. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

Proof. The key difference from the proof Theorem 4.8 is that because the graph is subcubic we may reduce the number of colours in the set Σ to $\Sigma = \{\mathbf{even}, \mathbf{1}_1, \mathbf{1}_2\}$, which means that we merge colours $\mathbf{0}, \mathbf{2}$ into a single colour **even**. Observe that we can do this since in the forget node we ensure that a vertex is of even degree and in subcubic graph if a vertex

is of even degree then it is of degree zero or two. Since there are no join nodes in a path decomposition we do not have problems with joining colour vectors from two bags efficiently.

The rest of the proof of Theorem 4.8 remains. □

4.3.2 The directed case

Theorem 4.10. *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 function ω and an integer W, computes $|\mathcal{C}_W|$ modulo 2.

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,

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

integers $0 \leq i, b \leq |V|$, $0 \leq w \leq 2N|V|$ and $s \in \Sigma^{B_x}$ define

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

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

The algorithm computes $A_x(i,b,w,s)$ for all bags $x \in T$ in a bottom-up fashion for all reasonable values of i, b, w and s. 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 $\mathrm{insubs}, \mathrm{outsubs} : \Sigma \to 2^{\Sigma}$.

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

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

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

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

In the above formula by s' we denote $s[u \to \alpha_u, v \to \alpha_v]$. 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 $\mathrm{outsubs}$ we need to have $\alpha_u \in \mathrm{outsubs}(s(u))$ and $\alpha_v \in \mathrm{insubs}(s(v))$. 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 either both indegree and outdegree zero or both indegree and outdegree one.

- **Join bag**: We have two children y and z. Figure 4.1 shows how two individual states of a vertex in B_y and B_z combine to a state of this vertex in B_x. XX indicates that two states do not combine. The correctness of the table is easy to check.

	$\mathbf{00}$	$\mathbf{01}_1$	$\mathbf{01}_2$	$\mathbf{10}_2$	$\mathbf{10}_1$	$\mathbf{11}$
$\mathbf{00}$	$\mathbf{00}$	$\mathbf{01}_1$	$\mathbf{01}_2$	$\mathbf{10}_2$	$\mathbf{10}_1$	$\mathbf{11}$
$\mathbf{01}_1$	$\mathbf{01}_1$	XX	XX	XX	$\mathbf{11}$	XX
$\mathbf{01}_2$	$\mathbf{01}_2$	XX	XX	$\mathbf{11}$	XX	XX
$\mathbf{10}_2$	$\mathbf{10}_2$	XX	$\mathbf{11}$	XX	XX	XX
$\mathbf{10}_1$	$\mathbf{10}_1$	$\mathbf{11}$	XX	XX	XX	XX
$\mathbf{11}$	$\mathbf{11}$	XX	XX	XX	XX	XX

Figure 4.1: The join table of DIRECTED PARTIAL CYCLE COVER where it is indicated which states combine to which other states.

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 4.1. We can now write the

recurrence formula for join bags.

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

A straightforward computation of the above formula leads to $36^t|V|^{O(1)}$ time complexity. We now show how to use the Generalized Subset Convolution to obtain a better time bound.

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

$$\phi(\mathbf{00}) = 0 \quad \phi(\mathbf{01}_1) = 1 \quad \phi(\mathbf{01}_2) = 2 \quad \phi(\mathbf{10}_2) = 3 \quad \phi(\mathbf{10}_1) = 4 \quad \phi(\mathbf{11}) = 5$$
$$\rho(\mathbf{00}) = 0 \quad \rho(\mathbf{01}_1) = 1 \quad \rho(\mathbf{01}_2) = 1 \quad \rho(\mathbf{10}_2) = 1 \quad \rho(\mathbf{10}_1) = 1 \quad \rho(\mathbf{11}) = 2$$

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 $\mathbf{1}$'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 according to Table 4.1 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 4.1 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 2.12, the values $A_x(i,b,w,s)$ for a join bag x can be computed in time $6^t|V|^{O(1)}$.

It is easy to see that the above recurrence leads to a dynamic programming algorithm that computes the parity of $|\mathcal{S}_W|$ for all values of W in $6^t|V|^{O(1)}$ time, since $|\mathcal{C}_W| = A_r(k,\ell,W,\emptyset)$ and $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$. Moreover, as we count the parities and not the numbers A_x themselves, all arithmetical operations can be done in constant time. Thus, the proof of Theorem 4.10 is finished. □

4.4 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.

4.4.1 Exact k-Leaf Spanning Tree

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 4.11. *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 nodes (i.e., vertices of degree at least 2). Using similar arguments as in Remark 4.3, we may assume that we are given a vertex $v_1 \in V$ that is required to be an internal node 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 a 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. $\mathcal{R}^k_W$ 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. $\mathcal{S}^k_W$ to be the family of sets $X \in \mathcal{R}^k_W$, such that $G[X]$ is connected;
3. $\mathcal{C}^k_W$ to be the family of pairs $(X, (X_1, X_2))$, where $X \in \mathcal{R}^k_W$ and (X_1, X_2) is a consistent cut of $G[X]$ with $v_1 \in X_1$.

The condition that for $X \in \mathcal{S}^k_W$ the graph $G[X]$ is connected, together with $|X| = |V| - 1$ gives us that each $X \in \mathcal{S}^k_W$ induces a spanning tree. Thus, $\mathcal{S}^k_W$ is indeed a family of spanning trees of exactly k leaves with root v_1 of degree at least 2.

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 prove formally that for any W and k we have $|\mathcal{S}^k_W| \equiv |\mathcal{C}^k_W|$. By a similar argument as in Lemma 3.2 for each $X \in \mathcal{R}^k_W$ there exist $2^{\mathsf{cc}(G[X])-1}$ consistent cuts of $G[X]$, and the claim follows.

To finish the proof we need to show how to compute $|\mathcal{C}^k_W|$ 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 $\overline{\mathcal{C}}^\ell_W$ 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 consistent 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) induces a consistent cut $(Y_1 \cup (N_{G[X]}(Y_1) \cap R), Y_2 \cup (N_{G[X]}(Y_2) \cap R))$ 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 $2^{\mathsf{cc}(G[X])-1}$ consistent cuts of $G[X]$, for fixed X and R we have $(X, R, C) \in \overline{\mathcal{C}}_W^\ell$ for exactly $2^{\mathsf{cc}(G[X])-1}$ 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 $\overline{\texttt{CountC}}(\omega, W, \ell, \mathbb{T})$ that, given a nice tree decomposition $\mathbb{T}$, weight function ω and integers W and ℓ computes $|\overline{\mathcal{C}}^{\ell}_{W}|$ modulo 2. Now we can use dynamic programming on the tree decomposition.

Recall that in the definition of $\overline{\mathcal{C}}^{\ell}_{W}$ 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 a side of the cut for vertices in R.

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 $s \in \{\mathbf{1}_1, \mathbf{1}_2, \mathbf{0}_0, \mathbf{0}_1\}^{B_x}$ define

$$\begin{aligned}
\mathcal{R}_x(\ell, w, m, d) = \Big\{ (X, R) \;\Big|\; & X \subseteq E_x \ \wedge\ R \subseteq V_x \ \wedge\ |X| = m \ \wedge\ |R| = \ell \ \wedge\ \omega(X) = w \\
& \wedge\ \deg_{G[X]}(v_1) = d \ \wedge\ (v \in R \setminus B_x \Rightarrow \deg_{G[X]}(v) = 1) \\
& \wedge\ (v \in R \cap B_x \Rightarrow \deg_{G[X]}(v) \leq 1) \Big\} \\
\mathcal{C}_x(\ell, w, m, d) = \Big\{ (X, R, (Y_1, Y_2)) \Big|\ & (X, R) \in \mathcal{R}_x(\ell, w, m, d) \ \wedge\ (v_1 \in V_x \Rightarrow v_1 \in Y_1) \\
& \wedge\ (Y_1, Y_2) \text{ is a consistent cut of } G(V_x \setminus R, X) \Big\} \\
A_x(\ell, w, m, d, s) = \Big| \Big\{ & (X, R, (Y_1, Y_2)) \in \mathcal{C}_x(\ell, w, m, d) \Big| \\
& (s(v) = \mathbf{1}_j \Rightarrow v \in Y_j) \ \wedge\ (s(v) = \mathbf{0}_j \Rightarrow (v \in R \wedge \deg_{G[X]}(v) = j)) \Big\} \Big|
\end{aligned}$$

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 $x \in \mathbb{T}$ 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**:
$$\begin{aligned}
A_x(\ell, w, m, d, s[v \to \mathbf{0}_0]) &= [v \neq v_1] A_y(\ell - 1, w, m, d, s) \\
A_x(\ell, w, m, d, s[v \to \mathbf{0}_1]) &= 0 \\
A_x(\ell, w, m, d, s[v \to \mathbf{1}_1]) &= A_y(\ell, w, m, d, s) \\
A_x(\ell, w, m, d, s[v \to \mathbf{1}_2]) &= [v \neq v_1] A_y(\ell, w, m, d, s)
\end{aligned}$$
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**:

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

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 recursive formula for the join bags.

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

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 recursive 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 2.10, 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 $|\overline{\mathcal{C}}^{\ell}_W|$ for all values of W and ℓ in $4^t|V|^{O(1)}$ time, since $|\overline{\mathcal{C}}^{\ell}_W| = \sum_{d\geq 2} A_r(\ell, W, |V|-1, d, \emptyset)$. 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 $|\overline{\mathcal{C}}^{\ell}_W|$ for $0 \leq \ell \leq |V|-1$ we can compute all values of $|\mathcal{C}^k_W|$ and $|\mathcal{S}^k_W|$ modulo 2. Thus, the proof of Theorem 4.11 is finished. □

4.4.2 Exact k-Leaf Outbranching

An arborescence is a directed graph in which, for a vertex r called the root and any other vertex v, there is exactly one directed path from r to v. In other words, it is a directed, rooted tree in which all edges point away from the root. A vertex of outdegree 0 in an arborescence is called a leaf. A spanning arborescence of a directed graph is called an outbranching (see Fig. 4.2 for an example).

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 an outbranching in D rooted at r with exactly k leaves?

Theorem 4.12. *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 $1/2$.*

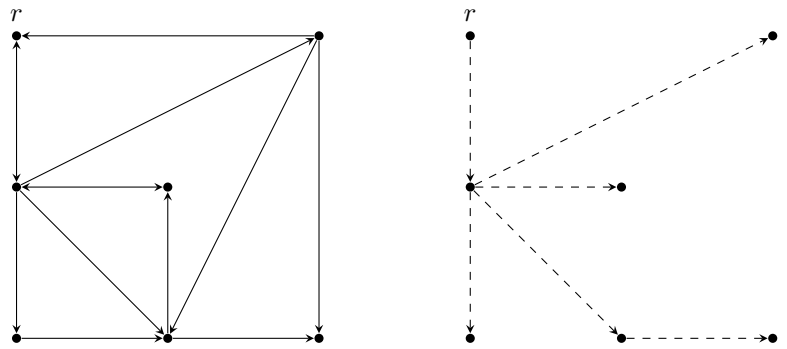


Figure 4.2: A graph and one of its outbranchings rooted at r. The presented outbranching has exactly 4 leaves.

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 a 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. $\mathcal{R}^k_W$ to be the family of sets $X \subseteq A$, such that $\omega(X) = W$, $\mathrm{indeg}_{G[X]}(v_1) = 0$ and $\mathrm{indeg}_{G[X]}(v) = 1$ if $v \neq v_1$, and $G[X]$ contains exactly k vertices of outdegree zero. Note that each weakly connected component of $G[X]$ for $X \in \mathcal{R}^k_W$ is either a directed unicyclic graph (a graph with exactly one cycle) or an arborescence rooted at v_1.

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

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

The condition that for $X \in \mathcal{S}^k_W$ the graph $G[X]$ is weakly connected, together with the condition on the indegrees of vertices gives us that each $X \in \mathcal{S}^k_W$ is of size $|V| - 1$ and induces a spanning arborescence of G rooted at v_1.

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 $|\mathcal{S}^k_W| \equiv |\mathcal{C}^k_W|$. By a similar argument as in Lemma 3.2 for each $X \in \mathcal{R}^k_W$ there exist $2^{\mathtt{cc}(G[X])-1}$ consistent cuts of $G[X]$, and the claim follows (here $\mathtt{cc}(G[X])$ denotes the number of weakly connected components of $G[X]$).

To finish the proof we need to show now to compute $|\mathcal{C}^k_W|$ 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 $\overline{\mathcal{C}}^\ell_W$ 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 consistent 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 $\overline{\texttt{CountC}}(\omega, W, \ell, \mathbb{T})$ that, given a nice tree decomposition $\mathbb{T}$, weight function 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.

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

$$\begin{aligned}
\mathcal{R}_x(\ell, w) = \Big\{ (X, R) \;\Big|\; & X \subseteq E_x \;\wedge\; R \subseteq V_x \;\wedge\; |R| = \ell \;\wedge\; \omega(X) = w \\
& \wedge\; (v \in V_x \setminus B_x \Rightarrow \operatorname{indeg}_{G[X]}(v) = [v \neq v_1]) \;\wedge\; (v \in B_x \Rightarrow \operatorname{indeg}_{G[X]}(v) \leq [v \neq v_1]) \\
& \wedge\; (v \in R \Rightarrow \operatorname{outdeg}_{G[X]}(v) = 0) \Big\} \\
\mathcal{C}_x(\ell, w) = \Big\{ (X, R, (Y_1, Y_2)) \Big|\; & (X, R) \in \mathcal{R}_x(\ell, w) \;\wedge\; v_1 \notin Y_2 \\
& \wedge\; (Y_1, Y_2) \text{ is a consistent cut of } G(V_x \setminus R, X) \Big\} \\
A_x(\ell, w, s, s_{\text{in}}) = \Big| \Big\{ (X, R, (Y_1, Y_2)) \in \mathcal{C}_x(\ell, w) \Big|\; & (s(v) = \mathbf{1}_j \Rightarrow v \in Y_j) \\
& \wedge\; (s(v) = \mathbf{0} \Rightarrow v \in R) \;\wedge\; (\forall_{v \in B_x} s_{\text{in}}(v) = \operatorname{indeg}_{G[X]}(v)) \Big\} \Big|
\end{aligned}$$

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 $x \in \mathbb{T}$ 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**:

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

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

- **Introduce edge bag**:

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

  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**:

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

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

  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 recursive 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 2.10, 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 $0 \leq \ell \leq |V| - 1$ we can compute all values of $|\mathcal{C}_W^k|$ and $|\mathcal{S}_W^k|$ modulo 2. Thus, the proof of Theorem 4.12 is finished. □

4.5 Maximum Full Degree Spanning Tree

In this section 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 4.13. *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 $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$. By a similar argument as in Lemma 3.2 for each $X \in \mathcal{R}_W$ there exist $2^{\mathtt{cc}(G[X])-1}$ 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.

For a bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \leq i \leq |V|$, $0 \leq b < |V|$,

$0 \leq w \leq N(|V|-1)$, $s_{\text{cut}} \in \{\mathbf{1},\mathbf{2}\}^{B_x}$ and $s_{\text{deg}} \in \{\mathbf{0},\mathbf{1}\}^{B_x}$ define

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

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

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

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

The algorithm computes $A_x(i,b,w,s_{\text{cut}},s_{\text{deg}})$ 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 $A_x(i,b,w,s_{\text{cut}},s_{\text{deg}})$ 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**:
$$\begin{aligned} A_x(i,b,w,s_{\text{cut}}[v \to \mathbf{1}],s_{\text{deg}}[v \to \mathbf{0}]) &= A_y(i,b,w,s_{\text{cut}},s_{\text{deg}}) \\ A_x(i,b,w,s_{\text{cut}}[v \to \mathbf{2}],s_{\text{deg}}[v \to \mathbf{0}]) &= [v \neq v_1]A_y(i,b,w,s_{\text{cut}},s_{\text{deg}}) \\ A_x(i,b,w,s_{\text{cut}}[v \to \alpha],s_{\text{deg}}[v \to \mathbf{1}]) &= 0 \end{aligned}$$
We make sure that v_1 belongs to X_1.

- **Introduce edge bag**:
$$\begin{aligned} A_x(i,b,w,s_{\text{cut}},s_{\text{deg}}) = {} & [s_{\text{cut}}(u) = s_{\text{cut}}(v)]A_y(i,b-1,w-\omega(uv),s_{\text{cut}},s_{\text{deg}}) \\ & + \sum_{\alpha_u,\alpha_v \in \{\mathbf{0},\mathbf{1}\}} A_y(i,b,w,s_{\text{cut}},s_{\text{deg}}[u \to \alpha_u, v \to \alpha_v]) \\ & \text{if } s_{\text{deg}}(u) = s_{\text{deg}}(v) = \mathbf{1} \\ A_x(i,b,w,s_{\text{cut}},s_{\text{deg}}) = {} & [s_{\text{cut}}(u) = s_{\text{cut}}(v)]A_y(i,b-1,w-\omega(uv),s_{\text{cut}},s_{\text{deg}}) \\ & \text{otherwise} \end{aligned}$$
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**:

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

  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 $s_{\deg}(v) = \mathbf{1}$ in x if and only if $s_{\deg}(v) = \mathbf{1}$ 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 $s_{\deg}^{-1}(\mathbf{1}) \subseteq B_x$. Let us define

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

  Consequently we have

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

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 2.10. 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 4.13. □

4.6 Graph Metric Travelling Salesman Problem

The GRAPH METRIC TRAVELLING SALESMAN PROBLEM is a special case of the TRAVELLING SALESMAN PROBLEM where the metric is induced by shortest paths in an undirected graph. Recently this problem received considerable attention [42, 62] due to two independently obtained approximation algorithms that break the $3/2$-approximation ratio of the classical algorithm of Christofides.

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 4.14. *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 (in particular the value of k is polynomially bounded in $|V|$). Furthermore note that for a **yes**-instance there exists a closed walk which uses each edge at most twice, since a solution is an Eulerian subgraph and if an edge is used at least three times then we may use it two times less while still obtaining 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. $\mathcal{R}^i_W$ 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. $\mathcal{S}^i_W$ to be the set of solutions, that is solution candidates $\phi \in \mathcal{R}^i_W$ such that the graph $G[\phi^{-1}(\{1,2\})]$ is connected;

3. $\mathcal{C}^i_W$ to be the family of pairs $(\phi,(X_1,X_2))$, where $\phi \in \mathcal{R}^i_W$, $v_1 \in X_1$, 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 $\mathcal{S}^i_W \neq \emptyset$.
The Count part. To use Algorithm 1 we need to formally prove that for any i and W we have $|\mathcal{S}^i_W| \equiv |\mathcal{C}^i_W|$. By a similar argument as in Lemma 3.2 for each $\phi \in \mathcal{R}^i_W$ 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 $|\mathcal{C}^i_W|$ modulo 2 in time $4^t n^{O(1)}$ using dynamic programming.

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 $s_{\deg} \in \{\mathbf{0},\mathbf{1}\}^{B_x}$ define

$$\mathcal{R}_x(i,w) = \Big\{\phi \in \{0,1,2\}^{E_x} \;\Big|\; \sum_{e\in E_x}\phi(e) = i \;\wedge \sum_{e\in E_x,\phi(e)>0}\omega((e,\phi(e))) = w \;\wedge\; \forall_{v\in V_x\setminus B_x}|\{uv \in E_x : \phi(uv) = 1\}| \mod 2 = 0\Big\}$$

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

$$A_x(i,w,s_{\mathrm{cut}},s_{\deg}) = \Big|\Big\{(\phi,(X_1,X_2)) \in \mathcal{C}_x(i,w)\Big|(s_{\mathrm{cut}}(v) = j \Rightarrow v \in X_j) \;\wedge\; \forall_{v\in B_x} s_{\deg}(v) \equiv |\{uv \in E_x : \phi(uv) = 1\}|\Big\}\Big|$$

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 in the multigraph induced by ϕ. Hence $A_x(i, w, s_{\text{cut}}, s_{\text{deg}})$ is the number of pairs from $\mathcal{C}_x(i, w)$ with a fixed interface on vertices from B_x.

The algorithm computes $A_x(i, w, s_{\text{cut}}, s_{\text{deg}})$ 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 $A_x(i, w, s_{\text{cut}}, s_{\text{deg}})$ 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**:

$$A_x(i, w, s_{\text{cut}}[v \to \mathbf{1}], s_{\text{deg}}[v \to \mathbf{0}]) = A_y(i, w, s_{\text{cut}}, s_{\text{deg}})$$
$$A_x(i, w, s_{\text{cut}}[v \to \mathbf{2}], s_{\text{deg}}[v \to \mathbf{0}]) = [v \neq v_1]A_y(i, w, s_{\text{cut}}, s_{\text{deg}})$$

  We make sure that v_1 belongs to X_1.

- **Introduce edge bag**:

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

  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 degree 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 $s_{\deg}^{-1}(\mathbf{1}) \subseteq B_x$. Let us define

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

Consequently we have

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

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 2.15. 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 4.14. □

Chapter 5

Solution size parametrization

In this chapter we show how the Cut&Count technique may be used to obtain FPT algorithms when parameterized by the solution size. We study vertex deletion problems in which the remaining graph has to be of constant treewidth. These problems are FEEDBACK VERTEX SET, CONNECTED VERTEX COVER and CONNECTED FEEDBACK VERTEX SET and improve the best known FPT algorithms for all three problems. The main idea behind the new results is the combination of the *iterative compression* technique, developed by Reed et al. [67], and the Cut&Count technique.

In the last section of this chapter we give an evidence that it is hard to improve the algorithm for CONNECTED VERTEX COVER, that is we show that unless Strong Exponential Time Hypothesis fails there does not exist an algorithm with $(2-\varepsilon)^k|V|^{O(1)}$ running time which computes the parity of the number of connected vertex covers of size k in a given graph $G = (V, E)$. To the best of our knowledge this is the first example of a problem parameterized by the solution size where there exists some evidence showing that it might be optimal.

5.1 Feedback Vertex Set

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 [18].

Theorem 5.1 (Theorem 1.5, 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, call it B, of size $k + 1$, we need to construct a feedback vertex set of size at most k or determine that none such exists.

As B is a feedback vertex set, the graph induced by the rest of the vertices is a forest. Thus we can construct a tree decomposition of the graph G_{i+1} of width at most $k + 2$ by creating a tree decomposition of the forest of width 1 and adding the whole set B to each bag. 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 4.1, 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 4.2, uses exponential space. However, in each step when computing $A_x(a, b, c, w, s)$ the algorithm refers only to values $A_y(a', b', c', w', s')$, where $s' = s$ on the intersection of the domains of s and s'. In our case the intersection of every two bags of the tree decomposition contains B. Therefore we can reorder the computation in the following manner: for every evaluation $\overline{s} : B \to \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}$ we fix it as the „core" evaluation for every bag in the decomposition and run the algorithm to compute all the values $A_x(a, b, c, w, s)$, where $s|_B = \overline{s}$. Such a computation takes polynomial time and space. As there are 3^{k+1} such possible evaluations $\overline{s}$, the algorithm runs in $3^k n^{O(1)}$ time and in polynomial space. We make n independent runs of the algorithm in order to assure that the probability of a false negative is at most $\frac{1}{2^n}$.

Once we have done this, we already tested with high probability whether the desired feedback vertex set exists or not. If the answer is negative, we answer NO. Otherwise we need to explicitly construct the set A_{i+1} in order to use it in the next step of the iterative compression. We make use of the algorithm for CONSTRAINED FEEDBACK VERTEX SET, given by Theorem 4.2. The algorithm considers the vertices of G_{i+1} one by one, building a set K which at the end will be the set A_{i+1} we want to construct. We begin with $K = \emptyset$ and preserve an invariant that at each step there is a feedback vertex set of size at most k containing the set K. When considering the vertex v, we test in $3^k n^{O(1)}$ time whether the graph admits a constrained feedback vertex set of size at most k with $S = K \cup \{v\}$, making n independent runs of the algorithm given by Theorem 4.2 in order to reduce the probability of a false negative to at most $\frac{1}{2^n}$. If the answer is positive, we can safely add v to K as we know that there is a feedback vertex set of size at most k containing $K \cup \{v\}$ (recall our algorithms do not return false positives). Otherwise we simply proceed to the next vertex. The computation terminates when K is already a feedback vertex set or when we have exhausted all vertices. Observe that if G_{i+1} admits a feedback vertex set of size at most k, this construction will terminate building a feedback vertex set A_{i+1} of size at most k unless there was an error in at least one of the tests. If we exhaust all vertices, we answer NO, as an error has occurred. Note that in each run of the algorithm for CONSTRAINED FEEDBACK VERTEX SET we can reorder the computation in the same way as in the previous paragraph

to reduce space usage to polynomial.

Observe that the described algorithm at most $n^2 + n$ times makes n independent runs of the algorithm from Theorem 4.2 as a subroutine: at most $n + 1$ times in each of the n steps of the iterative compression. Each of these groups of runs has the probability of a false negative bounded by $\frac{1}{2^n}$, thus the probability of a false negative is bounded by $\frac{n^2+n}{2^n}$, which is lower than $\frac{1}{2}$ for n large enough. □

5.2 Connected Vertex Cover

Now we proceed to the algorithm for CONNECTED VERTEX COVER. The previously best FPT algorithm is due to Binkele-Raible [6], 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 5.2. *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 the CONNECTED VERTEX COVER problem is *contraction–*closed. This means that if a graph H admits a connected vertex cover A of size at most k, then H' obtained from H by contracting an edge of H (and reducing possible 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 previous paragraph ensures that we can proceed as in the proof of Theorem 5.1, namely construct connected vertex covers for $G_1, G_2, \ldots, G_n$ consecutively, and the only thing we have to show is how to construct a connected vertex cover of size k in G_{i+1} given a connected vertex cover A_i of size k in G_i, or determine that none exists.

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

Now we are going to test whether G_{i+1} admits a connected vertex cover of size at most k. We could apply the algorithm from Theorem 4.4. As in the proof of Theorem 5.1, also this dynamic programming algorithm during the computation of $A_x(i, w, s)$ refers only to values $A_y(i', w', s')$ for s' such that $s = s'$ on the intersection of domains of s and s'. Therefore, similarly as before, we would iterate through all possible evaluations

$\overline{s} : B \to \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}$, each time computing all the values $A_x(i, W, s)$ such that $s|_B = \overline{s}$ in polynomial time, thus using only polynomial space in the whole algorithm. Unfortunately, the algorithm given by Theorem 4.4 runs in $3^k n^{O(1)}$ time.

We can, however, reduce the complexity by bounding the number of reasonable evaluations $\overline{s} : B \to \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}$ by $3^3 \cdot 2^{k-1}$. The set B induces in G_{i+1} a graph 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 $\overline{s}$ in the following manner. For the root r and the two additional vertices we choose any value from $\{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}$ for $\overline{s}$, 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 evaluations, depending on the evaluation of its parent u:

- if $\overline{s}(u) = \mathbf{0}$, 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 $\overline{s}(u) = \mathbf{1}_j$, the two possible options are $\mathbf{0}$ and $\mathbf{1}_j$, as otherwise the evaluation $\overline{s}$ 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 $\overline{s}$, which yields an algorithm with running time $2^k n^{O(1)}$, using polynomial space. As previously, we make n independent runs of the algorithm in order to reduce the probability of a false negative to at most $\frac{1}{2^n}$.

Once we have tested whether G_{i+1} admits a connected vertex cover of size at most k, we can construct it explicitly similarly as in the proof of Theorem 5.1 using the algorithm for CONSTRAINED CONNECTED VERTEX COVER. We consider vertices one by one, each time determining whether the vertex can be inserted into the connected vertex cover we are constructing by running the algorithm from Theorem 4.4 n times. Observe that all these runs can be done in $2^k n^{O(1)}$ time and polynomial space complexity using the same technique as in the testing. Thus we succeed in constructing A_{i+1} unless at least one of the tests returns a false negative.

The algorithm makes at most $n^2 + n$ groups of n independent runs of algorithm from Theorem 4.4. Therefore the probability of a false negative is bounded by $\frac{n^2+n}{2^n}$ which is less than $\frac{1}{2}$ for n large enough. □

5.3 Connected Feedback Vertex Set

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. [60], and runs in $46.2^k n^{O(1)}$ time complexity.

Theorem 5.3. *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 5.2, 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 5.2. Namely we 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. 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 a set 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 4.7 would yield an algorithm with running time $4^k n^{O(1)}$. Once again this algorithm, 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 5.1 and 5.2 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 $4^3 3^{k-1}$. The graph $G[B]$ consists 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 5.1 and 5.2. 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 4.7. 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, by the union bound 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 4.7. □

5.4 CVC as hard as CNF-SAT

In this section we present a chain of reductions showing that it is not possible to determine the parity of the number of connected vertex covers of size k in $(2-\varepsilon)^k |V|^{O(1)}$ time for any $\varepsilon > 0$ unless the Strong Exponential Time Hypothesis fails. To the best of our knowledge this is the first evidence that an existing algorithm for a problem parameterized by the solution size is the fastest possible.

5.4.1 From CNF-Sat to Hitting Set

Let us begin by recalling the following result of Calabro et al [16].

k-UNIQUE-CNF-SAT
Input: A CNF formula Φ consisting of m clauses of size at most k on n variables having at most one satisfying assignment.
Question: Is there a satisfying assignment for Φ?

Theorem 5.4 ([16]). *If there exists $\varepsilon > 0$ such that for any positive integer k there exists an algorithm for the k-UNIQUE-CNF-SAT problem running in $(2-\varepsilon)^n n^{O(1)}$ time then the Strong Exponential Time Hypothesis fails.*

We will also use the Sparsification Lemma proved by Impagliazzo et al [49].

Lemma 5.5 (Sparsification Lemma, Theorem 1 and Corollary 1 of [49]). *For every $\varepsilon > 0$ and positive integer k, there is a constant C so that any k-CNF-SAT formula Φ with n variables can be expressed as $\Phi = \bigvee_{i=1}^{t} \Psi_i$, where $t \leq 2^{\varepsilon n}$ and each Ψ_i is a k-CNF-SAT formula with at most Cn clauses and at most n variables.*

Moreover, this disjunction can be computed by an algorithm running in time $2^{\varepsilon n}n^{O(1)}$ *and the number of satisfying assignments for each* Ψ_i *is not greater than the number of satisfying assignments of* Φ.

We define the following variant of the satisfiability problem.

(k_1, k_2)-$\oplus$CNF-SAT
Input: A CNF formula Φ consisting of m clauses of size at most k_1 on n variables, where $m \leq k_2 n$.
Question: Is the number of satisfying assignments for Φ odd?

Using Sparsification Lemma we prove the following theorem.

Theorem 5.6. *If there exists* $\varepsilon > 0$ *such that for any positive integers* k_1, k_2 *there exists an algorithm for the* (k_1, k_2)-$\oplus$CNF-SAT *problem running in* $(2-\varepsilon)^n n^{O(1)}$ *time then there exists* $\varepsilon_2 > 0$ *such that for any positive integer* k *there exists an algorithm for the* k-UNIQUE-CNF-SAT *problem running in* $(2-\varepsilon_2)^n n^{O(1)}$ *time.*

Proof. Let k be any positive integer and let Φ be a formula of k-UNIQUE-CNF-SAT. Set ε' so that $(2-\varepsilon)2^{\varepsilon'} = 2 - \varepsilon_2$ for some $\varepsilon_2 > 0$. Now use the Sparsification Lemma on the formula Φ with ε'. As a result in time $2^{\varepsilon' n}n^{O(1)}$ we obtain a set $\{\Psi_i : 1 \leq i \leq t\}$ of at most $2^{\varepsilon' n}$ k-CNF-SAT formulas with at most Cn clauses each. Moreover by the property of the Sparsification Lemma that ensures that the number of satisfying assignments does not increase we infer that each Ψ_i is satisfiable iff it has an odd number of solutions. Therefore for each Ψ_i we can use the $(2-\varepsilon)^n n^{O(1)}$ time algorithm.

The total running time is $2^{\varepsilon' n}n^{O(1)} + 2^{\varepsilon' n}(2-\varepsilon)^n n^{O(1)}$ which is upper bounded by $(2-\varepsilon_2)^n n^{O(1)}$ and the theorem follows. □

Now we present a reduction from (k_1, k_2)-$\oplus$CNF-SAT to (p_1, p_2)-$\oplus$HITTING SET.

(p_1, p_2)-$\oplus$HITTING SET
Input: A set system $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = m$,$|U| = n$, for every $S \in \mathcal{F}$, $|S| \leq p_1$ and $m \leq p_2 n$.
Question: Is the number of $X \subseteq U$, with $X \cap S \neq \emptyset$ for each $S \in \mathcal{F}$, odd?

Theorem 5.7. *If there exists* $\varepsilon > 0$ *such that for any positive integers* p_1, p_2 *there exists an algorithm for the* (p_1, p_2)-$\oplus$HITTING SET *problem running in* $(2-\varepsilon)^n n^{O(1)}$ *time then there exists* $\varepsilon_2 > 0$ *such that for any positive integers* k_1, k_2 *there exists an algorithm for the* (k_1, k_2)-$\oplus$CNF-SAT *problem running in* $(2-\varepsilon_2)^n n^{O(1)}$ *time.*

Proof. Consider some positive integers k_1, k_2. Let us define the following procedure transforming an instance of (k_1, k_2)-$\oplus$CNF-SAT (that is a formula Φ) into a set system.

Assume $\Phi = C_1 \wedge \ldots \wedge C_m$ over n variables $v_1, \ldots, v_n$ where each C_i is a clause of size at most k_1. Let p be a positive odd integer such that $(2-\varepsilon)^{1+(2\lceil \log p \rceil + 1)/p} = (2-\varepsilon_2)$ for some $\varepsilon_2 > 0$. Note that p is a constant. W.l.o.g. we assume that p divides n since otherwise we can add dummy variables together with clauses ensuring that the added variables are false in any satisfying assignment. Let us denote $g = n/p$. Create the set system $\mathcal{F} \subseteq 2^U$ as follows.

1. Let $p' = p + 2\lceil \log p \rceil$ be an odd integer (recall that p is odd) and let $U = \{u_1, \ldots, u_{n'}, e_1, \ldots, e_g\}$ with $n' = p' \cdot g$.
2. Partition the set of variables $\{v_1, \ldots, v_n\}$ of Φ into g blocks V_i of size p, i.e. $V_i = \{v_{pi+1}, \ldots, v_{p(i+1)}\}$.
3. Partition the set $\{u_1, \ldots, u_{n'}\}$ into g blocks U_i of size p', i.e. $U_i = \{u_{p'i+1}, \ldots, u_{p'(i+1)}\}$.
4. For each block U_i arbitrarily choose an injective function $\psi_i \colon 2^{V_i} \to \binom{U_i}{\lceil p'/2 \rceil}$. This exists since
$$\left|\binom{U_i}{\lceil p'/2 \rceil}\right| = \binom{p'}{\lceil p'/2 \rceil} \geq \frac{2^{p'}}{p'} \geq \frac{2^p p^2}{p + 2\lceil \log p \rceil} \geq 2^p = \left|2^{V_i}\right|.$$
We think of ψ_i as a mapping that given an assignment to the variables of V_i associates with it a subset of U_i of size $\lceil p'/2 \rceil$.
5. For each block U_i, for each $X \in \binom{U_i}{\lceil p'/2 \rceil}$ add the set X to $\mathcal{F}$.
6. For each block U_i, for each $X \in \binom{U_i}{\lfloor p'/2 \rfloor}$ such that $\psi_i^{-1}(\{U_i \setminus X\}) = \emptyset$, add the set X to $\mathcal{F}$.
7. For every clause C of Φ, do the following:
   - Let $I = \{1 \leq j \leq g \mid C \text{ contains a variable of block } V_j\}$;
   - For every $i \in I$, define $\mathcal{A}_i$ as the set
   $$\left\{ X \in \binom{U_i}{\lfloor p'/2 \rfloor} \;\middle|\; \psi_i^{-1}(\{U_i \setminus X\}) \text{ contains an assignment of } V_i \text{ not satisfying } C \right\};$$
   - For every tuple $(A_i)_{i \in I}$ with $A_i \in \mathcal{A}_i$, add the set $\bigcup_{i \in I} A_i$ to $\mathcal{F}$.
8. For every block U_i for every $X \in \binom{U_i}{\lfloor p'/2 \rfloor}$, add the set $X \cup \{e_i\}$ to the set family.

Let $\mathcal{R} \subseteq 2^U$ be the family containing all sets $X \subseteq U$, such that X contains exactly $\lceil p'/2 \rceil$ elements from each block U_i (note that a set in $\mathcal{R}$ may contain some elements from $\{e_1, \ldots, e_g\}$). Moreover let $\mathcal{H} \subseteq 2^U$ be the family of hitting sets of $\mathcal{F}$ and denote $\mathcal{H}_0 = \mathcal{H} \setminus \mathcal{R}$ and $\mathcal{H}_1 = \mathcal{H} \cap \mathcal{R}$.

Lemma 5.8. *We have* $|\mathcal{H}_0| \equiv 0 \pmod 2$.

Proof. For every hitting set $X \in \mathcal{H}$ and block U_i, we know that $|X \cap U_i| \geq \lceil p'/2 \rceil$ since otherwise a subset of the set $U_i \setminus X$ added in Step 5 is not hit by X.

Assume that $X \notin \mathcal{R}$ is a hitting set of $\mathcal{F}$. Let $I \subseteq \{1, \ldots, g\}$ be the set of indices i such that X contains at least least $\lceil p'/2 \rceil + 1$ elements from the block U_i. Note that $I \neq \emptyset$.

Assume that $\mathcal{H}_0 \neq \emptyset$ (otherwise the lemma obviously follows). We construct a fixed point free involution $\pi : \mathcal{H}_0 \to \mathcal{H}_0$ defined as $\pi(X) = X \triangle \{e_i : i \in I\}$ (recall that the operator $\triangle$ denotes the symmetric difference of two sets). The function π is fixed point

free since $I \neq \emptyset$. Moreover $\pi(X) \in \mathcal{H}_0$ since from each block U_i for $i \in I$ the set X contains enough elements to hit all the sets in $\mathcal{F}$ containing the element e_i added in Step 8. Finally $\pi(\pi(X)) = X$ and consequently π is a fixed point free involution, and therefore $|\mathcal{H}_0|$ is even. □

Lemma 5.9. *The number of satisfying assignments of* Φ *is equal to* $|\mathcal{H}_1|$.

Proof. Define $\psi\colon 2^V \to 2^U$ as $\psi(A) = \bigcup_{i=1}^{g} \psi_i(A \cap V_i) \cup \{e_1, \ldots, e_g\}$. Note that ψ is injective, since for every i, ψ_i is injective. Furthermore $\psi(2^V) \subseteq \mathcal{R}$ since for each $A \in 2^V$ the set $\psi(A)$ contains exactly $\lceil \frac{p'}{2} \rceil$ elements from each block U_i.

Hence to prove the lemma, it is sufficient to prove that (1) A is a satisfying assignment iff $\psi(A) \in \mathcal{H}_1$, and (2) for each set $X \in \mathcal{H}_1$ we have $\psi^{-1}(X) \neq \emptyset$.

For the forward direction of (1), note that the sets added in Step 5 are hit by the pigeonhole principle since $|\psi_i(A \cap V_i)| = \lceil \frac{p'}{2} \rceil$. For the sets added in Step 6, consider the following. The set X of size $\lfloor p'/2 \rfloor$ is added because for some i, $\psi_i^{-1}(\{U_i \setminus X\}) = \emptyset$. Thus $\psi_i(A \cap V_i)$ automatically hits X. For the sets added in Step 7, for every clause C the added sets $\bigcup_{i \in I} A_i$ are hit by $\psi_i(A \cap V_i)$ where V_i is a group of variables satisfying C in assignment A (and this exists since A is a satisfying assignment). Finally since for each $i = 1, \ldots, g$ we have $e_i \in X$ all the sets added in Step 8 are hit by X. Therefore X is a hitting set and by the definition of $\mathcal{R}$ we have $X \in \mathcal{R}$.

For the reverse direction of (1), let A be an assignment such that $\psi(A)$ is a hitting set from $\mathcal{H}_1$. We show that A is a satisfying assignment of Φ. Suppose for the sake of contradiction that a clause C is not satisfied by A, and let I be as defined in Step 7 for this C. Since $\psi(A)$ is a hitting set, $|\psi(A) \cap U_i| \geq \lceil \frac{p'}{2} \rceil$ for every i because it hits all sets added in Step 5. Even more precise, $|\psi(A) \cap U_i| = \lceil \frac{p'}{2} \rceil$ because $\psi(A) \in \mathcal{R}$. Therefore, $|U_i \setminus \psi(A)| = \lfloor \frac{p'}{2} \rfloor$, and so $U_i \cap \psi(A) = U_i \setminus (U_i \setminus \psi(A))$ is a member of $\mathcal{A}_i$ for every i. This means that in Step 7 the set $\bigcup_{i \in I} A_i$ with $A_i = U_i \setminus \psi(A)$ was added, but this set is not hit by $\psi(A)$. So it contradicts that $\psi(A)$ is a hitting set.

For (2), let $X \in \mathcal{H}_1$. Note that X contains exactly $\lceil \frac{p'}{2} \rceil$ elements from each block by the definition of $\mathcal{H}_1$. Assume that there is no assignment $A \subseteq V$ such that $\psi(A) = X$. Recall that X is a hitting set. Since X hits all the sets added in Step 6, we infer that $\psi_i^{-1}(\{X \cap U_i\}) \neq \emptyset$ for each i. However, this contradicts the non-existence of $A \subseteq V$ such that $\psi(A) = X$. □

To finish the proof of Theorem 5.7 we note that each set in $\mathcal{F}$ is of size at most $k_1 p'$. Furthermore the cardinality of $\mathcal{F}$ is at most $2^{p'} n/p + k_2 2^{k_1 p'} n$. Thus $\mathcal{F} \subseteq 2^U$ is a valid instance of $(k_1 p', \lceil 2^{p'}/p + k_2 2^{k_1 p'} \rceil)$-$\oplus$HITTING SET and by Lemmas 5.8 and 5.9 the number of hitting sets of $\mathcal{F}$ is odd iff the number of satisfying assignments of Φ is odd. Finally note that our reduction works in polynomial time and since $|U| = n' + g = n(1 + (2\lceil \log p \rceil + 1)/p)$ by the definition of ε_2 the running time needed to solve the hitting set instance $\mathcal{F}$ is upper bounded by $(2 - \varepsilon_2)^n n^{O(1)}$. □

5.4.2 From Hitting Set to Set Cover

In this subsection we show a reduction from the (p_1, p_2)-$\oplus$HITTING SET problem to the following variant of SET COVER.

(p_1, p_2)-⊕SET COVER
Input: A set system $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = m$, $|U| = n$, for every $S \in \mathcal{F}$, $|S| \leq p_1$ and $m \leq p_2 n$.
Question: Is the number of $\mathcal{C} \subseteq \mathcal{F}$ with $\bigcup_{S \in \mathcal{C}} S = U$ odd?

Now we prove the following simple, but surprising lemma, which we think is of independent interest.

Lemma 5.10. *In any bipartite graph $G = (A \cup B, E)$ the number of independent sets is congruent to $|\{X \subseteq A : N(X) = B\}|$ modulo 2.*

Proof. Grouping on their intersection with A, the number of independent sets of G is equal to

$$\sum_{X \subseteq A} 2^{|B \setminus N(X)|} \equiv \sum_{\substack{X \subseteq A \\ |B \setminus N(X)|=0}} 2^0 = |\{X \subseteq A : N(X) = B\}|$$

and the lemma follows. □

Corollary 5.11. *For any set family $\mathcal{F} \subseteq 2^U$ the parity of the number of set covers is equal to the parity of the number of hitting sets.*

Proof. Let $G = (\mathcal{F} \cup U, E)$ be the bipartite graph where $(S, x) \in E$ iff $x \in S$. Note that the number of hitting sets of $\mathcal{F}$ is equal to $|\{X \subseteq U : N(X) = \mathcal{F}\}|$. Then by Lemma 5.10, the number of hitting sets is congruent to the number of independent sets of G modulo 2. And similarly, since the lemma is symmetric with respect to the two color classes of the bipartite graph, the number of set covers of $\mathcal{F}$ is also congruent to the number of independent sets of G modulo 2. □

The above corollary gives us the following theorem.

Theorem 5.12. *If there exists $\varepsilon > 0$ such that for any positive integers p_1, p_2 there exists an algorithm for the (p_1, p_2)-⊕SET COVER problem running in $(2 - \varepsilon)^n n^{O(1)}$ time then for any positive integers p_1, p_2 there exists an algorithm for the (p_1, p_2)-⊕HITTING SET problem running in $(2 - \varepsilon)^n n^{O(1)}$ time.*

5.4.3 From Set Cover to Connected Vertex Cover

First for any $\alpha > 0$ we define an intermediate problem p-⊕SET COVER$_\alpha$, that is, (p_1, p_2)-⊕SET COVER with the solution size bounded by αn and without the constraint on the number of sets in the family $\mathcal{F}$. Next we show that for every $\alpha > 0$ the p-⊕SET COVER$_\alpha$ problem is at least as hard as (p_1, p_2)-⊕SET COVER. Once we have this intermediate result, the reduction to CONNECTED VERTEX COVER follows easily.

p-⊕SET COVER$_\alpha$
Input: An integer t and a set system $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = m$, $|U| = n$, $t \leq \alpha n$, for every $S \in \mathcal{F}$, $|S| \leq p$.
Question: Is the number of $\mathcal{C} \subseteq \mathcal{F}$ with $|\mathcal{C}| = t$ such that $\bigcup_{S \in \mathcal{C}} S = U$ odd?

Theorem 5.13. *If there exist $\alpha, \varepsilon > 0$ such that for any positive integer p there exists an algorithm for the p-$\oplus$Set Cover$_\alpha$ problem running in $(2-\varepsilon)^n n^{O(1)}$ time, then there exists $\varepsilon_2 > 0$ such that for any positive integers p_1, p_2 there exists an algorithm for the (p_1, p_2)-$\oplus$Set Cover problem running in $(2-\varepsilon_2)^n n^{O(1)}$ time.*

Proof. As a proof we present a reduction which for fixed α transforms an instance $(\mathcal{F}', U')$ of (p_1, p_2)-$\oplus$Set Cover into polynomially many instances of the p-$\oplus$Set Cover$_\alpha$ problem, for some positive integer p.

In order to find the parity of the number of all set covers of the instance $(\mathcal{F}', U')$ we find the parity of the number of set covers of a particular size. That is we iterate over all possible sizes of a set cover $j = 1, \ldots, |\mathcal{F}|'$. Let us assume that we want to find the parity of the number of set covers of size j and for each positive integer $j' < j$ we know the parity of the number of set covers of $(\mathcal{F}', U')$ of size j'. Let q be the smallest power of two satisfying $\frac{|\mathcal{F}'|}{q} + 2 \leq \alpha|U'|$. We assume that $\alpha|U'| \geq 3$ since otherwise the instance is small and we can solve it by brute force (recall that α is a given constant). Observe that q is upper bounded by a constant independent of n since $|\mathcal{F}'| \leq p_2 n$.

We create a temporary set system $(\mathcal{F}_0, U_0)$ to ensure that the size of the set covers we are looking for is divisible by q. Let $r = j \bmod q$. We make $(\mathcal{F}_0, U_0)$ by taking the set system $(\mathcal{F}', U')$ and adding $q - r$ new elements to the universe U_0 and also $q - r$ singleton sets of the new elements to the family $\mathcal{F}_0$. Now we are looking for the parity of the number of set covers of size $j_0 = j + (q - r)$ in $(\mathcal{F}_0, U_0)$. Observe that for each $j' < j_0$ we know the parity of the number of set covers of size j' in $(\mathcal{F}_0, U_0)$ since it is equal to the parity of set covers of $(\mathcal{F}', U')$ of size $j' - (q - r) < j$ which we already know.

To obtain a p-$\oplus$Set Cover$_\alpha$ instance we set $U^* = U_0$ and we form a family $\mathcal{F}^*$ of all unions of exactly q sets from $\mathcal{F}_0$, that is for each of $\binom{|\mathcal{F}_0|}{q}$ choices of q sets $S_1, \ldots, S_q \in \mathcal{F}_0$ we add to $\mathcal{F}^*$ the set $\bigcup_{i=1}^{q} S_i$ (note that $\mathcal{F}_0$ might be a multiset). Finally we set $t^* = j_0/q$ which is an integer since $j + (q - r)$ is divisible by q. Observe that $t^* \leq \frac{j}{q} + 1 \leq \alpha|U| - 1$, by the definition of q, however $(\mathcal{F}^*, U^*, t^*)$ might not be a proper instance of $p_1 q$-$\oplus$Set Cover$_\alpha$, since $\mathcal{F}^*$ could be a multiset. Note that each subset of U^* appears in $\mathcal{F}^*$ at most $(2^{qp_1})^q = 2^{q^2 p_1}$ times, since $\mathcal{F}_0$ has no duplicates and each set in $\mathcal{F}^*$ is a union of exactly q sets from $\mathcal{F}_0$. To overcome this technical obstacle we make a new instance $(\mathcal{F}, U, t)$, where as U we take U^* with $z = 1 + q^2 p_1$ elements added, $U = U^* \cup \{e_1, \ldots, e_z\}$. We use elements $\{e_1, \ldots, e_{z-1}\}$ to make sets from $\mathcal{F}^*$ different in $\mathcal{F}$ by taking a different subset of $\{e_1, \ldots, e_{z-1}\}$ for duplicates. Additionally we add one set $\{e_1, \ldots, e_z\}$ to the family $\mathcal{F}$ and set $t = t^* + 1$. In this way we obtain $(\mathcal{F}, U, t)$, that is a proper $(qp_1 + z)$-$\oplus$Set Cover$_\alpha$ instance since $t = t^* + 1 \leq \alpha|U|^* \leq \alpha|U|$. Observe that in the final instance we have $|U| \leq n + q + z$ and $|\mathcal{F}| \leq (p_2 n + q)^q + 1$, which is a polynomial since p_1, p_2, q and z are constants.

To summarize the reduction, we have taken an instance of (p_1, p_2)-$\oplus$Set Cover and iterated over the size of solution. Next we made the size divisible by q by adding additional elements to the universe and created a multiset family $\mathcal{F}^*$ which we made a set family by differentiating equal sets using additional elements of the universe. Our goal was to decide whether the p-$\oplus$Set Cover instance $(\mathcal{F}', U')$ (for $p = qp_1 + z$) has odd number of set covers, which means that we want to control the correspondence between the parity of the number of solutions in each part of the construction. Observe that the only step

of the construction which has nontrivial correspondence between the number of solutions of the former and the latter instance is the grouping step where we transform an instance $(\mathcal{F}_0, U_0, j_0)$ into a multiset instance $(\mathcal{F}^*, U^*, t^*)$.

Hence we assume that we know the parity of the number of set covers of size $t^* = j_0/q$ in $(\mathcal{F}^*, U^*)$ as well as the parity of the number of set covers of size j' for each $j' < j_0$ in $(\mathcal{F}_0, U_0)$. Our objective is to compute the parity of the number of set covers of size j_0 in $(\mathcal{F}_0, U_0)$ in polynomial time and for this reason we introduce a few definitions and lemmas. Recall that each set in $\mathcal{F}^*$ corresponds to a union of exactly q sets in $\mathcal{F}_0$ and let $\Gamma : \mathcal{F}^* \to 2^{\mathcal{F}_0}$ be a function that for each set in $\mathcal{F}^*$ assigns a set of exactly q sets from $\mathcal{F}_0$ that it was made of. Moreover let $\mathcal{S}^* \subseteq 2^{\mathcal{F}^*}$ be the set of set covers of size t^* in $(\mathcal{F}^*, U^*)$ and let $\mathcal{S}_0 \subseteq 2^{\mathcal{F}_0}$ be the set of set covers of size *at most* j_0 in $(\mathcal{F}_0, U_0)$. We construct a mapping $\Phi : \mathcal{S}^* \to \mathcal{S}_0$ which maps each set cover $A \in \mathcal{S}^*$ to a set cover $A_0 \in \mathcal{S}_0$ such that A_0 is exactly the set of sets from $\mathcal{F}_0$ used in the t^* unions of q sets from $\mathcal{F}_0$, that is $\Phi(A) = \bigcup_{X \in A} \Gamma(X)$. In the following lemma we prove that for a set cover $A_0 \in \mathcal{S}_0$ the size of $\Phi^{-1}(A_0)$ depends solely on the size of A_0.

Lemma 5.14. *Let $A_0, B_0 \in \mathcal{S}_0$ such that $|A_0| = |B_0|$. Then $|\Phi^{-1}(A_0)| = |\Phi^{-1}(B_0)|$.*

Proof. Let $A_0 = \{X_1, \ldots, X_a\}$ be a set from $\mathcal{S}_0$, where each $X_i \in \mathcal{F}_0$. Observe that for any $A \in \mathcal{S}$ we have $\Phi(A) = A_0$ if and only if $\bigcup_{i=1}^{a} \Gamma(X_i) = A$. Consequently $|\Phi^{-1}(A_0)|$ is equal to the number of set covers of size t^* in the set system $(\binom{A_0}{q}, A_0)$ and hence $|\Phi^{-1}(A_0)|$ depends only on the size of A_0. □

Now we prove that for each set cover $A_0 \in \mathcal{S}_0$ of size j_0 an odd number of set covers from $\mathcal{S}^*$ is mapped by Φ to A_0.

Lemma 5.15. *For any nonnegative integers a, b such that $b \leq a$ the binomial coefficient $\binom{a}{b}$ is odd iff $\mathrm{ones}(b) \subseteq \mathrm{ones}(a)$, where $\mathrm{ones}(x)$ is the set of indices containing ones in the binary representation of x.*

Proof. For a nonnegative integer x by $f(x)$ let us denote the greatest integer i such that $x!$ is divisible by 2^i, that is

$$\begin{aligned} f(x) &= \sum_{i \geq 1} \left\lfloor \frac{x}{2^i} \right\rfloor \\ &= \Big(\sum_{i \geq 1} \frac{x}{2^i}\Big) - \frac{1}{2} \cdot \left|\left\{i \geq 1 : \left\lfloor \frac{x}{2^{i-1}} \right\rfloor \text{ is odd}\right\}\right| \\ &= \Big(\sum_{i \geq 1} \frac{x}{2^i}\Big) - \frac{|\mathrm{ones}(x)|}{2} \end{aligned}$$

Since $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ we infer that $\binom{a}{b}$ is odd iff $f(a) = f(b) + f(a-b)$, which by the above formula is equivalent to $|\mathrm{ones}(a)| = |\mathrm{ones}(b)| + |\mathrm{ones}(a-b)|$. However for any nonnegative integers x, y we have $\mathrm{ones}(x+y) \leq \mathrm{ones}(x) + \mathrm{ones}(y)$ and moreover $\mathrm{ones}(x+y) = \mathrm{ones}(x) + \mathrm{ones}(y)$ iff there are no carry-operations when adding x to y, which is equivalent to $\mathrm{ones}(x) \cap \mathrm{ones}(y) = \emptyset$.

Therefore by setting $x = b$ and $y = a - b$ we infer that $\binom{a}{b}$ is odd iff $\mathrm{ones}(b) \cap \mathrm{ones}(a-b) = \emptyset$ which is equivalent to $\mathrm{ones}(b) \subseteq \mathrm{ones}(a)$ and the lemma follows. □

Lemma 5.16. *Let $A_0 \in \mathcal{S}_0$ such that $|A_0| = j_0$ then $|\Phi^{-1}(A_0)|$ is odd.*

Proof. Since $|\Phi^{-1}(A_0)|$ is equal to the number set covers of size t^* in the set system $(\binom{A_0}{q}, A_0)$ and $|A_0| = j_0 = t^*q$ we infer that $|\Phi^{-1}(A_0)|$ is equal to the number of unordered partitions of A_0 into sets of size q. Hence $|\Phi^{-1}(A_0)| = \prod_{i=0}^{t^*-1} \binom{j_0-1-iq}{q-1}$. Since j_0 is divisible by q and q is a power of two using Lemma 5.15 we have $|\Phi^{-1}(A_0)| \equiv 1 (\mathrm{mod} 2)$. □

For $j = 1, \ldots, j_0$ by s_j let us denote the parity of the number of set covers of $(\mathcal{F}_0, U_0)$ of size j modulo 2. Recall that we know the value of s_j for each $j < j_0$ and we want to compute s_{j_0} knowing also $|\mathcal{S}^*| \bmod 2$. By Lemma 5.14 we can define d_j for $j = 1, \ldots, j_0$, that is the value of $|\Phi^{-1}(A_0)| \bmod 2$ for a set $A_0 \in \mathcal{S}_0$ of size j. By Lemma 5.16 we know that d_{j_0} equals one. Thus we have the following congruence modulo 2.

$$|\mathcal{S}^*| \quad = \quad \sum_{A_0 \in \mathcal{S}_0} |\Phi^{-1}(A_0)| \equiv \sum_{j=1}^{j_0} s_j d_j = s_{j_0} + \sum_{j=1}^{j_0-1} s_j d_j \,.$$

Hence knowing $|\mathcal{S}^*| \bmod 2$ and all values s_j for $j < j_0$ in order to compute s_{j_0} it is enough to compute all the values d_j, what we can do in polynomial time thanks to the following lemma.

Lemma 5.17. *For each $j = 1, \ldots, j_0$ we can calculate the value of d_j in polynomial time.*

Proof. Again we use that fact that for a set $A_0 \in \mathcal{S}_0$ we have that $|\Phi^{-1}(A_0)|$ is equal to the number set covers of size t^* in the set system $(\binom{A_0}{q}, A_0)$. Using the inclusion-exclusion principle modulo two we obtain the following formula when $|A_0| = j$.

$$|\Phi^{-1}(A_0)| \equiv \sum_{X \subseteq A_0} \left| \left\{ \mathcal{H} \subseteq \binom{X}{q} \middle| |\mathcal{H}| = t^* \right\} \right| = \sum_{i=0}^{j} \binom{j}{i} \binom{\binom{i}{q}}{t^*},$$

Where the second equality follows by grouping all summands $X \subseteq A_0$ with $|X| = i$ for every $0 \leq i \leq |A_0|$. □

Consequently by solving a polynomial of n number of instances of the $\oplus$SET COVER$_\alpha$ problem with universe size bounded by $n + q + z$ and set family size bounded by $(p_2 n + q)^q + 1$ we verify whether the initial formula ϕ has a satisfying assignment, which finishes the prove of Theorem 5.13. □

We can now obtain the following result.

Theorem 5.18. *If there exists $\varepsilon > 0$ and an algorithm running in $(2-\varepsilon)^k |V|^{O(1)}$ time determining the parity of the number of connected vertex covers of size k, then there exists $\varepsilon_2 > 0$ such that for any positive integer p one can solve the p-$\oplus$SET COVER$_\alpha$ problem in $(2-\varepsilon_2)^n n^{O(1)}$ time.*

Proof. For a set system $(\mathcal{F}, U)$ by an incidence graph we denote a bipartite graph $G = (\mathcal{F} \cup U, E)$ where in the set E we have all the edges between a set from $\mathcal{F}$ and its elements. Given an instance $(\mathcal{F}, U)$ of p-$\oplus$SET COVER$_\alpha$, we create an instance of the counting modulo 2 variant of the CONNECTED VERTEX COVER problem with G being obtained from the incidence graph of $(\mathcal{F}, U)$ by adding a vertex s adjacent to all vertices corresponding to sets and adding pendant vertices for every element of $U \cup \{s\}$.

It is easy to see that for every i, there exists a set cover of $(\mathcal{F}, U)$ of size $i \leq \alpha n$ iff there exists a connected vertex cover of G of size at most $i + |U| + 1 \leq |U|(1 + \alpha) + 1$ since without loss of optimality we can take all vertices having a pendant vertex, and then connecting these vertices is equivalent to covering all elements of U with sets in $\mathcal{F}$.

Now let us study the number of connected vertex covers of size j of G for every j. Note that for any connected vertex cover C, $C \cap \mathcal{F}$ must be a set cover of $(\mathcal{F}, U)$ by the connectivity requirement. Hence we group all connected vertex covers in G depending on which set cover in $(\mathcal{F}, U)$ their intersection with $\mathcal{F}$ is. Let c_j be the number of connected vertex covers of G of size j and s_i be the number of set covers of size i in $(\mathcal{F}, U)$, then:

$$c_j = \sum_{i=1}^{j-|U|-1} s_i \binom{|U|+1}{j-i-|U|-1}$$

It is not hard to see that s_i modulo 2 can be determined in polynomial time once $(c_1, \ldots, c_{i+|U|+1})$ modulo 2 are computed by recovering s_1 up to s_i in increasing order of i, since for $i = j - |U| - 1$ we have $\binom{|U|+1}{j-i-|U|-1} = 1$. □

By Theorems 5.4, 5.6, 5.7, 5.12, 5.13, and 5.18 we prove the main theorem of this section.

Theorem 5.19. *There is no algorithm running in $(2-\varepsilon)^k |V|^{O(1)}$ time determining the parity of the number of connected vertex covers of size k for $\varepsilon > 0$ unless the Strong Exponential Time Hypothesis is false.*

Chapter 6

Maximizing disconnectivity is hard under ETH

In Chapter 4 we have shown that a lot of well-known algorithms running in $2^{O(\mathbf{tw}(G))}|V|^{O(1)}$ 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?

In this chapter we provide an 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 $2^{o(p \log p)}|V|^{O(1)}$, where p denotes the width of a given path decomposition of the input graph (note that $\mathbf{pw}(G) \geq \mathbf{tw}(G)$ for every graph G). 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 $2^{o(p \log p)}|V|^{O(1)}$ and consequently prove Theorems 1.3 and 1.4.

6.1 On maximizing the number of connected components

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 6.1 (Theorems 1.3 and 1.4 restated)**.** *Assuming ETH, there is no* $2^{o(p \log p)}|V|^{O(1)}$ *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.*

We start our reductions from $k \times k$ HITTING SET and $k \times k$ PERMUTATION HITTING SET problems. Te problems were introduced and analyzed by Lokshtanov et al. [59]. 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 $i \in [k]$). Let us recall the definitions of those problems.

$k \times k$ 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 6.2 ([59], Theorem 2.4)**.** *Assuming ETH, there is no* $2^{o(k \log k)}m^{O(1)}$ *time algorithm for* $k \times k$ HITTING SET *nor for* $k \times k$ PERMUTATION HITTING SET.

We first prove the bound for MAXIMALLY DISCONNECTED DOMINATING SET by a quite simple reduction from $k \times k$ HITTING SET. This is done in Section 6.2. Then, in Section 6.3 we prove the bound for undirected CYCLE PACKING, by quite involved reduction from $k \times k$ PERMUTATION HITTING SET. In Section 6.4 we provide a reduction to directed CYCLE PACKING and in Section 6.5 we provide a reduction to MAX CYCLE COVER in both variants.

6.2 Maximally Disconnected Dominating Set

In this section 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 $r := k$.

6.2.1 Gadgets

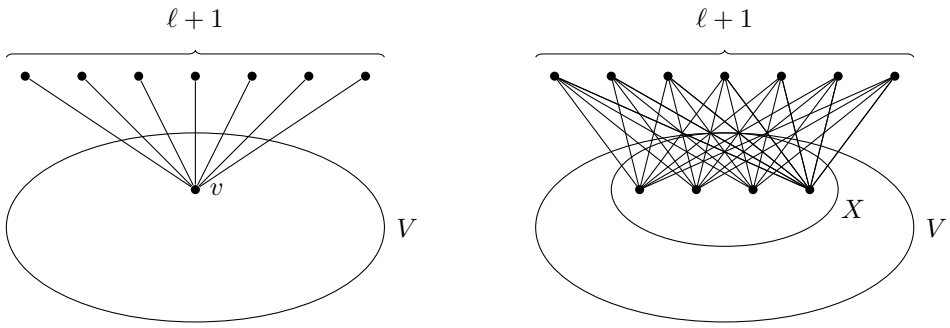


Figure 6.1: Force gadget and one-in-many gadget.

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 6.3. *By adding a* force gadget *for vertex $v \in V$ we mean the following construction: we introduce $\ell + 1$ new vertices of degree one, adjacent to v (see Fig. 6.1).*

Lemma 6.4. *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 6.5. *By adding a* one-in-many gadget *to vertex set $X \subseteq V$ we mean the following construction: we introduce $\ell + 1$ new vertices of degree $|X|$, adjacent to all vertices in X (see Fig. 6.1).*

Lemma 6.6. *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 6.7. *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 $p + 1$. □

6.2.2 Construction

Let $S_i^{\texttt{row}} = \{i\} \times [k]$ be the set containing all elements in the i-th row in the set $[k] \times [k]$. We denote $\mathcal{S} = \{S_s : 1 \leq s \leq m\} \cup \{S_i^{\texttt{row}} : 1 \leq i \leq k\}$. 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 6.8. *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 $A \in \mathcal{S}$). The lemma follows by Lemma 6.7. □

6.2.3 From hitting set to dominating set

Lemma 6.9. *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 $S_i^{\texttt{row}}$.

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, $D \cap X^A = \{x^A_{i_A,j_A}\}$ 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 $1 \leq j \leq k$:

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

Note that D_j is a partition of D into k pairwise disjoint sets, since S contains exactly one element from each row. 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. □

6.2.4 From dominating set to hitting set

Lemma 6.10. *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 at most $\ell - 2k = k + m$ vertices of D 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 consists 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 $x^{S_i^{\text{row}}}_{i,f(i)}$ be the unique vertex in $D \cap X^{S_i^{\text{row}}}$. 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, and hence if suffices to show that S intersects each of the sets S_j for $j = 1, \ldots, m$.

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., $(i,j) \in S$, since $G[D]$ contains the edges $p_j^R x^{S_i^{\text{row}}}_{i,j}$ and $x^{S_i^{\text{row}}}_{i,j} p_i^L$. 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 the set $S_s \in \mathcal{S}$. Let $x^{S_s}_{i,j}$ be the unique vertex in $D \cap X^{S_s}$. Note that $x^{S_s}_{i,j}$ 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 $(i,j) \in S \cap S_s$. □

6.3 Undirected Cycle Packing

6.3.1 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 6.11. *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 $p + 2$. □

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 6.1 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 6.3.2 we describe the *r-in-many gadget*, a tool used widely in the construction. In Section 6.3.3 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 6.3.4 and 6.3.5 respectively. Reductions to directed CYCLE PACKING and to MAX CYCLE COVER are in Sections 6.4 and 6.5 respectively.

6.3.2 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 6.12. *Let H be a multigraph and X be an arbitrary subset of vertices of H. By an* r-in-many gadget attached to X *($1 \leq r < |X|$) we mean the following construction: we introduce $(|X| - r)$ new vertices $\{u_i : 1 \leq i \leq |X| - r\}$ and for each $1 \leq i \leq |X| - r$ 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, consisting of two edges xu_i for some $1 \leq i \leq |X| - r$ and $x \in X$, is called a* gadget short cycle*.*

Definition 6.13. *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 $1 \leq r_i < |X_i|$ for $1 \leq i \leq d$. Let G be a multigraph constructed from H by attaching to X_i a r_i-in-many gadget, for all $1 \leq i \leq d$. 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 6.14. *Let G be a gadget extension of H, and let X_i and r_i be as in Definition 6.13. If $\mathcal{C}_H$ is a family of vertex-disjoint cycles in H satisfying the following property: for each $1 \leq i \leq d$ 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*.*

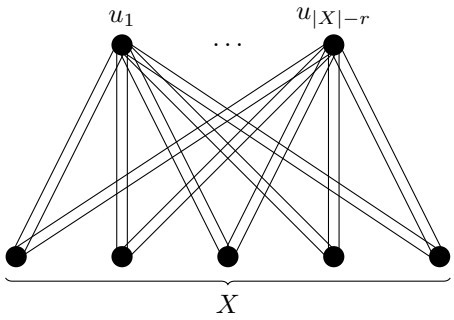


Figure 6.2: The r-in-many gadget attached to set X.

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

Lemma 6.15. *Let G be a gadget extension of H, and let X_i and r_i be as in Definition 6.13. 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 $1 \leq i \leq d$, 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 $1 \leq i \leq d$ 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 safely assume that the r-in-many gadgets are used as in the proof of Lemma 6.15.

Lemma 6.16. *Let G be a gadget extension of H, and let X_i and r_i be as in Definition 6.13. 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 $1 \leq i \leq d$, 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. □

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

Lemma 6.17. *Let G be a gadget extension of H, and let X_i and r_i be as in Definition 6.13. Assume that we are given a path decomposition of H of width p, such that for each $1 \leq i \leq d$ 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 $1 \leq i \leq d$ and let V_i be a bag containing X_i. We introduce bags V_i^j, $1 \leq j \leq |X_i| - r_i$, 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 $|V_i^j| = |V_i| + 1$. □

6.3.3 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 $1 \leq i \leq k$;
   (b) vertices $p_{i,j}^C, q_{i,j}^C$ for $1 \leq i, j \leq k$; for each $1 \leq i \leq k$ we denote $X_i^p = \{p_{i,j}^C : 1 \leq j \leq k\}$ and $X_i^q = \{q_{i,j}^C : 1 \leq j \leq k\}$;

(c) vertices $x^L_{i,s}, x^R_{i,s}, y^L_{i,s}, y^R_{i,s}$ for $1 \leq i \leq k$ and $1 \leq s \leq m$;

(d) and vertices $x^C_{i,s}, y^C_{i,s}, x^Z_{i,s}, y^Z_{i,s}, z^C_{i,s}$ for $1 \leq s \leq m$ and $(i,j) \in S_s$ (recall that there is at most one element in each row in S_s); we denote $X^x_s = \{x^C_{i,s} : (i,j) \in S_s\}$, $X^y_s = \{y^C_{i,s} : (i,j) \in S_s\}$ and $X^z_s = \{z^C_{i,s} : (i,j) \in S_s\}$.

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^Z_i and p^R_j are connected into full bipartite graph with vertices $p^C_{i,j}$ inserted into the middle of each edge, i.e., for all $1 \leq i, j \leq k$ we add edges $p^Z_i p^C_{i,j}$ and $p^C_{i,j} p^R_j$. Similar construction is performed for vertices q^Z_i, q^R_j and $q^C_{i,j}$, i.e., for all $1 \leq i, j \leq k$ we add edges $q^Z_i q^C_{i,j}$ and $q^C_{i,j} q^R_j$.

3. For each $1 \leq i \leq k$, vertices $x^L_{i,s}$ and $y^L_{i,s}$ are arranged into path from p^Z_i to q^Z_i, i.e., $x^L_{i,s} y^L_{i,s} \in E$ for $1 \leq s \leq m$, $y^L_{i,s} x^L_{i,s+1} \in E$ for $1 \leq s < m$ and $q^Z_i y^L_{i,m}, p^Z_i x^L_{i,1} \in E$. By $\mathcal{P}^L_i$ we denote the path from p^Z_i to q^Z_i

4. For each $1 \leq i \leq k$, vertices $x^R_{i,s}$ and $y^R_{i,s}$ are arranged into path from p^R_i to q^R_i, i.e., $x^R_{i,s} y^R_{i,s}$ for $1 \leq s \leq m$, $y^R_{i,s} x^R_{i,s+1} \in E$ for $1 \leq s < m$ and $p^R_i x^R_{i,1}, q^R_i y^R_{i,m} \in E$. By $\mathcal{P}^R_i$ we denote the path from p^R_i to q^R_i.

5. For each $1 \leq s \leq m$, if $(i,j) \in S_s$, we add a path $(x^L_{i,s}, x^Z_{i,s}, x^C_{i,s}, x^R_{j,s})$.

6. Similarly, for each $1 \leq s \leq m$, if $(i,j) \in S_s$, we add a path $(y^L_{i,s}, y^Z_{i,s}, y^C_{i,s}, y^R_{j,s})$.

7. Moreover, for each $1 \leq s \leq m$ and $(i,j) \in S_s$ we add a cycle $(x^Z_{i,s}, z^C_{i,s}, y^Z_{i,s})$.

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^p_i and X^q_i for $1 \leq i \leq k$ we attach 1-in-many gadgets;

2. to vertex sets X^x_s and X^y_s for $1 \leq s \leq m$ we attach 1-in-many gadgets;

3. for $1 \leq s \leq m$ we attach a $(|X^z_s| - 1)$-in-many gadget to X^z_s.

Clearly, the above construction can be done in polynomial time. Note that we can pack the following number of gadget short cycles in G:

$$\begin{aligned}\ell_G &:= \sum_{i=1}^{k} (|X^p_i| - 1 + |X^q_i| - 1) + \sum_{s=1}^{m} (|X^x_s| - 1 + |X^y_s| - 1 + |X^z_s| - (|X^z_s| - 1)) \\ &= 2k^2 - 2k - m + 2\sum_{s=1}^{m} |S_s|.\end{aligned}$$

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.

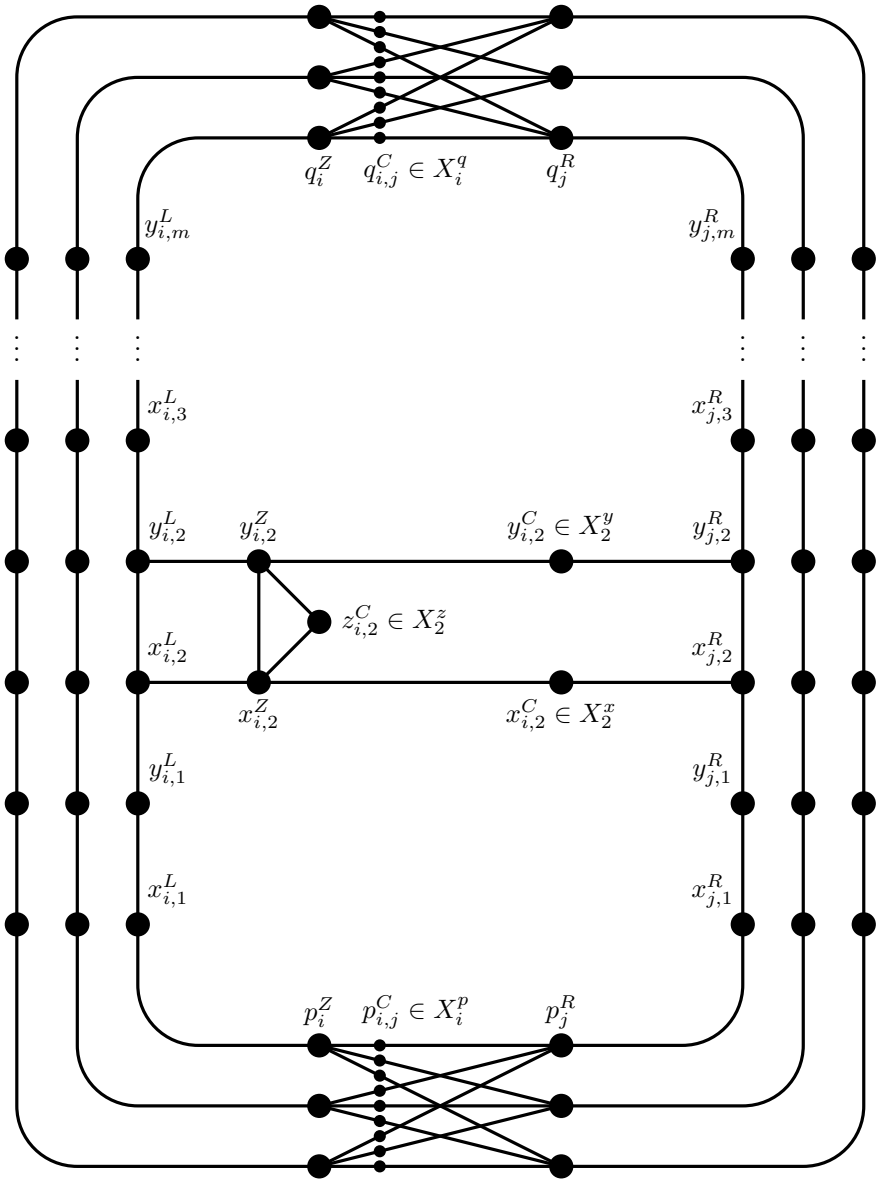


Figure 6.3: The part of the graph H with the main frame and the part for an element $(i, j) \in S_s$. Recall that the gadget safe in H family may cover at most one element of X_i^p, X_i^q, X_s^x and X_s^y and cannot cover whole X_s^z.

Lemma 6.18. *In polynomial time we can construct a path decomposition of G of width $11k$.*

Proof. By Lemma 6.17, 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 the 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 $1 \leq s \leq m$. □

Let us note that the above bound is not optimal, but we need only $O(k)$ bound.

6.3.4 From hitting set to disjoint cycles

We prove Point 2 by the following lemma:

Lemma 6.19. *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 6.15, 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 $(y^L_{i,s_j}, y^Z_{i,s_j}, y^C_{i,s_j}, y^R_{f(i),s_j})$, the subpath of $\mathcal{P}^R_{f(i)}$ from $y^R_{f(i),s_j}$ to $x^R_{f(i),s_{j+1}}$, the path $(x^R_{f(i),s_{j+1}}, x^C_{i,s_{j+1}}, x^Z_{i,s_{j+1}}, x^L_{i,s_{j+1}})$ and the subpath of $\mathcal{P}^L_i$ from $x^L_{i,s_{j+1}}$ to y^L_{i,s_j};

3. $C(i,h(i))$ consists of the path $(y^L_{i,s_{h(i)}}, y^Z_{i,s_{h(i)}}, y^C_{i,s_{h(i)}}, y^R_{f(i),s_{h(i)}})$, the subpath of $\mathcal{P}^R_{f(i)}$ from $y^R_{f(i),s_j}$ to $q^R_{f(i)}$, the path $(q^R_{f(i)}, q^C_{i,f(i)}, q^Z_i)$ and the subpath of $\mathcal{P}^L_i$ from q^Z_i to $y^L_{i,s_{h(i)}}$.

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. $X^p_i \setminus \{p^C_{i,f(i)}\}$ and $X^q_i \setminus \{q^C_{i,f(i)}\}$ for $1 \leq i \leq k$;
2. $X^x_s \setminus \{x^C_{i_s,s}\}$ and $X^y_s \setminus \{y^C_{i_s,s}\}$ for $1 \leq s \leq m$;
3. $z^C_{i_s,s} \in X^z_s$ 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. 6.4. □

6.3.5 From disjoint cycles to hitting set

In this section we prove Point 3 by the following lemma:

Lemma 6.20. *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 6.16 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 6.19.

First let us analyze subgraph $H[L \cup R \cup C]$. Note that this subgraph is a forest containing:

1. k paths $\mathcal{P}^L_i$ for $1 \leq i \leq k$ without the endpoints, i.e., without p^Z_i and q^Z_i;
2. k trees consisting of paths $\mathcal{P}^R_j$ ($1 \leq j \leq k$) with attached leaves $p^C_{i,j}$, $q^C_{i,j}$ for $1 \leq i \leq k$ and $x^C_{i,s}$, $y^C_{i,s}$ for $1 \leq s \leq m$, $(i,j) \in S_s$.
3. $\sum_{s=1}^m |S_s|$ isolated vertices $z^C_{i,s}$, $1 \leq s \leq m$, $(i,j) \in S_s$.

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 $H[L \cup R \cup C]$:

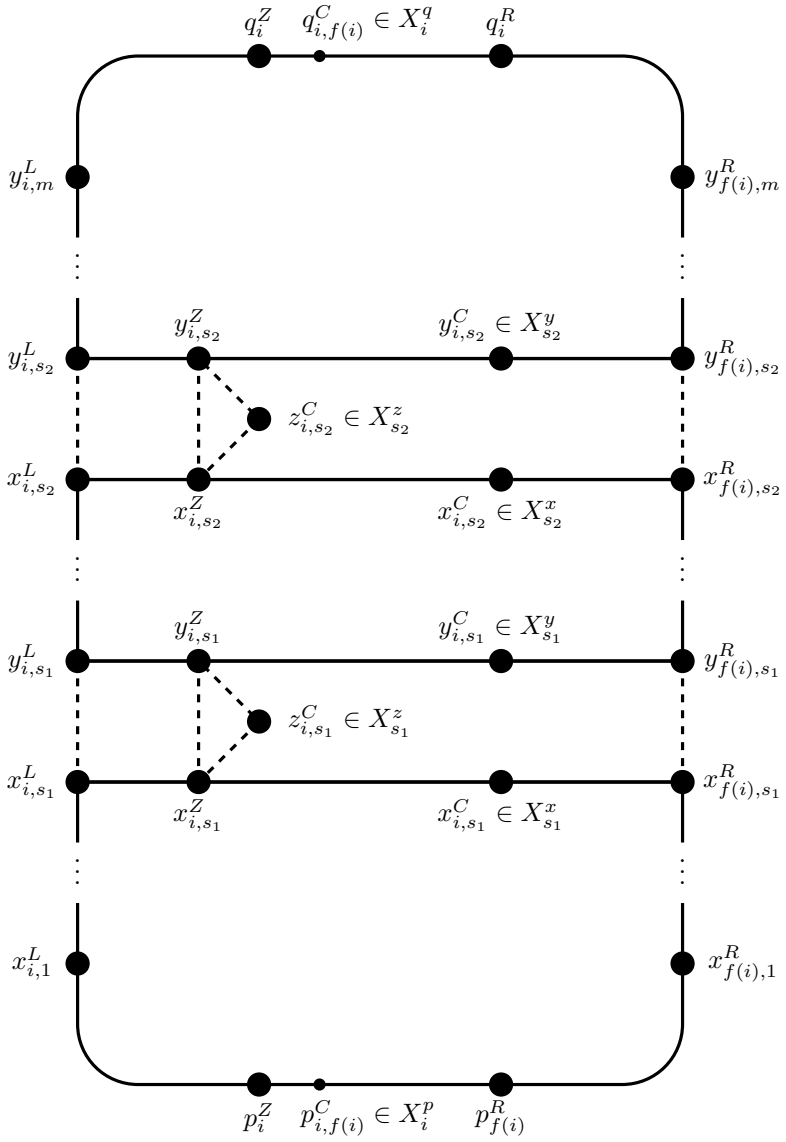


Figure 6.4: An example how to pack three cycles for $(i, f(i)) \in S$ and $h(i) = 2$.

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 $y_{i,s}^C$.

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 $z^C_{i,s}$ be a (arbitrarily chosen) vertex not covered by $\mathcal{C}$, where $(i,j) \in S_s$. Let us focus on the vertex $x^Z_{i,s}$. It has three neighbours apart from $z^C_{i,s}$: vertices $x^L_{i,s}$, $x^C_{i,s}$ and $y^Z_{i,s}$. The vertex $x^C_{i,s}$ has degree two, and the other neighbour is $x^R_{j,s}$. As $x^L_{i,s} \in C(i, s-1)$ and $x^R_{j,s} \in C(f^{-1}(j), s-1)$, the vertex $x^Z_{i,s}$ 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 $x^Z_{i,s}$ and $y^Z_{i,s}$ 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 $(x^L_{i,s}, x^Z_{i,s}, x^C_{i,s}, x^R_{j,s})$, 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 $y^Z_{i,s}$. The vertex $x^Z_{i,s}$ is used on cycle $C(i, s-1)$, $y^Z_{i,s} \notin C(i, s-1)$ and we assumed the vertex $z^C_{i,s}$ is not covered by $\mathcal{C}$. Thus, $y^Z_{i,s}$ lies on a cycle C with path $(y^L_{i,s}, y^Z_{i,s}, y^C_{i,s}, y^R_{f(i),s})$. As $x^L_{i,s}, x^R_{f(i),s} \in C(i, s-1)$, and we are allowed to cover only one vertex from X^y_s, C contains a path $(x^L_{i,s+1}, y^L_{i,s}, y^Z_{i,s}, y^C_{i,s}, y^R_{f(i),s}, y^R_{f(i),s+1})$ (if $s < m$) or $(q^Z_i, y^L_{i,s}, y^Z_{i,s}, y^C_{i,s}, y^R_{f(i),s}, q^R_{f(i)})$ (if $s = m$).

Now focus on vertices $x^Z_{i',s}$ for $i' \neq i$. As $x^C_{i,s}$ is covered by $\mathcal{C}$, the vertex $x^C_{i',s}$ cannot be covered too. Thus, $x^Z_{i',s}$ and $y^Z_{i',s}$ 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 $x^L_{i',s}$ does not lie on $C^{i'}$, thus $C^{i'} = (x^Z_{i',s}, y^Z_{i',s}, z^C_{i',s})$.

As vertices $x^Z_{i',s}$ and $y^Z_{i',s}$ are covered by the cycle $(x^Z_{i',s}, z^C_{i',s}, y^Z_{i',s})$, the cycle $C(i', s-1)$ contains the path $(x^L_{i',s}, y^L_{i',s}, x^L_{i',s+1})$ (if $s < m$) or $(x^L_{i',s}, y^L_{i',s}, q^Z_{i'})$ (if $s = m$). As vertices $x^C_{i,s}$ and $y^C_{i,s}$ are covered by cycles $C(i, s-1)$ and C, the cycle $C(i', s-1)$ contains path $(x^R_{f(i'),s}, y^R_{f(i'),s}, x^R_{f(i'),s+1})$ (if $s < m$) or $(x^R_{f(i'),s}, y^R_{f(i'),s}, q^R_{f(i')})$ (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. □

6.4 From undirected to directed Cycle Packing

In this section we provide a reduction from undirected to directed CYCLE PACKING, proving Theorem 6.1 for directed CYCLE PACKING.

Lemma 6.21. *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*.

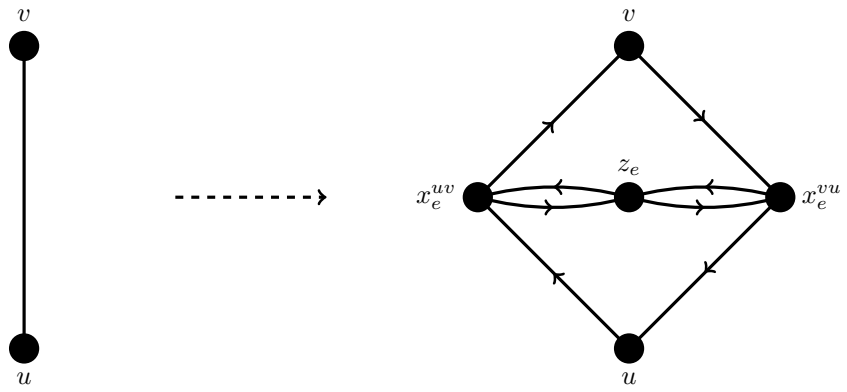


Figure 6.5: The construction of the gadget replacing edge uv.

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 set of 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 $p + 3$. □

6.5 From Cycle Packing to Max Cycle Cover

In this section we provide a reduction from CYCLE PACKING to MAX CYCLE COVER that proves Theorem 6.1 for MAX CYCLE COVER, both in directed and undirected setting. Formally, we prove the following lemma

Lemma 6.22. *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.

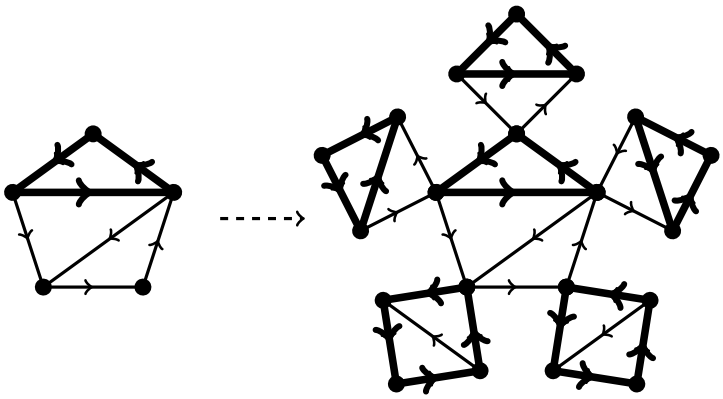


Figure 6.6: An example of the reduction from CYCLE PACKING to MAX CYCLE COVER in the directed setting, together with the conversion of cycle families.

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 $a_v, c_v \in C_v$ and $C_v \subseteq \{a_v, b_v, c_v, v\}$. 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 the 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 $p + 3$. □

Chapter 7

Tightness of the Cut&Count technique under SETH

Following the framework introduced by Lokshtanov et al. [58], we prove that an improvement in the base of the exponent in a number of our algorithms would contradict SETH. In other words in this chapter we prove lower bounds gathered in Column C of Table 1.1.

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) [58]. 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. Our lower bounds show that this is not an artifact of the Cut&Count technique, but rather an intrinsic characteristic of these problems.

In this chapter we provide reductions in the spirit of [58] that prove the following running time lower bounds for arbitrary $\varepsilon > 0$:

- $(3-\varepsilon)^p|V|^{O(1)}$ for CONNECTED VERTEX COVER,
- $(4-\varepsilon)^p|V|^{O(1)}$ for CONNECTED DOMINATING SET,
- $(4-\varepsilon)^p|V|^{O(1)}$ for CONNECTED FEEDBACK VERTEX SET,
- $(4-\varepsilon)^p|V|^{O(1)}$ for CONNECTED ODD CYCLE TRANSVERSAL,
- $(3-\varepsilon)^p|V|^{O(1)}$ for FEEDBACK VERTEX SET.

As a consequence we obtain two additional lower bounds. The $(4-\varepsilon)^p|V|^{O(1)}$ lower bound 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 [34]. The $(3-\varepsilon)^p|V|^{O(1)}$ lower bound 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.

7.1 Overview

In the proofs we follow the same approach as Lokshtanov et al. [58] in the lower bounds for problems without the connectivity requirement. We mostly base on the VERTEX COVER and DOMINATING SET lower bounds of [58], 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 sections 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 [58], we prove pathwidth bounds for the constructed graphs using mixed search game. Let us recall the informal definition from [58].

Definition 7.1 ([71, 58])**.** *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 7.2 ([71])**.** *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.

7.2 Connected Vertex Cover

Theorem 7.3. *Assuming SETH, for every constant $\varepsilon > 0$ there is no 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 $1 \leq t \leq n'$ and $1 \leq \ell \leq \eta$ we create a path $\mathcal{P}_{t,\ell}$ consisting of $2a$ vertices $v^{\alpha}_{t,\ell,k}$, $0 \leq k < a$ and $1 \leq \alpha \leq 2$, arranged in the following order:

$$v^1_{t,\ell,0}, v^2_{t,\ell,0}, v^1_{t,\ell,1}, \ldots, v^1_{t,\ell,a-1}, v^2_{t,\ell,a-1}.$$

Furthermore we connect all vertices $v^2_{t,\ell,k}$ ($0 \leq k < a$) and $v^1_{t,\ell,0}$ to the root r. To simplify further notation we denote $v^1_{t,\ell,a} = r$. 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, $1 \leq t \leq n'$, and a position k, $0 \leq k < a$. For each $1 \leq \ell \leq \eta$ we create three vertices $h^{\alpha}_{t,\ell,k}$, $1 \leq \alpha \leq 3$ that are pairwise adjacent and all are adjacent to the root r. Moreover, we add edges $h^1_{t,\ell,k}v^1_{t,\ell,k}$, $h^2_{t,\ell,k}v^2_{t,\ell,k}$ and $h^3_{t,\ell,k}v^1_{t,\ell,k+1}$. Let $\mathcal{H}_{t,\ell,k} = \{h^{\alpha}_{t,\ell,k} : 1 \leq \alpha \leq 3\}$, $\mathcal{H}_{t,k} = \bigcup_{\ell=1}^{\eta} \mathcal{H}_{t,\ell,k}$. 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 $\{h^{s_\ell}_{t,\ell,k} : 1 \leq \ell \leq \eta\}$. For each sequence $S \in \{1,2,3\}^\eta$ we add three vertices $x^S_{t,k}$, $x^{S*}_{t,k}$ and $y^S_{t,k}$, where $x^S_{t,k}$ 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 $x^S_{t,k}x^{S*}_{t,k}$, $x^S_{t,k}y^S_{t,k}$ and $y^S_{t,k}r$. In the graph G the vertices $x^{S*}_{t,k}$ are of degree one, thus any connected vertex cover in G needs to include all vertices $x^S_{t,k}$. Let $\mathcal{Y}_{t,k} = \{y^S_{t,k} : S \in \{1,2,3\}^\eta\}$ for $1 \leq t \leq n'$, $0 \leq k < a$ 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 $y^S_{t,k}$ 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 $3^\eta \geq 2^\beta$ vertices $x^S_{t,k}$ 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 $0 \leq j < (2\eta n' + 1)$. 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 $1 \leq t \leq n'$. 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 $0 \leq j < (2\eta n' + 1)$ we add an edge $y^S_{t,mj+i}c_{i,j}$.

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 7.2.

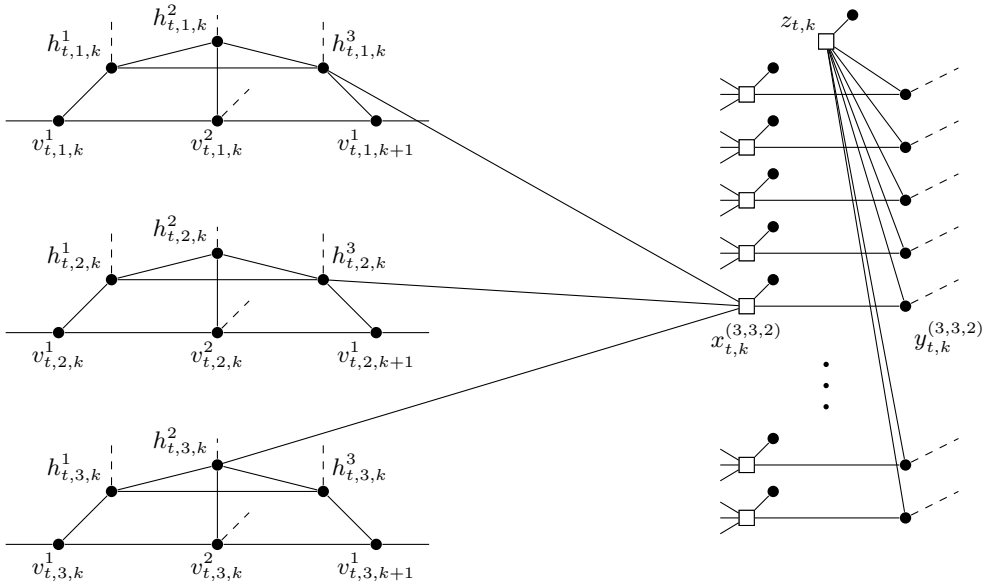


Figure 7.1: Group gadget $\mathbf{B}_{t,k}$ for $\eta = 3$. Dashed edges are connecting a vertex with the root r. Vertices that have a pendant neighbour and thus need to be included in any connected vertex cover of G are presented as squares.

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 7.4. *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 $0 \leq i < m$, $0 \leq j < (2\eta n' + 1)$, $x^S_{t,k}$ 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 $|X_{\text{force}}| = 1 + a + (3^\eta + 1)n'a$.

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

$$X_h = \bigcup_{t=1}^{n'} \bigcup_{k=0}^{a-1} \bigcup_{\ell=1}^{\eta} \{h^\alpha_{t,\ell,k} : \alpha \neq S_t(\ell)\}.$$

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 $v^2_{t,\ell,k}$ if $S_t(\ell) = 2$, and all vertices $v^1_{t,\ell,k}$ 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 $|X_v| = \eta n' a$.

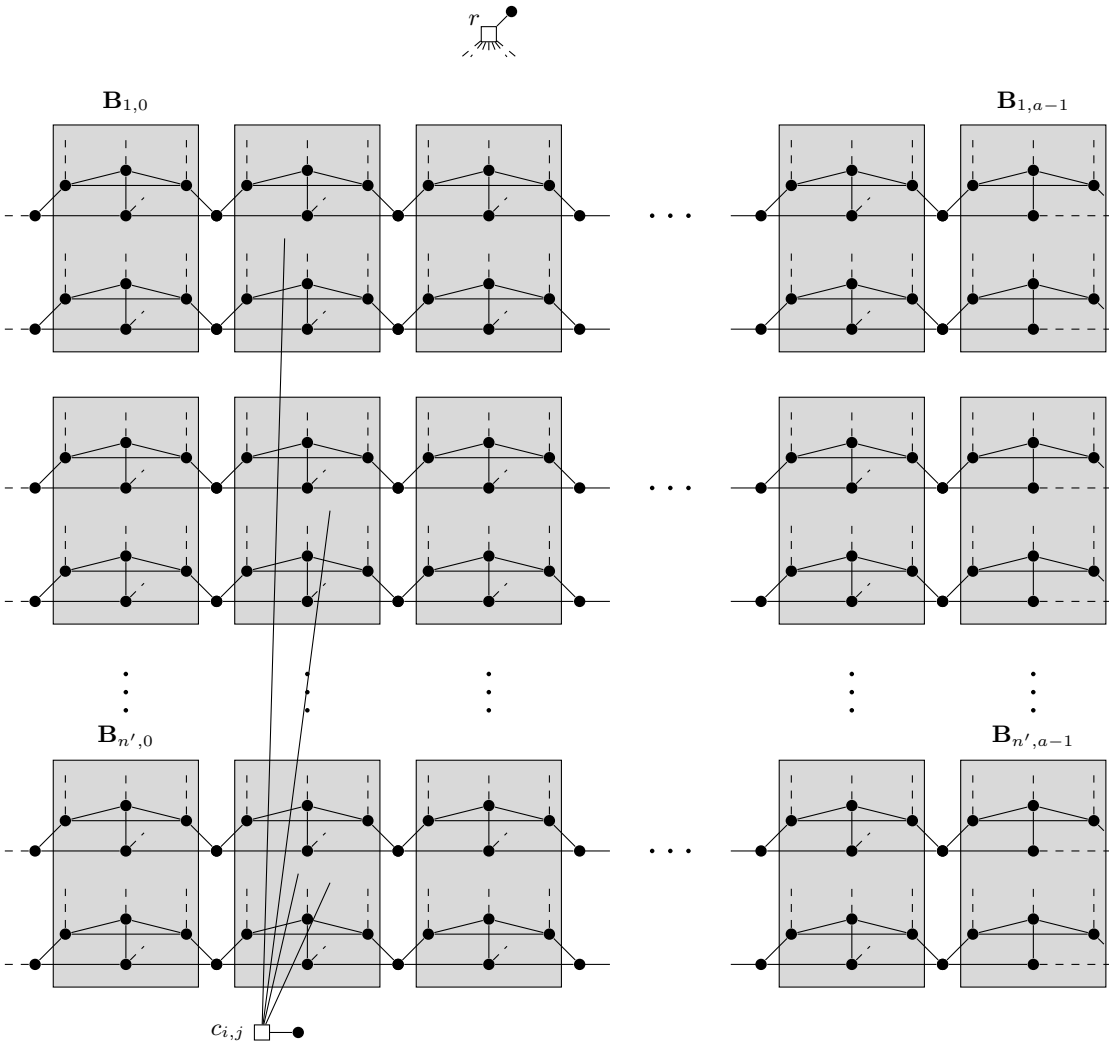


Figure 7.2: Gray rectangles represent group gadgets. Dashed edges connect have the root vertex r as one of the endpoints. Vertices that have a pendant neighbour and thus need to be included in any connected vertex cover of G are presented as squares.

Finally, let us define the set X_y to be the set of all vertices $y^{S_t}_{t,k}$ 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^*, $c^*_{i,j}$, $x^{S*}_{t,k}$ and $z^*_{t,k}$ are isolated in $G \setminus X$, as their single neighbours in G are included in X_{force}.

2. The vertices $v^{\alpha}_{t,\ell,k}$ 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 $v^{\alpha}_{t,\ell,k}$ from $\mathcal{H}$ are in X_h.

3. The vertices $h^{S_t(\ell)}_{t,\ell,k}$ are isolated in $G \setminus X$, since their single neighbours on paths $\mathcal{P}_{t,\ell}$ are in X_v and all other neighbours of $h^{S_t(\ell)}_{t,\ell,k}$ are in X_{force} and in X_h.

4. Finally, the vertices $y^S_{t,k}$ 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.

2. If vertices $v^2_{t,\ell,k}$ belong to X_v, they are connected to r by direct edges. Otherwise, the vertices $v^1_{t,\ell,k}$ are connected via vertices $h^1_{t,\ell,k}$ or $h^3_{t,\ell,k-1}$ (with the exception of vertices $v^1_{t,\ell,0}$ that are connected directly).

3. Each vertex $x^S_{t,k}$ for $S \neq S_t$ is connected to r via any vertex $h^{S(\ell)}_{t,\ell,k}$ for which $S(\ell) \neq S_t(\ell)$.

4. Vertices $x^{S_t}_{t,k}$ and $z_{t,k}$ are connected to r via $y^{S_t}_{t,k}$.

5. Finally, each vertex $c_{i,k}$ is connected to r via any vertex $y^{S_t}_{t,k}$ for which the assignment ϕ on variables from F_t satisfies the clause C_i.

□

Lemma 7.5. *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 $1 \leq i \leq m$, $0 \leq j < (2\eta n' + 1)$, $x^S_{t,k}$ 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 $y^S_{t,k}$ 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 $h^{S_{t,k}(\ell)}_{t,\ell,k} \notin X_h$ for $1 \leq \ell \leq \eta$.

Note that the vertex $x^{S_{t,k}}_{t,k} \in X_{\text{force}}$ does not have a neighbour in X_h, thus, to connect it to the root r, we need to have $y^{S_{t,k}}_{t,k} \in X_y$. 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., $y^S_{t,k} \in X$ 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 $|X_v \cap \{v^\alpha_{t,\ell,k} : 1 \leq \alpha \leq 2\}| = 1$. Let $\alpha(t,\ell,k)$ be such that $v^{\alpha(t,\ell,k)}_{t,\ell,k} \in X_v$. 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 $v^1_{t,\ell,k+1}h^3_{t,\ell,k}$ is not covered by X. Moreover, $h^1_{t,\ell,k+1} \in X_h$, as otherwise $v^1_{t,\ell,k+1}$ is isolated in $G[X]$, and $h^2_{t,\ell,k+1} \in X_h$, as otherwise the edge $v^2_{t,\ell,k+1}h^2_{t,\ell,k+1}$ 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 $v^1_{t,\ell,k}h^1_{t,\ell,k}$ is not covered by X. Thus $\alpha(t,\ell,k+1) = 1$, as otherwise the edge $v^2_{t,\ell,k}v^1_{t,\ell,k+1}$ is not covered by X, and $S_{t,k+1}(\ell) \neq 2$, as otherwise the edge $v^2_{t,\ell,k+1}h^2_{t,\ell,k+1}$ 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 $y^S_{t,mj+i} \in X$ and the assignment of F_t that corresponds to S satisfies C_i. However, we know that $y^S_{t,mj+i} \in X$ implies that $S = S_{t,mj+i}$, and thus ϕ satisfies C_i. □

Pathwidth bound

Lemma 7.6. *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. □

Proof of Theorem 7.3. 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 7.4, 7.5 ensure correctness, whereas Lemma 7.6 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. □

7.3 Connected Dominating Set

Theorem 7.7. *Assuming SETH, for every constant $\varepsilon > 0$ there is no 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 $F_1, \ldots, F_{n'}$, each of size two, hence $n' = n/2$. The pathwidth of G will be roughly n'.

First, we add to the graph G 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:

$$v^1_{t,0}, h^1_{t,0}, v^2_{t,0}, h^2_{t,0}, v^1_{t,1}, \ldots, h^2_{t,a-1}.$$

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.

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 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 $0 \leq j < n+1$. 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 $v^1_{t,mj+i}$ and if x^1_t occurs negatively in C_i then we connect $c_{i,j}$ with $v^2_{t,mj+i}$. Similarly if x^2_t occurs positively in C_i then we connect $c_{i,j}$ with $h^1_{t,mj+i}$ and if x^2_t occurs negatively in C_i then we connect $c_{i,j}$ with $h^2_{t,mj+i}$. Intuitively taking the vertex $v^1_{t,mj+i}$ into a connected dominating set corresponds to setting x^1_t to true, whereas taking the vertex $h^1_{t,mj+i}$ into a connected dominating set 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 $K = 1 + n' \cdot 2a$ be the size of the connected dominating set we ask for.

Correctness

Lemma 7.8. *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 $F_t = \{x^1_t, x^2_t\}$ and for each $0 \leq k < a$ we include into X:

1. the vertex $v^1_{t,k}$ if $\phi(x^1_t)$ is true, and $v^2_{t,k}$ otherwise;
2. the vertex $h^1_{t,k}$ if $\phi(x^2_t)$ is true, and $h^2_{t,k}$ 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.

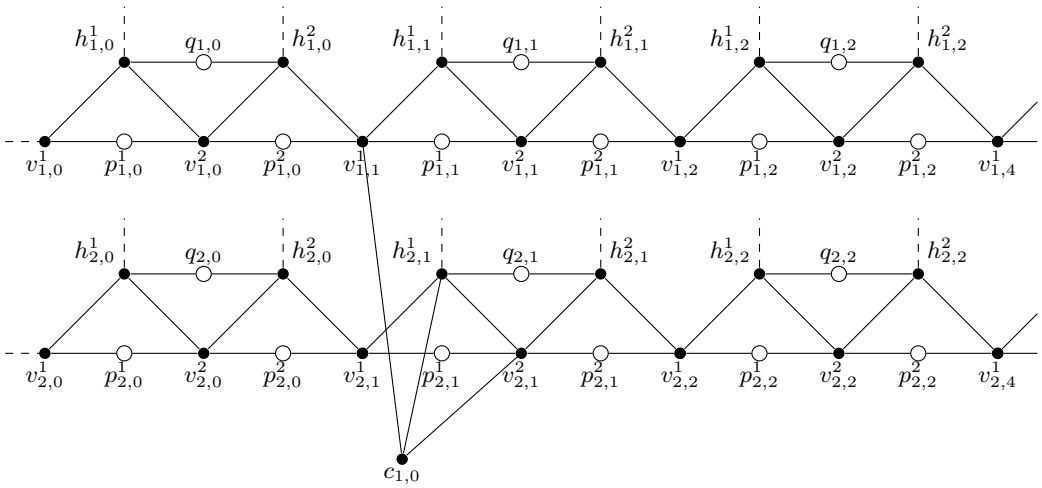


Figure 7.3: Part of the construction for CONNECTED DOMINATING SET. Dashed edges are connecting a vertex with the root r. Empty circles represent guard vertices.

1. r^* and all vertices in $\mathcal{H}$ are dominated by the root r.

2. All guards $p^1_{t,k}$, $p^2_{t,k}$ and $q^1_{t,k}$ are dominated by $X \cap (\mathcal{H} \cup \mathcal{V})$ (with the possible exception of $p^2_{t,a-1}$ that is dominated by r).

3. All vertices in $\mathcal{V}$ are dominated by $X \cap \mathcal{H}$ (with the possible exception of $v^1_{t,0}$ that is dominated by r).

4. Finally, each clause vertex $c_{i,j}$ is dominated by any vertex $v^\alpha_{t,mj+i}$ or $h^\alpha_{t,mj+i}$ 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 $v^1_{t,k} \in X$, then $v^1_{t,k}$ is connected to the root via $h^2_{t,k-1}$ or $h^1_{t,k}$, with the exception of $v^1_{t,0}$, that is connected to r directly;

2. if $v^2_{t,k} \in X$, then $v^2_{t,k}$ is connected to the root via $h^1_{t,k}$ or $h^2_{t,k}$.

□

Lemma 7.9. *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 $p^1_{t,k}$ and $q_{t,k}$ ensure that for each $1 \leq t \leq n'$ and $0 \leq k < a$ at least one vertex $v^\alpha_{t,k}$ and at least one vertex $h^\alpha_{t,k}$ ($1 \leq \alpha \leq 2$) 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 $v^{\alpha}_{t,k}$ and exactly one vertex $h^{\alpha}_{t,k}$ 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 $F_t = \{x^1_t, x^2_t\}$ we define:

1. $\phi_k(x^1_t)$ to be true if $v^1_{t,k} \in X$ and false if $v^2_{t,k} \in X$;
2. $\phi_k(x^2_t)$ to be true if $h^1_{t,k} \in X$ and false if $h^2_{t,k} \in X$.

We now show that the assignments ϕ_k cannot differ much for all indices $0 \leq k < a$. Note that for each block $F_t = \{x^1_t, x^2_t\}$ and $0 \leq k < a-1$:

1. if $\phi_k(x^1_t)$ is true, then $\phi_{k+1}(x^1_t)$ is also true, as otherwise $v^2_{t,k}, v^1_{t,k+1} \notin X$ and the guard $p^2_{t,k}$ is not dominated by X;
2. if $\phi_k(x^2_t)$ is true, then $\phi_{k+1}(x^2_t)$ is also true, as otherwise $h^2_{t,k}, h^1_{t,k+1} \notin X$ and the vertex $v^2_{t,k}$ is either not dominated by X (if $v^2_{t,k} \notin X$) or isolated in $G[X]$ (if $v^2_{t,k} \in X$).

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 assignment $\phi = \phi_{mj}$ 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. □

Pathwidth bound

Lemma 7.10. *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 ($0 \leq k < a$) there are searchers on all vertices $v^1_{t,k}$ for $1 \leq t \leq n'$. Let $0 \leq i < m$ and $0 \leq j < n+1$ be integers such that $k = i + mj$. We place a searcher on $c_{i,j}$. Then, for each $1 \leq t \leq n'$ in turn we put 7 searchers on vertices $p^1_{t,k}$, $v^2_{t,k}$, $p^2_{t,k}$, $v^1_{t,k+1}$, $h^1_{t,k}$, $h^2_{t,k}$ and $q_{t,k}$, and then remove 7 searchers from vertices $v^1_{t,k}$, $p^1_{t,k}$, $v^2_{t,k}$, $p^2_{t,k}$, $h^1_{t,k}$, $h^2_{t,k}$ 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. □

Proof of Theorem 7.7. 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 7.8, 7.9 ensure correctness, whereas Lemma 7.10 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. □

7.4 Connected FVS and Connected OCT

Theorem 7.11. *Assuming SETH, for every constant $\varepsilon > 0$ there is no 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 7.12. *Assuming SETH, for every constant $\varepsilon > 0$ there is no 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 7.11 and 7.12 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 $u^1_{vw}, u^2_{vw}, u^3_{vw}$ and edges vu^1_{vw}, $u^1_{vw}w$, vu^2_{vw}, $u^2_{vw}u^3_{vw}$, $u^3_{vw}w$. In the graph G the vertices u^α_{vw} 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^α_{vw} *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^1r^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 $1 \leq t \leq n'$ 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:

$$v^1_{t,0}, h^1_{t,0}, v^2_{t,0}, h^2_{t,0}, v^1_{t,1}, \ldots, h^2_{t,a-1}.$$

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. Moreover, we connect all vertices $h^{\alpha}_{t,k}$ ($1 \leq t \leq n'$, $0 \leq k < a$, $1 \leq \alpha \leq 2$) to the vertex r^{α}. 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 pentagon edges $v^1_{t,k}v^2_{t,k}$, $v^2_{t,k}v^1_{t,k+1}$ and $h^1_{t,k}h^2_{t,k}$.

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 $0 \leq j < n+1$. 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

Lemma 7.13. *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 $F_t = \{x^1_t, x^2_t\}$ and for each $0 \leq k < a$ we include into X:

1. the vertex $v^1_{t,k}$ if $\phi(x^1_t)$ is true, and $v^2_{t,k}$ otherwise;
2. the vertex $h^1_{t,k}$ if $\phi(x^2_t)$ is true, and $h^2_{t,k}$ otherwise.

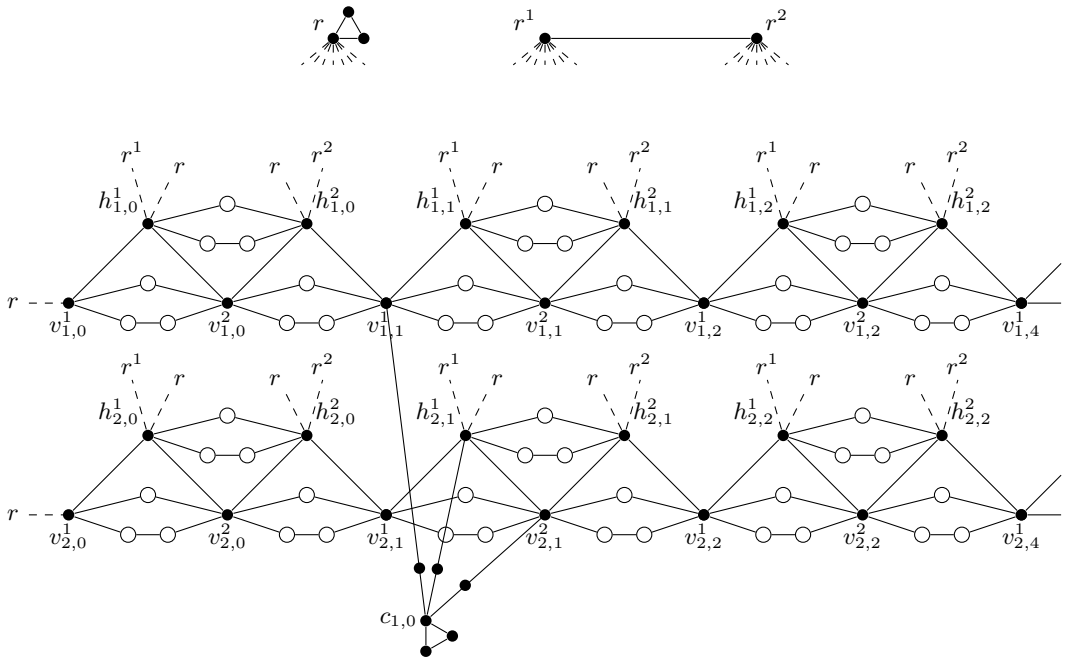


Figure 7.4: Part of the construction for CONNECTED FEEDBACK VERTEX SET and CONNECTED ODD CYCLE TRANSVERSAL. Dashed edges have one endpoint r, r^1 or r^2. Empty circles represent guard vertices in the pentagon edges.

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 $0 \leq j < n+1$ we add to the set X exactly one neighbour of $c_{i,j}$, namely we add the vertex $c^\gamma_{i,j,t,\beta}$, 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^{**}, $c^*_{i,j}$, $c^{**}_{i,j}$, $c^\gamma_{i,j,t,\beta}$ ($0 \leq i < m$, $0 \leq j < n+1$, $1 \leq t \leq n'$, $1 \leq \beta \leq 2$, $\gamma \in \{v, h\}$) 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 $c^\gamma_{i,j,t,\beta}$, 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 $c^\gamma_{i,j,t,\beta}$, 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^1r^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 $1 \leq t \leq n'$ and $0 \leq k < a$:

1. if $v^1_{t,k} \in X$, then $v^1_{t,k}$ is connected to root via $h^2_{t,k-1}$ or $h^1_{t,k}$, with the exception of $v^1_{t,0}$, that is connected to r directly;
2. if $v^2_{t,k} \in X$, then $v^2_{t,k}$ is connected to root via $h^1_{t,k}$ or $h^2_{t,k}$.

We are left with the vertices $c_{i,j}$ and their neighbours chosen to X for $0 \leq i < m$, $0 \leq j < n+1$. 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. □

Lemma 7.14. *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 $h^1_{t,k}h^2_{t,k}$ and $v^1_{t,k}v^2_{t,k}$ ($1 \leq t \leq n'$, $0 \leq k < a$) 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 $|X| \leq K = 1 + 2a + n' \cdot 2a$, 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 $0 \leq k < a$ we construct an assignment ϕ_k as follows. For each block $F_t = \{x^1_t, x^2_t\}$ we define:

1. $\phi_k(x^1_t)$ to be true if $v^1_{t,k} \in X$ and false if $v^2_{t,k} \in X$;
2. $\phi_k(x^2_t)$ to be true if $h^1_{t,k} \in X$ and false if $h^2_{t,k} \in X$.

We now show that the assignments ϕ_k cannot differ much for all indices $0 \leq k < a$. Note that for each block $F_t = \{x^1_t, x^2_t\}$ and $0 \leq k < a-1$:

1. if $\phi_k(x^1_t)$ is true, then $\phi_{k+1}(x^1_t)$ is also true, as otherwise X contains no endpoint of the pentagon edge $v^2_{t,k}v^1_{t,k+1}$.
2. if $\phi_k(x^2_t)$ is true, then $\phi_{k+1}(x^2_t)$ is also true, as otherwise $h^2_{t,k}, h^1_{t,k+1} \notin X$ and either the vertex $v^2_{t,k}$ is not connected to the root in $G[X]$ (if $v^2_{t,k} \in X$, since vertices from $\mathcal{C}$ are leaves in $G[X]$) or $G \setminus X$ contains a cycle of length five consisting of vertices $v^2_{t,k}$, $h^2_{t,k}$, r^2, r^1 and $h^1_{t,k+1}$ (if $v^2_{t,k} \notin X$).

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 assignment $\phi = \phi_{mj}$ 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. □

Pathwidth bound

Lemma 7.15. *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 ($0 \leq k < a$) there are searchers on all vertices $v^1_{t,k}$ for $1 \leq t \leq n'$. Let $0 \leq i < m$ and $0 \leq j < n+1$ 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 $1 \leq t \leq n'$ in turn we put $O(1)$ searchers on vertices $v^2_{t,k}$, $v^1_{t,k+1}$, $h^1_{t,k}$, $h^2_{t,k}$, the guard vertices of pentagon edges $v^1_{t,k}v^2_{t,k}$, $v^2_{t,k}v^1_{t,k+1}$,$h^1_{t,k}h^2_{t,k}$, and all vertices $c^\gamma_{i,j,t,\beta}$, and then remove searchers from the vertices $v^1_{t,k}$, $v^2_{t,k}$, $h^1_{t,k}$, $h^2_{t,k}$, all aforementioned guard vertices and vertices $c^\gamma_{i,j,t,\beta}$. 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. □

Proof of Theorems 7.11 and 7.12. 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 7.13, 7.14, together with an observation that any connected feedback vertex set is also a connected odd cycle transversal in G, ensure correctness, whereas Lemma 7.15 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. □

7.5 Feedback Vertex Set

Theorem 7.16. *Assuming SETH, for every constant $\varepsilon > 0$ there is no 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 [58].

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 $v^\alpha_{t,\ell,k}$, $0 \leq k < a$ and $1 \leq \alpha \leq 3$, arranged in the following order:

$$v^1_{t,\ell,0}, v^2_{t,\ell,0}, v^3_{t,\ell,0}, v^1_{t,\ell,1}, \ldots, v^1_{t,\ell,a-1}, v^2_{t,\ell,a-1}, v^3_{t,\ell,a-1}.$$

Let $\mathcal{V}_{t,\ell,k} = \{v^\alpha_{t,\ell,k} : 1 \leq \alpha \leq 3\}$, $\mathcal{V}_{t,k} = \bigcup_{\ell=1}^{\eta} \mathcal{V}_{t,\ell,k}$ and $\mathcal{V} = \bigcup_{t=1}^{n'} \bigcup_{k=0}^{a-1} V_{t,k}$.

Third, for each two consecutive vertices $v^\alpha_{t,\ell,k}$, $v^{\alpha'}_{t,\ell,k'}$ on the path $\mathcal{P}_{t,\ell}$ we introduce vertices $h^{\alpha,1}_{t,\ell,k}$ and $h^{\alpha,2}_{t,\ell,k}$ connected by a triangle edge. Furthermore, we add edges $h^{\alpha,1}_{t,\ell,k} v^\alpha_{t,\ell,k}$, $h^{\alpha,1}_{t,\ell,k} r$, $h^{\alpha,2}_{t,\ell,k} v^{\alpha'}_{t,\ell,k'}$, $h^{\alpha,2}_{t,\ell,k} r$. Let $\mathcal{H}$ be the set of all vertices $h^{\alpha,\gamma}_{t,\ell,k}$.

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 $v^\alpha_{t,\ell,k}$, $h^{\alpha,\gamma}_{t,\ell,k}$ ($1 \leq \ell \leq \eta$, $1 \leq \alpha \leq 3$, $1 \leq \gamma \leq 2$) 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 $p^\alpha_{t,\ell,k}$ ($1 \leq \alpha \leq 3$), pairwise connected by triangle edges. Moreover, for each $1 \leq \alpha \leq 3$ we connect $p^\alpha_{t,\ell,k}$ and $v^\alpha_{t,\ell,k}$ by a triangle edge. Let $\mathcal{P}$ be the set of all vertices $p^\alpha_{t,\ell,k}$ 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 $q^{S,\alpha}_{t,\ell,k}$ ($1 \leq \alpha \leq 3$), pairwise connected by triangle edges. Second, we connect $q^{S,\alpha}_{t,\ell,k}$ with $v^\alpha_{t,\ell,k}$

($1 \leq \alpha \leq 3$) with a triangle edge. Third, we introduce a vertex $x^S_{t,k}$ and connect all vertices $q^{S,s_\ell}_{t,\ell,k}$ ($1 \leq \ell \leq \eta$) and the vertex $x^S_{t,k}$ into a cycle $\mathcal{Q}^S_{t,k}$. Fourth, we connect all vertices $x^S_{t,k}$ 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 $y^S_{t,k}$ and $z^S_{t,k}$ and triangle edges $x^S_{t,k}y^S_{t,k}$ and $y^S_{t,k}z^S_{t,k}$. Let $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ be the sets of all vertices $x^S_{t,k}$, $y^S_{t,k}$ and $z^S_{t,k}$, 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 $x^S_{t,k}$ 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 $z^S_{t,mj+i}$ 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.

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 7.6.

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 7.17. *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

$$X_v = \{v^{S_t(\ell)}_{t,\ell,k} : 1 \leq t \leq n', 1 \leq \ell \leq \eta, 0 \leq k < a\}$$
$$X_p = \{p^\alpha_{t,\ell,k} : 1 \leq t \leq n', 1 \leq \ell \leq \eta, 0 \leq k < a, 1 \leq \alpha \leq 3, \alpha \neq S_t(\ell)\}$$
$$X_q = \{q^{S,\alpha}_{t,\ell,k} : 1 \leq t \leq n', 1 \leq \ell \leq \eta, 0 \leq k < a, S \in \{1,2,3\}^\eta, 1 \leq \alpha \leq 3, \alpha \neq S_t(\ell)\}$$
$$X_x = \{x^{S_t}_{t,k} : 1 \leq t \leq n', 0 \leq k < a\}$$
$$X_y = \{y^S_{t,k} : 1 \leq t \leq n', 0 \leq k < a, S \in \{1,2,3\}^\eta, S \neq S_t\}$$
$$X_z = \{z^{S_t}_{t,k} : 1 \leq t \leq n', 0 \leq k < a\}$$

Moreover, we define the set $X_h \subseteq \mathcal{H}$ to contain, for each $1 \leq t \leq n'$, $1 \leq \ell \leq \eta$, $0 \leq k < a$, $1 \leq \alpha \leq 3$ and $(k,\alpha) \neq (a-1,3)$, the vertex $h^{\alpha,1}_{t,\ell,k}$ if $v^\alpha_{t,\ell,k} \notin X_v$ and

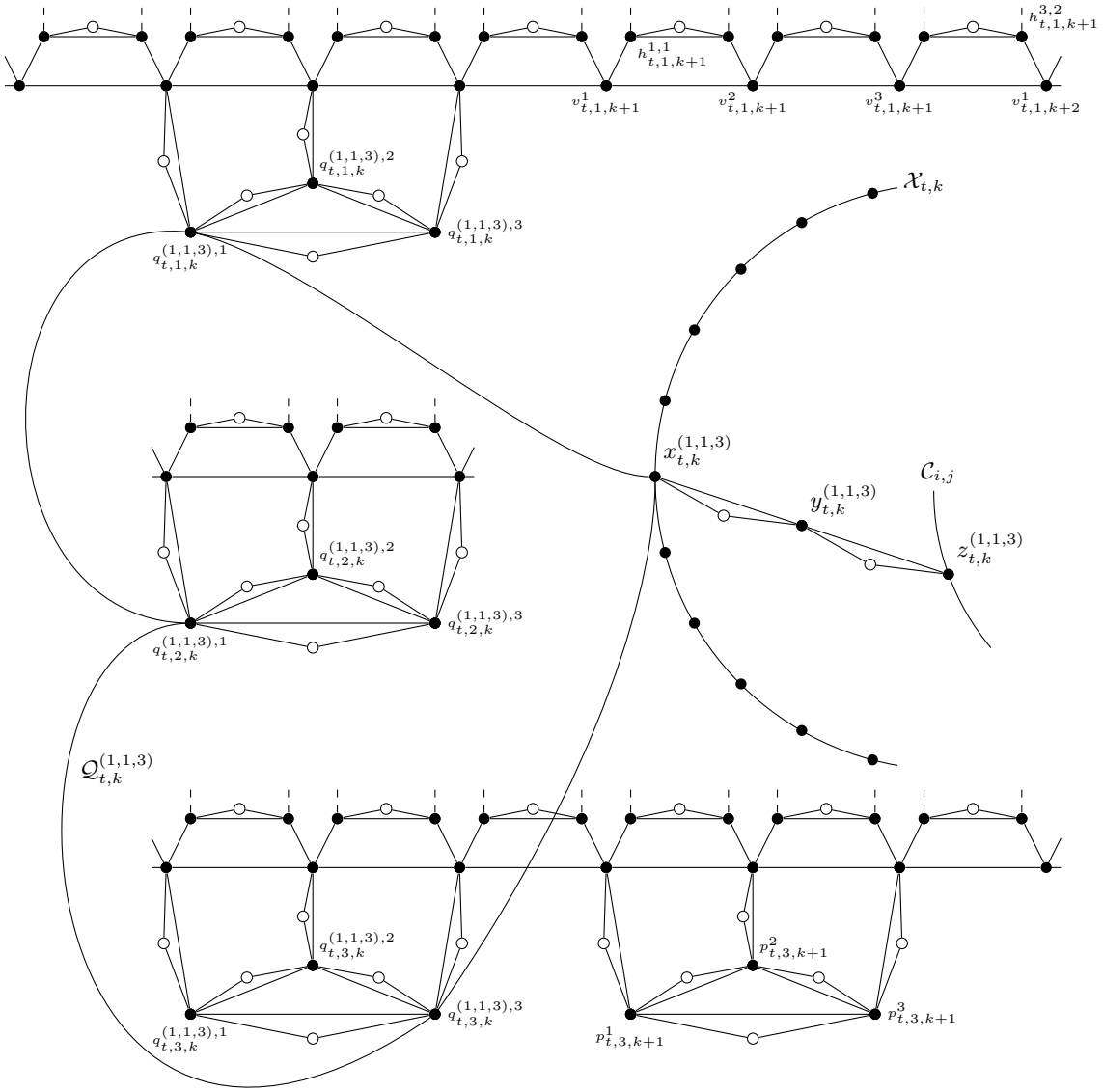


Figure 7.5: Part of the construction around group gadget $\mathbf{B}_{t,k}$ for $\eta = 3$. Dashed edges are connecting a vertex with the root r. Empty circles represent guard vertices.

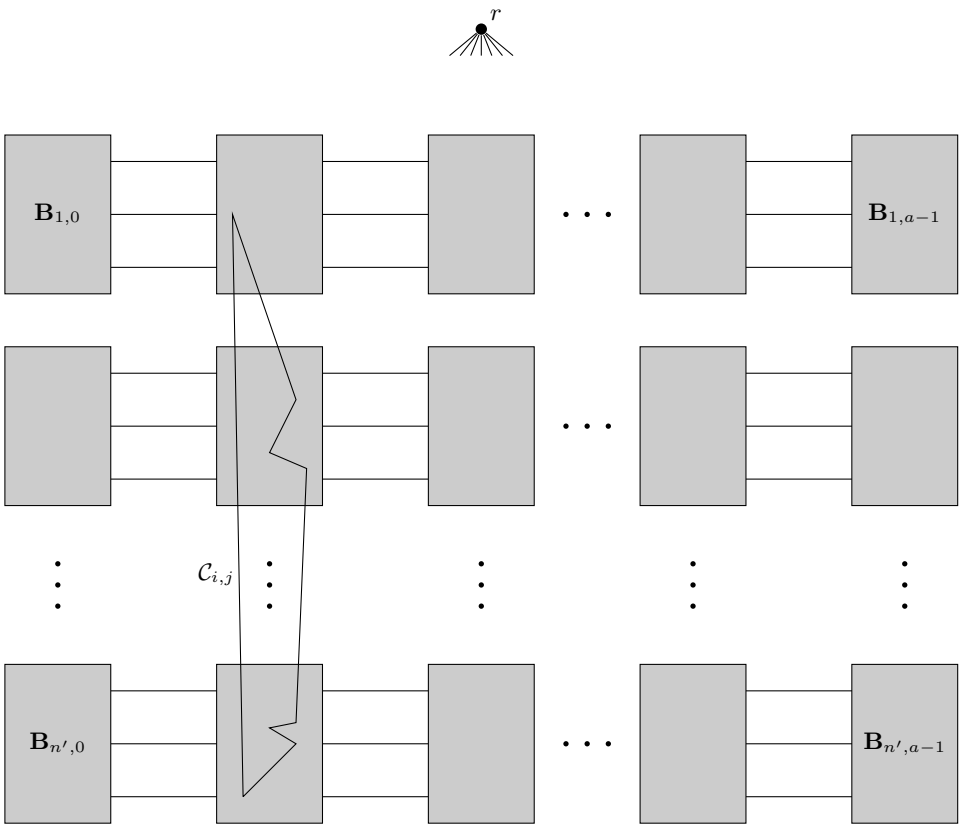


Figure 7.6: The arrangement of the group gadgets in G.

the vertex $h^{\alpha,2}_{t,\ell,k}$ 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 $h^{\alpha,1}_{t,\ell,k}$ and $h^{\alpha,2}_{t,\ell,k}$.
2. X includes two out of three vertices in each triple $p^{\alpha}_{t,\ell,k}$, $1 \leq \alpha \leq 3$, and in each triple $q^{S,\alpha}_{t,\ell,k}$, $1 \leq \alpha \leq 3$.
3. If $p^{\alpha}_{t,\ell,k} \notin X$ or $q^{S,\alpha}_{t,\ell,k} \notin X$, then $v^{\alpha}_{t,\ell,k} \in X$.
4. If $y^{S}_{t,k} \notin X$ then both $x^{S}_{t,k}$ and $z^{S}_{t,k}$ 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 $p^{\alpha}_{t,\ell,k} \notin X$ or $q^{S,\alpha}_{t,\ell,k} \notin X$, then $v^{\alpha}_{t,\ell,k} \in X$. 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 $v^{\alpha}_{t,\ell,k}$ and $v^{\alpha'}_{t,\ell,k'}$ 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 $p^{\alpha}_{t,\ell,k} \notin X$ or $q^{S,\alpha}_{t,\ell,k} \notin X$, then $v_{t,\ell,k} \in X$ and if $x^{S}_{t,k} \notin X$ then $y^{S}_{t,k} \in X$. 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 $\mathcal{Q}^{S}_{t,k} \setminus X$ are of degree at most two, except for the vertex $x^{S}_{t,k}$. If $S \neq S_t$, then for any $1 \leq \ell \leq \eta$ such that $S(\ell) \neq S_t(\ell)$ we have $q^{S,S(\ell)}_{t,\ell,k} \in X_q$ and the cycle $\mathcal{Q}^{S}_{t,k}$ is intersected by X. On the other hand, if $S = S_t$, then $x^{S}_{t,k} \in X_x$. 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 $x^{S_t}_{t,k} \in X_x$ and we infer that $G_2 \setminus X$ is a forest.

As for $\mathcal{Y}$, note that if $y^{S}_{t,k} \notin X$ then $S = S_t$ and $x^{S}_{t,k}, z^{S}_{t,k} \in X$, and $y^{S}_{t,k}$ 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 $\mathcal{C}_{i,j}$, $0 \leq i < m$, $0 \leq j < 2\eta n' + 1$. Consider a clause C_i and an index $0 \leq j < 2\eta n' + 1$. As ϕ satisfies C_i, there exists a block F_t, such that a variable from this block satisfies C_i. Then $z^{S_t}_{t,mi+j}$ is both on the cycle $\mathcal{C}_{i,j}$ and in X, and $G_3 \setminus X$ is a forest, too. □

Lemma 7.18. *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 beginning 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 $y^S_{t,k}$ and $z^S_{t,k}$ 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 $\mathcal{X}_{t,k}$, 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 $\{q^{S,\alpha}_{t,\ell,k} : 1 \leq \alpha \leq 3\}$ 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 $\{p^\alpha_{t,\ell,k} : 1 \leq \alpha \leq 3\}$ need to be included in X. Moreover, if $p^\alpha_{t,\ell,k} \notin X$ for some $1 \leq \alpha \leq 3$, then $v^\alpha_{t,\ell,k} \in X$, as otherwise the triangle edge $v^\alpha_{t,\ell,k}p^\alpha_{t,\ell,k}$ 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 $h^{\alpha,1}_{t,\ell,k}h^{\alpha,2}_{t,\ell,k}$ 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 $\{p^\alpha_{t,\ell,k} : 1 \leq \alpha \leq 3\}$. If it is wholly contained in X, then $X \setminus \{p^1_{t,\ell,k}\} \cup \{v^1_{t,\ell,k}\}$ 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 $\{v^\alpha_{t,\ell,k} : 1 \leq \alpha \leq 3\}$.

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^1_A, h^2_A, h^1_B, h^2_B, edges rh^1_A, rh^2_A, rh^1_B, rh^2_B, $h^1_A v_A$, $h^2_A v_B$, $h^1_B v_B$, $h^2_B v_C$ and triangle edges $h^1_A h^2_A$, $h^1_B h^2_B$. Note that $|\{h^1_A, h^2_A, h^1_B, h^2_B\} \setminus X| = 2$, 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 $v^\alpha_{t,\ell,k} \in X_v$ 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 $\{q^{S,\alpha}_{t,\ell,mj+i} : 1 \leq \alpha \leq 3\}$ for $1 \leq \ell \leq \eta$, $S \in \{1,2,3\}^\eta$. As $q^{S,\alpha}_{t,\ell,mj+i}$ is connected to $v^{\alpha}_{t,\ell,mj+i}$ by a triangle edge, we infer that $q^{S,\alpha}_{t,\ell,mj+i} \notin X$ iff $v^{\alpha}_{t,\ell,mj+i} \in X$, which is equivalent to $S_t(\ell) = \alpha$. Thus $x^{S_t}_{t,mj+i} \in X$, as otherwise the cycle $\mathcal{Q}^{S_t}_{t,mj+i}$ is disjoint with X. As $|X_x| = K_x$, the set X_x contains exactly one vertex out of each cycle $\mathcal{X}_{t,k}$, and we infer that $x^{S}_{t,mj+i} \notin X$ for $S \neq S_t$. Recall that $x^{S}_{t,mj+i}$ and $y^{S}_{t,mj+i}$ are connected by a triangle edge, thus $y^{S}_{t,mj+i} \in X$ 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 $y^{S}_{t,mj+i}z^{S}_{t,mj+i}$, and we infer that if $z^{S}_{t,mj+i} \in X$ 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 $z^{S}_{t,mj+i} \in X$ and the assignment of the variables of the block F_t that corresponds to S satisfies C_i. However, we have proven that $z^{S}_{t,mj+i} \in X$ implies $S = S_t$, thus ϕ satisfies C_i and the proof is finished. □

Pathwidth bound

Lemma 7.19. *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 7.16. 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 7.17, 7.18 ensure correctness, whereas Lemma 7.19 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. □

Chapter 8

Conclusions and open problems

For several years it was known that most of the local problems (where by local we mean that a solution can be verified by checking separately the neighbourhood of each vertex), standard dynamic programming techniques give $c^{\mathbf{tw}}|V|^{O(1)}$ time algorithms for a constant c. The main consequence of the Cut&Count technique as presented in this dissertation is that problems which can be formulated as a local constraint with an additional upper bound on the number of connected components also admit $c^{\mathbf{tw}}|V|^{O(1)}$ time algorithms. 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 derandomization for treewidth parametrizations 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)?

Bibliography

[1] Manindra Agrawal and Somenath Biswas. Primality and identity testing via Chinese remaindering. *Journal of the ACM*, 50(4):429–443, 2003.

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

[3] Vikraman Arvind and Partha Mukhopadhyay. Derandomizing the isolation lemma and lower bounds for circuit size. In A. Goel, K. Jansen, J. D. P. Rolim, and R. Rubinfeld, editors, *11th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems/12th International Workshop on Randomization and Computation, APPROX/RANDOM 2008*, volume 5171 of *Lecture Notes in Computer Science*, pages 276–289. Springer, 2008.

[4] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Dániel Marx. Approximation schemes for steiner forest on planar graphs and graphs of bounded treewidth. In L. J. Schulman, editor, *42nd Annual ACM Symposium on Theory of Computing, STOC 2010*, pages 211–220. ACM Press, 2010.

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

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

[7] Andreas Björklund. Determinant sums for undirected Hamiltonicity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, pages 173–182. IEEE Computer Society, 2010.

[8] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: Fast subset convolution. In D. S. Johnson and U. Feige, editors, *39th Annual ACM Symposium on Theory of Computing, STOC 2007*, pages 67–74. ACM Press, 2007.

[9] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. The travelling salesman problem in bounded degree graphs. In L. Aceto, I. Damgård, L. A.

Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *35th International Colloquium on Automata, Languages and Programming (1), ICALP 2008*, volume 5125 of *Lecture Notes in Computer Science*, pages 198–209. Springer, 2008.

[10] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *arXiv.org. The Computing Research Repository*, abs/1007.1161, 2010.

[11] Hans L. Bodlaender. On disjoint cycles. *International Journal of Foundations of Computer Science*, 5(1):59–68, 1994.

[12] Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.

[13] Glencora Borradaile, Claire Kenyon-Mathieu, and Philip N. Klein. A polynomial-time approximation scheme for Steiner tree in planar graphs. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *18th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007*, pages 1285–1294. Society for Industrial and Applied Mathematics, 2007.

[14] Nicolas Bourgeois, Bruno Escoffier, Vangelis Th. Paschos, and Johan M. M. van Rooij. A bottom-up method and fast algorithms for max independent set. In H. Kaplan, editor, *12th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2010*, volume 6139 of *Lecture Notes in Computer Science*, pages 62–73. Springer, 2010.

[15] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved LP-based approximation for Steiner tree. In L. J. Schulman, editor, *42nd Annual ACM Symposium on Theory of Computing, STOC 2010*, pages 583–592. ACM Press, 2010.

[16] Chris Calabro and Ramamohan Paturi. -sat is no harder than decision-unique--sat. In *Fourth International Computer Science Symposium in Russia, CSR 2009*, volume 5675 of *Lecture Notes in Computer Science*, pages 59–70. Springer, 2009.

[17] David G. Cantor and Erich Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991.

[18] Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set new measure and new structures. In H. Kaplan, editor, *12th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2010*, volume 6139 of *Lecture Notes in Computer Science*, pages 93–104. Springer, 2010.

[19] Suresh Chari, Pankaj Rohatgi, and Aravind Srinivasan. Randomness-optimal unique element isolation with applications to perfect matching and related problems. *SIAM Journal on Computing*, 24(5):1036–1050, 1995.

[20] Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *Journal of Computer and System Sciences*, 74(7):1188–1198, 2008.

[21] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[22] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On Problems as Hard as CNF-Sat. *Manuscript*, 2011.

[23] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *52th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2011*, page to appear.

[24] Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, and W. Thomas, editors, *36th International Colloquium on Automata, Languages and Programming (1), ICALP 2009*, volume 5555 of *Lecture Notes in Computer Science*, pages 304–315. Springer, 2009.

[25] Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. *Theoretical Computer Science*, 411(40-42):3701–3713, 2010.

[26] 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 L. Wang, editor, *13th Annual International Conference on Computing and Combinatorics, COCOON 2005*, volume 3595 of *Lecture Notes in Computer Science*, pages 859–869. Springer, 2005.

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

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

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

[30] Frederic Dorn, Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Beyond bidimensionality: Parameterized subexponential algorithms on directed graphs. *arXiv.org. The Computing Research Repository*, abs/1001.0821, 2010.

[31] Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Fast subexponential algorithm for non-local problems on graphs of bounded genus. In L. Arge and R. Freivalds, editors, *10th Scandinavian Workshop on Algorithm Theory, SWAT 2006*, volume 4059 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 2006.

[32] Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Catalan structures and dynamic programming in H-minor-free graphs. In S.-H. Teng, editor, *19th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008*, pages 631–640. Society for Industrial and Applied Mathematics, 2008.

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

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

[35] Rodney G. Downey and Michael R. Fellows. Fixed parameter tractability and completeness. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory: Current Research*, pages 191–225. Cambridge University Press, 1993.

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

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

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

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

[40] Fedor V. Fomin and Dimitrios M. Thilikos. A simple and fast approach for solving problems on planar graphs. In V. Diekert and M. Habib, editors, *21st International Symposium on Theoretical Aspects of Computer Science, STACS 2004*, volume 2996 of *Lecture Notes in Computer Science*, pages 56–67. Springer, 2004.

[41] H. Gebauer. On the number of Hamilton cycles in bounded degree graphs. In *5th Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2008*, pages 241–248. Society for Industrial and Applied Mathematics, 2008.

[42] Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the Traveling Salesman Problem. In *52th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2011*, page to appear.

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

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

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

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

[47] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley-Longman, 2nd edition, 2001.

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

[49] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[50] Kazuo Iwama and Takuya Nakashima. An improved exact algorithm for cubic graph TSP. In G. Lin, editor, *13th Annual International Conference on Computing and Combinatorics, COCOON 2007*, volume 4598 of *Lecture Notes in Computer Science*, pages 108–117. Springer, 2007.

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

[52] Iyad A. Kanj, Michael J. Pelsmajer, and Marcus Schaefer. Parameterized algorithms for feedback vertex set. In R. G. Downey, M. R. Fellows, and F. K. H. A. Dehne, editors, *1st International Workshop on Parameterized and Exact Computation, IWPEC 2004*, volume 3162 of *Lecture Notes in Computer Science*, pages 235–247. Springer, 2004.

[53] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *A Symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, 1972.

[54] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, 2005.

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

[56] Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *35th International Colloquium on Automata, Languages and Programming (1), ICALP 2008*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2008.

[57] Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, and W. Thomas, editors, *36th International Colloquium on Automata, Languages and Programming (1), ICALP 2009*, volume 5555 of *Lecture Notes in Computer Science*, pages 653–664. Springer, 2009.

[58] Daniel Lokshtanov, Daniel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *22st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*, 2011. Accepted for publication, to appear.

[59] Daniel Lokshtanov, Daniel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. In *22st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*, 2011. Accepted for publication, to appear.

[60] Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. FPT algorithms for connected feedback vertex set. In Md. S. Rahman and S. Fujita, editors, *4th International Workshop on Algorithms and Computation, WALCOM 2010*, volume 5942 of *Lecture Notes in Computer Science*, pages 269–280. Springer, 2010.

[61] Daniel Mölle, Stefan Richter, and Peter Rossmanith. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. *Theory of Computing Systems*, 43(2):234–253, 2008.

[62] Tobias Mömke and Ola Svensson. Approximating graphic TSP by matchings. In *52th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2011*, page to appear.

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

[64] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2006.

[65] Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for undirected feedback vertex set. In P. Bose and P. Morin, editors, *13th International Symposium on Algorithms and Computation, ISAAC 2002*, volume 2518 of *Lecture Notes in Computer Science*, pages 241–248. Springer, 2002.

[66] 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.

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

[68] Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

[69] Donald J. Rose. On simple characterizations of k-trees. *Discrete Mathematics*, 7(3-4):317–322, 1974.

[70] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

[71] Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Mixed searching and proper-path-width. *Theoretical Computer Science*, 137(2):253–268, 1995.

[72] Jan Arne Telle and Andrzej Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. In F. K. H. A. Dehne, J.-R. Sack, N. Santoro, and S. Whitesides, editors, *3th Workshop on Algorithms and Data Structures, WADS 1993*, volume 709 of *Lecture Notes in Computer Science*, pages 610–621. Springer, 1993.

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

[74] William T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 22(2):107–111, 1947.

[75] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In A. Fiat and P. Sanders, editors, *17th Annual European Symposium on Algorithms, ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009.

[76] Johan M. M. van Rooij, Jesper Nederlof, and Thomas C. van Dijk. Inclusion/exclusion meets measure and conquer. In A. Fiat and P. Sanders, editors, *17th Annual European Symposium on Algorithms, ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 554–565. Springer, 2009.

[77] Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters*, 109(6):315–318, 2009.

[78] Frank Yates. The design and analysis of factorial experiments. Technical Communication 35, Common-wealth Bureau of Soils, Harpenden, United Kingdom, 1937.

[79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In E. W. Ng, editor, *International Symposium on Symbolic and Algebraic Computation, EUROSAM 1979*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.

Appendix A

Problem definitions

(k_1, k_2)-$\oplus$CNF-SAT
Input: A CNF formula Φ consisting of m clauses of size at most k_1 on n variables, where $m \leq k_2 n$.
Question: Is the number of satisfying assignments for Φ odd?

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

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

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

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

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

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 $\deg_G(v) = \deg_T(v)$?

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

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?

FEEDBACK VERTEX SET
Input: An undirected graph $G = (V, E)$ and an integer k.
Question: Does there exist a set feedback vertex set $Y \subseteq V$ (i.e. $G[V \setminus Y]$ is a forest) of size at most k?

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?

(p_1, p_2)-$\oplus$HITTING SET
Input: A set system $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = m$,$|U| = n$, for every $S \in \mathcal{F}$, $|S| \leq p_1$ and $m \leq p_2 n$.
Question: Is the number of $X \subseteq U$, with $X \cap S \neq \emptyset$ for each $S \in \mathcal{F}$, odd?

$k \times k$ 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 $1 \leq i \leq m$?

(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)?

(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)?

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

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

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

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

(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) 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?

$k \times k$ HITTING SET
Input: A family of sets $S_1, S_2 \dots S_m \subseteq [k] \times [k]$, 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 $1 \leq i \leq m$?

(p_1, p_2)-$\oplus$SET COVER
Input: A set system $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = m$, $|U| = n$, for every $S \in \mathcal{F}$, $|S| \leq p_1$ and $m \leq p_2 n$.
Question: Is the number of $\mathcal{C} \subseteq \mathcal{F}$ with $\bigcup_{S \in \mathcal{C}} S = U$ odd?

p-$\oplus$SET COVER$_\alpha$
Input: An integer t and a set system $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = m$, $|U| = n$, $t \leq \alpha n$, for every $S \in \mathcal{F}$, $|S| \leq p$.
Question: Is the number of $\mathcal{C} \subseteq \mathcal{F}$ with $|\mathcal{C}| = t$ such that $\bigcup_{S \in \mathcal{C}} S = U$ odd?

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?

k-UNIQUE-CNF-SAT
Input: A CNF formula Φ consisting of m clauses of size at most k on n variables having at most one satisfying assignment.
Question: Is there a satisfying assignment for Φ?

WEIGHTED STEINER TREE
Input: An undirected graph $G = (V, E)$ together with a weight function $c : E \to \mathbb{Z}_+$, a set of terminals $T \subseteq V$ and an integer K
Question: Does there exist a tree in G that contains all terminals T and the sum of its edges is at most K?

Index