Space and Time Efficient Structural Improvements of Dynamic Programming Algorithms

Jesper Nederlof

Dissertation for the degree of Philosophiae Doctor (PhD)

University of Bergen
Norway

November 2011

Table of Contents

dedicated to my parents

Glossary

$\uplus$ Operator equivalent to $\vee$ except that $\texttt{true} \uplus \texttt{true}$ is not defined. 20

$\mathbf{0}$ All-zero vector/matrix. 20

$\mathbf{1}$ All-one vector/matrix. 20

$|x|$ Absolute value. 22

$\boldsymbol{A}$ Matrix. 20

$\boldsymbol{a}$ Row vector. 20

$\boldsymbol{A}^{(i)}$ The i^{th} column of $\boldsymbol{A}$. 20

a_i The entry of $\boldsymbol{a}$ at index i. 20

$\mathbb{B}$ The boolean semiring. 22

$\mathbb{C}$ Field of complex numbers. 22

$\equiv$ Congruence modulo 2 unless specified otherwise with $(\mathrm{mod}\,\cdot)$. 20

$\mathbb{D}$ Disjoint union structure: Set $\{\texttt{false}, \texttt{true}\}$ with $\uplus$ as operator. 22

Δ Symmetric difference. 19

$\circ$ Hadamard product of two matrices or vectors. 21

$\imath$ The imaginary unit. 22

$\otimes$ Kronecker product. 21

$\mathcal{M}_-$ Min-sum Semiring. 22

$\mathcal{M}_+$ Max-sum Semiring. 22

`mpl` Maximum product length of a circuit. 71

`msl` Maximum sum length of a circuit. 71

$\mathbb{N}$ Set of all positive integers. 22

$\mathcal{O}$ Suppresses constant factors. 23

$\mathcal{O}^\star$ Suppresses factors polynomial in the *input size*. 23

$\tilde{\mathcal{O}}$ Suppresses factors polylogarithmic in the *leading term of the argument*. 23

ω_m m^{th} root of unity. 22

$\mathbb{Q}$ Set of rational numbers. 22

rk rank of a matrix. 21

$\mathbb{R}$ Set of rational numbers. 22

$\mathbb{R}^+$ Set of rational numbers. 22

$\mathcal{R}^{m\times n}$ All $m \times n$ matrices with entries from $\mathcal{R}$. 22

$\mathcal{R}^{\mathcal{G}}$ All vectors indexed by $\mathcal{G}$ with entries from $\mathcal{R}$ and pairwise operation addition and multiplication in $\mathcal{R}$. 22

$\mathcal{R}[\mathcal{G}]$ Group-algebra like structure. 62

$\texttt{trun}_\ell$ Truncate after ℓ most significant bits after decimal point. 22

$\mathbb{U}$ Union structure: Set $\{\texttt{false}, \texttt{true}\}$ with $\vee$ as operator.. 22

$\mathbb{Z}$ Set of integers. 22

$\mathbb{Z}_m$ Additive group of integer mod m. 22

Part I

Introduction

Chapter 1

Introduction

Vehicle navigation, school timetabling, oil network design, risk management in nuclear power plants, train scheduling, search engines, multimedia compression, airline revenue management, weather forecast computation, DNA sequence alignment, community facility allocation, internet traffic routing, information retrieval, text typesetting, insurance forecasts, automatic pilot assistance, music information retrieval, face recognition and data encryption are just a few of the many cornerstones of modern society that crucially depend on *algorithms*.

Example 1 — Primary School Integer Multiplication Algorithm

$$
\begin{array}{r}
\ 9\,6\,2\,5\,9\,8\,7\,3\,3 \\
\times\ 9\,8\,2\,4\,5\,1\,6\,5\,3 \\
\hline
2\,8\,8\,7\,7\,9\,6\,1\,9\,9 \\
4\,8\,1\,2\,9\,9\,3\,6\,6\,5 \\
5\,7\,7\,5\,5\,9\,2\,3\,9\,8 \\
9\,6\,2\,5\,9\,8\,7\,3\,3 \\
4\,8\,1\,2\,9\,9\,3\,6\,6\,5 \\
3\,8\,5\,0\,3\,9\,4\,9\,3\,2 \\
1\,9\,2\,5\,1\,9\,7\,4\,6\,6 \\
7\,7\,0\,0\,7\,8\,9\,8\,6\,4 \\
8\,6\,6\,3\,3\,8\,8\,5\,9\,7 \\
\hline
9\,4\,5\,7\,0\,6\,7\,1\,6\,4\,1\,1\,5\,5\,5\,6\,4\,9
\end{array}
$$

An algorithm is a list of strict instructions one should follow to solve a computational task in the same way one should follow a recipe for a culinary task. Consider for example the task of multiplying large integers. The algorithm that children learn in primary school is the following: first memorize all the outcomes of every multiplication of the single-digit integers (for example $9{\cdot}7 = 63$); then how to multiply any integer with a single-digit number ($9{\cdot}286 = 18{\cdot}100+72{\cdot}10+54 = 2574$,

and yes even here we use an algorithm to add the numbers by remembering the carry); and finally how to multiply two large numbers as in Example 1.

When we analyse the algorithm of Example 1, we see that if the two integers both consist of n digits, we need about n^2 multiplications and additions of single-digit integers to compute their product. Interestingly, there are also much more efficient algorithms. The first one, published in 1963 ([KO63]), runs in $n^{\lg 3}$ time, while the current fastest ones use at most $n \cdot \log n \cdot \log\log n \cdot 2^{c \log^* n}$ additions and multiplications [DKSS08, Für09] where c is a small constant and $\log^* n$ is the number of times one has to take the logarithm of n to get below 2.

Example 2 — Integer Factorization There is a now widely used encryption system called *RSA* (named after its inventors Rivest, Shamir and Adleman [RSA78]) that relies on the hardness of the *factoring problem*: given an integer c, finding integers a, b not equal to 1 such that $a \cdot b = c$[(1)]. When stated oversimplified, b could be a message, and the following could be an encrypted message:

$c =$ 123018668453011775513049495838496272077285356959533479219732245215172640050726365751874520219978646938995647494277406384592519255732630345373154826850791702612214291346167042921431160222124047927473779408066535141959745985690214341 3.

Hence, to attack the encryption system we have to find the divisor b of the given huge integer. Utilizing hundreds of machines for 2 years, a and b in this specific case were found [KAF+10]:

$c =$ 334780716989568987860441698482126908177047949837137685689124313889828837938780022876147116525317430877378144679994893674604366679959042824463379962795263227915816 · 43430876426760322838157396665112792333734171433968102700927987363089 17

Curiously, the computer can check whether the found integers are indeed a solution to the problem asked in a fraction of a second, and even the (dedicated) reader could verify this himself using pen and paper and the algorithm for integer multiplication (although using this algorithm it will require about 50.000 single-digits additions and multiplications). How can we find divisors a, b for any integer with an algorithm like we did with the multiplication problem? Note that, if one finds a more efficient algorithm for this problem, one could circumvent many (if not almost all) security systems. The first approach would be *brute force*: if $a \cdot b = c$, then we know that the smallest of a and b is at most $\sqrt{c}$, so we try all integers $i = 2, \ldots, \sqrt{c}$ and check each time whether c/i is an integer. If so, we found our a and b. For computing c/i there is another algorithm taught at primary school called 'long division' which is about as fast as the multiplication algorithm. But, of course, $\sqrt{c}$ operations is still intractable.

(1) In this system c is the product of two primes, and hence a and b are unique.

Even more interestingly, the inverse problem (the so-called *factoring* problem, see Example 2) seems to be a lot harder. That is, suppose somebody gives us the integer $c = 945706716411555649$ from Example 1, and asks whether we can find integers a, b not equal to 1 such that $a \cdot b = c$. Modern computers will still be able to find such integers a, b in reasonable time in this specific example, but the problem quickly becomes intractable when we go to a higher values.

In Example 2 a *brute-force* algorithm is given. The perhaps most prominent subject in this thesis is the following: can we improve brute-force algorithms? For example, in Example 2, does there exists some 'intuition' such that we can make educated guesses about the divisor of the given integer c? Is there an algorithm for making such an educated guess for every large integer, or is the best algorithm just trying all candidate divisors? This a fundamental question with far-reaching consequences, but still only a humble special case of one of the big questions in theoretical computer science.

1.1 Big Questions in Theoretical Computer Science

In this section we will very informally introduce some very famous open questions in the field of theoretical computer science. Most research done in this thesis should be seen as humble contributions to 'smaller' variants of these questions, i.e., special cases or less fundamental variants.

Since, especially for the first problem discussed below, there has already been a lot of philosophical discussion about the importance and about possible answers to the problems, we will mainly state some interesting quotes from the theoretical computer science community. The connection with this thesis will be discussed in the next chapter.

P versus NP: can creativity be automated?

Informally, the class $\mathcal{P}$ ('polynomial time') is the set of all computational tasks that can be solved 'efficiently', that is, we can *find* a solution efficiently. The class $\mathcal{NP}$ ('non-deterministic polynomial time') is the set of all computational tasks where a suggested solution can be *verified* efficiently. It is known that every task in $\mathcal{P}$ is also in $\mathcal{NP}$[(2)]. The $\mathcal{P}$ versus $\mathcal{NP}$ problem is the converse: is every problem in $\mathcal{NP}$ also in $\mathcal{P}$? Without doubt, this is the most famous and appealing open question in computer science. The question was, to our knowledge, first informally asked by Gödel in a letter to Von Neumann in 1954 (this is only known since the 1990's, [Sip92]). The class $\mathcal{P}$ was introduced (implicitly) in [Cob65, Edm65], and in [Edm65] the class $\mathcal{NP}$ was also (implicitly) introduced and it was conjectured that $\mathcal{P} \neq \mathcal{NP}$. Many interesting philosophical things have been said about the $\mathcal{P}$ versus $\mathcal{NP}$ question, and we recommend several surveys

[(2)] For our oversimplified definitions this is not immediately true.

[Wig09, Imp95, For09, Coo]. All computer scientists agree that someone proving $\mathcal{P} = \mathcal{NP}$ would be an astonishing event, two examples:

> [On $\mathcal{P}$ versus $\mathcal{NP}$]*In a very strong sense, this possibility is utopia for the quest for knowledge and technological development by humans.* (..) *There would be a short program which, given detailed constraints on any engineering task, would quickly generate a design which meets the given criteria, if one exists. The design of new drugs, cheap energy, better strains of food, safer transportation, and robots that would release us from all unpleasant chores, would become a triviality.*
> — A. Wigderson [Wig09]

> *$\mathcal{P} = \mathcal{NP}$ would also have big implications in mathematics. One could find short fully logical proofs for theorems but these fully logical proofs are usually extremely long. But we can use the Occam razor principle to recognize and verify mathematical proofs as typically written in journals. We can then find proofs of theorems that have reasonably length proofs say in under 100 pages. A person who proves $\mathcal{P} = \mathcal{NP}$ would walk home from the Clay Institute not with one million-dollar check but with seven (actually six since the Poincaré Conjecture appears solved).*[(3)]
> — L. Fortnow [For09]

In [Coo71, Lev73], the notation of $\mathcal{NP}$-completeness was introduced. Informally speaking, a computational task is *$\mathcal{NP}$-complete* if the existence of an efficient algorithm would imply that $\mathcal{P} = \mathcal{NP}$. Starting with [Kar72], numerous computational problems are now known to be $\mathcal{NP}$-complete (the perhaps most basic 300 are given in [GJ79]), and mainly because of this the $\mathcal{P}$ versus $\mathcal{NP}$ question is so important:

> *A keyword search in Melvyl, the University of California's online library reveals that about 6.000 papers each year have the term '$\mathcal{NP}$-complete' on their title, abstract, or list of keywords. This is more than each of the terms 'compiler', 'database', 'expert', 'neural network' and 'operating system'. Even more surprising is the diversity of the disciplines with papers referring to $\mathcal{NP}$-completeness. They range from statistics and artificial life to automatic control and nuclear engineering.*
> — C. Papadimitriou [Pap98]

Nowadays thousands of completely different computational tasks are known to be $\mathcal{NP}$-complete. This has surprising consequences: for example, paraphrasing

[(3)] The $\mathcal{P}$ versus $\mathcal{NP}$ problem is one of the seven millenium prize problems [Coo]; a resolution of such a problem will be rewarded with one million dollar.

[Wig09], optimally packing a large set of suitcases in a big van optimally requires the same kind of creativity as (dis-)proving a mathematical statement.

Although at this moment very little progress towards the direct resolution has been made[(4)], it is often stated that the general belief is that $\mathcal{P} \neq \mathcal{NP}$. The main motivation often used is however very weak: many believe that $\mathcal{P} \neq \mathcal{NP}$ because nowadays thousands of completely different problems are known to be $\mathcal{NP}$-complete and nobody ever found an efficient algorithm for *any* of them (since that person would prove $\mathcal{P} = \mathcal{NP}$). Let us also mention another possible explanation for this general belief:

> *People have a strong sense that creativity was absolutely essential in* [discoveries like Wiles' proof of Fermat, Einstein's relativity, Darwin's evolution, etc.]*, and will be essential for important future ones. While elusive to define, people feel that creativity, ingenuity or leap-of-thought which lead to discoveries are the domain of very singular, talented and well-trained individuals, and that the process leading to discovery is anything but the churning of a prespecified procedure or recipe. These few stand in sharp contrast to the multitudes who can appreciate the discoveries after they are made.*
>
> — A. Wigderson [Wig09]

The most interesting reason in the context of this thesis is however stated in our, perhaps more down to earth, final quotation on this subject:

> *The main argument in favor of $\mathcal{P} \neq \mathcal{NP}$ is the total lack of fundamental progress in the area of exhaustive search. This is, in my opinion, a very weak argument. The space of algorithms is very large and we are only at the beginning of its exploration.*
>
> — M. Vardi [Hem02]

In Chapter 2, we will discuss the "area of exhaustive search" and related subjects, and weaker versions of the $\mathcal{P}$ versus $\mathcal{NP}$ problem. Naturally, proving $\mathcal{P} = \mathcal{NP}$ would have major consequences for the work in this thesis. On the other hand, proving $\mathcal{P} \neq \mathcal{NP}$ would give more strength to the presented work.

P versus RP: how valuable is randomness for solving computational problems?
Apart from time, another resource that is perhaps less fundamental and intuitive is *randomness*. At first sight it may be surprising that randomness helps to obtain more efficient computation:

> *Anyone who considers arithmetic methods of producing random digits is, of course, in a state of sin.*
>
> — J. von Neumann [von51]

[(4)] for some strict definition of 'direct'

But at the current point in time, it really seems that randomization helps to get *some* improvement: this could be in terms of less time or memory usage, but also in terms of providing a much easier algorithm. The reader can convince himself of the utility of randomization by reading Example 3.

Example 3 — Matrix product verification Suppose we are given three $n \times n$ matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ and we want to decide whether $\boldsymbol{AB} = \boldsymbol{C}$. One naive approach would be to compute $\boldsymbol{AB}$ and compare it with $\boldsymbol{C}$. The fastest known algorithm for multiplying two $n \times n$ matrices uses $\mathcal{O}(n^{2.371})$ operations [CW82], so this algorithm will be rather slow. Another naive approach would be to choose some $1 \leq i, j \leq n$ at random, compute $(\boldsymbol{AB})_{ij}$, and check whether it is equal to C_{ij}. However, in the worst-case, $\boldsymbol{C}$ differs only from $\boldsymbol{AB}$ in one entry and then the probability that the algorithm detects the difference is only n^{-2}. Boosting this to constant probability by repeating the procedure n^2 times results in an algorithm using $\mathcal{O}(n^3)$ operations.

There is a less natural but more efficient approach (we follow the description of [MR96, Section 7.1]). Choose a vector $\boldsymbol{r} \in \{0,1\}^n$ uniformly at random (that is, toss an unbiased coin n times and let r_i be the i^{th} outcome for $1 \leq i \leq n$); compute $\boldsymbol{A}(\boldsymbol{Br})$, and $\boldsymbol{Cr}$, return `YES` if the outcomes are equal, and `NO` otherwise. This algorithm only requires $\mathcal{O}(n^2)$ operations, and if it returns `NO`, $\boldsymbol{AB}$ and $\boldsymbol{C}$ are different. On the other hand, the probability that $\boldsymbol{x} = \boldsymbol{ABr}$ and $\boldsymbol{y} = \boldsymbol{Cr}$ are equal when $\boldsymbol{AB}$ and $\boldsymbol{C}$ are different is at most $\frac{1}{2}$. To see the latter claim, note that if $\boldsymbol{AB} \neq \boldsymbol{C}$, then $\boldsymbol{D} = \boldsymbol{AB} - \boldsymbol{C}$ is a matrix with non-zero entries. Then $\boldsymbol{x} = \boldsymbol{y}$ if and only if $\boldsymbol{Dr}$ is the all-zero vector. Since $\boldsymbol{D}$ has a non-zero entry, let D_{ij} be a non-zero entry. Then the probability that $(\boldsymbol{Dr})_i = 0$ is at most $\frac{1}{2}$ by the Principle of Deferred Decisions (see also [MR96], Section 3.5), because after all elements of $\boldsymbol{r}$ except the j^{th} element are fixed, setting r_i either to 1 or 0 will imply $d_i \neq 0$. More randomness-efficient variants of this were given in [KS93, CS93, NN93].

Informally, the $\mathcal{P}$ versus $\mathcal{RP}$[(5)] problem is the following: if an algorithm could be solved efficiently using randomization, could it also be solved efficiently deterministically? Although this problem is not resolved yet, there is a strong general opinion that the answer to this question is 'yes'. There are some 'reasonable' complexity-theoretic assumptions (informally stated, that there exists exponential time constructible functions that require exponential size circuits to compute [AB09, Theorem 20.7]) that directly imply that $\mathcal{P} = \mathcal{RP}$. On the other hand, it is known that proving $\mathcal{P} = \mathcal{RP}$ (or even giving $2^{n^{o(1)}}$ time deterministic algorithms for some problems in $\mathcal{RP}$ such as the Polynomial Identity Testing problem) will require proving some kind of circuit lower bounds, which is believed

[(5)] Usually, the slightly different $\mathcal{P}$ versus $\mathcal{BPP}$ problem is discussed.

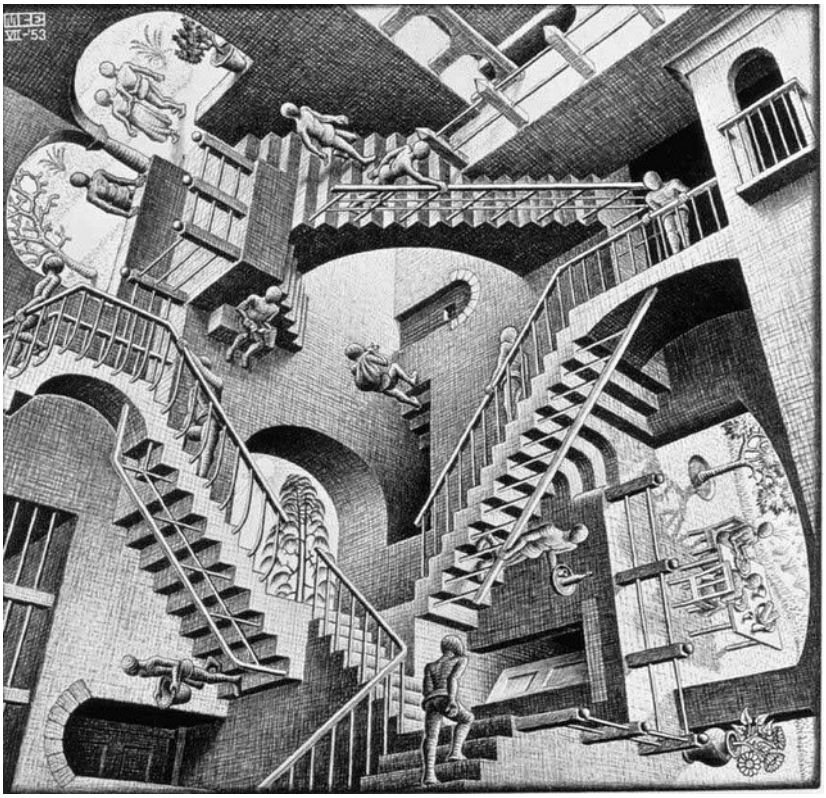

Figure 1.1 – Escher's ´Relativity', 1953.

to be a tough challenge by complexity theorists. Obviously this question is very important for theoretical computer science in general, but a resolution either way will not necessarily have direct consequences for the work presented in this thesis.

L versus NL: how valuable is memory for solving computational problems?

Suppose you are trapped in a huge building (see Figure 1.1 to get the feeling). The building consists of many rooms that all have a different number written on the floor, and in each room there are many doors numbered $1, 2, \ldots$. Of course, you want to determine whether there exists a way out, and if so, even escape. This problem would be easy if you were given a (sufficiently large) piece of paper and a pen: simply start making a map of the building, remembering unopened doors, and keep exploring all rooms until you find an exit, or until you have visited all the rooms. But suppose you are not given paper and pen. Can you ever get out of the building? And if you do not manage to find the exit, can you ever be sure that there actually exists an exit? In [AKL+79], this problem was studied. It was shown that if you are given, instead of a paper and pen, a set of *dice*, there is a way of finding the exit with constant probability (say, probability

at least $\frac{1}{2}$). The algorithm is very intuitive: every time you enter a new room, use the dice to choose a new door to enter at random. In [AKL+79] it was shown, using elementary linear algebra (see for example [AB09, Section 21.1]) that if you do this, say, n^4 times where n is the number of rooms in the building, then you will find the exit with reasonable probability no matter how the building is designed. Later, a surprising derandomization of this result was given in [Rei08], although this algorithm applies to the current scenario under some mild technical conditions.

To explain the $\mathcal{L}$ versus $\mathcal{NL}$ problem informally, we change the above scenario a little. Suppose now that in the building there are also doors that are one-way in the sense that once we go through them we cannot go back (as for example the doors at the exit of the tax-free zone in an airport). Also, we know that in any room there is a 'trapdoor' you can choose to jump in that brings you back to the starting room (this is to prevent you from ever getting stuck). Again, if you are given a (sufficiently large) piece of paper and pen, this problem is easy to solve by making a map of the building. But suppose you are not given these items, do you have a chance of finding the exit, even if you are given a set of dice? Can you ever be sure there is no exit?

Let it be clear that in the above, the 'you' represents the computer that has to solve the computational task of solving the maze, and the absence of the piece of paper represents the absence of a significant amount of working memory. Also, let it be clear that the above description of the $\mathcal{L}$ versus $\mathcal{NL}$ problem is still somewhat oversimplified. See Section 1.2 for a more detailed description.

It is not clear what the general opinion about the $\mathcal{L}$ versus $\mathcal{NL}$ question is, but it is clear that it is the most important problem concerning space-efficiency. In contrary to the $\mathcal{P}$ versus $\mathcal{NP}$ problem, little guesses have been made on its outcome. One of the few:

> *We believe that $\mathcal{NL}$ differs from $\mathcal{L}$, but not with the same depth of conviction as for the other complexity classes. Also, the resolution of the $\mathcal{L}$ versus $\mathcal{NL}$ problem may not give real insights about the higher complexity classes under consideration. Still, the separation of $\mathcal{L}$ from $\mathcal{P}$, $\mathcal{NP}$ or $\mathcal{PH}$ would be a major achievement as would be the separation of $\mathcal{P}$ from $\mathcal{PH}$ or $\mathcal{PSPACE}$, even if not rewarded by a million dollars*[3].
> — J. Hartmanis *[Har03]*

A proof of $\mathcal{L} = \mathcal{NL}$ would have major consequences for the work presented in this thesis. The main reason is that the result could be used to make basically any algorithm as space-efficient as it could possible be: a natural lower bound on the space requirement is obtained by noticing that to execute an algorithm, the computer has to memorize where in the code it is). On the other hand, a proof of $\mathcal{L} \neq \mathcal{NL}$ would only strengthen the results.

1.2 Big Questions more Detailed

The Model of Computation

Now we will revisit the questions from Section 1.1, but discuss them in a more formal way. Our definitions will only be as formal as necessary, rather following terminology from textbooks on algorithms such as [CLRS01] than from textbooks on computational complexity ([AB09, Sip97]). An *algorithm* is (as discussed in the beginning of this chapter) any step-by-step approach that accepts some data as *input*, given at some block of read-only memory and writes some data, called *the output*, to another block of write-only memory. Besides that, the algorithm is also allowed to use a block of *working memory*. Usually, we will call an algorithm A a $t(n)$-*time* $s(n)$-*space algorithm* if it is guaranteed to require at most $t(n)$ steps and $s(n)$ units of working memory for every possible number n of bits of the input (the so-called *input size*). As we will see, n does not always need to stand for the number of bits of the input. If A is a n^c-time algorithm where n is the number of bits of the input and c is some constant, then A is called a *polynomial time algorithm*. If A is a n^c-space algorithm where n is the number of bits of the input and c is some constant, then A is called a *polynomial space algorithm*.

An important nuance throughout this thesis lies in the definition of a 'step' and 'memory units' as mentioned above. For example, the definition that is often used (for example in [CLRS01]) is that a step is any basic arithmetic operations such as $\vee, \wedge, +, *, /$. However, if the arguments are very big, for example large integers or even infinite-precision complex numbers, this is not very realistic. For example in [Sha79] it is showed that the integer factoring problem from Example 2 can be solved in polynomial time if infinite precision is allowed. This model, where storing any integer in the working memory also is assumed to take a constant number of bits, is sometimes referred to as the *unit cost model* (see also [FK10]). The motivation for this model is that it is a simpler model, and that the above complication occurs very seldomly. In the alternative model, *the log-cost model*, arithmetic operations and storing data is assumed to take a constant number of steps and space only if the data is represented by a constant number of bits. Although occasionally some very simple arguments will be skipped, most results given in this thesis hold in the latter model (often exactly and occasionally up to lower-order terms).

A *computational task* (often simply called *problem*) consists of an unambiguously defined input/output relation; it is called a *decision problem* if the output consists of one bit, and an instance is called a `YES`-instance if the relation indicates that the output is 1, and a `NO`-instance otherwise. A *parameterized problem*[(6)] is a computational task together with a *complexity parameter*. If k is the complexity parameter, we will aim for running times of the type $f(k)n^c$, where $f : \mathbb{N} \to \mathbb{N}$, n

[(6)] This concept stems from the field of Fixed Parameter Tractability (Section 1.3).

is the input size and c is a constant. A *complexity class* is a set of computational classes, and they are always written in calligraphic fonts.

P versus NP

The class $\mathcal{P}$ is the class of all decision problems that can be solved in polynomial time. The class $\mathcal{NP}$ is the class of all problems of the following type: given an integer n and $f : \{0,1\}^* \to \{0,1\}$ such that evaluating f at a given y is in $\mathcal{P}$, find an $x \in \{0,1\}^n$ such that $f(x) = 1$. Note that this corresponds with the informal definition from Section 1.1. The construction we used to obtain the definition of $\mathcal{NP}$ from the definition of $\mathcal{P}$ is also called *adding non-determinism*.

A problem is called $\mathcal{NP}$*-complete* if giving an algorithm that solves it in polynomial time would imply that every problem in $\mathcal{NP}$ can be solved in polynomial time. We will now describe the most fundamental, and first known to be an, $\mathcal{NP}$-complete problem.

Definition 1.1 A *boolean circuit* C is a pair $C = (D, \lambda)$ where $D = (V, A)$ is a directed acyclic graph with maximum in-degree 2 and one sink. Elements of V are refered to as *gates*. All gates with in-degree 0 (also called *sources*) are called *input gates*. If $g \in V$ is not an input gate, then $\lambda(g) \in \{\vee, \wedge, \neg\}$. The *size* of C is defined to be $|V|$.

If C is a boolean circuit with n sources and $\boldsymbol{x} \in \{0,1\}^n$, we will slightly abuse notation by denoting a boolean variable for every gate in the following natural way: if s_i is the i^{th} source gate, $s_i = x_i$, and for other gates the value is defined recursively as the result of applying the logical operation the gate is labeled with on the values given by the in-neighbors. The output $C(x)$ is the value of the output gate.

It is worth mentioning that in Section 4.6 we will see a simple but for this thesis very important more general notion of a circuit.

CKT-Sat **Parameter:** n
Input: A Boolean circuit with n input gates.
Question: Is there $\boldsymbol{x} \in \{0,1\}^n$ such that $C(x) = 1$?

Theorem 1.1 — (Coo71, Lev73) CKT-Sat is $\mathcal{NP}$-complete.

There are several subfields of theoretical computer science that indirectly try to cope with the $\mathcal{P}$ versus $\mathcal{NP}$ problem such as approximation algorithms, average case complexity, Fixed Parameter Tractability (FPT), heuristics, lower bounds in the restricted computational models, restricted instances, but we will not further elaborate on them in this thesis. It should be mentioned that in practice, the subfield developing heuristics seems to be the most successful. We will discuss the subfield of exact complexity (and a bit of FPT), which is most relevant for this thesis, in Chapter 2.

P versus RP

The complexity class $\mathcal{RP}$ consists of all decision problems that can be solved by a polynomial time algorithm with bounded one-sided error probability, that is, if the algorithm states that the problem is a `YES`-instance it is correct, and if the problem is a `YES` instance, the algorithm will detect this with constant error probability. If an algorithm has one-sided constant error probability, the error-probability can be made very small (e.g. $\mathcal{O}(2^{-n})$) by repeating the algorithm n times and returning `YES` if and only if one of the executions resulted in a `YES`-answer. If the original algorithm uses $\mathcal{O}(k)$ random bits the new algorithm uses $\mathcal{O}(nk)$ random bits. A more randomness-efficient fast construction is possible with only $\mathcal{O}(n+k)$ random bits using expander graphs (see for example [AB09, Subsection 21.2.5]).

L versus NL

The class $\mathcal{L}$ is the class of all problems that can be solved in logarithmic space. The class $\mathcal{NL}$ is obtained from the class $\mathcal{L}$ by adding non-determinism.

The most notable recent breakthrough in the field of space bounded computation is that the following problem, that we already saw in disguised form in Section 1.1, is proven to be in $\mathcal{L}$ [Rei08].

U-Reach
Input: An undirected graph $G = (V, E)$ and vertices $s, t \in V$.
Question: Is there a path from s to t?

At first sight the difference with the following problem (which we also saw in disguised form in Section 1.1) looks rather innocent:

Short D-Reach **Parameter:** n, k
Input: A directed graph $D = (V, A)$, integer k and vertices $s, t \in V$.
Question: Is there a path from s to t of length at most k?

However, whereas U-Reach $\in \mathcal{L}$, Short D-Reach is $\mathcal{NL}$-complete meaning that proving that is in $\mathcal{L}$ would imply that $\mathcal{L} = \mathcal{NL}$. Unfortunately, very little significant[7] progress has been made on the $\mathcal{L}$ versus $\mathcal{NL}$ problem, since the most recent significant space-efficient algorithm is a rather naive (but very important) one from the 1970's:

Theorem 1.2 — (Sav70) There is an algorithm solving the Short D-Reach problem in $\mathcal{O}((2n)^{\log k})$ time and $\mathcal{O}(\lg n \cdot \lg k)$ space.

[7] For a sufficiently strict definition of 'significant', many interesting papers on the subject have been appeared, see [Kin10] for a recent overview.

Proof The proof is a direct application of the Divide&Conquer technique: if $k = 1$ the problem is trivial. If $k > 1$, there exists a path from s to t of length at most k if and only if there exists a vertex v such that there exists a path from s to v of length at most $\lfloor k/2 \rfloor$ and a path from v to t of length at most $\lceil k/2 \rceil$, so we can try all $v \in V$ and proceed independently with the two subproblems. The running time of this algorithm $T(n,k)$ satisfies the recurrence $T(n,k) = 2n \cdot T(n,k/2) = 2n^{\lg k}$ and it is easy to see that the space usage of this algorithm is bounded by $\mathcal{O}(\lg n \cdot \lg k)$.■

It is worth mentioning that with exactly the same proof, it is usually (see [Sav70, AB09, Sip97]) stated that

$$\mathcal{NSPACE}(f(n)) \subseteq \mathcal{SPACE}(f(n)^2),$$

where $\mathcal{SPACE}(f(n))$ are all problems that have $f(n)$-algorithms solving them and $\mathcal{NSPACE}$ is obtained from $\mathcal{PSPACE}$ by adding non-determinism. This second variant might look a lot more general, but the main reason that the Short D-Reach is so important is that for any algorithm, the directed configuration graph of all its possible states in fact implies a reachability problem (see for example [Sip97, Section 8.1]). Finally let us mention that it is not known whether there exists a faster space-efficient algorithm than the one from 1.2; stated in hypothesis form:

Hypothesis 1 The Short D-Reach problem cannot be solved in polynomial time and $\mathcal{O}(k \cdot \log^c n)$ space for some constant c.

1.3 Notation and Preliminaries

Sets, Sequences, Functions, Logic

We use the convention that sets are denoted by capital letters. Sometimes, sets are given as unordered lists $\{..\}$, while sequences (= row-vectors) are given as ordered lists between parentheses $(..)$. Let A, B be two sets. Then $A \times B$ denotes the Cartesian product of A and B; A^B denotes the set of all $|B|$-dimensional vectors indexed by elements from B with elements from A; we also use the shorthand A^n for denoting the $(n-1)$-fold Cartesian product of A. If C is another set, $A^{B \times C}$ denotes the set of all $|B| \times |C|$ matrices with elements from A, where the rows are indexed by elements from B and columns are indexed by elements from C; we also use the shorthand $A^{m \times n}$ for the set of $m \times n$ matrices with elements from A.

If $f : A \to B$ then A is called the *domain* of f and B is called the *codomain* of f. If $X \subseteq A$, $e \in B$, and $Y \subseteq B$, then $f(X) = \cup_{x \in X} f(x)$, $f^{-1}(e) = \{a \in X : f(a) = e\}$ and $f^{-1}(Y) = \cup_{a \in Y} f^{-1}(a)$. If $f : A \to \mathbb{N}$, and $X \subseteq A$, $f(X)$ also denotes $\sum_{e \in X} f(e)$. As usual $A \cup B$ denotes the *union* of A and B, $A \cap B$ denotes the *intersection* of A and B and $A \Delta B$ denotes the *symmetric difference* of A and B. Boolean values will be denoted by `true` or `false`, and 1 or 0, interchangeably. The usual logical

operations of disjunction, conjunction and negation are denoted respectively by $\vee, \wedge, \neg$. Additionally we will use $\uplus$ which is the operation $\{0,1\} \times \{0,1\} \to \{0,1\}$ defined by $a \uplus b = a \vee b$ if $a \wedge b \neq 1$, and undefined otherwise.

We let 2^A denote the *power set* of A (that is, the set consisting of all subsets of A). A subset of 2^A is usually called a set family and is denoted by calligraphic letters. For every $A \subseteq U$ we will occasionally interpret A as an element of $\{0,1\}^n$; hence extending the $\uplus$ operation in the natural way, it denotes the disjoint union of A and B, that is $A \uplus B$ is defined to be $A \cup B$ if $A \cap B = \varnothing$ and undefined otherwise.

We will use Iverson's bracket notation [Ive62]: for a predicate or boolean p, we let $[p]$ denote 1 if p is `true` and 0 otherwise. This notation extends naturally in the context of other algebraic structures than the integers. The congruence symbol $\equiv$, if not specified with $(\bmod m)$, denotes congruence modulo 2. In this thesis log, lg, and ln denote the logarithms with respectively base $10, 2$ and e.

Graphs

An *undirected graph* $G = (V, E)$ consists of a set V of *vertices* and a set E of unordered pairs of distinct vertices (i.e. there are no so-called 'self-loops'). Edges are interchangeably denoted by $\{u, v\}$, (u, v) and uv, where $u, v \in V$. The *(open) neighborhood* $N(v) : V \to 2^V$ is the function that gives all vertices with which v occurs in an unordered pair, the *closed neighborhood* $N[v] = N(v) \cup \{v\}$ and $|N(v)|$ is the *degree* of v.

A *directed graph* (or: *digraph*) $D = (V, A)$ consists of a set V of *vertices* and a set $A \subseteq V \times V$ of *arcs* consisting of ordered pairs of distinct vertices (i.e., there are no so-called 'self-loops'). Arcs are interchangeably denoted by (u, v) and uv. The set $N^-(v)$ of *in-neighbors* of a vertex $v \in V$ are all vertices u such that $(u, v) \in A$ and the set $N^+(v)$ of *out-neighbors* of a vertex $v \in V$ are all vertices u such that $(v, u) \in A$. The *in-degree* (or *fan-in*) and *out-degree* (or *fan-out*) of v are defined as $|N^-(v)|$ and $|N^+(v)|$, respectively. A vertex with in-degree 0 is called a *source* and a vertex with out-degree 0 is called a *sink*.

A *walk of length* k in an (un)directed graph G is a sequence $(v_1, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in E$ for every $1 \leq i < k$. A *path of length* k in an (un)directed graph G is a sequence $(v_1, \ldots, v_k)$ such that $v_i \neq v_j$ for $i < j$ and $(v_i, v_{i+1}) \in E$ for every $1 \leq i < k$. A *cycle of length* k is a path $(v_1, \ldots, v_k)$ such that $(v_k, v_1) \in E$. A (di)graph is called *acyclic* if it doesn't contain any cycles.

Matrices and Vectors

In this thesis, boldface lower-case characters (like $\boldsymbol{a}$) always denote vectors while boldface upper-case characters (like $\boldsymbol{A}$) will always denote matrices. If $\boldsymbol{a}$ is a vector, a_i denotes the element of $\boldsymbol{a}$ indexed by i. Similarly, if $\boldsymbol{A}$ is a matrix, $\boldsymbol{A}^{(i)}$ denotes the column of $\boldsymbol{A}$ indexed by i. The vectors $\mathbf{0}, \mathbf{1}$ are defined to be the all-zero and all-one vectors, respectively. The *inner-product* of two row-vectors $\boldsymbol{a}, \boldsymbol{b}$ of equal dimension is defined as $\boldsymbol{a} \cdot \boldsymbol{b}^T$ (where $\cdot$ refers to the standard

matrix multiplication). The *Hadamard product*, denoted as $\circ$ is just point-wise multiplication, that is if $\boldsymbol{A} \circ \boldsymbol{B} = \boldsymbol{C}$ then for every i, j it holds that $C_{ij} = A_{ij}B_{ij}$.

If $\boldsymbol{A}$ is an $m \times n$ matrix and $\boldsymbol{B}$ is a $p \times q$ matrix, then the *Kronecker product* $\boldsymbol{A} \otimes \boldsymbol{B}$ is the $mp \times nq$ block matrix defined as follows:

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} A_{11}\boldsymbol{B} & \cdots & A_{1n}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ A_{m1}\boldsymbol{B} & \cdots & A_{mn}\boldsymbol{B} \end{pmatrix}.$$

It is easy to see that the following, so-called *mixed product property*, holds:

$$(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{C} \otimes \boldsymbol{D}) = \boldsymbol{AC} \otimes \boldsymbol{BD} \tag{1.1}$$

The *rank of a matrix* $\boldsymbol{A}$, $\mathrm{rk}(A)$, is the maximum number of linearly independent columns (or equivalently, rows (see [Ant94, Theorem 5.6.1])) vectors of $\boldsymbol{A}$.

Theorem 1.3 — (Ant94),Theorem 5.6.8. If $\boldsymbol{A}$ is an $m \times n$ matrix, then the following are equivalent:

1. $\boldsymbol{Ax}=\boldsymbol{0}$ has only the trivial solution (that is, $\boldsymbol{x} = \boldsymbol{0}$),
2. the column vectors of $\boldsymbol{A}$ are linearly independent,
3. $\boldsymbol{Ax} = \boldsymbol{b}$ has at most one solution (none or one) for every $1 \times m$ matrix $\boldsymbol{b}$.

Moreover, if the column vectors of $\boldsymbol{A}$ are linearly independent, it can be decided whether $\boldsymbol{x}$ with $\boldsymbol{Ax} = \boldsymbol{b}$ exists and if so it can be found in polynomial time, using Gauss elimination.

Other algebraic notation

Given a function (also called *operation*) $\bullet : V \times V \to W$, we say it

1. is *closed*, if $a \bullet b \in V$ for every $a, b \in V$,
2. is *associative*, if $(a \bullet b) \bullet c = a \bullet (b \bullet c)$ for every $a, b, c \in V$,
3. has *an identity*, if there exists $1 \in V$ (called the identity) such that $a \bullet 1 = a = 1 \bullet a$ for every $a \in V$,
4. has *an inverse*, if for every $a \in V$ there exists a unique $a^{-1} \in V$ such that $a \bullet a^{-1} = 1 = a^{-1} \bullet a$.

The pair $(V, \bullet)$ is called a *group* if it satisfies all of the above properties, and a *semigroup* if it is at least closed and associative. The pair $(V, \bullet)$ is called *commutative* if for every $a, b \in V$ it holds that $a \bullet b = b \bullet a$. A commutative group is sometimes also referred to as an *Abelian group*. If we are given another operation $\oplus : V \times V \to V$, we say that $\oplus$ and $\bullet$ *distribute* if for every $a, b, c \in V$ it holds

that $a \bullet (b \oplus c) = a \bullet b \oplus a \bullet c$. The triple $(V, \oplus, \bullet)$ is called a *ring* if $(V, \oplus)$ is an Abelian group and $(V, \bullet)$ is a semigroup. A ring $(V, \oplus, \bullet)$ is called a *commutative ring* if $(V, \bullet)$ is a commutative semigroup. In a triple $(V, \oplus, \bullet)$ the identity of $\oplus$ is referred to as 0 (or, *the zero*) and the identity of $\bullet$ is referred to as 1 (*the one*), if they exist. A *field* is a commutative ring $(V, \oplus, \bullet)$ such that $(V \setminus 0, \bullet)$ is a group. The notion of a *semiring* is equivalent to the notion of a ring, except that for the first operator $\oplus$ every element must have an inverse. The first operator is usually referred as addition while the second one is referred to as multiplication. For two groups $G_1 = (V_1, \bullet_1)$, $G_2 = (V_2, \bullet_2)$, the *direct product of* G_1 *and* G_2 is the group $G_1 \oplus G_2 = (V_1 \times V_2, \bullet_3)$ where $(a_1, a_2) \bullet_3 (b_1, b_2) = (a_1 \bullet_1 b_1, a_2 \bullet_2 b_2)$ for every $a_1, b_2 \in V_1$ and $a_2, b_2 \in V_2$.

For a semiring $\mathcal{R}$ and a finite set $\mathcal{G}$, we write $\mathcal{R}^{\mathcal{G}}$ for the ring consisting of the set $\mathcal{R}^{\mathcal{G}}$ (the set of all vectors over $\mathcal{R}$ with coordinates indexed by elements of $\mathcal{G}$), equipped with coordinate-wise addition + and multiplication $\circ$ (the Hadamard product). That is, for $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{R}^{\mathcal{G}}$ and $\boldsymbol{a} + \boldsymbol{b} = \boldsymbol{d}$, $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c}$ we set $a_z b_z = c_z$ and $a_z + b_z = d_z$ for each $z \in \mathcal{R}$, where the juxtaposition and + denote addition and multiplication in $\mathcal{R}$, respectively. $\mathcal{R}^{m \times n}$ refers to all $m \times n$ matrices with elements from $\mathcal{R}$. For $\boldsymbol{v} \in \mathcal{R}^{\mathcal{G}}$ denote by $\operatorname{supp}(\boldsymbol{v}) \subseteq \mathcal{G}$ the *support* of $\boldsymbol{v}$, that is, $\operatorname{supp}(\boldsymbol{v}) = \{z \in \mathcal{G} \mid v_z \neq 0\}$, where 0 is the additive identity element of $\mathcal{R}$. A vector $\boldsymbol{v}$ is called *singleton* if $|\operatorname{supp}(\boldsymbol{v})| = 1$. We denote by $\langle w, z \rangle$ the singleton with value w on index z (that is, $\langle w, z \rangle_y = w[y = z]$ for all $y \in \mathcal{G}$).

The *natural numbers*,*real numbers*, *positive real numbers*, *rational numbers*, *integers*, and the *integers modulo* m are denoted by $\mathbb{N}$,$\mathbb{R}$,$\mathbb{R}^+\mathbb{Q}$,$\mathbb{Z}$,$\mathbb{Z}_m$ (sometimes they also refer to the appropriate algebraic structures with the natural addition and multiplication). In this thesis, $\mathbb{D}$ and $\mathbb{U}$ refer to the group-like structures consisting of the set $\{0, 1\}$ and respectively equipped with $\cup$ and $\vee$ as group operations. The *boolean semiring* $\mathbb{B}$ consists of the set $\{0, 1\}$ and $\vee$ and $\wedge$ as operators. The *min-sum semiring* $\mathcal{M}_-$ and the *max-sum semiring* $\mathcal{M}_+$ are the semirings consisting of $\mathbb{N} \cup \infty$ (respectively $\mathbb{N} \cup -\infty$) with min (respectively max) as addition operation and integer addition as multiplication operator.

The field of complex numbers is denoted by $\mathbb{C}$, and its elements are written as $x = a + bi$ where a is the real part and b is the imaginary part; the *absolute value of* x, denoted $|X|$ is defined as $\sqrt{x^2 + y^2}$. The m^{th} root of unity of the complex field is denoted by ω_m; in this thesis we will fix $\omega_m = e^{2\pi\imath/m}$ where e is the Euler constant. For a precision integer ℓ, the $\texttt{trun}_\ell(c)$ operation accepts a (infinite precision) (complex) number as argument an removes all bits after the decimal point except the ℓ most significant ones.

Partially Ordered Sets

A partial order is a binary relation $\leq$ over a set P which is reflexive, antisymmetric, and transitive, i.e., for all $a, b, c \in P$ we have that:

- $a \leq a$ (reflexivity),

- if $a \leq b$ and $b \leq a$, then $a \leq b$ (antisymmetry),
- if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity).

A partially ordered set (poset) is a pair $(P, \leq)$ where $\leq$ is a partial order. An element $c \in P$ is said to be an *upper bound* of a set $X \subseteq P$ if $x \leq c$ holds for all $x \in X$. The set X is said to have a *join* in P if there exists an upper bound $c \in P$ (called *the join*) of X such that, for all upper bounds d of X, it holds that $c \leq d$. For $X \subseteq P$, let us write $\bigvee X$ for the join of X; for the join of $X = \{a, b\}$ we write simply $a \vee b$. The *open interval* (x, y) is the poset induced by the set $\{e \in P : x < e < y\} \subseteq P$. We write $[x, y)$, $(x, y]$ and $[x, y]$ for the analogous (half-)closed interval. A *chain* is a set $C = \{c_1, \ldots, c_l\} \subseteq P$ where $c_1 < \cdots < c_l$. A chain C is *odd* if $|C|$ is odd and *even* otherwise.

Asymptotic Notation

Formally, for every integer δ and function $f : \mathbb{N}^\delta \to \mathbb{N}$, $\mathcal{O}(f(\boldsymbol{n}))$ denotes the set of all functions $g : \mathbb{N}^\delta \to \mathbb{N}$ such that there exists $c \in \mathbb{N}$ and $\boldsymbol{n}_0$ such that for every $\boldsymbol{n} \geq \boldsymbol{n}_0$ (that is, it is bigger in every coordinate): $g(\boldsymbol{n}) \leq c \cdot f(\boldsymbol{n})$. Similarly, $\tilde{\mathcal{O}}(f(\boldsymbol{n}))$ denotes the set of all g such that $g(\boldsymbol{n}) \in \mathcal{O}(f(\boldsymbol{n}) \cdot \log^c f(\boldsymbol{n}))$ for some c (that is, $\tilde{\mathcal{O}}$ suppresses terms that are polylogarithmic in *leading terms*). In the context of given problem instance of input size m, we also let $\mathcal{O}^\star(f(\boldsymbol{n}))$ denote the set $\cup_c \mathcal{O}(f(\boldsymbol{n}) \cdot m^c)$ (that is, $\mathcal{O}^\star$ suppresses term that are polynomial in the *input size*). As is common, we will abuse notation by saying that an algorithm runs in $\mathcal{O}(n)$ time, $2^{\mathcal{O}(c)}$ time or a function *is* $\mathcal{O}(n^2)$. To the author's best knowledge the definition of $\mathcal{O}^\star$ has been introduced for the first time in [Woe01a].

Complexity Classes

As discussed before, $\mathcal{P}$ is the class of all decision problems that can be solved by a deterministic algorithm in time bounded by a polynomial in the input size and $\mathcal{NP}$ are all decision problems solvable by a non-deterministic algorithm running in time bounded by a polynomial in the input size. Informally, non-determinism means that the algorithm can split itself into two copies that run independently without any further cost in time. The class consisting of all problems that can be solved in polynomial time with two-sided (respectively, one-sided) error probability at most 2/3 is denoted by $\mathcal{BPP}$ ($\mathcal{RP}$). The class $\mathcal{EXP}$ is the class of problems solvable in polynomial time and the class $\mathcal{NEXP}$ is the class of all problems solvable in non-deterministic exponential time. The class $\mathcal{P}$/poly is the class of problems solvable in polynomial time by polynomial sized non-uniform circuits, or equivalently, solvable by an algorithm that takes a polynomial amount of advice in polynomial time: there exists an advice function $a : \mathbb{N} \to \{0, 1\}^*$ such that the algorithm gets along with the usual input of n bits the advice string $a(n)$ of $\mathcal{O}^\star(1)$ length. It is worth mentioning that it is known that $\mathcal{BPP} \subseteq \mathcal{P}$/poly by Adleman's theorem [Adl78]. For more background on complexity theory and more thorough definitions we refer to [Sip97].

Parameterized Complexity

An instance of a *parameterized problem* is a pair consisting of a normal problem instance and a *parameter*, often denoted by an integer k. A parameterized problem is *Fixed Parameter Tractable* (equivalently, it is in $\mathcal{FPT}$) if there exists a function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm that solves the problem in $\mathcal{O}^\star(f(k))$ time. The class $\mathcal{W}[P]$ is the class of all parameterized problems that are in $\mathcal{FPT}$ if the following problem is in $\mathcal{FPT}$. For more information and more through definitions on this subjects we refer to the textbooks [DF99, FG06, Nie06]

Weighted Circuit Sat	**Parameter:** k
Input: A boolean circuit C on n variables and integer k.	
Question: Is there $\boldsymbol{x} \in \{0,1\}^n$ with $\mathrm{supp}(\boldsymbol{x}) = k$ and $C(\boldsymbol{x}) = \texttt{true}$?	

Treewidth

The definition of treewidth is as follows:

Definition 1.2 — Tree Decomposition, (RS84) A *tree decomposition* of a (undirected or directed) graph $G = (V, E)$ is a tree $\mathbb{T}$ in which each vertex $x \in \mathbb{T}$ has an assigned set of vertices $B_x \subseteq V$ (called a *bag*) such that $\bigcup_{x \in \mathbb{T}} B_x = V$ with the following properties:

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

Dynamic programming algorithms on tree decompositions are often presented on nice tree decompositions introduced by Kloks [Klo94]. We refer to the tree decomposition definition given by Kloks as to a *standard nice tree decomposition*.

Definition 1.3 — Standard Nice Tree Decomposition 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 1.4 — Nice Tree Decomposition A *nice tree decomposition* is a tree decomposition with one special bag z called the *root* with $B_z = \varnothing$ and in which each bag is one of the following types:

- **Leaf bag**: a leaf x of $\mathbb{T}$ with $B_x = \varnothing$.
- **Introduce vertex bag**: an internal vertex x of $\mathbb{T}$ with one child vertex y for which $B_x = B_y \cup \{v\}$ for some $v \notin B_y$. This bag is said to *introduce* v.
- **Introduce edge bag**: an internal vertex x of $\mathbb{T}$ labeled with an edge $uv \in E$ with one child bag y for which $u, v \in B_x = B_y$. This bag is said to *introduce* uv.
- **Forget bag**: an internal vertex x of $\mathbb{T}$ with one child bag y for which $B_x = B_y \setminus \{v\}$ for some $v \in B_y$. This bag is said to *forget* v.
- **Join bag**: an internal vertex x with two child vertices l and r with $B_x = B_r = B_l$.

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

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

The *width* of a tree decomposition $\mathbb{T}$ is the size of the largest bag of $\mathbb{T}$ minus one, and the *treewidth*, $\mathbf{tw}(\mathbb{T})$, of a graph G is the minimum width over all possible tree decompositions of G.

Given a tree decomposition, a standard nice tree decomposition of equal width can be found in polynomial time [Klo94] 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.

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, and it is easy to see that these contain no join bags.

Observation 1.1 For every graph it holds that its treewidth is at most its pathwidth.

Finding a path or tree decomposition of width at most t, or determining that none exists, can be done in $\mathcal{O}^\star(2^{\mathcal{O}(t^3)})$ [Bod96] (see also [DF99, Section 6.3]).

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 see [BK08, HKK05].

Chapter 2

Exact Algorithms for NP-hard Problems

Let us recall two of the possible viewpoints mentioned in Section 1.1 arguing for $\mathcal{P} \neq \mathcal{NP}$ (let us emphasize that the persons quoted do not necessarily share these viewpoints):

1. creativity cannot be automated. Creativity, ingenuity or leap-of-thought can only be achieved by very talented people and cannot be simulated on a brainless computer with a prescribed recipe (mentioned in Wigderson's quote),

2. for the last 50 years, there has been a total lack of fundamental progress in the area of exhaustive search (mentioned in Vardi's quote).

The last statement can be justified since it is known that scientists were already thinking about improving exhaustive search from the 1950's, as can be read from the following (famous) quote:

> "*It would be interesting to know, for example, what the situation is in the case of determining whether a number is a prime number, and in the case of finite combinatorial problems, how strongly* in general *the number of steps vis-à-vis the blossen Probieren*[(1)] *can be reduced.*"
> — K. Gödel, in a letter to Von Neumann from 1956 [Har89, Sip92]

And indeed, while the first question has been resolved (see [AKS04] for a deterministic algorithm), we still seem to have little clue for the second more general question.

The question 'Can exhaustive search be improved?' is arguably not much less fundamental than the $\mathcal{P}$ versus $\mathcal{NP}$ problem: if there would exist a generic

[(1)] the "simple testing", "trying out", or "exhaustive search"

recipe to ignore a very large part of the search space would this not simulate 'creativity'? If not, exactly how much must exhaustive search be improved in order to be able to conclude that 'creativity can be automated'? It seems like *any* non-trivial improvement could be called 'an astonishing event'. In hypothesis form, the question is formalised as follows:

Hypothesis 2 For every $\delta < 1$, CKT-SAT can not be solved in $\mathcal{O}^\star(2^{\delta n})$ time.

Note that proving Hypothesis 2 seems to be harder than disproving it, since it is (a lot more) stronger than the statement $\mathcal{P} \neq \mathcal{NP}$.

Exact algorithms for NP-hard problems

The CKT-SAT problem seems to be the most fundamental $\mathcal{NP}$-complete problem, but of course we do not need to restrict ourselves to this problem: for many other problems optimal solutions are often required (see also Papadimitriou's quote from Chapter 1) and it makes sense to ask for an $\mathcal{NP}$-complete problem, what the largest input size is such that we can solve all instances up to that size. From the positive perspective, it is plausible that the techniques we have to develop for these problems will be helpful to attack Hypothesis 2. From the negative perspective, we could use Hypothesis 2 and show for other problems that there is some lower bound on the computation time needed. Note that here, the $\mathcal{NP}$-completeness framework initiated by Karp [Kar72, Coo71, Lev73] only tells us that all the problems are equivalent with respect to whether they can be solved in polynomial time, so in some sense we are trying to refine this framework when concerning ourselves with strong lower bounds.

The field in theoretical computer science studying these kind of questions is the field of *exact algorithms for hard problems*. Early exponential time algorithms were already explicitly[(2)] given in the 1960's: for example an $\mathcal{O}^\star(2^n)$-time algorithm for the TRAVELING SALESMAN PROBLEM [HK62]. The field received not much attention in the next thirty years, but since then it has grown tremendously. See [vR11, Section 1.2] for a recent more detailed historical account. Several surveys [Sch05, Woe01a, Woe04, Woe08], recent PhD theses [Gas08, Mni10, vR11, Sau08, Sch11, Tra10], and a book [FK10] are devoted or highly related to the subject. Also worth mentioning is a recent breakthrough that improves an almost 50 years algorithm for HAMILTONIAN PATH [Bjö10b].

Back to Satisfiability

Although we said that not much significant progress has been made on resolving Hypothesis 2, we would like to briefly mention two results: first is a connection to circuit lower bounds:

[(2)] More implicit but earlier examples that could be mentioned are the algorithm for the SUBSET SUM problem [Bel54] from the 1950's, or even the more than two thousand years old sieve of Erosthanes that was called "brute force" in Example 2.

Theorem 2.1 — (Wil10) Suppose there is a superpolynomial function $s(n)$ such that CKT-SAT can be solved in $2^n n^{\mathcal{O}(1)}/s(n)$ time by a deterministic algorithm, then $\mathcal{NEXP} \subseteq \mathcal{P}/\text{poly}$

Thus, this even increases the stakes concerning Hypothesis 2. In this thesis we will not elaborate much more on circuit lower bounds. Another recent result more related to this thesis is the following result:

Theorem 2.2 — (PP10) If there exists a randomized algorithm that, given an instance of the CKT-SAT problem, runs in $\mathcal{O}(|C|^k)$ time and outputs

- `NO` if the given instance is a `NO`-instance,
- `YES` with probability at least $2^{-\delta n}$ for some $\delta < 1$ if the given instance is a `YES`-instance,

then there exists a $\mu < 1$ depending on δ and k such that CKT-SAT can be solved in $\mathcal{O}(2^{\mathcal{O}(n^\mu \lg^{1-\mu} m)})$ time.

The authors call an algorithm of the type required in Theorem 2.2, i.e., an algorithm running in polynomial time that has exponentially small one-sided success probability, an **OPP** algorithm. The conclusion of Theorem 2.2 seems very strong (Theorem 2.3 gives an implication), so the theorem can be interpreted as an upper bound on the power of **OPP** algorithms. Many exponential time algorithms can be easily turned into **OPP** algorithms where the error probability is the inverse of the running time of the original algorithm, most notably the Davis-Putnam procedure (see for example [FK10, Chapter 3]) and local search (see [FK10, Chapter 8] or [Sch99]), so one could interpret Theorem 2.2 as an indication that these technique on their own are not powerful enough[(3)]. For the big majority of techniques in this thesis, it seems very hard to turn it into an **OPP** technique.

The following interesting connection with Fixed Parameter Tractability is known:

Theorem 2.3 — (ADF95), see also (FG06), Theorem 3.25.

$$\text{CKT-SAT can be solved in } \mathcal{O}^\star(2^{\delta n}) \text{ for some } \delta < 1 \leftrightarrow \mathcal{FPT} = \mathcal{W}[P]$$

There is an important special case of the CKT-SAT problem, that has been well-studied and has been the basis for two well-known hypotheses that are similar to Hypothesis 2. A *k-CNF-formula over n variables* $\boldsymbol{v} = (v_1, \ldots, n_v)$ is a logical

[(3)] Indeed, a recent algorithm for a variant of the CKT-SAT uses non-**OPP** techniques [Wil11].

formula that is a conjunction of *clauses* $\phi = C_1 \wedge \ldots \wedge C_m$ where every clause is a disjunction of literals $C_i = l_{i,1} \vee l_{i2} \vee \ldots \vee l_{i,\ell}$ where $\ell \leq k$ and every literal is either v_j or $\neg v_j$ for some $1 \leq j \leq n$. The formula ϕ is said to be *satisfied* by a variable assignment $\boldsymbol{v} \in \{\texttt{true}, \texttt{false}\}^n$ if substituting it makes ϕ `true`.

k-CNF-SAT **Parameter:** n
Input: A k-CNF formula ϕ of n variables.
Question: Does there exists a $\boldsymbol{v} \in \{\texttt{true}, \texttt{false}\}^n$ that satisfies ϕ?

For convenience, let σ_k be the smallest constant such that the k-CNF-SAT problem can be solved in $\mathcal{O}^\star(2^{\sigma_k n})$ time. The following hypotheses our well-known and used as assumptions for obtaining several lower bounds. We will also use them in Chapters 7 and 8.

Exponential Time Hypothesis (ETH) (IP01). For every k, $\sigma_k(\text{CNF-SAT}) > 0$.

Strong Exponential Time Hypothesis (SETH) (IP01). $\lim_{k\to\infty} \sigma_k(\text{CNF-SAT}) = 1$.

The following lemma is also useful in proving lower bounds and was used to prove that solving many problems in $\mathcal{O}^\star(2^{o(n)})$ time is not possible under the ETH in [IP01].

Lemma 2.1 — Sparsification Lemma,(IPZ01, CIP06) There is an algorithm that accepts an integer $k \geq 2$, a real $0 < \epsilon \leq 1$, a k-CNF formula ϕ as input, and outputs a sequence $\phi_1, \ldots, \phi_s$ of k-CNF formulas in $s \cdot \mathcal{O}^\star(1)$ time such that

- $s \leq 2^{\epsilon n}$,
- $\text{sol}(\phi) = \cup_i \text{sol}(\phi_i)$, where $\text{sol}(\phi)$ is the set of satisfying assignments of ϕ,
- for every $1 \leq i \leq s$, each literal occurs at most $\mathcal{O}(\frac{k}{\epsilon})^{3k}$ times in a clause.

It is worth mentioning that again there is a known connection with Fixed Parameter Tractability:

Theorem 2.4 — See for example (FG06),Chapter 16. The ETH fails $\leftrightarrow \mathcal{FPT} = \mathcal{M}[1]$[(4)].

[(4)] For a definition of $\mathcal{M}[1]$ we refer to [FG06].

Exact Algorithms versus Fixed Parameter Tractability

Due to historical reasons, it is common to distinguish two different types of areas of study concerning exact algorithms for $\mathcal{NP}$-hard problems that are actually very similar: "Fixed Parameter Tractability (FPT)" (see also Section 1.3) and "exact exponential algorithms". The main difference is that the complexity parameter in problems in the FPT area need not be related to the input size at all (note that this is even a necessary condition to make the typical FPT question "is this problem in $\mathcal{FPT}$?" non-trivial). On the other hand, there are many problems (including all those studied in this thesis) that are known to be (often trivially) in $\mathcal{FPT}$, but where we are interested in obtaining space/time bounds of the type $\mathcal{O}^\star(f(k))$ where $f(k)$ grows as slowly as possible. Clearly these questions are in the intersection of both areas, so this thesis is about both "FPT" and "exact exponential algorithms", and results from both areas of study will appear to be useful.

The remainder of this chapter is organized as follows: in Section 2.1 we will give an introduction to the dynamic programming technique that is central in this thesis and give some example applications, and in Section 2.2 we will give an overview of the thesis and give a first brief description of what the role of dynamic programming in the thesis will exactly be.

2.1 Dynamic Programming

Dynamic programming is, together with exhaustive search, one of the most basic techniques for designing algorithms, and in particular exact algorithms for $\mathcal{NP}$-hard problems. The idea of dynamic programming is the following: to solve a given problem, we have to solve different parts of the problem (subproblems), and afterwards combine the solutions of the subproblems (subsolutions) to obtain a global solution. The naive way to do this would be to enumerate all subsolutions, try every combination and see which combine to a global solution. Dynamic programming seeks to implement this step in a more efficient way by identifying all relevant properties and treating all subsolutions with the same properties as one, by tabulating for each property whether there is a solution exhibiting it.

Hence basically, dynamic programming algorithms work with a very large table of data stored in memory, and iteratively compute table entries out of previously computed table entries. The procedure that computes new table entries from old ones is often very easy and that is why it is convenient to formalize a dynamic programming algorithm with a recurrence. Let us proceed with the perhaps easiest application of the technique to the Subset Sum problem, defined as follows:

SUBSET SUM **Parameter:** t
Input: Set $S = \{1, \ldots, n\}$ and function $\omega : S \to \mathbb{N}$ and an integer t.
Question: Does there exist a subset $X \subseteq S$ such that $\omega(X) = t$?

We first define the table entries: for $0 \leq i \leq n$ and $0 \leq j \leq t$, define $A(i,j) = \exists X \subseteq \{1, \ldots, i\} : \omega(X) = j$. It is easy to see that the following recurrence holds:

$$A(i,j) = \begin{cases} \texttt{true} & \text{if } i = 0, \quad (2.1a) \\ A(i-1,j) \vee A(i-1,j-a_i) & \text{otherwise.} \quad (2.1b) \end{cases}$$

Now, we can use (2.1) to compute $A(n,t)$ and it is easy to see that the given instance of SUBSET SUM is a YES-instance if and only if $A(n,t) = \texttt{true}$. To compute $A(i,t)$ for every $1 \leq i \leq n$ and $0 \leq j \leq t$, simply (a) create an array $A[i,j]$ (b) set $A[0,j]$ according to (2.1a) (c) for every $1 \leq i \leq n$ compute the entries $A[i,j]$ using the stored entries $A[i-1,j]$ (d) remove the entries $A[i-1,j]$ from the working memory. This gives an algorithm solving the SUBSET SUM problem in $\mathcal{O}(nt)$ time and $\mathcal{O}(t)$ space (note that this is not polynomial time since t could be exponential in the input size).

To clarify the discussion after Theorem 2.2, let us note that it is unclear how to turn this algorithm into an **OPP** algorithm: it is not known how to solve SUBSET SUM in polynomial time with one-sided error probability at most $\mathcal{O}^\star(1 - t^{-1})$.

Another easy application of dynamic programming is to the SHORT D-REACH problem defined in Chapter 1. Suppose we are given an instance consisting of a digraph $D = (V, A)$, integer $k \leq |V|$ and vertices s, t. For every $1 \leq i \leq k$ and $v \in V$, we define $A(i,v)$ to be true if and only if there is a path from s to v of length at most i.

$$A(i,v) = \begin{cases} [s = v] & \text{if } i = 1, \quad (2.2a) \\ \bigvee_{u \in N^-(v)} A(i-1,u) & \text{otherwise,} \quad (2.2b) \end{cases}$$

and this can easily be turned into an algorithm solving the problem in $\mathcal{O}(nk)$ time (note that this is not optimal since a simple breadth-first search also suffices). Of course, the SHORT D-REACH problem is not $\mathcal{NP}$-hard; indeed the above algorithm is polynomial time. It is however easily turned into an $\mathcal{NP}$-complete problem by studying *succinct variants*. Early work studying such variants is given in [PY86, GW84, FKVV98, Wag86], and they are the canonical way to define $\mathcal{NEXP}$-complete problems.

With a boolean circuit or formula C with $2n$ variables we can associate a digraph $D = (\{0,1\}^n, A)$ where $(\boldsymbol{u}, \boldsymbol{v}) \in A$ if and only if $(u_1, \ldots, u_n, v_1, \ldots, v_n)$ satisfies C. We call D the graph *represented* by C.

CIRCUIT-SUCCINCT SHORT D-REACH **Parameter:** n
Input: A boolean circuit C on $2n$ variables, integer $k \in \mathcal{O}^\star(1)$ and $\boldsymbol{s}, \boldsymbol{t} \in \{0,1\}^n$.
Question: Is there a path of length at most k in the graph represented by C?

FORMULA-SUCCINCT SHORT D-REACH **Parameter:** n
Input: A CNF-formula C on $2n$ variables, integer $k \in \mathcal{O}^\star(1)$ and $\boldsymbol{s}, \boldsymbol{t} \in \{0,1\}^n$.
Question: Is there a path of length at most k in the graph represented by C?

It would be interesting to know whether Savitch's theorem can be improved here, or to show that such an improvement would either imply an improvement of Savitch's theorem in the general case or has some complexity theoretic consequences. In other words, can we exploit the fact that the graph is given by a succinct representation? It seems that, for example, improving over Savitch's theorem by giving a faster polynomial space algorithm for the CIRCUIT-SUCCINCT SHORT D-REACH implies a faster polynomial space algorithm for a large class of dynamic programming algorithms: informally, if an exponential time dynamic programming algorithm stores only boolean values in the table and only *disjunctions* of recursive entries occur, then evaluating a table entry can straightforwardly and efficiently be formulated as an instance of CIRCUIT-SUCCINCT SHORT D-REACH.

Concerning time, there are simple reductions to show that an $\mathcal{O}^\star(2^{\delta 2n})$-time algorithm for CIRCUIT-SUCCINCT SHORT D-REACH or FORMULA-SUCCINCT SHORT D-REACH would contradict Hypothesis 2 or the SETH, respectively: obtain C' from C by adding four variables $w_1, \ldots, w_n$ and add clauses in order to make sure that $(w_1, \ldots, w_4, v_1, \ldots, v_n)$ is satisfying assignment of C' if and only if

$$(w_1, \ldots, w_4) \in \{\{\texttt{true}, \texttt{true}, \texttt{true}, \texttt{false}\} \cup \{\texttt{false}, \texttt{true}, \texttt{false}, \texttt{false}\}\} \vee$$
$$((w_1, \ldots, w_4) = \{\texttt{true}, \texttt{false}, \texttt{false}, \texttt{true}\} \wedge (v_1, \ldots, v_n) \text{ satisfies } C.)$$

As vertices $\boldsymbol{s}, \boldsymbol{t} \in \{0,1\}^n$ taking $\boldsymbol{s} = \mathbf{1}$ and $\boldsymbol{t} = \mathbf{0}$ suffices.

Dynamic Programming on Graph Decompositions

A very important and widely used application of dynamic programming is for problems on graphs that are decomposed in a specific way, for example by a tree decomposition: we now proceed by giving an example for the following problem:

INDEPENDENT SET
Input: An undirected graph $G = (V, E)$ and integer k.
Question: Is there a subset $X \subseteq V$ with $|X| = k$ and $u, v \in X \to uv \notin E$?

Theorem 2.5 — Folkore, see for example (Nie06),Theorem 10.14. There exists an algorithm that given a tree decomposition of G of width t solves Independent Set in $\mathcal{O}^\star(2^t)$ time.

Proof We first define the table entries of the dynamic programming algorithm we will give: for every bag $x \in \mathbb{T}$, $0 \leq i \leq k$ and $S \subseteq B_x$,

$$A_x(i,S) = \texttt{true} \leftrightarrow \exists X \subseteq V_x : |X| = i \wedge u,v \in X \rightarrow uv \notin E_x,$$

or less formally, $A_x(i,S)$ indicates whether G_x has an independent set of size k. The algorithm computes $A_x(i,S)$ for all bags $x \in \mathbb{T}$ in a bottom-up fashion for all values $0 \leq i \leq k$ and $S \subseteq B_x$. We now give the recurrence for $A_x(i,S)$ that is used by the dynamic programming algorithm. In order to simplify notation let v the vertex introduced and contained in an introduce bag, uv the edge introduced in an introduce edge bag, and let y,z stand for the left and right child of x in $\mathbb{T}$ if present.

- **Leaf bag**:

  $$A_x(i,\varnothing) = [i = 0]$$

- **Introduce vertex bag**:

  $$A_x(i,S) = \Big(v \in S \wedge A_y(i-1, S \setminus v)\Big) \vee \Big(v \notin S \wedge A_y(i,S)\Big)$$

  If $v \in S$, it contributes with 1 to the size of the independent set in G_y, otherwise it does not.

- **Introduce edge bag**:

  $$A_x(i,S) = (u \notin S \vee v \notin S) \wedge A_y(i,S)$$

  Here we check whether the introduced edge violates the independent set.

- **Forget bag**:

  $$A_x(i,S) = A_y(i, S \cup \{v\}) \vee A_y(i, S \cup \{v\})$$

  In the child bag the vertex v can either be in the independent set or not.

- **Join bag**:

$$A_x(i,S) = \bigvee_{j=0}^{i+|S|} A_y(j,S) \wedge A_z(i+|S|-j,S)$$

The two independent sets of G_y and G_z together form an independent set of size i of G_x if they are consistent in the overlapping vertex set S, and their sizes sum up to i, taking care that vertices in S are accounted twice, and every independent set of G_x can be decomposed accordingly.

It is easy to see that using the above recurrences, $A_x(k,\varnothing)$ can be computed in $\mathcal{O}^\star(2^k)$ time, and it is follows by definition that $A_x(k,\varnothing) = \texttt{true}$ if and only if the given instance is a YES-instance. ■

Recently, it was shown that in some sense Theorem 2.5 is optimal unless the SETH fails:

Theorem 2.6 — (LMS11a) Unless the SETH fails, there is no algorithm that given a tree decomposition of G of width t solves Independent Set in $\mathcal{O}^\star(2^{\delta t})$ time for some $\delta < 1$.

It is conceivable[5] that the above algorithm can be related as well to the Formula-Succinct Short D-Reach concerning both the time and space requirements using constructions similar to the proof of Theorem 2.6.

Open Question 1 Can Circuit-Succinct Short D-Reach or Formula-Succinct Short D-Reach be solved in $\mathcal{O}^\star(1)$ space and $\mathcal{O}^\star(2^{cn})$ time for some constant c? Or would it have any complexity-theoretic consequences?

More Information on DP

There are many crucial applications of the dynamic programming technique to theoretical computer science, we refer to [CLRS01, Chapter 15] for a few of the most basic ones. General studies and classifications of dynamic programming algorithms were initiated already several times. For example in the field of parallel algorithms [KGGK94], monadic / polyadic and series/parallel types are distinguished [KGGK94], in the field of approximation algorithms generic results have been obtained [Woe01b] (also, in [GKM+11] a very recent generic approximative counting algorithm has been given), and models of dynamic programming algorithm are considered in [BODI11, Hel89, KH67]. A generic result using dynamic programming for graphs with given efficient tree decompositions is Courcelle's Theorem [DF99, Section 6.5]. Logarithmic space versions of Theorem 2.5 where t is constant were recently obtained in [EJT10].

Also, we refer to [Dre02] for an historical account (including the origin of the name), and [Bel54] for a reprint of the famous book of Bellman.

[5] the author does not have a full proof

2.2 On this Thesis

In this section we will describe the contents of this thesis. Recall the name:

"Space and Time Efficient Structural Improvements of Dynamic Programming Algorithms"

In this thesis, we search for *generic improvements of naive dynamic programming* in the same way as the search for *generic improvements of exhaustive search* as discussed in the beginning of this chapter. We believe that these two questions are somehow dual: exhaustive search is very insensitive to the number (and the complexity) of the constraints imposed on the variables, while dynamic programming is very insensitive to the number of variables[(6)] (or: number of candidate solutions).

Since dynamic programming is very space-inefficient by definition, improvements could refer to saving time or/and saving space. Theorem 2.5 and the discussion of Circuit-Succinct Short D-Reach and Formula-Succinct Short D-Reach already suggest that improving some naive dynamic programming schemes in time is as hard as improving exhaustive search, but could we still get some generic positive results?

The remainder of this thesis is organised as follows: in part II, Chapter 3, we give some space-efficient algorithms using the principle of inclusion/exclusion (or Möbius inversion). Although the results are non-trivial, this should be considered as a warm-up since we study a more general framework in Chapter 5.

In Chapter 4 we recall preliminary knowledge and introduce notation that facilitates reading Part III. We recall a number of known transformations and discuss two techniques similar to the ones appearing in Part III. More importantly we recall the notion of an algebraic circuit that will be used to describe classes of dynamic programming algorithms that will be studied in Part III.

In Chapter 5 we give a class of dynamic programming algorithms that can be turned into space-efficient ones without increasing the running time significantly. This will be based on two transformations. As mentioned earlier, the results in (the second half of) that chapter generalize (almost all) the results of Chapter 3. Oversimplified, the algorithms are obtained by implicitly applying an invertible transformation to the dynamic programming table.

Chapter 6 continues the study initiated in Chapter 5: in order to save space and time, we first implicitly apply a hash function (i.e. non-invertible transformation) to the dynamic programming table and then follow the approach of Chapter 5. We combine this with exhaustive search to give an algorithm for the Linear Sat problem. Also, in a technical result, we propose a new hash function that could useful for further improvements of dynamic programming or exhaustive search algorithms.

[(6)] A more concrete setting of this duality is studied in Subsections 8.2.2 and 9.1.

In Chapter 7 we study the class of dynamic programming algorithms on graphs of small treewidth for connectivity problems. The main contribution is a hash function mapping the number of solutions to the number of so-called 'cut objects' which is used to improve the time and space requirements of known algorithms. We also show that for some connectivity problems such an improvement is harder to obtain, and that it is hard to improve our new algorithms (where hard refers to breaking a complexity theoretical assumption).

In Chapter 8 we study the relation between dynamic programming algorithms and branching algorithms from the hardness perspective. Our main aim is to show that improving 'naive' dynamic programming based algorithms for problems as Steiner Tree and Set Cover implies improving exhaustive k-CNF-Sat. We achieve this aim partially by showing such a result for the parity versions of Steiner Tree and Set Cover. We also relate possible improvements of exhaustive search algorithms for several problems to each other.

In Part IV, Chapter 9, we give a conclusion, relate the chapters to each other, and give more directions for further research. Moreover, we discuss one lemma from the paper [NvR10] concerning symmetry.

The thesis is based on (parts of) the following papers:

[Ned09]. Jesper Nederlof. Fast polynomial-space algorithms using Möbius inversion: Improving on Steiner tree and related problems. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 713–725. Springer, 2009

[LN10]. Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In Leonard J. Schulman, editor, *STOC*, pages 321–330. ACM, 2010

[KKN11]. Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. On homomorphic hashing for coefficient extraction. 2011. Manuscript

[CNP+11a]. 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 *FOCS*, 2011. To appear. Full version: [CNP+11b]

[CDL+11]. Marek Cygan, Holger Dell, Daniel Lokstahnov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-Sat. 2011. Manuscript

[NvR10]. Jesper Nederlof and Johan M. M. van Rooij. Inclusion/exclusion branching for partial dominating set and set splitting. In Venkatesh Raman and Saket Saurabh, editors, *IPEC*, volume 6478 of *Lecture Notes in Computer Science*, pages 204–215. Springer, 2010

Part II

Prologue

Chapter 3

Fast Polynomial-space Algorithms using Möbius Inversion: Improving on Steiner Tree and Related Problems

This chapter is based on a part of the following paper:

[Ned09]. Jesper Nederlof. Fast polynomial-space algorithms using Möbius inversion: Improving on Steiner tree and related problems. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 713–725. Springer, 2009

In 2006, Björklund et al. drew new attention to the well-known principle of inclusion-exclusion (see [BHK09] for the journal version): they gave $\mathcal{O}^*(2^n)$-time algorithms for several set partition problems, the most prominent one being k-Coloring. They also mention a simple adjustment to their algorithm to achieve an $\mathcal{O}^*(2.24^n)$-time algorithm with polynomial space for k-Coloring. Also related to this are the older $\mathcal{O}^*(2^n)$-time polynomial-space algorithms for #Hamiltonian path by Karp [Kar82] and (implicitly) Kohn et al. [KGK77], and for #Perfect Matching by Björklund and Husfeldt [BH08].

Our algorithms presented in this chapter heavily rely on the work of Björklund et al. [BH08, BHKK07, BHKK08, BHKK10b, BHK09]. The results can be read from Table 3.1.

Steiner Tree is one of the most well-studied $\mathcal{NP}$-complete problems. For this problem, the Dreyfus-Wagner [DW72] dynamic programming algorithm (see also Section 5.2.2) has been the fastest exact algorithm for over 30 years. However, recently Björklund et al. [BHKK07] gave an $\mathcal{O}^*(2^k)$-time algorithm for the variant

Problem	Poly-space	By	Exp-space	By	Results
Steiner Tree with unit weights	$5.96^k n^{\mathcal{O}(\log k)}$ $\mathcal{O}^\star(1.60^n)$	[FGK08] [FGK08]	$\mathcal{O}^\star(2^k)$ $\mathcal{O}^\star(1.36^n)$	[BHKK07] [FGK08]	$\mathcal{O}^\star(2^k)$ $\mathcal{O}^\star(1.36^n)$
Steiner Tree	$5.96^k n^{\mathcal{O}(\log k)}$	[FGK08]	$\mathcal{O}^\star((2+\epsilon)^k)$ $\mathcal{O}^\star(2^k C)$	[FKM+07] [BHKK07]	$\mathcal{O}^\star(2^k C)^\dagger$
Degree Constrained Spanning Tree			$\mathcal{O}^\star(5.92^n)$	[AFS09]	$\mathcal{O}^\star(2^n)$
Maximum Internal Spanning Tree			$\mathcal{O}^\star(3^n)$	[FGR09]	$\mathcal{O}^\star(2^n)$
#Spanning Forests			$\mathcal{O}^\star(2^n)$	[BHKK08]	$\mathcal{O}^\star(2^n)$
Cover Polynomial	$\mathcal{O}^\star(3^n)$	[BHKK08]	$\mathcal{O}^\star(2^n)$	[BHKK08]	$\mathcal{O}^\star(2^n)$
Convex Tree Coloring			$\mathcal{O}^\star(2^k C)$	[PHN08]	$\mathcal{O}^\star(2^k C)^\dagger$
#Perfect Matching	$\mathcal{O}^\star(2^n)$[(1)]	[BH08]	$\mathcal{O}^\star(1.62^n)$[(1)]	[Koi09]	$\mathcal{O}^\star(1.95^n)$

Table 3.1 – The results of this work compared to the relevant previous results. The number of vertices on the input graph is denoted by n. The running times of the algorithms of this work are given in the last column. These new algorithms use polynomial space, except the ones indicated with the †, which use $\mathcal{O}^\star(C)$ space with C being the maximum weight involved.

with bounded integer weights with k terminals, and Fuchs et al. [FKM+07] gave an $\mathcal{O}^\star(c^k)$-time algorithm for the general case, for any $c > 2$. Both algorithms use $\Omega(2^k)$ space. In 2008, Fomin et al. [FGK08] initiated the study of polynomial space algorithms for Steiner Tree. They gave polynomial space algorithms with running times bounded by $5.96^k n^{\mathcal{O}(\log k)}$ and $\mathcal{O}^\star(1.60^n)$ where n is the number of vertices in the graph. They pose the question whether Steiner Tree is fixed parameter tractable with respect to k when there is a polynomial space restriction. We answer this question affirmatively by providing an algorithm that runs in $\mathcal{O}^\star(2^k)$ time and meets the restriction. Using the techniques of [FGK08], this also leads to a polynomial-space $\mathcal{O}^\star(1.36^n)$-time algorithm.

The Max Internal Spanning Tree (MIST) and Degree Constrained Spanning Tree (DCST (also called Min-Max Degree Spanning Tree) problems are natural generalizations of Hamiltonian path. In [FGR09], Fernau

[(1)] Recently Björklund improved this by giving a polynomial space $\mathcal{O}^\star(1.41^n)$ time algorithm [Bjö11].

et al. ask if there exists an $\mathcal{O}^\star(2^n)$-time algorithm to solve MIST. In [GSS08], Gaspers et al. ask if there exists an $\mathcal{O}^\star(2^n)$-time algorithm solving DCST. We answer both questions by giving polynomial-space algorithms with this running time.

The Cover polynomial of a directed graph, introduced by Graham and Chung [CG95], generalizes all problems that can be solved using two operations named deletion and contraction of edges, and is designed to be the directed analogue of the Tutte polynomial. We improve the $\mathcal{O}^\star(3^n)$-time polynomial-space algorithm of Björklund et al. [BHKK08] to an $\mathcal{O}^\star(2^n)$-time polynomial-space algorithm. We also give the same improvement for #Spanning forests, which is one particular case of the Tutte polynomial. For more information about the Tutte polynomial we refer to [BHKK08].

Finally, we give two new algorithms for Convex Tree Coloring and #Perfect matching (counting the number of perfect matchings of a graph). (A special case of) the Convex Tree Coloring problem was studied in [PHN08], where a $\mathcal{O}^\star(2^k C)$ time and $\mathcal{O}^\star(C)$ space algorithm was given. We will continue this study by emphasizing the space usage aspect. Finally, we show that #Perfect matching can be solved in $\mathcal{O}^\star(1.95^n)$ time and polynomial space, improving over the previous $\mathcal{O}^\star(2^n)$ time polynomial space algorithm of Björklund and Husfeldt [BH08][(1)]. It is worth to mention that with exponential space, one can count perfect matchings of general graphs even in $\mathcal{O}^\star(1.62^n)$ time due to Koivisto [Koi09].

This chapter is organized as follows: we first recall the principle of Inclusion-Exclusion and the well-known Hamiltonian path algorithm in Section 3.1. After this we provide in Section 3.2 a natural extension by introducing the concept of *branching walks* and give the resulting algorithms. In the remaining sections we prove the remaining results that are not primarily based on branching walks.

Preliminary Definitions

Recall that a *walk of length* k in G is a sequence $W = (v_1, \ldots, v_{k+1}) \in V^{k+1}$ such that $(v_i, v_{i+1}) \in E$ for each $0 < i \leq k$. We say W is *from* v if $v_1 = v$, and W is *cyclic* if $v_1 = v_{k+1}$. Furthermore, $v \in V$ is said to be *visited* by W if $v_i = v$ for some $1 \leq i \leq k$. For a rooted tree T we will use $\mathtt{rt}(T)$ to denote the root of T. For a vertex $v \in V(T)$ we will also use $\mathtt{pa}(v)$ to denote the parent of v. In this chapter $[k]$ denotes $\{1, \ldots, k\}$ for an integer k.

Also given a graph $G_1 = (V_1, E_1)$, a *homomorphism from* G_1 *to* G is a function $\varphi : V_1 \to V$ such that $(u, v) \in E_1$ implies $(\varphi(u), \varphi(v)) \in E$. Note that φ directly corresponds to a walk in the case that G_1 is a path. We will use the notation $\varphi(X) = \{\varphi(u) \mid u \in X\}$ and $\varphi(Y) = \{(\varphi(u), \varphi(v)) \mid (u, v) \in Y\}$ for $X \subseteq V_1$ and $Y \subseteq E_1$. If $G_2 = (V_2, E_2)$ is a third graph with $V_1 \cap V_2 = \varnothing$ and $\varphi_1 : V_1 \to V$ and $\varphi_2 : V_2 \to V$ are homomorphisms, we will use $\varphi_1 \cup \varphi_2$ to denote the homomorphism $V_1 \cup V_2 \to V$ defined by $\varphi(v) = \varphi_1(v)$ and $\varphi(w) = \varphi_2(w)$ with $v \in V_1$ and $w \in V_2$.

Model

We assume integers in the input are given in binary, and we will prove our results for the unit-cost computation model where any arithmetic operation and storing any integer is assumed to take constant time and space. However, all our results also hold in the (more realistic) log-cost computation model where arithmetic operations and storing integers only take constant time and space if they are constant-sized. Also see Section 1.2 for more discussion on this issue.

3.1 Inclusion-Exclusion

Here we will give a brief introduction to inclusion-exclusion, following [Ned08]. In a nutshell, inclusion-exclusion is a way of computing a quantity through expressing it as a number of intersections. This can be useful since in many settings it is easier to reason about these intersection instead.

Four Sets in a Venn-Diagram

Suppose we are given 5 requirements $A_1, \ldots, A_5$ of a set U which are illustrated in Venn-diagram Figure 3.1. For notational ease, assume $A_i = \varnothing$ for $i < 1$ and $i > 5$. We want to compute $|\bigcap_i A_i|$, without using any union operator.
First, in (a) we sum over all sets, counting elements of frequency i exactly i times. To compensate, we subtract the size of the intersection between each pair of sets in (b), subtracting elements of frequency i exactly $\frac{i*(i-1)}{2}$ times. After adding all intersections of 3 sets and subtracting elements in 4 sets, every element in one of the sets is exactly counted once. Concluding, we obtain the following identity:

$$|\bigcup_i A_i| = \sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \sum_{i<j<k<l} |A_i \cap A_j \cap A_k \cap A_l|.$$

The Theorem

Theorem 3.1 — Folklore Let U and V be sets and for every $v \in V$, let P_v be a subset of V. Use $\overline{P_v}$ to denote $U \setminus P_v$. With the convention $\cap_{i \in \varnothing} \overline{P_i} = U$, we have:

$$|\bigcap_{v \in V} P_v| = \sum_{F \subseteq V} (-1)^{|F|} |\bigcap_{v \in F} \overline{P_v}|. \tag{3.1}$$

Proof For $R \uplus F \uplus O = V$, define

$$N(R, F, O) = \left|\left(\bigcap_{v \in R} P_v\right) \cap \left(\bigcap_{i \in F} \overline{P_v}\right)\right|.$$

(intuitively, R, F, O contain all the indices of respectively all 'required', 'forbidden' and 'optional' subsets). Then De Morgan's law states that for every $v \in V$,

$$N(R \cup \{v\}, F, O) = N(R \setminus \{v\}, F, O \cup \{v\}) - N(R \setminus \{v\}, F \cup \{v\}, O). \tag{3.2}$$

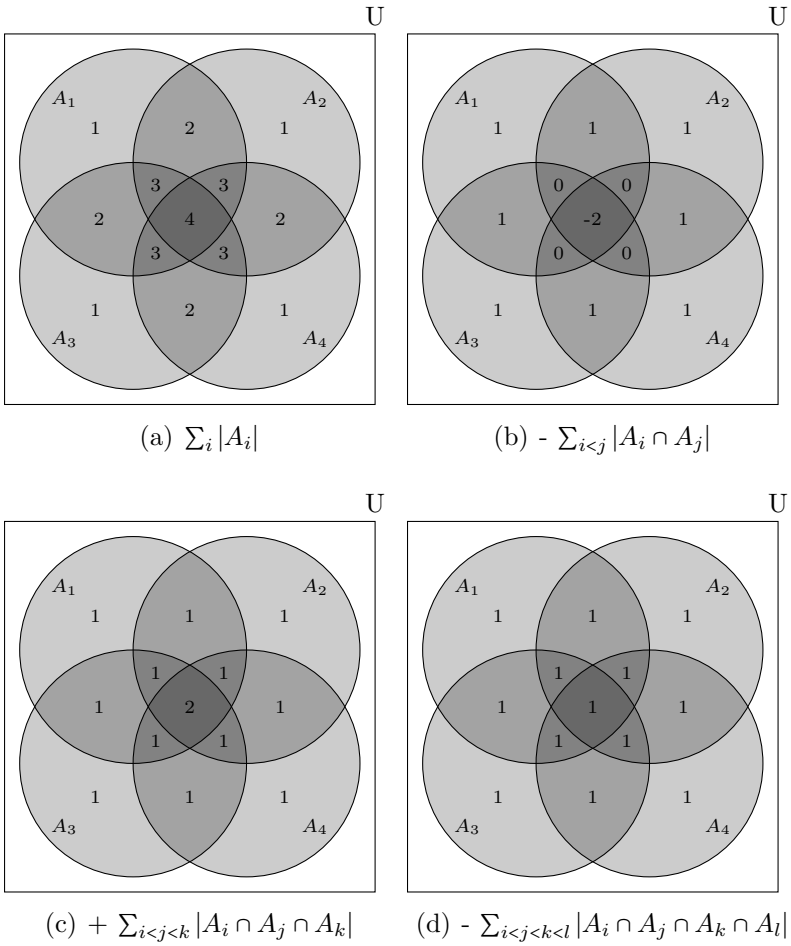

Figure 3.1 – An example of an inclusion-exclusion formula

Iteratively applying this equation gives $N(R, \varnothing, \varnothing) = \sum_{F \subseteq V} (-1)^{|F|} N(\varnothing, F, R \setminus F)$ which is easily seen to be equivalent to (3.1). ■

In this chapter, we call any application of the above theorem an *Inclusion-Exclusion-formulation* (IE-formulation). In the context of this chapter it is convenient to use the following terminology: We refer to the set U as the *universe*, to R as the *requirement space* and to P_v as a *property*. Moreover, given a set $F \subseteq R$, we call the task of computing $|\cap_{v \in F} \overline{P_v}|$, the *simplified problem*. Note that if the simplified problem can be solved in $\mathcal{O}^\star(t(n))$ time and $\mathcal{O}^\star(s(n))$ space, the left-hand side of (3.1) can be determined in $\mathcal{O}^\star(2^n t(n))$-time and $\mathcal{O}^\star(s(n))$ space in the straightforward manner. All algorithms in this chapter will exploit this observation.

Application to the Hamiltonian Path Problem

To illustrate Theorem 3.1, we will first recall the following IE-formulation due to Karp [Kar82] for counting Hamiltonian paths. Given a (possibly directed) graph $G = (V, E)$, a Hamiltonian path is a walk that contains each vertex of G exactly once[(2)].

Theorem 3.2 — (Kar82) Hamiltonian paths of a graph on n vertices can be counted in $\mathcal{O}^\star(2^n)$ time and polynomial space.

Proof Let $G = (V, E)$. In the context of Theorem 3.1, define the universe U as all walks of length $n-1$ in G, the requirement space $R = V$, and P_v as all walks of length $n-1$ that visit vertex v, for every $v \in V$. With these definitions, the left-hand side of (3.1), $|\bigcap_{v\in V} P_v|$, is the number of Hamiltonian paths in G. Now it remains to show how to solve the simplified problem. Given $F \subseteq V$ and $x \in V \setminus F$, let $w_F(x,k)$ be the number of walks from x of length k in $G[V \setminus F]$. Then $w_F(x,k)$ admits the following recurrence:

$$w_F(x,k) = \begin{cases} 1 & \text{if } k = 0, \\ \sum_{t\in N(x)\setminus F} w_F(t,k-1) & \text{otherwise.} \end{cases} \tag{3.3}$$

Also, notice that

$$|\bigcap_{v\in F} \overline{P_v}| = \sum_{s\in V\setminus F} w_F(s, n-1),$$

and hence the simplified problem can be solved in polynomial time using dynamic programming on (3.3) (the parameter F is fixed but is added for clarity). Thus it takes $\mathcal{O}^\star(2^n)$ time and polynomial space to evaluate (3.1), and the theorem follows. ■

3.2 Branching Walks

Definition 3.1 A *branching walk* B in $G = (V, E)$ is a pair (T, φ) where T is an ordered rooted tree and $\varphi : V(T) \to V$ is a homomorphism from T to G. For a vertex $x \in V$, B *is from* x if $\varphi(\mathtt{rt}(T)) = x$. B *visits* a vertex $v \in V$ if $v \in \varphi(V(T))$. The *length* of B is $|E(T)|$.

A branching walk is a natural generalization of a walk: Notice that a branching walk (T, φ) is a walk in the special case that T is a path rooted at an endpoint. Since we will count distinct branching walks, we emphasize that two branching walks (T_1, φ_1) and (T_2, φ_2) are distinct if there is no isomorphism $\psi : T_1 \to T_2$ such that $\varphi_1(v) = \varphi_2(\psi(v))$ for every $v \in V(T_1)$.

[(2)] Note that this is slightly different from the usual definition, since a path corresponds to two walks in both directions. So we will actually obtain a number that is exactly twice the number of Hamiltonian paths.

3.2.1 Steiner Tree

In this section we will give an extension of the technique in the previous section to obtain a new IE-formulation for the Steiner Tree problem, which is defined as follows:

Steiner Tree	**Parameter:** $k = \lvert K\rvert$
Input: $G = (V, E)$, $c \in \mathbb{Z}^+$, weight function $\omega : E \to [C] \setminus 0$ and terminals $K \subseteq V$	
Question: Is there a subtree (V', E') of G such that $K \subseteq V'$ and $\sum_{e \in E'} \omega(e) \leq C$?	

Given a branching walk (T, φ) in the input graph G, the quantity $\sum_{e \in E(T)} \omega(\varphi(e))$ is said to be its *weight*.

Lemma 3.1 Let $s_0 \in K$. There exists a subtree $S = (V', E')$ of G such that $K \subseteq V'$ and $w(E') \leq C$ if and only if there exists a branching walk $B = (T, \varphi)$ from s_0 of weight at most C such that $K \subseteq \varphi(V(T))$.

Proof For the forward direction, assume S to be ordered by fixing an arbitrary order. Then define $B = (S, \varphi)$, with $\varphi : V' \to V'$ the identity function, and let $\texttt{rt}(S) = s_0$ (this is possible since $s_0 \in K \subseteq V(S)$). Then, clearly $\omega(\varphi(E(S))) \leq c$. For the backward direction, notice that $(\varphi(V(T)), \varphi(E(T)))$ is connected and if we let S be a spanning tree of $(\varphi(V(T)), \varphi(E(T)))$, it has the required properties. ■

Consider the following IE-formulation: let $s_0 \in K$ be an arbitrarily chosen terminal, and define the universe U to be the set of all branching walks (T, φ) from s_0 of weight at most C. Let the requirement space R be K. For every $v \in K$, define a property $P_v \subseteq U$ that consists of all branching walks in U that visit v and have weight at most C. It follows that the left-hand side of (3.1), $\lvert\bigcap_{v \in K} P_v\rvert$, is the number of branching walks of weight at most C that contain all terminals. By Lemma 3.1, this quantity is larger than 0 if and only if the instance of Steiner Tree is a `yes`-instance. Hence we can restrict our goal to determining $\lvert\bigcap_{v \in K} P_v\rvert$. For this we use Theorem 3.1.

Before we proceed, first let us recall a basic combinatorial problem: Let $\mathcal{T}_n$ be the set of all distinct ordered rooted trees on n edges (also called the *Catalan numbers*). We will give a recurrence for $\lvert\mathcal{T}_n\rvert$. Let

$$\gamma : \bigcup_{i,j} (\mathcal{T}_i \times \mathcal{T}_j) \leftrightarrow \mathcal{T}_{i+j+1} \tag{3.4}$$

be the gluing operation such that $T = \gamma(T_1, T_2)$ is obtained by connecting $\texttt{rt}(T_1)$ and $\texttt{rt}(T_2)$, setting $\texttt{rt}(T) = \texttt{rt}(T_2)$, letting the first child of $\texttt{rt}(T)$ be $\texttt{rt}(T_1)$, being followed by the children of $\texttt{rt}(T)$ in T_2. Clearly γ is a bijection so $\lvert\mathcal{T}_n\rvert = \sum_{i=0}^{n-i-1} \lvert\mathcal{T}_i\rvert\lvert\mathcal{T}_{n-i-1}\rvert$.

We now continue applying Theorem 3.1 by showing how the simplified problem can be solved in $\mathcal{O}^\star(C)$ time and space. For $F \subseteq K$, define $\mathcal{B}_F(x, W)$ to be all branching walks (T, φ) of weight at most W from x in $G[V \setminus F]$, where $x \in V \setminus F$. Hence $U = \mathcal{B}_\emptyset(x, C)$. Let $b_F(x, W) = |\mathcal{B}_F(x, W)|$. First, note that the simplified problem is to compute

$$|\bigcap_{v \in F} \overline{P_v}| = b_F(s_0, C)$$

for a given set $F \subseteq K$ of terminals. Now $b_F(s_0, C)$ can be computed using the following lemma in combination with dynamic programming:

Lemma 3.2 Let $F \subseteq K$ and $x \in V \setminus F$, then

$$b_F(x, W) = \begin{cases} 1 & \text{if } W = 0, \quad (3.5a) \\ \sum_{t \in N(x) \setminus F} \sum_{W_1 + W_2 = W - \omega(x,t)} b_F(t, W_1)\, b_F(x, W_2) & \text{otherwise.} \quad (3.5b) \end{cases}$$

Proof There is one branching walk B of weight 0, $B = (T, \varphi)$, from x with T being a single vertex and φ mapping this single vertex to x, hence Case 3.5a. For the second case, define a function

$$\gamma' : \bigcup_{t \in N(x) \setminus F} \bigcup_{W_1, W_2} (\mathcal{B}_F(t, W_1) \times \mathcal{B}_F(x, W_2)) \leftrightarrow \mathcal{B}_F(x, W_1 + W_2 + \omega(x, t))$$

by $\gamma'((T_1, \varphi_1), (T_2, \varphi_2)) = (\gamma(T_1, T_2), \varphi_1 \cup \varphi_2)$. Now γ' is a bijection since γ is a bijection as stated in (3.4) and for the branching walk $(\gamma(T_1, T_2), \varphi)$ it must hold that $\varphi(\mathtt{rt}(T_1)) \in N(\varphi(\mathtt{rt}(T_2)))$. Hence Case 3.5b follows. ■

Theorem 3.3 Steiner tree can be solved in $\mathcal{O}^\star(2^k C)$ time and $\mathcal{O}^\star(C)$ space.

Proof Due to Lemma 3.1 the considered IE-formulation solves Steiner Tree, and we can use dynamic programming on (3.5b) to compute the simplified problem in $\mathcal{O}^\star(C^2)$ time. This can be further reduced to $\mathcal{O}^\star(C)$ time in a standard manner with the Fast Fourier Transform. Then the theorem follows from Theorem 3.1. ■

The following result is a consequence of Theorem 3.3 and the considerations of Section 4.2 in [FGK08]:

Theorem 3.4 The Steiner Tree with unit weights problem can be solved in $\mathcal{O}^\star(1.36^n)$ time using polynomial space.

Proof Modify Algorithm `steiner` as described in Section 4.2 in [FGK08], except that we replace Step 4 of `steiner` with the algorithm due to Theorem 3.3. This clearly does not change the worst-case running time as claimed in Theorem 5 of [FGK08], and it is easy to see that the new algorithm uses polynomial space. ■

3.2.2 Degree Constrained Spanning Tree

DEGREE CONSTRAINED SPANNING TREE (DCST) **Parameter:** n
Input: $G = (V, E)$, $1 \leq c \leq n$
Question: Is there a spanning tree of G with maximum degree at most c?

Analoguous to Lemma 3.1, we will use the following lemma to reduce our problem to computing some quantity involving branching walks:

Lemma 3.3 There exists a spanning tree of G of maximum degree at most c if and only if there exists a branching walk $B = (T, \varphi)$ of length $n-1$ such that $\varphi(V(T)) = V$ and the maximum degree of T is at most c.

Proof For the forward direction, assume S to be a spanning tree of G of maximum degree at most c and order it by fixing an arbitrary order. Then define $B = (S, \varphi)$, with $\varphi : V(S) \to V(S)$ the identity function, and choose $\mathtt{rt}(S)$ arbitrarily. For the backward direction, notice that $(\varphi(V(T)), \varphi(E(T)))$ is a tree of maximum degree at most c. ■

Now we can use the following IE-formulation: Define the universe U as all branching walks (T, φ) of length $n-1$ such that the maximum degree of T is at most c. Define the requirement space $R = V$ and P_v to be all branching walks in U visiting v, for every $v \in V$. That is, let $P_v = \{(T, \varphi) \in U : v \in \varphi(V(T))\}$. For $F \subseteq V$, define $\mathcal{D}_F(x, j, g)$ as all branching walks $(T, \varphi) \in U$ from x of length j in $G[V \setminus F]$ such that the degree of the root of T is at most g and the degree of every other vertex is at most c. Hence $U = \cup_{x \in V \setminus F} \mathcal{D}_\varnothing(x, n-1, c)$. Let $d_F(x, j, g) = |\mathcal{D}_F(x, j, g)|$. We apply Theorem 3.1, and conclude that the simplified problem is to compute the number of branching walks in the universe that avoid F, and hence

$$\Big|\bigcap_{v \in F} \overline{P_v}\Big| = \sum_{x \in V \setminus F} d_F(x, n-1, c).$$

This can be done in polynomial time with dynamic programming using the following lemma:

Lemma 3.4

$$d_F(x, j, g) = \begin{cases} [g \geq 0] & \text{if } j = 0, \quad (3.6a) \\ \displaystyle\sum_{t \in N(x) \setminus F} \sum_{j_1 + j_2 = j-1} d_F(t, j_1, c-1)\, d_F(x, j_2, g-1) & \text{otherwise.} \quad (3.6b) \end{cases}$$

Proof To see that Case 3.6a holds, notice that $d_F(x, 0, g) = 0$ if g is negative since the root has a non-negative degree, and otherwise there is one branching walk (T, φ) of length zero obtained by letting T be the tree on one vertex and letting φ be the function mapping this vertex to x.

For Case 3.6b, let us define $\gamma'((T_1,\varphi_1),(T_2,\varphi_2))$ to be the branching walk $(\gamma(T_1,T_2),\varphi_1\cup\varphi_2)$, where γ is the gluing operation from (3.4). Then we claim that γ' is a bijection

$$\gamma' : \bigcup_{t\in N(x)\setminus F}\bigcup_{j_1,j_2}(\mathcal{D}_F(t,j_1,c-1)\times\mathcal{D}_F(x,j_2,g)) \leftrightarrow \mathcal{D}_F(x,j_1+j_2+1,g+1).$$

To see this, define a (c,g)-tree as a tree of maximum degree c with the degree of its root at most g, and notice that $\gamma(T_1,T_2)$ is a (c,g)-tree if and only if T_1 is a $(c,c-1)$-tree and T_2 is a $(c,g-1)$-tree. Then Case 3.6b follows by combining with the arguments in the proof of Lemma 3.2. ■

Theorem 3.5 The Degree Constrained Spanning Tree problem can be solved in $\mathcal{O}^\star(2^n)$ time and polynomial space.

Proof Due to Lemma 3.3 the considered IE-formulation solves DCST, and we can use dynamic programming on (3.4) to compute the simplified problem in polynomial time. Then the theorem follows from Theorem 3.1. ■

3.2.3 Maximum Internal Spanning Tree

A vertex of a tree is called *internal* if its degree is at least 2.

Maximum internal Spanning Tree **Parameter:** n
Input: $G=(V,E)$, $1\le c\le n$.
Question: Is there a spanning tree of G with at least c internal vertices?

Lemma 3.5 There exists a spanning tree of G with at least c internal vertices if and only if there exists a branching walk $B=(T,\varphi)$ of length $n-1$ such that $\varphi(V(T))=V$ and T has at least c internal vertices.

Proof For the forward direction, assume S to be a spanning tree of G with at least c internal vertices and order it by fixing an arbitrary order. Define $B=(S,\varphi)$, with $\varphi : V(S)\to V(S)$ the identity function. It clearly has the required properties. For the backward direction, notice that $(V,\varphi(E(T)))$ is a spanning tree with at least c internal vertices since $\phi(E(T))=n-1$. ■

Consider the following IE-formulation: Define the universe U to be all branching walks (T,φ) of length $n-1$ such that T has at least c internal vertices. For $v\in V$, let $P_v=\{(T,\varphi)\in U : v\in\varphi(V(T))\}$. For $F\subseteq V$ and $\delta\in\mathbb{Z}_-\cup\{0,1,2\}$, define $\mathcal{M}^\delta_F(x,j,g)$ as all branching walks (T,φ) in $G[V\setminus F]$ of length j from x such that $|\{v\in V(T)\setminus \mathtt{rt}(T) : v \text{ is internal}\}|+[d(\mathtt{rt}(T))\ge\delta]=g$. Let $m^\delta_F(x,j,g)=|\mathcal{M}^\delta_F(x,j,g)|$, then

Lemma 3.6

$$m_F^\delta(x,j,g) = \begin{cases} [g = [\delta \le 0]] & \text{if } j = 0, \quad (3.7a) \\ \sum_{t \in N(x) \setminus F} \sum_{g_1+g_2=g} \sum_{j_1+j_2=j-1} m_F^1(t,j_1,g_1)\, m_F^{\delta-1}(x,j_2,g_2) & \text{otherwise.} \quad (3.7b) \end{cases}$$

Proof To see that Case 3.7a holds, notice that there is only one branching walk of length 0 from x and its tree-part is a single vertex, so g must be equal to 1 if $\delta \le 0$ and equal to 0 otherwise by definition. For Case 3.7b, first define $\gamma'((T_1,\varphi_1),(T_2,\varphi_2))$ as the branching walk $(\gamma(T_1,T_2),\varphi_1 \cup \varphi_2)$, where γ is the gluing operation from (3.4). Then we claim that γ' is a bijection

$$\gamma' : \bigcup_{t \in N(s) \setminus F} \bigcup_{g_1,g_2} \bigcup_{j_1,j_2} (\mathcal{M}_F^1(t,j_1,g_1) \times \mathcal{M}_F^{\delta-1}(s,j_2,g_2)) \leftrightarrow \mathcal{M}_F^\delta(s, j_1+j_2+1, g_1+g_2).$$

To see this, note that $\gamma(T_1,T_2)$ has g' internal vertices not equal to the root if and only if T_1 has g_1' and T_2 has g_2' internal vertices not equal to the root with $g_1' + g_2' = g'$ and

$$d_{\gamma(T_1,T_2)}(\mathtt{rt}(\gamma(T_1,T_2))) = 1 + d_{T_2}(\mathtt{rt}(T_2)).$$

Hence the δ and g accumulators are modified correctly. Then Case 3.7b follows by combining with the arguments in the proof of Lemma 3.2. ■

Theorem 3.6 The Maximum Internal Spanning Tree problem can be solved in $\mathcal{O}^\star(2^n)$ time and polynomial space.

Proof Use the IE-formulation as described above. The simplified problem is to compute $|\bigcap_{v \in F} \overline{P_v}|$, which is equal to $\sum_{s \in V \setminus F} m_F^2(s, n-1, c)$ since the root needs at least two neighbors in order to be an internal vertex. It follows from Lemma 3.5 that there exists $B \in \bigcap_{v \in V} P_v$ if and only if there exists a spanning tree with at least c leaves. ■

3.2.4 Counting Spanning Forests

Define a *c-spanning forest* to be an acyclic spanning subgraph with exactly c connected components. In this section we will address the following problem:

#Spanning Forests **Parameter:** n
Input: $G = (V,E)$, $1 \le c \le n$
Question: The number of acyclic spanning subgraphs of G with exactly c connected components.

Assume that a total ordering $\prec$ on the vertex set V is given. Given a subset $X \subseteq V$, let $\min X$ be the minimum element of X with respect to $\prec$.

Definition 3.2 A *sorted branching walk* is a branching walk (T, φ) such that for every $v \in V(T)$ with children $c_1, \ldots, c_l$, numbered with respect to the order of T, $\varphi(c_i) \prec \varphi(c_j)$ for every $i < j$ and $\varphi(\mathtt{rt}(v)) = \min \varphi(V(T))$.

It can be noted that the definition implies no two children of vertex $v \in V(T)$ can be mapped to the same vertex in G.

Lemma 3.7 There is a bijection between the set of all subtrees S of G, and the set of all sorted branching walks $B = (T, \varphi)$ of length $|\varphi(E(T))|$ such that $\varphi(E(T))$ induces a subtree.

Proof Note that it is sufficient to show that for every subtree S of G, there is exactly one a sorted branching walk $B = (T, \varphi)$ of length $|\varphi(E(T))|$ such that $\varphi(E(T)) = E(S)$.

Since $\varphi(V(T)) = V(S)$ and B is sorted, we know that $\varphi(\mathtt{rt}(T)) = \min V(T)$. Since $|E(S)| = |E(T)|$, φ is a bijection and hence S and T are isomorphic. Moreover $\varphi(c_i) \prec \varphi(c_j)$ whenever c_i occurs before c_j as a child of a vertex in T since B is sorted. This implies that T and φ are uniquely determined by S. ∎

Define $\mathcal{B}_F(l)$ to be all sorted branching walks (T, φ) of length l in G such that $F \cap \varphi(V(T)) = \varnothing$. Also, define $b_F(l) = |\mathcal{B}_F(l)|$.

Lemma 3.8 The number of c-spanning forests is equal to

$$\sum_{F \subseteq V} (-1)^{|F|} \frac{1}{c!} \sum_{l_1 + \ldots + l_c = n-c} \prod_{j=1}^{c} b_F(l_j). \tag{3.8}$$

Proof We apply Theorem 3.1. Define U to be the set of all families f of sorted branching walks of total length $n-c$ (that is, the sum of the length of all members of f is $n-c$). Let the requirement space be V, and for every $v \in V$ define P_v to be all elements $f \in U$ such that there is $(T, \varphi) \in f$ with $v \in \varphi(V(T))$. It is not hard to see that a term of the summation of (3.8) (ignoring $(-1)^{|F|}$) is equal to $|\cap_{v \in F} \overline{P_v}|$.

Applying Theorem 3.1, it remains to show that the number of c-spanning forests is $|\cap_{v \in V} P_v|$. To see this, notice that for a family of sorted branching walks $f = \{(T_1, \varphi_1), \ldots, (T_c, \varphi_c)\}$, $\cup_{i=1}^{c} \varphi_i(V_i) = V$ if and only if $\cup_{i=1}^{c} \varphi_i(E_i)$ is a c-spanning forest. Then every c-spanning forest corresponds to exactly one set of branching walks of total length $n-c$ due to Lemma 3.7. The lemma follows. ∎

Define $\mathcal{B}_F(x, j, g)$ as all sorted branching walks (T, φ) from x of length j such that $\varphi(V(T)) \cap F = \varnothing$ and no child of the root of T is mapped to one of the first $g-1$ neighbors of x in $G[V \setminus F]$ with respect to the ordering $\prec$. Define $b_F(x, j, g) = |\mathcal{B}_F(x, j, g)|$. Use $N_F^q(x)$ to denote the q^{th} element of the set $N(x) \setminus F$ with respect to the ordering $\prec$.

Lemma 3.9

$$b_F(x,j,g) = \begin{cases} 1 & \text{if } \ j = 0, \qquad (3.9a) \\ 0 & \text{else if } g > |N(x) \cap F|, (3.9b) \\ \quad b_F(x,j,g+1) + & \\ \sum\limits_{j_1+j_2=j-1} b_F(N_F^g(x), j_1, 1)\ b_F(x, j_2, g+1) & \text{otherwise.} \qquad (3.9c) \end{cases}$$

Proof For Case 3.9a, notice there is exactly one branching walk (T,φ) with T being a single vertex and with φ mapping this vertex to x.

To see Case 3.9b, notice that since $g > |N(x) \cap F|$, in any branching walk in $\mathcal{B}_F(x,j,g)$ the root can not have any children but since $j > 0$ this must be the case, and hence such a branching walk does not exist.

For Case 3.9c, first notice that $\mathcal{B}_F(x,j,g+1)$ are exactly all branching walks $(T,\varphi) \in \mathcal{B}_F(x, j_1+j_2+1, g)$ where no child of the root of T is mapped to $N_F^g(x)$. Define $\gamma'((T_1,\varphi_1),(T_2,\varphi_2))$ as the branching walk $(\gamma(T_1,T_2), \varphi_1 \cup \varphi_2)$, where γ is the gluing operation from (3.4). Then it is not too hard to see that γ' is a bijection

$$\gamma' : \bigcup_{j_1,j_2} \Big(\mathcal{B}_F(N_F^g(x), j_1, 1) \times \mathcal{B}_F(x, j_2, g+1) \Big) \leftrightarrow \mathcal{B}_F(x, j_1+j_2+1, g) \setminus \mathcal{B}_F(x, j_1+j_2+1, g+1)$$

since $\gamma'(B_1,B_2)$ and $\gamma'(B_1',B_2')$ are distinct whenever either B_1 and B_1' or B_2 and B_2' are distinct. Then Case 3.9c follows by combining with the arguments in the proof of Lemma 3.2. ■

Theorem 3.7 The #SPANNING FORESTS problem can be solved in $\mathcal{O}^\star(2^n)$ time and polynomial space.

Proof Let $\overline{F} = V \setminus F$ and $b_F(l) = \sum_{x \in \overline{F}} b_{\overline{F}[x]}(x,l,1)$, where $\overline{F}[x]$ stands for the set of all elements e in $\overline{F}$ such that $x \prec e$. Using dynamic programming in combination with Lemma 3.9, the values $b_F(l)$ can be computed for every $1 \le l \le n$ for a fixed F. Also using standard dynamic programming, the simplified problem (that is, the summand of (3.9a) ignoring the $(-1)^{|F|}$) can be computed in polynomial time. Hence (3.9a) can be evaluated within the claimed resource bounds and the theorem follows from Lemma 3.8. ■

3.3 Cover Polynomial

We use $x^{\underline{i}}$ for the falling factorial $\frac{x!}{(x-i)!}$. The cover polynomial of a directed graph $D = (V,A)$ can be defined as (see also [BHKK08, CG95]):

$$\sum_{i,j} c(i,j) x^{\underline{i}} y^{\underline{j}}$$

where $c(i,j)$ is defined as the number of ways to partition V into i directed paths and j directed cycles of D. In this section we will address the following problem:

Cover Polynomial	**Parameter:** n
Input: A graph $G = (V, E)$ on n vertices.	
Question: The coefficients $c(i,j)$ for every $0 \le i, j \le n$.	

Since paths and cycles with l edges contain $l+1$ and l vertices respectively, the sum of the lengths of the paths and cycles in such a partition will be $n-i$. Moreover, if V is covered by i paths and j cycles with lengths summing up to $n-1$, the path and cycles are disjoint because of the size restriction. Recall from Subsection 3.1 that $w_F(s,j)$ is the number of walks starting from s of length j avoiding F. Similarly, define $w_F(j)$ as all walks of length j avoiding F and $\hat{w}_F(j)$ as the number of cyclic walks of length j avoiding F.

Lemma 3.10

$$c(i,j) = \sum_{F \subseteq V} (-1)^{|F|} \frac{1}{i!j!} \sum_{l_1 + \ldots l_{i+j} = n-i} \Big(\prod_{k=1}^{i} w_F(l_k) \Big) \Big(\prod_{k=i+1}^{i+j} \hat{w}_F(l_k) \Big). \tag{3.10}$$

Proof We apply Theorem 3.1. Define U as all combinations of i walks and j cyclic walks with lengths summing up to exactly $n-i$. Define the requirement space to be V, and for every $v \in V$ let P_v be all combinations of walks and cyclic walks in U such that at least one of the (cyclic) walks visits v. It is easy to see that each summand (ignoring the $(-1)^{|F|}$) equals $|\cap_{v \in F} \overline{P_v}|$, and since $|\cap_{v \in F} P_v| = c(i,j)$ by the above discussion, the lemma follows from Theorem 3.1. ■

Theorem 3.8 The Cover Polynomial problem can be solved in $\mathcal{O}^\star(2^n)$ time and polynomial space.

Proof By using minor modifications of the dynamic programming algorithm using (3.3), for every $F \subseteq V$ and $0 \le j \le n$, both $w_F(j)$ and $\hat{w}_F(j)$ can be computed in polynomial time. Using straightforward dynamic programming, the simplified problem can be obtained from the values $w_F(j)$ and $\hat{w}_F(j)$. Using this, (3.10) can be evaluated within the claimed resource bounds. ■

3.4 Convex Tree Coloring

In this section we solve a generalization of the Convex Recoloring problem studied in [PHN08] and improve upon one of their results. Let $T = (V, E)$ be a tree and $\omega : E(T) \times [k] \times [k] \to [C]$. A *k-coloring* is a function $\gamma : V \to [k]$. As before,

for $X \subseteq V$ let $\varphi(X)$ be $\cup_{v \in X} \varphi(v)$. The coloring γ is *minimal* if $\gamma(V) = [k]$. Define $\omega(\gamma) = \sum_{(u,v) \in E(T)} \omega((u,v), \gamma(u), \gamma(v))$. The coloring γ is *convex* if for every $x, y, z \in V$ such that y is in the (unique) path from x to z it holds that $\gamma(x) = \gamma(z)$ implies $\gamma(x) = \gamma(y)$, or more informally: γ is convex if all its color classes induce a connected subtree. It is worth mentioning that in [PHN08] the slightly different Convex Tree Recoloring (CTR) was studied, but that CTR is a special case of CTC.

Convex Tree Coloring (CTC)	**Parameter:** k
Input: A tree $T = (V, E)$ with a weight function $\omega : E \times [k] \times [k] \to [C]$.	
Question: Is there a convex coloring $\gamma : V \to [k]$ with $\omega(\gamma) \leq C$?	

Let us first note that we can safely assume that T is in fact binary, since if it is not, we can insert vertices and force them to have the same color by using an appropriate weight function. Also note we can restrict ourselves to minimal colorings by k vertices of degree one to arbitrary vertices and setting $\omega(\cdot, \cdot, \cdot) = 0$. Hence the colors assigned to the added vertices do not matter at all, but if a color is not minimal, it can be extended to a minimal one using the new added vertices. Thus we can restrict ourselves to solving the following variant:

Minimal Convex Binary Tree Coloring (MCBTC)	**Parameter:** k
Input: A binary tree $T = (V, E)$ with a weight function $\omega : E \times [k] \times [k] \to [C]$.	
Question: Is there a minimal convex coloring $\gamma : V \to [k]$ with $\omega(\gamma) \leq C$?	

For a given coloring γ, use $\texttt{pce}(\gamma)$ to denote the number of poly-chromatic edges, i.e. the number of edges $(y, z) \in E(T)$ such that $\varphi(y) \neq \varphi(z)$.

Lemma 3.11 The instance of MCBTC is a `yes`-instance if and only if there exists a k-coloring γ such that $\texttt{pce}(\gamma) = k - 1$, and $\omega(\gamma) \leq C$.

Proof In a tree, the fact that γ gives k monochromatic connected components is equivalent with $\texttt{pce}(\gamma) = k - 1$ since after contracting all monochromatic edges, we again obtain a tree.

Using Lemma 3.11, we can reduce our problem to determining the existence of a minimal k-coloring with $k-1$ polychromatic edges. We will use Theorem 3.1: For notational convenience add a vertex to T, make it adjacent to an arbitrarily chosen other vertex of T and let the added vertex be the root of T. For the added edge, set all corresponding weights to 0. Define the universe $U = \{\gamma : V \to [k] \mid \texttt{pce}(\gamma) = k - 1 \wedge \omega(\gamma) \leq C\}$. Define the requirement space to be $[k]$ with for each $1 \leq v \leq k$ a requirement P_v being all elements γ of U with $v \in \gamma(V)$. Then $|\cap_{v \in [l]} P_v| > 0$ if and only if the current instance is a `yes`-instance by Lemma 3.11. Define $\mathcal{C}_F(s, W, g, p)$ as all k-colorings γ of $T[s]$ with $\texttt{pce}(\gamma) = g + [\gamma(\texttt{rt}(T)) \neq p]$,

$\gamma(V(T[s])) \cap F = \varnothing$ and $\omega(\gamma) \leq W$ and let $c_F(s, W, g, p) = |\mathcal{C}_F(s, W, g, p)|$. Also note that $|\cap_{v \in F} \overline{P_v}| = c_F(\mathtt{rt}(T), C, k-1, 1)$ (the color of the root is not relevant here so we just set it to 1). It remains to show how to compute $c_F(\mathtt{rt}(T), C, k-1, p)$:

Lemma 3.12

$$c_F(s, W, g, p) = \begin{cases} [\omega((s, \mathtt{pa}(s)), p, p) \leq W] & \text{if } s \text{ is leaf} \wedge g = 0, (3.11a) \\ \sum_{q \in [l] \setminus (F \cup \{p\})} [\omega((s, \mathtt{pa}(s)), q, p) \leq W] & \text{if } s \text{ is leaf} \wedge g = 1, (3.11b) \\ 0 & \text{if } s \text{ is leaf} \wedge g > 1, (3.11c) \\ \hat{c}_F(s, j, g, p) & \text{otherwise,} \quad (3.11d) \end{cases}$$

where $\hat{c}_F(s, W, g, p) =$

$$\sum_{\substack{g_1 + g_2 = l \\ W_1 + W_2 = W - \omega((s, \mathtt{pa}(s)), p, p)}} c_F(s_1, W_1, g_1, q) c_F(s_2, W_2, g_2, q) + \quad (3.12)$$

$$\sum_{q \in [l] \setminus (F \cup \{p\})} \sum_{\substack{g_1 + g_2 = l-1 \\ W_1 + W_2 = W - \omega((s, \mathtt{pa}(s)), q, p)}} c_F(s_1, W_1, g_1, q) c_F(s_2, W_2, g_2, q). \quad (3.13)$$

Proof For Case 3.11a, the only possible valid coloring is the one where $\gamma(s) = \gamma(\mathtt{pa}(s))$. For Case 3.11b, any coloring γ with $\gamma(s) \neq \gamma(\mathtt{pa}(s))$ and low enough cost is counted. For Case 3.11c, no such a coloring exists since a leaf is only adjacent to one edge. For Case 3.11d, s is not a child and hence has two children s_1 and s_2. It is not too hard to see that the term at (3.12) is the number of colorings in $\mathcal{C}_F(s, W, g, p)$ where $\gamma(s) = \gamma(\mathtt{pa}(s))$ and that the term at (3.13) is the number of colorings in $\mathcal{C}_F(s, W, g, p)$ where $\gamma(s) \neq \gamma(\mathtt{pa}(s))$. ■

Theorem 3.9 The Convex Tree Coloring problem can be solved in $\mathcal{O}^\star(2^k C)$ time and $\mathcal{O}^\star(C)$ space.

Proof By Lemma 3.11 and the discussion before it we can reduce Convex Tree Coloring to finding a minimal k-coloring γ with $k-1$ polychromatic edges and $\omega(\gamma) \leq C$. This can be solved using the IE-formulation as discussed above. The simplified problem can be solved in $\mathcal{O}^\star(C^2)$ time and $\mathcal{O}^\star(C)$ using Lemma 3.1 and using standard Fast Fourier Transform techniques this can be reduced to $\mathcal{O}^\star(C)$ time and space. The theorem then follows from Theorem 3.1. ■

3.5 Counting Perfect Matchings

In this section we will count the number of perfect matchings in a graph $G = (V, E)$. We let $|V| = 2n$. First, we arbitrarily partition the vertex set V into A

and B (thus, $A \cup B = V$, and $A \cap B = \varnothing$), with $|A| = |B| = n$. Let an *l-matching* of G be a perfect matching $M \subseteq E$ such that $|\{e \in M : |e \cap A| = 1\}| = l$, i.e. a perfect matching containing exactly l edges with exactly one endpoint in A.

We will need the following simple lemma:

Lemma 3.13 — (Ned08) Lemma 4.10, (AFS09) Theorem 4 Given an independent set $S \subseteq V$, the number of perfect matching of G can be computed in $\mathcal{O}^\star(2^{2n-|S|})$.

We will first give two algorithms that we combine later. Both algorithms solve the same problem, but with different running times. Afterwards we show how to combine the two to obtain the main result of this section.

Lemma 3.14 l-matchings can be counted in $\mathcal{O}^*(\binom{n}{l}2^n)$ time and polynomial space.

Proof Given a perfect matching M, define

$$L(M) = \{a \in A : \exists (a,b) \in M \wedge b \in B\}.$$

That is, $L(M)$ is the set of all vertices in A that are matched with a vertex in b in M. Then, if we define $f(X)$ as the number of perfect matchings M such that $L(M) = X$, then we can compute the number of l-matchings according to

$$\#\text{l-matchings} = \sum_{X \in \binom{A}{l}} f(X) \tag{3.14}$$

Define $g(X, B)$ as the number of perfect matchings in the graph obtained from $G[X \cup B]$ by making X into an independent set. Then $f(X) = \mathtt{pm}(A \setminus X)g(X, B)$ where $\mathtt{pm}(A \setminus X)$ is the number of perfect matching in $G[A \setminus X]$. Using Lemma 3.13 both $\mathtt{pm}(A \setminus X)$ and $g(X, B)$ can be computed in polynomial space and time $\mathcal{O}^\star(2^{n-l})$ and $\mathcal{O}^\star(2^n)$ respectively. Hence the lemma follows. ∎

We proceed to the next algorithm:

Lemma 3.15 l-matchings can be counted in $\mathcal{O}^*(\binom{n}{\frac{n}{2}+\frac{l}{2}}2^n)$ time and polynomial space.

Proof Arbitrarily choose a total ordering $\prec$ on A and use the shorthand $k = \frac{n+l}{2}$. We will use Theorem 3.1: Define the universe U to be

$$U = \left\{\left((u_i, v_i)\right)_{i \le k} \in \left((A \times V) \cap E\right)^k \mid \forall i < j : u_i \prec u_j\right\} \times E^{n-k}$$

and the requirement space to be V. Define P_v to be

$$\left\{\left((u_1, v_1), \ldots, (u_n, v_n)\right) \in U \mid v \in \{u_1, \ldots, u_n\} \cup \{v_1, \ldots, v_n\}\right\}$$

for each $v \in V$. We claim that $|\cap_{v\in V} P_v|$ is $2^n(\frac{(n-l)}{2})!$ times the number of l-matchings. To see this, notice that for every l-matching there are exactly $2^n(\frac{n-l}{2})!$ elements in $|\cap_{v\in F} \overline{P_v}|$ obtained by all permutations of the $\frac{n-l}{2}$ edges contained in $G[B]$ and then flipping the order of the vertices of the edge. Moreover, every element in $|\cap_{v\in F} \overline{P_v}|$ can obtained in this way from exactly one l-matching.

Thus, combining Theorem 3.1 with the above we obtain that the number of l-matchings is

$$\frac{1}{2^n(\frac{n-l}{2})!} \sum_{F\subseteq V} (-1)^{|F|} |\bigcap_{v\in F} \overline{P_v}|. \tag{3.15}$$

We proceed by *trimming* (see also [BHKK10b]): Observe that $|\cap_{v\in F} \overline{P_v}| = 0$ if $|F \cap A| > (n-k)$ since $u_1, \ldots, u_k \in A$ are distinct for any $((u_1, v_1), \ldots, (u_n, v_n)) \in U$. Hence for evaluating (3.15), we can restrict ourselves to sum over $F \subseteq V$ such that $|F \cap A| \leq (n-k)$. The number of $F \subseteq V$ such that $|F \cap A| \leq (n-k)$ is $\mathcal{O}^\star(\binom{n}{n-k}2^n) = \mathcal{O}^\star(\binom{n}{k}2^n)$. Hence to prove the lemma, it suffices to show that $|\cap_{v\in F} \overline{P_v}|$ can be computed in polynomial time for a fixed F.

Let $\{a_1, \ldots, a_n\}$ be obtained by sorting A with respect to the total ordering $\prec$. Define

$$p_F(r,m) = \left\{ \Big((u_i, v_i) \Big)_{i\leq m} \in \Big(\big((A \setminus F) \times (V \setminus F) \big) \cap E \Big)^m \mid \forall i < j : u_i \prec u_j \prec a_r \right\}. \qquad \blacksquare$$

Then we have $|\cap_{v\in F} \overline{P_v}| = p_F(n,k) * |E(G[B \setminus F])|^{n-k}$ and it is easy to see that $p_F(n,k)$ can be computed according to

$$p_F(r,m) = \begin{cases} [m=1] d_{G[V\setminus F]}(v_1) & \text{if } r = 1, \quad (3.16a) \\ p_F(r-1,m) + d_{G[V\setminus F]} p_F(r-1, m-1) & \text{otherwise}, \quad (3.16b) \end{cases}$$

hence $p_F(n,k)$ and $|\cap_{v\in F} \overline{P_v}|$ can be computed in polynomial time and the lemma follows.

Theorem 3.10 The number of perfect matchings of a graph G can be computed in $\mathcal{O}^\star(1.95^{|V(G)|})$ time and polynomial space.

Proof Sum over all $0 \leq l \leq n$ over the number of l-matchings computed by the algorithm of minimum running time among the algorithms of Lemmas 3.14 and 3.15. The running time of this algorithm is

$$\max_{0\leq l\leq n} \min \left\{ \binom{n}{l} 2^n, \binom{n}{\frac{1}{2}n + \frac{1}{2}l} 2^n \right\} = \max_{0\leq l\leq n} \min \left\{ \binom{n}{l}, \binom{n}{\frac{1}{2}n + \frac{1}{2}l} \right\} 2^n. \tag{3.17}$$

Since $0 \leq p \leq p' \leq \frac{n}{2}$ implies $\binom{n}{p} \leq \binom{n}{p'}$, (3.17) is maximized if $\min\{l, \frac{n}{2} - \frac{l}{2}\}$ is maximized, which is at $l = \frac{n}{3}$. Hence (3.17) is equal to $\binom{n}{\frac{n}{3}}2^n = \mathcal{O}^\star(1.89^n 2^n))$, where the latter is due to standard approximations (see for example Lemma 4 of [BK06]). $\blacksquare$

3.6 Concluding Remarks

It is worth mentioning that, as the proofs might suggest, most of results of this chapter admit a generalization. The study of such a generalization was initiated in the conference version of this work and finally was given in [LN10] and will be discussed in Chapter 5. There, Theorem 3.3 is also improved by giving a $\mathcal{O}^\star(2^k C)$ time and polynomial space algorithm.

The Inclusion-Exclusion formula admits a strong generalization that we will introduce in Chapter 4 and use in Section 6.3. It is also worth mentioning that the improvements of the Inclusion-Exclusion formula have been investigated in the setting of approximative counting or set systems with special properties, see for example [KLS96].

Chapter 4

Transformations and Circuits

In this chapter we will give an introduction to the use of a number of transformations in designing algorithms and we will set up the stage for the next chapters, in particular Chapters 5 and 6. Often, the transformations we will discuss are quite simple and can be naturally represented as a single matrix. In general, one can of course argue about whether some algorithm uses a transformation or not, and when something is actually worth calling a transformation. For example, in the textbook on algorithms by Levitin [Lev03], the author speaks about three variations of the so-called 'transform-and-conquer' method:

1. **Instance simplification**– transformation to a simpler or more convenient instance of the same problem,
2. **Representation change**– transformation to a different representation of the same instance,
3. **Problem reduction**– transformation to an instance of a different problem for which an algorithm is already available.

In this and the subsequent chapters, we will focus on the second and third variant. Typically, we introduce a general problem involving so-called *algebraic circuits* (see Section 4.6) that are useful for problem reduction and show that it can be solved by using variant 2 or 3 (and occasionally variant 1).

4.1 Transformations

In this section we will discuss some classical examples of the 'representation change' variants mentioned above to show how this can actually be useful. Except the embedding of the min-sum semiring, we will not need any material covered this section so it can safely be skipped. However, we feel it serves as a good introduction to what we consider to be the 'representation change' variant.

Complex Number Multiplication

Perhaps the most fundamental and classical example is the application for complex number multiplication due to Gauss: given four reals $a, b, c, d \in \mathbb{R}$, we want to compute the reals $e, f \in \mathbb{R}$ such that $e + f\imath = (a + b\imath)(c + d\imath)$.

One way, using transformations, would be to compute the polar representations of the complex numbers: compute r_1, r_2, ϕ_1, ϕ_2 such that $(a+b\imath) = r_1(\cos\phi_1 + \imath\sin\phi_1)$ and $(a+b\imath) = r_2(\cos\phi_2+\imath\sin\phi_2)$. Then compute r_3, ϕ_2 such that $(e+f\imath) = r_3(\cos\phi_3+\imath\sin\phi_3)$; this only takes two operations since $r_3 = r_1 \cdot r_2$ and $\phi_3 = \phi_1+\phi_2$. Then compute e, f from r_3, ϕ_3. Unfortunately this is not very efficient: we already need three computations to compute r_1, r_2 (using Pythagoras' Theorem) and $r_3 = r_1 \cdot r_2$, and even more important we need to evaluate trigonometric functions which is costly. However, Gauss observed that without using these functions, it is also possible to reduce the number of multiplications by *transforming* the problem to the problem of three totally different multiplications:

$$k_1 \leftarrow c \cdot (a + b) \quad k_2 \leftarrow a \cdot (d - c) \quad k_3 \leftarrow b \cdot (c + d) \qquad e = k_1 - k_3 \text{ and } f = k_1 + k_2.$$

Matrix Multiplication and the Min-Sum Semiring

Another famous and similar example is Strassen's algorithm for multiplying two $n \times n$ matrices. It is based on 2×2 matrix multiplication: here we are given elements $a, b, \ldots, h$ of a semiring $\mathcal{R}$ and want to compute $r, s, t, u \in \mathcal{R}$ where

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & f \\ g & h \end{pmatrix}.$$

Like before we focus on minimizing the number of multiplications in $\mathcal{R}$ since typically, they take more time than additions. It is easy to verify that computing r, s, t, u in the trivial manner requires 8 multiplications. In 1969, Strassen [Str69] (see also [CLRS01], Section 28.2) showed that if $\mathcal{R}$ is a ring, the problem of computing r, s, t, u can be *transformed* to the problem of computing 7 totally different multiplications. Combined with straightforward recursion where the ring elements are $\lceil n/2 \rceil \times \lceil n/2 \rceil$ matrices this gives an algorithm for multiplying two $n \times n$ matrices over rings using only $n^{\lg 7}$ ring operations. Let ω be the smallest number such that matrix multiplication over any ring $\mathcal{R}$ can be performed using $\mathcal{O}(n^\omega)$ ring operations. After Strassen's algorithm, a series of improvements were found and the current best is in [CW82] proving $\omega < 2.376$.

However, in the case that $\mathcal{R}$ is not a ring, a similar improvement is not known. In particular the case where $\mathcal{R}$ is the min-sum semiring $\mathcal{M}_-$ is interesting and heavily studied. Note that if $\boldsymbol{A}, \boldsymbol{B} \in \mathcal{M}_-^{n\times n}$ and $\boldsymbol{AB} = \boldsymbol{C}$ then for every $1 \le i, j \le n$ $c_{ij} = \min_{1\le k\le n} a_{ik} + b_{kj}$. This matrix product is sometimes called the *distance-product* (see [SZ99]).

It is an open question (see among others [SZ99, WW10]) whether the distance product of two $n \times n$ matrices can be computed in $\mathcal{O}(n^{3-\epsilon} \log M)$ time. Solving

this question in the affirmative would imply an improved algorithm for the All Pair Shortest Path problem (see [SZ99]), Maximum Edge-Weighted Triangle (see for example [WWY10], where also a $\mathcal{O}(n^{3-\epsilon}\log M)$ time algorithm for the more special case of Maximum Node-Weighted Triangle is given) and Minimum Weight Cycle (see [RW11]).

It is, however, already known that the distance product of two $n \times n$ matrices with positive integer entries that are at most M, can be computed in $\tilde{\mathcal{O}}(n^{\omega} M \lg M)$ time. Note that this can be very slow because M could be exponential in the input size since the integers are given in binary. But we elaborate more on this now since it will be particularly useful in the next chapter. The idea is to embed the min-sum semiring $\mathcal{M}_-$ into the polynomial ring $\mathbb{Z}[\mathbb{Z}_{2M+1}]$ and then apply a fast matrix multiplication algorithm. Let us denote the embedding by $\eta : \mathcal{M}_- \to \mathbb{Z}[\mathbb{Z}_{d+1}]$, and $\eta^{-1} : \mathbb{Z}[\mathbb{Z}_{d+1}] \to \mathcal{M}_-$ be the function extracting the element from $\mathcal{M}_-$ back (note that we slightly abuse notation since η and η^{-1} are not inverses in the strict mathematical sense). Then, if $2M \leq d$ and we let $\eta(a) = \langle 1, a \rangle$:

$$\eta^{-1}(\boldsymbol{a}) = \min_{i \in \operatorname{supp}(a)} i; \quad \min(a,b) = \eta^{-1}(\eta(a) + \eta(b)); \; a+b = \eta^{-1}(\eta(a) * \eta(b)). \tag{4.1}$$

Moreover, since all integers resulting from the embedding are either 0 or 1, the operations in $\mathbb{Z}[\mathbb{Z}_d]$ can be performed using only $\mathcal{O}(d \log d)$ time using the Fast Fourier Transform (see Section 4.3). This embedding will turn out to be useful whenever a technique requiring an additive inverse is applied to a minimization/maximization problem.

Matrix Diagonalization

As mentioned before, transformations can be useful to speed up computation. The reason is naturally that in the transformed domain, the computation is significantly easier than in the original: in the above examples, the computation of four (eight) particular multiplications was transformed to the problem of computing three (seven) multiplications. In the following we restrict ourselves to the important special case of linear transformations (that is, matrices) and recall some basic linear algebra that indicate when they are useful for speeding up computation.

Consider Example 4. Note that the matrix $\boldsymbol{T}$ is particularly useful since $\boldsymbol{T}^{-1}\boldsymbol{A}\boldsymbol{T}$ is a diagonal matrix and diagonal matrix multiplication is easier than general matrix multiplication. Let us say that a matrix $\boldsymbol{T}$ *diagonalizes* a matrix $\boldsymbol{A}$ if $\boldsymbol{T}^{-1}\boldsymbol{A}\boldsymbol{T}$ is a diagonal matrix. It is well understood when a matrix is diagonalizable and similar decompositions are known as well, but for more details we refer to any undergraduate textbook on linear algebra (see for example [Ant94]). We will see in the subsequent chapters that the above scheme of (i) applying a transformation (ii) doing the required computation in the transformed domain,

and (iii) applying the inverse transformation, is advantageous in many situations.

Example 4 — Computing Fibonacci numbers. Consider the Fibonacci numbers f_i defined by $f_0 = f_1 = 1$ and for $n > 1$, $f_n = f_{n-1} + f_{n-2}$. Then it is easy to see that for $n \geq 1$, it holds that

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix}, \text{ since } \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix} = \begin{pmatrix} f_{n+2} & f_{n+1} \\ f_{n+1} & f_n \end{pmatrix}.$$

Moreover, this straightforwardly gives an algorithm using $\mathcal{O}(\log n)$ operations for computing f_n. But, this algorithm seems to need storage for at least 4 Fibonacci numbers at each step. Since these Fibonacci numbers are particularly big, one might wonder whether there is an alternative way. Let $\varphi = \frac{1+\sqrt{5}}{2}$ (also called the *golden ratio*) and let

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \boldsymbol{T} = \begin{pmatrix} \varphi & 1 \\ 1 & -\varphi \end{pmatrix}. \text{ Then, note that } \boldsymbol{T}^{-1} = \frac{1}{\varphi + 2}\begin{pmatrix} \varphi & 1 \\ 1 & -\varphi \end{pmatrix}.$$

The transformation $\boldsymbol{T}$ is the one that will simplify the computation of F_n. To see that this helps, note that

$$\boldsymbol{T}^{-1}\boldsymbol{A}\boldsymbol{T} = \frac{1}{\varphi + 2}\begin{pmatrix} \varphi & 1 \\ 1 & -\varphi \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} \varphi & 1 \\ 1 & -\varphi \end{pmatrix} = \frac{1}{\varphi + 2}\begin{pmatrix} 3\varphi + 1 & 0 \\ 0 & 1 - 2\varphi \end{pmatrix}.$$

Now $f_n = (\boldsymbol{A}^n)_{12}$ can be computed according to $A^n = \boldsymbol{T}(\boldsymbol{T}^{-1}\boldsymbol{A}\boldsymbol{T}^n)\boldsymbol{T}^{-1}$, where the powering of the expression between parentheses is powering of a diagonal matrix, and the crux is that this is a easier than general matrix multiplication. Hence this scheme only requires storage for two numbers roughly proportional to f_n. Note that very similarly one can also simply use the known closed formula for the Fibonacci numbers.

4.2 Group Algebra-like Structures

An important source of matrices for which there are explicit diagonalizing matrices known are so called group algebra-like structures. Let us start with giving the most general formal definition:

Definition 4.1 Given a semiring $\mathcal{R}$ and set $\mathcal{G}$ equipped with a binary operation $\cdot : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$, $\mathcal{R}[\mathcal{G}]$ is defined as the semiring consisting of the set $\mathcal{R}^{\mathcal{G}}$ equipped with the addition and multiplication operators defined as follows: if $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}$ are in $\mathcal{R}[\mathcal{G}]$ with $\boldsymbol{a} + \boldsymbol{b} = \boldsymbol{c}$ and $\boldsymbol{a} * \boldsymbol{b} = \boldsymbol{d}$, then for every $z \in \mathcal{G}$ it holds that $c_z = a_z + b_z$ and $d_z = \sum_{x \cdot y = z} a_x b_y$.

Note that in the above definition, it is easy to see that $\mathcal{R}[\mathcal{G}]$ actually is a semiring. Depending on the corresponding axioms that $\mathcal{R}$ and $\mathcal{G}$ satisfy, $\mathcal{R}[\mathcal{G}]$ is usually called (semi)group (semi)ring/algebra.

As illustration, let us discuss an embedding η of $\mathcal{R}[\mathcal{G}]$ into the ring of all $|\mathcal{G}| \times |\mathcal{G}|$ matrices with entries from $\mathcal{R}$. If $\boldsymbol{A} = \eta(\boldsymbol{a})$, then $A_{x,z} = \sum_{x \cdot y = z} a_y$. Given $\boldsymbol{A}$ and assuming $\mathcal{G}$ is a semigroup that has an identity, $\boldsymbol{a}$ can be easily re-obtained using $a_z = A_{e,z}$, where e is the identity of $\mathcal{G}$.

Example 5 — Embedding of $\mathbb{Z}[\mathbb{Z}_3]$ into $\mathbb{Z}^{3\times3}$. Let $\boldsymbol{a} = (1, 2, 4)$, $\boldsymbol{b} = (5, 6, 7)$, $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$ and $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ be the corresponding embeddings, then

$$\boldsymbol{AB} = \begin{pmatrix} 1 & 4 & 2 \\ 2 & 1 & 4 \\ 4 & 2 & 1 \end{pmatrix} \begin{pmatrix} 5 & 7 & 6 \\ 6 & 5 & 7 \\ 7 & 6 & 5 \end{pmatrix} = \begin{pmatrix} 43 & 39 & 44 \\ 44 & 43 & 39 \\ 39 & 44 & 43 \end{pmatrix},$$

and it is easy to check that indeed $\boldsymbol{c} = (43, 44, 39)$.

An important case which is the main subject of study in Section 4.3 is $\mathcal{R}[\mathbb{Z}_m]$. In this special case, the matrices resulting from the embedding are of the special type defined as follows:

Definition 4.2 A matrix $\boldsymbol{C}$ is *circulant* if there exists a vector $\boldsymbol{c} \in \mathcal{R}^m$ such that:

$$\boldsymbol{C} = \begin{pmatrix} c_0 & c_{m-1} & \dots & c_1 \\ c_1 & c_0 & \dots & c_2 \\ \vdots & \vdots & \ddots & \vdots \\ c_{m-1} & c_{m-2} & \dots & c_0 \end{pmatrix}.$$

To see that $\mathcal{R}[\mathbb{Z}_m]$ can indeed be embedded by exclusively using circulant matrices, note that if $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathcal{R}[\mathbb{Z}_m]$ and

$$\begin{pmatrix} c_0 & c_{m-1} & \dots & c_1 \\ c_1 & c_0 & \dots & c_2 \\ \vdots & \vdots & \ddots & \vdots \\ c_{m-1} & c_{m-2} & \dots & c_0 \end{pmatrix} = \begin{pmatrix} a_0 & a_{m-1} & \dots & a_1 \\ a_1 & a_0 & \dots & a_2 \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-1} & a_{m-2} & \dots & a_0 \end{pmatrix} \begin{pmatrix} b_0 & b_{m-1} & \dots & b_1 \\ b_1 & b_0 & \dots & b_2 \\ \vdots & \vdots & \ddots & \vdots \\ b_{m-1} & b_{m-2} & \dots & b_0 \end{pmatrix},$$

then for every $0 \le i < n$, $c_i = \sum_{j+k \equiv i \pmod m} a_j b_k$ which is indeed consistent with the multiplication operator of $\mathcal{R}[\mathbb{Z}_m]$.

In the next subsection we will see a transformation that diagonalizes all $m \times m$ circulant matrices. However, in this thesis we will use the following notion of diagonalization directly rather then the notion of matrix diagonalization:

Definition 4.3 A matrix $\boldsymbol{T}$ *diagonalizes* $\mathcal{R}[\mathcal{G}]$ if for every $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{R}[\mathcal{G}]$ $(\boldsymbol{a} * \boldsymbol{b})\boldsymbol{T} = \boldsymbol{a}\boldsymbol{T} \circ \boldsymbol{b}\boldsymbol{T}$.

4.3 Fourier Transform

In this section we study group algebras of the type $\mathbb{C}[\mathbb{Z}_{m_1} \oplus \mathbb{Z}_{m_2} \oplus \ldots \oplus \mathbb{Z}_{m_\delta}]$ where δ is an integer, $(m_1, m_2, \ldots, m_\delta) = \boldsymbol{m} \in \mathbb{N}^\delta$ and $\oplus$ refers to the direct product (refer to Section 1.3). It is known that every Abelian group $\mathcal{A}$ is isomorphic to $\mathbb{Z}_{m_1} \oplus \mathbb{Z}_{m_2} \oplus \ldots \oplus \mathbb{Z}_{m_\delta}$ for some $\boldsymbol{m} \in \mathbb{N}^\delta$ by the fundamental Theorem of finite Abelian groups (see for example [Ter99, Theorem 1 in Chapter 10]) so all structural properties in this section apply as well to $\mathbb{C}[\mathcal{A}]$ where $\mathcal{A}$ is an arbitrary finite Abelian group.

Definition 4.4 — The Fourier transform For a positive integer m, the Fourier transform $\boldsymbol{F}_m$ is the matrix

$$\boldsymbol{F}_m = \begin{pmatrix} \omega_m^{0*0} & \omega_m^{0*1} & \ldots & \omega_m^{0*(m-1)} \\ \omega_m^{1*0} & \omega_m^{1*1} & \ldots & \omega_m^{1*(m-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_m^{(m-1)*0} & \omega_m^{(m-1)*1} & \ldots & \omega_m^{(m-1)*(m-1)} \end{pmatrix}.$$

Recall from Section 1.3, that in the above definition, ω_N is the N^{th} root of unity. The Fourier transform is invertible, and its inverse looks very similar:

Lemma 4.1 — Fourier inversion The inverse of $\boldsymbol{F}_m$ is given by

$$\boldsymbol{F}_m^{-1} = \frac{1}{m} \begin{pmatrix} \omega_m^{-0*0} & \omega_m^{-0*1} & \ldots & \omega_m^{-0*(m-1)} \\ \omega_m^{-1*0} & \omega_m^{-1*1} & \ldots & \omega_m^{-1*(m-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_m^{-(m-1)*0} & \omega_m^{-(m-1)*1} & \ldots & \omega_m^{-(m-1)*(m-1)} \end{pmatrix}$$

Proof This boils down to showing that $\frac{1}{m}\sum_{k=0}^{m-1} \omega_N^{ik-kj} = [i = j]$. If $i = j$, the left hand is clearly equal to 1. If $i \neq j$, we have

$$\frac{1}{m}\sum_{k=0}^{m-1} (\omega_m^{i-j})^k = \frac{1 - (\omega_m^{i-j})^m}{1 - \omega_m^{i-j}} = \frac{1 - (\omega_m^m)^{i-j}}{1 - \omega_m^{i-j}} = 0,$$

where we use the geometric progression summation formula for the first equality and the special properties of ω_m for the second equality. ∎

Now we will consider $\boldsymbol{F}_{\boldsymbol{m}} = \boldsymbol{F}_{m_1} \otimes \boldsymbol{F}_{m_2} \otimes \ldots \otimes \boldsymbol{F}_{m_\delta}$ (recall from Section 1.3 that $\otimes$ is the Kronecker product). By the mixed product property (refer to (1.1)), we know that

$$\boldsymbol{F}_{\boldsymbol{m}^{-1}} = \boldsymbol{F}_{m_1}^{-1} \otimes \boldsymbol{F}_{m_2}^{-1} \otimes \ldots \otimes \boldsymbol{F}_{m_\delta}^{-1}. \tag{4.2}$$

Theorem 4.1 — Fourier diagonalization Let $\boldsymbol{m} = (m_1, m_2, \ldots, m_\delta)$ and $\mathcal{A} = \mathbb{Z}_{m_1} \oplus \ldots \oplus \mathbb{Z}_{m_\delta}$. Then the Fourier transform $\boldsymbol{F}_{\boldsymbol{m}} = \boldsymbol{F}_{m_1} \otimes \boldsymbol{F}_{m_2} \otimes \ldots \otimes \boldsymbol{F}_{m_\delta}$ diagonalizes $\mathbb{C}[\mathcal{A}]$.

Proof We have to prove that for every $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{C}[\mathcal{A}]$, $(\boldsymbol{a} * \boldsymbol{b})\boldsymbol{F} = \boldsymbol{a}\boldsymbol{F} \circ \boldsymbol{b}\boldsymbol{F}$. To see that this is true let $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. Then, denoting the group operation of $\mathcal{A}$ with $+$, for every $\boldsymbol{z} \in \mathcal{A}$:

$$\begin{aligned}
((\boldsymbol{a} * \boldsymbol{b})\boldsymbol{F})_{\boldsymbol{z}} = (\boldsymbol{c}\boldsymbol{F})_{\boldsymbol{z}} = \boldsymbol{c}\boldsymbol{F}^{(\boldsymbol{z})} &= \sum_{\boldsymbol{y} \in \mathcal{A}} \left(\sum_{\boldsymbol{w}+\boldsymbol{x}=\boldsymbol{y}} a_{\boldsymbol{w}} b_{\boldsymbol{x}} \right) \prod_{i=1}^{\delta} \omega_{m_i}^{y_i z_i} \\
&= \sum_{\boldsymbol{w}, \boldsymbol{x} \in \mathcal{A}} a_{\boldsymbol{w}} b_{\boldsymbol{x}} \prod_{i=1}^{\delta} \omega_{m_i}^{(w_i + x_i) z_i} \\
&= \Big(\sum_{\boldsymbol{w} \in \mathcal{A}} a_{\boldsymbol{w}} \prod_{i=1}^{\delta} \omega_{m_i}^{w_i z_i} \Big) \Big(\sum_{\boldsymbol{x} \in \mathcal{A}} b_{\boldsymbol{x}} \prod_{i=1}^{\delta} \omega_{m_i}^{x_i z_i} \Big) \\
&= (\boldsymbol{a}\boldsymbol{F} \circ \boldsymbol{b}\boldsymbol{F})_{\boldsymbol{z}}.
\end{aligned}$$

■

Similarly, it can be verified that $\boldsymbol{F}$ diagonalizes all circulant matrices, that is $\boldsymbol{F}^{-1}\boldsymbol{C}\boldsymbol{F}$ is diagonal for every circulant matrix $\boldsymbol{C}$ but we will not use this. On a side-note, generalizations of the Fourier transform to non-Abelian group are known as well (see for example [Ter99, Chapter 15]), that "block-diagonalize" the matrices resulting from the embedding as discussed in Subsection 4.2.

Fast Fourier Transform

The following theorem, given without proof, is a generalization of the famous Cooley-Tukey algorithm [CT65] (sometimes also attributed to, among others, Gauss or Yates [Yat37], see [HJB84] for a historical account). The idea is to give an algorithm that exploits a factorization of the Fourier matrix $\boldsymbol{F}$. See also [CLRS01] for a less technical proof of the original special case $\delta = 1$.

Theorem 4.2 — Fast Fourier Transform,see (BCT90, Theorem 3) Given $\boldsymbol{v} \in \mathbb{C}[\mathbb{Z}_{m_1} \oplus \mathbb{Z}_{m_2} \oplus \ldots \oplus \mathbb{Z}_{m_\delta}]$ where the complex numbers are represented by at most β bits, $\boldsymbol{v}\boldsymbol{F}$ and $\boldsymbol{v}\boldsymbol{F}^{-1}$ can be computed in $\mathcal{O}(|\mathcal{A}|(\lg|\mathcal{A}|)\beta(\lg^2\beta))$ time, where $\mathcal{A} = \mathbb{Z}_{m_1} \oplus \mathbb{Z}_{m_2} \oplus \ldots \oplus \mathbb{Z}_{m_\delta}$.

This claimed running time is not known to be optimal, even for the special case where $\delta = 1$. However, in a restricted class of algorithms it *is* known to be optimal [Pap79]. For a survey on further generalizations, we also refer to [MR97].

4.4 Möbius Inversion

For the notation and terminology used on partially ordered sets in this section we refer to Section 1.3. In the following, let P be a poset and let $\mathcal{R}$ be a ring.

Definition 4.5 The *Möbius function* $\mu : P \times P \to \mathcal{R}$ of P is defined for all $x, y \in P$ by

$$\mu(x,y) = \begin{cases} 1 & \text{if } x = y, \\ -\sum_{x \le y < z} \mu(y,z) & \text{if } x < z, \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

The *zeta transform* $\boldsymbol{\zeta}$ and *Möbius transform* $\boldsymbol{\mu}$ are the $|P| \times |P|$ matrices defined by $\zeta_{x,y} = [x \le y]$ and $\mu_{x,y} = \mu(x,y)$ for all $x, y \in P$.

Theorem 4.3 — Hall (Sta11, Proposition 3.8.5) For all $x, y \in P$, $\mu(x,y)$ is the number of even chains in (x,y) minus the number of odd chains in the interval (x,y).

Proof Use induction on number of elements in the interval (x,y). If $x = y$, (x,y) contains only the empty chain, which is even. If $x \le y$, group all chains on their smallest element y. The contribution of all these chains is exactly $-\mu(y,z)$ since odd chains are extended to even chains and vice-versa by adding y. ■

Theorem 4.4 — Möbius inversion (Folklore) The Möbius and zeta-transform are mutual inverses, that is: $\boldsymbol{\mu\zeta} = \boldsymbol{I}$ and $\boldsymbol{\zeta\mu} = \boldsymbol{I}$.

Proof This is equivalent to stating $\sum_{y \in P} [x \le y]\mu(y,z) = [x = z]$. If $x = z$, both sides are easily seen to be equal to 1. Otherwise, the left-hand side is easily seen to be the number of odd chains in $[x,z)$ minus the number of even chains in $[x,z)$. Note that in $[x,z)$ with $x < z$, taking the symmetric difference of a chain with x is a bijection between all even and odd chains and hence the quantity will be zero. The second part follows by very similar arguments. ■

Indeed, in the case that P is the subset lattice $(2^V, \cup)$, the equality $\boldsymbol{v} = \boldsymbol{v\zeta\mu}$ or written more explicitly,

$$v_X = \sum_{Y \subseteq X} \mu(Y,X) v_Y = \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} v_Y$$

is equivalent to the Inclusion-Exclusion equality (see Theorem 3.1). In retrospective, we see that the reason that Inclusion-Exclusion was useful in Chapter 3 is that, like the Fourier transform, $\boldsymbol{\zeta}$ is a diagonalizer as well:

Theorem 4.5 Let $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{R}[\mathbb{U}^V]$. Then $(\boldsymbol{a} * \boldsymbol{b})\boldsymbol{\zeta} = \boldsymbol{a}\boldsymbol{\zeta} \circ \boldsymbol{b}\boldsymbol{\zeta}$.

Proof If we let $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$, then for every $Z \subseteq V$:

$$(\boldsymbol{c}\boldsymbol{\zeta})_Z = \sum_{Y \subseteq Z} \sum_{W \cup X = Y} a_W b_X = \sum_{X,Y \subseteq Z} a_X b_Y = (\boldsymbol{a}\boldsymbol{\zeta} \circ \boldsymbol{b}\boldsymbol{\zeta})_Z.$$

■

Fast Zeta/Möbius-transform

The following theorem is usually attributed to Yates

Theorem 4.6 — Fast zeta/Möbius transform,(Yat37) Given $\boldsymbol{v} \in \mathcal{R}[\mathbb{U}_n]$ where $|v_X| \leq 2^\beta$ for every $X \subseteq \{1, \ldots, n\}$, $\boldsymbol{v}\boldsymbol{\zeta}$ and $\boldsymbol{v}\boldsymbol{\mu}$ can be computed in $\mathcal{O}(2^n n \beta \lg^2 \beta)$ time.

It is a subject of ongoing research to find similarly fast algorithms for other posets than the subset lattice $\mathbb{U}^n$ (see for example [BHK+11]).

4.5 Working Modulo 2

Another useful transformation is simply introducing a modulus 2. Note that this is quite different from the transformations mentioned before since it is not invertible: one could argue that all previous transformations are of the second variant mentioned in the beginning of this chapter, and in this section we will encounter a transformation of the third type.

If we aim to decide whether there exists a witness of a problem in $\mathcal{NP}$ and we are guaranteed that the number of witnesses is either zero or odd, we can focus on counting the number of witnesses modulo two instead. This sometimes is advantageous since in some situations the number of 'fake witnesses'[(1)] are guaranteed to be even. For making sure that the number of witnesses is either zero or odd, we will now recall a well-known technique.

Definition 4.6 Given a set U, set family $\mathcal{F} \subseteq 2^U$ and a function $\omega : U \to \mathbb{Z}$, a set $S \in \mathcal{F}$ is called a *minimizer of* ω *in* $\mathcal{F}$ if $\omega(S) = \min_{S' \in \mathcal{F}} \omega(S')$. The function ω is said to *isolate* the set family $\mathcal{F} \subseteq 2^U$ if there is a unique minimizer of ω in $\mathcal{F}$.

Recall here, that for $X \subseteq U$, $\omega(X)$ denotes $\sum_{u \in X} \omega(u)$. Now let us give the lemma, we provide a proof for completeness.

Lemma 4.2 — Isolation Lemma, (MVV87) 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

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

(1) for a cleverly chosen definition of 'fake witnesses'

Proof For every element $e \in U$, define

$$a(e) = \min_{e \notin S \in \mathcal{F}} \omega(S) - \min_{e \in S \in \mathcal{F}} \omega(S \setminus \{e\}).$$

Notice that for every element $e \in U$, $a(e)$ does not depend on $\omega(e)$. Hence, taking probability over all weight functions $\omega : U \to \{1, 2, \ldots, N\}$ uniformly at random, we know that for every element $e \in U$, $\text{prob}[a(e) = \omega(e)] \leq \frac{1}{N}$. Now assume $S_1, S_2 \in \mathcal{F}$ are both minimizers of ω in $\mathcal{F}$ such that $S_1 \neq S_2$. Let $e \in S_2 \setminus S_1$. Then we know that

$$\min_{e \notin S \in \mathcal{F}} \omega(S) = \omega(S_1) = \omega(S_2) = \min_{e \in S \in \mathcal{F}} \omega(S \setminus e) + \omega(e),$$

and subtracting $\min_{e \in S \in \mathcal{F}} \omega(S \setminus e)$ from both sides results in $a(e) = \omega(e)$. Then, we know that the probability that ω does not isolate $\mathcal{F}$ is

$$\text{prob}[\exists \text{two distinct minimizers of } \omega \text{ in } \mathcal{F}] \leq \text{prob}[\exists e : a(e) = \omega(e)] \leq \frac{|U|}{N},$$

where the last inequality follows from the union bound. ■

The original motivation of the Isolation Lemma was to give a fast parallel algorithm for constructing a maximum weight matching of a graph: first, the Isolation Lemma is used reduce the problem of deciding whether a matching of weight at least t exists to the problem to computing the parity of the number of matchings of size at least t. The latter can be reduced to computing the determinant of a certain matrix, and since the determinant of a matrix can be computed efficiently by a parallel algorithm due to [Pan85], this gives a fast (to be precise, $\mathcal{RNC}^2$) randomized parallel algorithm for deciding whether a matching of weight at least t exists. Second, due to the Isolation Lemma and the reduction all parallel processors will consider the same matching and hence it can also be constructed in the same resource bound. Before this, an efficient parallel algorithm for the *decision variant* was already known due to [Lov79] using *polynomial identity testing* based on an observation of [Tut47].

It is worth mentioning that in [CRS95], a lemma using fewer 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.

An alternative method to a similar end is obtained by using Polynomial Identity Testing [DL78, Sch80, Zip79] over a field of characteristic 2. This second method has been already used in the field of exact and parameterized algorithms [Bjö10b, BHKK10a, Kou08, KW09, Wil09]. The two methods do not differ much in their consequences: both use the same number of random bits (the most randomness efficient algorithms are provided in [AB03, CRS95]). Also, the challenge of giving a full derandomization seems to be equally difficult for both methods [AM08, KI04]. In this work, we choose to use the Isolation Lemma because it requires less preliminary knowledge.

4.6 Algebraic Circuits

As mentioned in the beginning of this chapter we use the notion of an (algebraic) circuit to define generic problems to which other problems to be solved later can be reduced. It is not surprising that circuits are useful for this purpose since they are already being studied as a model of computation in the field of algebraic complexity theory (see for example [AB09, Section 16.1]). In this section we will mainly introduce some terminology and easy observations that will be used throughout the next chapters.

Definition 4.7 — (Algebraic) circuit A *circuit* C is a triple $(D, \lambda, \mathcal{R})$ where $D = (V, A)$ is a directed acyclic multigraph with unique sink and indegree at most 2, $\mathcal{R}$ is a semiring and λ is a function $\lambda : V \to \{+, *\} \cup \mathcal{R}$ such that for every source $v \in V$, $\lambda(v) \in \mathcal{R}$ and for every vertex $v \in V$ that is not a source, $\lambda(v) \in \{+, *\}$.

Note that in the special case where $\mathcal{R}$ is the boolean semiring $\mathbb{B}$, the definition coincides with the definition of a boolean circuit as already encountered for the CKT-Sat problem (see Chapter 1).

In addition to the formal definition, we use the following natural terminology and notation. The function λ is a called the *labeling function*, while vertices of D are called *gates* of C. If v is a gate of C, $\lambda(v)$ is called the label of v. The sources and sink of D are referred to as *input gates* and the *output gate* of C, respectively. Gates of C that are not input gates are called either *addition* or *multiplication gates* of C depending on their label. If v is a gate of C, $C[v]$ is the circuit obtained from C by removing all its gates from which v cannot be reached in the directed graph of C. In all the above, we will omit the 'in C' part if this is clear from the context.

Definition 4.8 Every gate g of C is *associated* with a semiring element in the following natural way: if g is an input gate, we associate the label of g with g. If g is an addition gate we associate the ring element $e_1 + \ldots + e_d$ with g, and if g is a multiplication gate we associate the ring element $e_1 \cdot \ldots \cdot e_d$ with g where $e_1, \ldots, e_d$ are the ring elements associated with the d in-neighbors of g, and $+$ and $\cdot$ are the operations of the ring $\mathcal{R}$.

Often we will slightly abuse notation and directly address the ring element associated by a gate g by g itself.

We will often encounter circuits over a (semi)ring with a groundset of the type A^B where A, B are sets, so than the semiring elements are actually vectors indexed by B.

Definition 4.9 A circuit C is said to have *singleton inputs* if the label of every input gate of C is a singleton vector.

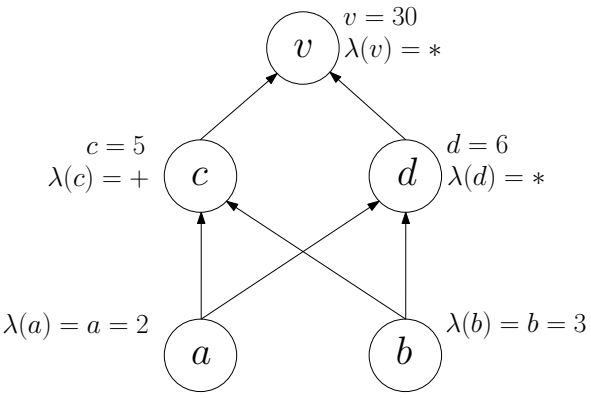

Figure 4.1 – A circuit over $\mathbb{Z}$ with labeling function λ and the associated values from Definition 4.8.

Most of the computational problems in the subsequent chapters will be formulated as circuits that have singleton inputs.

Definition 4.10 Let $\mathcal{R}$ and $\mathcal{S}$ be rings, let $h : \mathcal{R} \to \mathcal{S}$ be a homomorphism, and suppose that C is a circuit over $\mathcal{R}$. Then, the circuit C' over $\mathcal{S}$ *obtained by applying* h *to* C is defined as the circuit obtained from C by replacing for every input gate the label l by $h(l)$.

Note that the following is immediate from the definition of a homomorphism:

Observation 4.1 Suppose C is a circuit over a ring $\mathcal{R}$ with output $v \in \mathcal{R}$. Then the circuit over $\mathcal{S}$ obtained by applying a homomorphism $h : \mathcal{R} \to \mathcal{S}$ to C outputs $h(v) \in \mathcal{S}$.

Maximum Sum / Product Length

Consider the task of evaluating a circuit $C = (D, \mathbb{Z}, \lambda)$ where $0 \leq \lambda(g) \leq 2$ for every input gate g. If v is the output of C, naturally v can be determined using only an addition for every addition gate and a multiplication for every multiplication gate, so in the unit-cost model (refer to Section 1.2) this can be done in $\mathcal{O}(|C|)$. However, there is a problem if we move to the (recall, more realistic) log-cost model: there are circuits where the number of bits needed to represent the output in binary is exponential in $|C|$ (see Figure 4.2 for an example). Hence, the linear time algorithm is not realistic at all since there are circuits in reality that take much more time to evaluate.

To filter out these circuits and be able to provide algorithms that are efficient in the log-cost model, we define the following measure for how difficult evaluating a circuit is.

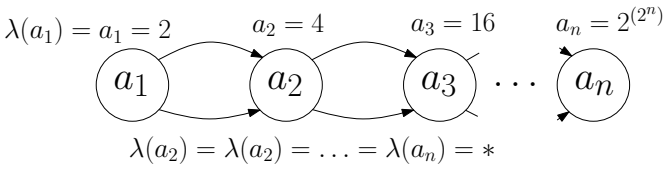

Figure 4.2 – A circuit over $\mathbb{Z}$ with labeling function λ and the associated values from Definition 4.8.

Definition 4.11 — Maximum sum length, Maximum product length For a given circuit $C = ((V, A), \lambda, \mathcal{R})$ with output gate v, define the *maximum sum length* $\texttt{msl}(C)$ and *maximum product length* $\texttt{mpl}(C)$ as follows:

$$\texttt{msl}(C) = \max_{X \subseteq A} \left| \left\{ \#\text{io-paths(X)} \;\middle|\; \text{for every } v \in V \text{ with } \lambda(v) = *,\ d^-_{G[X]}(v) = 1 \right\} \right|,$$
$$\texttt{mpl}(C) = \max_{X \subseteq A} \left| \left\{ \#\text{io-paths(X)} \;\middle|\; \text{for every } v \in V \text{ with } \lambda(v) = +,\ d^-_{G[X]}(v) = 1 \right\} \right|,$$

where #io-paths(X) is the number of paths in $D[X]$ from an input gate of C to the output gate v.

The names in this definition are justified by the fact that if one writes down the formula the circuit C computes, the maximum sum and product length are exactly the maximum number of concatenated summations and multiplications respectively. An equivalent and perhaps more useful characterization however is the following:

Observation 4.2 Let $C = (D, \lambda, \mathcal{R})$ be a circuit. Then, $\texttt{msl}(C)$ is equal to the output of the circuit $C' = (D, \kappa^+, \mathcal{M}_+)$ and $\texttt{mpl}(C)$ is equal to the output of the circuit $C'' = (D, \kappa^*, \mathcal{M}_+)$, where

$$\kappa^+(v) = \begin{cases} 1 & \text{if } v \text{ is an input gate,} \\ \max & \text{if } \lambda(v) = *, \\ + & \text{if } \lambda(v) = +, \end{cases} \qquad \kappa^*(v) = \begin{cases} 1 & \text{if } v \text{ is an input gate,} \\ \max & \text{if } \lambda(v) = +, \\ + & \text{if } \lambda(v) = *. \end{cases}$$

Proof Both claims are easy to prove by induction on $|C|$: if $|C| = 1$, v is an input gate than $\texttt{mpl}(C) = \texttt{msl}(C) = 1$ and both C' and C'' clearly output 1. Otherwise, in the second case we have to consider the input wire to include in X that maximizes #io-paths(X), which is by induction the in-neighbor maximizing its associated value. Similarly in the third case, we are allowed to include both input wires into X and the number of paths from an input to the current gate is exactly the sum of the number of paths from an input to an in-neighbor. ■

Now let us show now how Definition 4.11 can be used to bound the values computed by a circuit.

Observation 4.3 Given a circuit $C = (D, \lambda, \mathbb{R}_+)$ such that $2 \leq M \in \mathbb{N}$ and for every input gate a, $a \leq M$. Then for every gate g, it holds that

$$g \leq (M \cdot \mathtt{msl}(C[g]))^{\mathtt{mpl}(C[g])}.$$

Proof We give a proof by induction on $|C[g]|$: If $|C[g]| = 1$ then $\mathtt{msl}(C[g]) = \mathtt{mpl}(C[g]) = 1$ and clearly $g \leq M$ by assumption since g is an input gate. If $|C[g]| > 1$, let l and r be the in-neighbors of g. If g is an addition gate

$$\begin{aligned} g = l + r &\leq (M \cdot \mathtt{msl}(C[l]))^{\mathtt{mpl}(C[l])} + (M \cdot \mathtt{msl}(C[r]))^{\mathtt{mpl}(C[r])} \\ &\leq (M \cdot \mathtt{msl}(C[l]))^{\mathtt{mpl}(C[g])} + (M \cdot \mathtt{msl}(C[r]))^{\mathtt{mpl}(C[g])} \\ &\leq (M \cdot (\mathtt{msl}(C[l]) + \mathtt{msl}(C[r])))^{\mathtt{mpl}(C[g])} \\ &= (M \cdot (\mathtt{msl}(C[g]))^{\mathtt{mpl}(C[g])}, \end{aligned}$$

and if g is a multiplication gate

$$\begin{aligned} g = l * r &\leq (M \cdot \mathtt{msl}(C[l]))^{\mathtt{mpl}(C[l])} \cdot (M \cdot \mathtt{msl}(C[r]))^{\mathtt{mpl}(C[r])} \\ &\leq (M \cdot \mathtt{msl}(C[g]))^{\mathtt{mpl}(C[l])} \cdot (M \cdot \mathtt{msl}(C[g]))^{\mathtt{mpl}(C[r])} \\ &= (M \cdot (\mathtt{msl}(C[g])))^{\mathtt{mpl}(C[g])}. \end{aligned}$$

■

Circuit Evaluation

The task of evaluating a circuit is to compute the element associated to its output. It is known that, even for the special case of boolean circuits, the evaluation problem is $\mathcal{P}$-hard [AB09, Section 6.5.2]. Recall from Section 1.3 that the $\mathtt{trun}_\ell(\cdot)$ operation accepts a real or complex number as parameter and removes all bits behind the decimal point that are not the $\ell - 1$ most significant ones. So we have that $|c - \mathtt{trun}_\ell(c)| \leq 2^{-l}$ for every c and ℓ. Now consider Algorithm 1. It performs a natural scheme to determine the output of the circuit except it truncates intermediate values.

Let a' and b' be estimations of a and b respectively, let $c' = \mathtt{trun}_\ell(a' + b')$ and $c = a + b$. Note that actually $c' = a' + b'$ since the truncation operation is not effective because the sum operation does not produce new bits less significant bits. Then we have that

$$|c' - c| \leq |a' - a| + |b' - b|. \tag{4.4}$$

Now, suppose $d' = \mathtt{trun}_\ell(a' \cdot b')$ and $d = a \cdot b$. This yields the following error bound:

$$|d' - d| \leq |a' - a| \cdot |b| + |b' - b| \cdot |a| + |a' - a| \cdot |b' - b| + 2^{-\ell}. \tag{4.5}$$

Function `evalTrunc`$(C = (D, \lambda', \mathbb{C}), \ell)$

Find a topological ordering $v_1, \ldots, v_n$ of V where $v_n = v$ is the output of C.
for $i = 1$ to n **do**
  if v_i is an input gate **then**
    Set $v'_i \leftarrow \lambda'(v_i)$
  else
    Let $l, r < i$ be such that $N^-(v_i) = \{v_l, v_r\}$.
    if $\lambda = +$ **then**
      Set $v'_i \leftarrow \mathtt{trun}_\ell(v'_l + v'_r)$
    else
      Set $v'_i \leftarrow \mathtt{trun}_\ell(v'_l \cdot v'_r)$
return v'_n

Algorithm 1 – Truncated evaluation of circuits over $\mathbb{C}$. See also Lemma 4.3

Here, the $2^{-\ell}$ term comes from the truncation operation. Equipped with (4.4) and (4.5) we are ready to bound the error of approximation obtained by Algorithm 1. See also the similar analysis of Knuth of the Schönhage-Strassen algorithm for integer multiplication[Knu69].

Lemma 4.3 Let $C = (D = (V, A), \lambda, \mathbb{C})$ be a circuit with output gate v, let $2 \le M, \ell \in \mathbb{N}$, and $\lambda' : V \to \mathbb{C}$ be such that and for every input gate g of C:

1. $|\lambda(g)| \le M$,
2. $|\lambda'(g) - \lambda(g)| \le 2^{-\ell}$.

Then, if $v' = \mathtt{evalTrunc}(C = (D, \lambda', \mathbb{C}), \ell)$ (see Algorithm 1), it holds that

$$|v' - v| \le 2^{-\ell}(4M \cdot \mathtt{msl}(C) \cdot |C|)^{\mathtt{mpl}(C)}. \tag{4.6}$$

Moreover, algorithm `evalTrunc` runs in time $|C|(\mathtt{mpl}(C)\mathrm{polylog}(\mathtt{msl}(C) \cdot M) + \tilde{\mathcal{O}}(\ell))$ and uses $|C|(\ell + \mathrm{polylog}(\mathtt{msl}(C) \cdot M))$ space.

Proof Let $\hat{C} = (D, \hat{\lambda}, \mathbb{R}_+)$ with $\hat{\lambda}(g) = |g|$ for every input gate g, and for a gate g of C, denote $\hat{g}$ for the corresponding gate in $\hat{C}$. Then trivially, $|g| \le \hat{g}$. Note that without loss of generality, we can assume that $\hat{v} > 0$ since otherwise the labels of all input must be 0 and then the lemma trivially holds. We will first prove the bound on $|v' - v|$ by proving the following induction hypothesis with induction on $|C[g]|$. Note that it implies (4.6) when combined with Observation 4.3.

Claim 4.6.1 For every gate g of C with $\hat{g} > 0$ it holds that $|g' - g| \le 2^{-\ell}\hat{g}(4|C[g]|)^{\mathtt{mpl}(C[g])}$.

Proof (of the claim) For the base case $|C| = 1$ we know that g is an input gate, and the induction hypothesis follows from assumption 2. For the case $|C| > 1$, we know that g is not an input gate so it has 2 in-neighbors l and r. If g is an addition gate, (4.4) yields

$$\begin{aligned}
|g' - g| &\le |l' - l| + |r' - r| \\
&\quad \big(\text{induction hypothesis}\big) \\
&\le 2^{-\ell}\hat{l}(4|C[l]|)^{\mathtt{mpl}(C[l])} + 2^{-\ell}\hat{r}(4|C[r]|)^{\mathtt{mpl}(C[r])} \\
&\quad \big(\text{using } \mathtt{mpl}(C[l]), \mathtt{mpl}(C[r]) \le \mathtt{mpl}(C[l]) \text{ and } |C[l]|, |C[r]| \le |C[g]| - 1\big) \\
&\le 2^{-\ell}\hat{l}(4(|C[g]| - 1))^{\mathtt{mpl}(C[g])} + 2^{-\ell}\hat{r}(4(|C[g]| - 1))^{\mathtt{mpl}(C[g])} \\
&\quad \big(\text{substituting using } |l| + |r| \le |g|\big) \\
&\le 2^{-\ell}\hat{g}(4(|C[g]| - 1))^{\mathtt{mpl}(C[g])} \le 2^{-\ell}\hat{g}(4|C[g]|)^{\mathtt{mpl}(C[g])}.
\end{aligned}$$

If g is a multiplication gate, (4.5) yields

$$|g' - g| \le |l' - l||r| + |r' - r||l| + |l' - l||r' - r| + 2^{-\ell}. \tag{4.7}$$

Let us first bound the terms on the right hand side in (4.7) separately:

$$\begin{aligned}
|l' - l||r' - r| &\le 2^{-2\ell}\hat{l}(4|C[l]|)^{\mathtt{mpl}(C[l])}\hat{r}(4|C[r]|)^{\mathtt{mpl}(C[r])} \\
&\quad \big(\text{using } |C[l]|, |C[r]| \le |C[g]|\big) \\
&\le 2^{-2\ell}\hat{g}(4|C[g]|)^{\mathtt{mpl}(C[l])}(4|C[g]|)^{\mathtt{mpl}(C[r])} \\
&\quad \big(\text{using } \mathtt{mpl}(C[l]) + \mathtt{mpl}(C[r]) = \mathtt{mpl}(C[g])\big) \\
&\le 2^{-2\ell}\hat{g}(4|C[g]|)^{\mathtt{mpl}(C[g])} \le \frac{1}{4}2^{-\ell}\hat{g}(4|C[g]|)^{\mathtt{mpl}(C[g])},
\end{aligned} \tag{4.8}$$

$$\begin{aligned}
|l' - l||r| + |r' - r||l| &\le 2^{-\ell}\left(\hat{l}(4|C[l]|)^{\mathtt{mpl}(C[l])}\hat{r} + \hat{r}(4|C[r]|)^{\mathtt{mpl}(C[r])}\hat{l}\right) \\
&\quad \big(\text{using } \hat{l}\cdot\hat{r} = \hat{g};\ \mathtt{mpl}(C[l]), \mathtt{mpl}(C[r]) \le \mathtt{mpl}(C[g]) - 1\big) \\
&\le 2^{-\ell}\hat{g}4^{\mathtt{mpl}(C[g])-1}\left(|C[l]|^{\mathtt{mpl}(C[g])} + |C[r]|^{\mathtt{mpl}(C[g])}\right) \\
&\quad \big(\text{using } |C[l]|, |C[r]| \le |C[g]|\big) \\
&\le 2^{-\ell}\hat{g}4^{\mathtt{mpl}(C[g])-1}2|C[g]|^{\mathtt{mpl}(C[g])} \\
&\le \frac{1}{2}2^{-\ell}\hat{g}4^{\mathtt{mpl}(C[g])}|C[g]|^{\mathtt{mpl}(C[g])}.
\end{aligned} \tag{4.9}$$

Now, substituting (4.8) and (4.9) into (4.7) gives

$$\begin{aligned}|g' - g| &\leq \frac{1}{2}2^{-\ell}\hat{g}(4|C[g]|)^{\mathtt{mpl}(C[g])} + \frac{1}{4}2^{-\ell}\hat{g}(4|C[g]|)^{\mathtt{mpl}(C[g])} + 2^{-\ell} \\ &\quad \big(\text{using } \hat{g}(4C[g])\mathtt{mpl}(C[g]) \geq 1\big) \\ &\leq 2^{-\ell}\hat{g}(4|C[g]|)^{\mathtt{mpl}(C[g])},\end{aligned}$$

completing the proof of the claim. ■

It remains to prove the bound on the running time, which is easy to verify since a topological ordering can be found in $|C|\lg(|C|)$ time (see for example [CLRS01, Section 22.4]) and all arithmetic operations can be performed in the claimed time-bound using fast integer multiplication since (see for example [DKSS08, Für09]) they are represented by at most $\ell + \lg(M \cdot \mathtt{msl}(C))^{\mathtt{mpl}(C)}$ by Observation 4.3. ■

4.7 Color-Coding and the Koutis-Williams Approach

Suppose we are given a set U and, implicitly, a set family $\mathcal{F} \subseteq 2^U$. For example, if $G = (V, E)$ is a graph U could be E and $\mathcal{F}$ could be the set of paths of G. Roughly speaking, the Color-Coding and Koutis-Williams approach are both hashing methods that aim to determine whether there exists a set $S \in \mathcal{F}$ with $|S| = k$ in time $\mathcal{O}^\star(c^k)$ for some constant c.

Multilinear Monomial Detection	**Parameter:** k
Input: A circuit C over $\mathbb{B}[\mathbb{D}^n]$ with singleton inputs outputting $\boldsymbol{v}$, integer k.	
Question: Does there exist $\boldsymbol{e} \in \mathbb{D}^n$ such that $v_{\boldsymbol{e}} = \mathtt{true}$ and $\lvert\mathrm{supp}(\boldsymbol{e})\rvert = k$?	

We will need to use the following theorem as a blackbox (a proof follows for example from the methods from Chapter 5).

Theorem 4.7 — Folklore If $k = n$, Multilinear Monomial Detection can be solved in $\mathcal{O}^\star(2^k\mathtt{mpl}(C))$ time and $\mathcal{O}^\star(1)$ space.

It is worth to mention that it is possible to avoid the $\mathtt{mpl}(C)$ term if exponential space or randomization is allowed. We will also need the following:

Definition 4.12 A *k-perfect family of hash functions* $\mathcal{F}$ is a family of functions $\mathcal{F} = \{f : \{1, \ldots, n\} \to \{1, \ldots, k\}\}$ such for for every subset $X \subseteq \{1, \ldots, k\}$ there exists $f \in \mathcal{F}$ such that $f^{-1}(X) = \{1, \ldots, k\}$ (or equivalently, for every $x, y \in X$ it holds that $f(x) = f(y) \to x = y$).

Theorem 4.8 — (NSS95) For every n and k, there exist a k-perfect family of hash functions $\mathcal{F} = \{f : \{1,\dots,n\} \to \{1,\dots,k\}\}$ of size $e^k k^{O(\log k)} \log n$ that can be constructed in $e^k k^{O(\log k)} \text{poly}(n,k)$ time.

See also the textbook [FG06] for a proof of a similar result. Now we are ready to apply the Color-Coding approach to the Multilinear Monomial Detection problem.

Theorem 4.9 — (AYZ95) Multilinear Monomial Detection can be solved in $\mathcal{O}^\star((2e)^k \texttt{mpl}(C))$ time and $\mathcal{O}^\star(1)$ space.

Proof (sketch) Let $\{1,\dots,n\} = U$ for simplicity. The general idea is the following: we are looking for a subset $X \subseteq U$ with $|X| = k$ such that $v_X = \texttt{true}$. We transform the given instance by iteratively applying each hash function of $\mathcal{F}$ obtained from Theorem 4.8 to U. For every $e \in U$, choose a *color* $c(e) \in \{1,\dots,k\}$ at random. Define $\gamma : \mathbb{B}[\mathbb{D}^n] \to \mathbb{B}[\mathbb{D}^k]$ be the function such that for every $X \in \mathbb{D}^k$

$$\gamma(\boldsymbol{a})_X = \bigvee_{Y \subseteq U} (|X| = |Y|) \wedge (c(Y) = X) \wedge a_X.$$

Construct the circuit C' obtained from C by applying γ. Since C has singleton inputs, this can be done in polynomial time. Also, since γ is a homomorphism (this is non-trivial but we skip the derivation; it basically follows from the fact that c is a homomorphism from $\mathbb{D}^n$ to $\mathbb{D}^k$), the output of C' is $\gamma(\boldsymbol{v})$ by Observation 4.1. Then, it is easy to see that if $\gamma(\boldsymbol{v})_{\{1,\dots,k\}} = \texttt{true}$, the original instance is a YES-instance. Moreover, if there exists $X \subseteq U$, then there exists an $c \in \mathcal{F}$ such that $|X| = k$ $c(X) = \{1,\dots,k\}$, and then it is easy to see that $\gamma(\boldsymbol{v}) = \texttt{true}$. Hence, we can reduce the instance of Multilinear Monomial Detection to $|\mathcal{F}|$ instances where $k = n$. Then these can be solved using Theorem 4.7 and we know the answer of the given instance is YES if and only if at least one of the created instances is YES. ■

It should also be noted that variations of the Color-Coding technique that improve in special cases of the Multilinear Monomial Detection problem such as k–Path extension have been studied in [CKL+09] (see also [FLGS10]). Now we proceed to the Koutis-Williams approach. To explain this approach, it is useful to introduce some terminology that might explain the name of the problem studied.

Definition 4.13 A *monomial* M is a multisubset of U, and M is said to be *multilinear* if it is also a set. A *colored monomial* is a pair (M,γ) where M is a monomial and $\gamma : M \to K$. Additionally, a colored monomial (M,γ) is called *colorful* if $|\gamma(M)| = |M|$ (that is, γ is a bijection).

Recall from Section 1.3 the subtle definition of a function with a multiset as its codomain, and note that different copies of the same element of U are treated as distinct ones: coloring the first copy with 3 and the second with 4 is the different from the reverse. Hence, the number of different functions γ defined as above is $|K|^{|M|}$, as it is with normal functions.

Lemma 4.4 — Koutis,Williams (Kou08, KW09, Wil09) There exists an algorithm that given a circuit $C = (D = (V, A), \lambda, \mathbb{B}[\mathbb{N}^n])$ with singleton inputs outputting $\boldsymbol{v}$ and an integer k, constructs a circuit $C' = (D' = (V', A'), \lambda', \mathbb{Z}_2[\mathbb{D}^k \times \mathbb{Z}_{nk}])$ outputting $\boldsymbol{v}'$ in time $\text{poly}(|C|, n)$ such that

1. if there exists $\boldsymbol{e} \in \mathbb{D}^n$ such that $v_{\boldsymbol{e}} = \texttt{true}$ and $|\text{supp}(\boldsymbol{e})| = k$, then with probability at least $\frac{1}{2}$ it holds that $v'_{(\{1,\ldots,k\},w)} = 1$ for some integer $w \leq 2nk$,
2. otherwise $v'_{(\{1,\ldots,k\},w)} = 0$ for every $w \leq 2nk$.

We use the coloring technique used by Björklund [Bjö10b] (although we replace the original term 'labels' with 'colors' to avoid confusion) in combination with Lemma 4.2.

Proof (of Lemma 4.4, sketch) Note that we can assume without loss of generality that for every input gate g, $\lambda(g) = \langle \texttt{true}, e \rangle$ for some $e \in U$: given an input gate g with $\lambda(g) = \langle i, X \rangle$ where $X = \{e_1, \ldots, e_{|X|}\}$, we can replace it with $\langle i, \varnothing \rangle \prod_{i=1}^{n} \langle 1, e_i \rangle$. Now, for every element $e \in U$ and color $c \in \{1, \ldots, k\}$, choose a weight $\omega(e, c) \in \{1, \ldots, 2n\}$ uniformly at random.

Let $h : \mathbb{B}[\mathbb{D}^n] \to \mathbb{Z}_2[\mathbb{D}^k \times \mathbb{Z}_{nk}]$ be the function such that for every $\boldsymbol{a} \in \mathbb{B}[\mathbb{D}^n]$, $X \subseteq U$, and integer W

$$h(\boldsymbol{a})_{X,W} = \sum_{Y \subseteq U} \sum_{\gamma : Y \leftrightarrow X} \Big[\sum_{e \in X} \omega(e, \gamma(e)) = W \Big] a_Y. \tag{4.10}$$

Note that here and in the following, expressions like a_Y are boolean expression but actually are interpreted as elements of $\mathbb{Z}_2$ as $[a_Y]$ (that is, it denotes 1 if b_Y is `true` and 0 if a_Y is `false`).

Claim 4.9.1 h is a homomorphism.

Proof (of the Claim) It is easy to see that $\big(h(\boldsymbol{a}) + h(\boldsymbol{b})\big)_{X,W} = h(\boldsymbol{a} + \boldsymbol{b})_{X,W}$.Moreover,

$h(\boldsymbol{a} * \boldsymbol{b})_{X,W} =$

$$\sum_{Y \subseteq U} \sum_{\gamma : Y \leftrightarrow X} \Big[\sum_{e \in X} \omega(e, \gamma(e)) = W \Big] \sum_{Y_1 \uplus Y_2 = Y} a_{Y_1} b_{Y_2}$$

(Partition X, Y and γ into two parts, reorder summations.)

$$= \sum_{X_1 \uplus X_2 = X} \sum_{\substack{Y_1, Y_2 \subseteq U \\ Y_1 \cap Y_2 = \varnothing}} \sum_{\substack{\gamma_1 : Y_1 \leftrightarrow X_1 \\ \gamma_2 : Y_2 \leftrightarrow X_2}} \Big[\sum_{e \in X_1} \omega(e, \gamma_1(e)) + \sum_{e \in X_2} \omega(e, \gamma_2(e)) = W \Big] a_{Y_1} b_{Y_2}$$

(Guessing partition of W into W_1,W_2.)

$$= \sum_{\substack{X_1 \uplus X_2 = X \\ W_1 + W_2 = W}} \sum_{\substack{Y_1, Y_2 \subseteq U \\ Y_1 \cap Y_2 = \varnothing}} \sum_{\substack{\gamma_1 : Y_1 \leftrightarrow X_1 \\ \gamma_2 : Y_2 \leftrightarrow X_2}} \Big[\sum_{e \in X_1} \omega(e, \gamma_1(e)) = W_1 \Big] a_{Y_1} \cdot \Big[\sum_{e \in X_2} \omega(e, \gamma_2(e)) = W_2 \Big] b_{Y_2}$$

(See the comment below.)

$$\equiv \sum_{\substack{X_1 \uplus X_2 = X \\ W_1 + W_2 = W}} \sum_{Y_1, Y_2 \subseteq U} \sum_{\substack{\gamma_1 : Y_1 \leftrightarrow X_1 \\ \gamma_2 : Y_2 \leftrightarrow X_2}} \Big[\sum_{e \in X_1} \omega(e, \gamma_1(e)) = W_1 \Big] a_{Y_1} \cdot \Big[\sum_{e \in X_2} \omega(e, \gamma_2(e)) = W_2 \Big] b_{Y_2}$$

(Collecting dependent variables together and applying definition of h.)

$$= \sum_{\substack{X_1 \uplus X_2 = X \\ W_1 + W_2 = W}} h(a)_{X_1, W_1} \cdot h(b)_{X_2, W_2}$$

$$= \big(h(\boldsymbol{a}) * h(\boldsymbol{b})\big)_{X,W}.$$

To see that the congruence holds, notice that the quantity

$$\sum_{\substack{X_1 \uplus X_2 = X \\ W_1 + W_2 = W}} \sum_{\substack{Y_1, Y_2 \subseteq U \\ Y_1 \cap Y_2 \neq \varnothing}} \sum_{\substack{\gamma_1 : Y_1 \leftrightarrow X_1 \\ \gamma_2 : Y_2 \leftrightarrow X_2}} \Big[\sum_{e \in X_1} \omega(e, \gamma_1(e)) = W_1 \Big] a_{Y_1} \cdot \Big[\sum_{e \in X_2} \omega(e, \gamma_2(e)) = W_2 \Big] b_{Y_2}$$

(that is, the sum of the terms with intersecting Y_1 and Y_2 at the right-hand of the congruence) is always even since if we have two mappings γ_1, γ_2, we can take all elements in the intersection of Y_1 and Y_2 and exchange their images. It is easy to see that this also results in a different term with non-zero contribution since X_1, X_2 are disjoint. ■

Now, let C' be the circuit obtained from C by applying the homomorphism h. Note that this can easily be constructed in $\mathcal{O}^\star(1)$ time. Let $\boldsymbol{v}'$ be the output of C'. By Observation 4.1 and Claim 4.9.1, $\boldsymbol{v}' = h(\boldsymbol{v})$. Now for item 2 of the lemma, notice that in (4.10) only summands Y with $|Y| = |X|$ can have a non-zero contribution since otherwise there are no bijections γ. For item 1: if there exists $Y \in \mathbb{D}^n$ such that $v_{e=\texttt{true}}$ and $|\mathrm{supp}(\boldsymbol{e})| = k$, then we can apply Lemma 4.2. Let the universe be $U \times \{1, \ldots, k\}$; since the set of all bijections γ can be interpreted as a family of subsets of $U \times \{1, \ldots, k\}$, Lemma 4.2 states that with probability at least $\frac{1}{2}$ there exists a W such that $h(\boldsymbol{v})_{\{1,\ldots,k\},W} = 1$. ■

Using a small variant of Theorem 4.7 (see Chapter 5 for the relevant techniques) in combination with Lemma 4.4, the following can be obtained:

Corollary 4.1 — **Koutis,Williams ((Kou08, KW09, Wil09))** Multilinear Monomial Detection can be solved in $\mathcal{O}^\star(2^k)$ time by a randomized algorithm with constant one-sided error probability.

Part III

Main Contribution

Chapter 5

Saving Space by Algebraization

This chapter is based on the following paper:

[LN10]. Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In Leonard J. Schulman, editor, *STOC*, pages 321–330. ACM, 2010

Recall that an inherent property of dynamic programming algorithms is that they require a relatively big amount of working memory. This is because often many previously computed table entries are required for efficient computation of new entries. In this chapter we identify sufficient conditions for being able to turn dynamic programming algorithms into algorithms with roughly the same running time and significantly lower space usage. In particular, we show that if a dynamic programming algorithm can be formalized as a relatively short expression in one of some specific types of rings, then one can use algebraic transformations to evaluate the relevant part of the expression in a space-efficient and quick manner.

In Chapter 2 we saw one of the two canonical applications of dynamic programming, namely the Subset Sum and Knapsack problems. The algorithms by Bellman [Bel54], as explained in Section 2.1, are the most famous and fastest algorithms for these problems; especially since dynamic programming is the only known method able to solve these problems in pseudo-polynomial[(1)] time. However, as also mentioned before, these algorithms use almost as much space as time.

So, a natural and fundamental question is whether it is possible to have a pseudo-polynomial time algorithm for Subset Sum using only polynomial space. In this chapter we answer this question affirmatively by showing that Bellman's dynamic programming algorithm can be encoded as a relatively easy expression in a specific large ring, and that it is possible to reduce the space requirement of any dynamic programming algorithm of such type. In particular, we obtain an

[(1)] polynomial in the input size if integers are given in unary

algorithm with running time $\tilde{\mathcal{O}}(n^3t)$ time and $\tilde{\mathcal{O}}(n^2)$ space. For the more general Knapsack problem we give a $\tilde{\mathcal{O}}(n^4vw)$-time $\tilde{\mathcal{O}}(n^2)$-space algorithm.

Our methodology enables us to make many exponential-time and exponential-space dynamic programming algorithms run in polynomial space instead. This is useful because algorithms using both exponential time and space can run out of space long before they run out of time. Some old and recent results in the direction of polynomial-space exponential-time algorithms for $\mathcal{NP}$-hard problems include Karp's $\mathcal{O}^*(2^n)$ time algorithm [Kar82] for Hamiltonian Path (see also Section 3.1), Björklund, Husfeldt and Koivisto's Set Cover(2) algorithm [BHK09] running in time $\mathcal{O}^*(2^n)$, and the $\mathcal{O}^*(2^k)$ time algorithm for Steiner Tree with unit weights as introduced in Section 3.2.1. Our method unifies these results, and improves algorithms for weighted variants of these problems.

An interesting open problem is whether Traveling Salesman can be solved in $O^*(2^n)$ time and polynomial space [Woe04]. We make progress towards resolving this problem by giving a polynomial-space algorithm for Traveling Salesman running in $O^*(2^nt)$ time. For the two other problems, Weighted Set Cover and Weighted Steiner Tree we make similar improvements. In particular, our algorithms match the time bound of the best known pseudo-polynomial space algorithms for these problems.

This chapter is organized as follows. In Section 5.1 we give sufficient conditions to be able to save space for dynamic programming on tables indexed by integers (so-called numerical dynamic programming). In Section 5.2 we give sufficient conditions to be able to save space for dynamic programming on tables indexed by subsets. Then, in Section 5.3 we combine the results from the two previous sections. All the three sections will be concluded by a few applications.

Methodology and Contribution

Perhaps the best name for the mainly used method in this chapter is 'coefficient extraction' (see also [Lip10]): the idea is to express the given computational task as a relatively small circuit (see Definition 4.7) over a large ring. This ring typically consists of large vectors and the computational task is reduced to determine a specific element of the vector output by the small circuit, the so-called *coefficient*. Then transformations are used to determine the coefficient in a manner more efficient than trivially possible. All preliminary knowledge is already given in Chapter 4, so in this chapter it only remains to put all the pieces together. It should be noted that this elementary and generic scheme is already very old and often used ([CU03, CKSU05, CT65, Für09, Kou08, Man95], to name just a few). Still, we feel that our contribution is three-fold: (i) we make an explicit and formal study of the generic setting, (ii) we give some new applications, and (iii) we show that the small circuit used to solve a task often is directly implied by the known dynamic programming algorithm.

(2) with a polynomial number of sets

5.1 Numerical Dynamic Programming

In this section we will study circuits over the ring $\mathbb{Z}[\mathcal{A}]$, where $\mathcal{A}$ is an Abelian group. As we will see at the end of this section, these arguably represent the most famous applications of dynamic programming. We will give an algorithm more efficient than the trivial one by exploiting the Discrete Fourier Transform (DFT) (see Section 4.3). It should be noted that the main technicality here stems from the fact that the DFT requires that the base field (in this case $\mathbb{Z}$) in which the arithmetic finally is performed contains specific roots of unity. To overcome this, we use the following common approach of working with complex numbers rather than integers. However, since we work in the log-cost model (see Section 1.2), we have to work with approximations of the appropriate complex numbers since the roots of unity require infinite precision to store and work with exactly. Fortunately all the preliminary work is already done in Chapter 4 and we can now directly prove the main theorem of this section.

Theorem 5.1 There exists an algorithm that, given $\mathcal{A} = \mathbb{Z}_{m_1} \oplus \mathbb{Z}_{m_2} \oplus \ldots \oplus \mathbb{Z}_{m_\delta}$ and a circuit $C = (D, \lambda, \mathbb{Z}[\mathcal{A}])$ with output gate $\boldsymbol{v}$ and $t \in \mathcal{A}$, computes v_t using at most

$$|\mathcal{A}| \cdot \tilde{\mathcal{O}}(\lg|\mathcal{A}| \cdot |C| \cdot \texttt{mpl}(C) \cdot \text{polylog}(M \cdot \texttt{msl}(C) \cdot |C| \cdot) \lg M) \text{ time and}$$

$$\mathcal{O}\big(|C|(\texttt{mpl}(C) \lg(M \cdot \texttt{msl}(C) \cdot |C|) + \lg|\mathcal{A}|)\big) \text{ space,}$$

where $M = \max_{v \in (C)}\{x : \lambda(v) = \langle v, i\rangle\}$. Here we assume a group element $e \in \mathcal{A}$ is given as input by δ integers $(e_1, e_2, \ldots, e_\delta) = e \in \mathcal{A}$.

Proof We use `extractFourier` as described in Algorithm 2.

Function $\texttt{extractFourier}(C = (D, \mathbb{Z}[\mathbb{Z}_{m_1} \oplus \mathbb{Z}_{m_2} \oplus \ldots \oplus \mathbb{Z}_{m_\delta}], \lambda), \boldsymbol{t})$

1: Let $\ell = 2 \cdot \texttt{mpl}(C) \cdot \lg(4M \cdot \texttt{msl}(C) \cdot |C|)$. Set precision for truncation.

2: **return** $\texttt{round}\left(\frac{1}{|A|} \sum_{x_1=0}^{m_1-1} \cdots \sum_{x_\delta=0}^{m_\delta-1} \texttt{trun}_2\Big(\texttt{sub}(C, (x_1, \ldots, x_\delta), \ell) \cdot \texttt{trun}_\ell\big(\prod_{d=1}^{\delta} \omega_{m_d}^{x_d t_d}\big)\Big)\right)$

Using the Fourier inversion formula with sufficiently good approximations

Function $\texttt{sub}(C = (D, \mathcal{A}, \lambda), x, \ell)$ Return approximation of $(\boldsymbol{v}\boldsymbol{F}_{\boldsymbol{v}})_{\boldsymbol{x}}$.

3: **for** every input gate g of C **do**

4: Set $\kappa(g) \leftarrow \texttt{trun}_\ell\big(v \cdot \prod_{d=1}^{\delta} \omega_{m_d}^{e_d}\big)$, where $\langle v, \boldsymbol{e}\rangle = \lambda(g)$.

5: **return** $\texttt{evalTrunc}((D, \kappa, \mathbb{C}), \ell)$ See Algorithm 1.

Algorithm 2 – Implementing Theorem 5.1

For the analysis of Algorithm 2, we first analyse its correctness, and discuss its running time afterwards.

Recall that $\boldsymbol{F_m}$ is the Fourier matrix from Definition 4.4, so then considering Line 4 of Algorithm 2, we see that $\lim_{\ell\to\infty}\kappa(g) = \boldsymbol{g}\boldsymbol{F_m}$. Since $\boldsymbol{F_m}$ is a homomorphism from $\mathbb{C}[\mathcal{A}]$ to $\mathbb{C}$ by Theorem 4.1, it follows from Observation 4.1 and Lemma 4.3 that $\lim_{\ell\to\infty}\mathtt{sub}(C,x,\ell) = (\boldsymbol{v}\boldsymbol{F_m})_{\boldsymbol{x}}$. More precisely, we apply Lemma 4.3 assuming $M \geq 2$: it is easy to verify that both conditions of the lemma are met, so it follows that

$$|(\boldsymbol{v}\boldsymbol{F_m})_{\boldsymbol{x}} - \mathtt{sub}(C,x,\ell)| \leq 2^{-\ell}(4M\cdot\mathtt{msl}(C)\cdot|C|)^{\mathtt{mpl}(C)} \leq 2^{-\frac{1}{2}\ell}. \tag{5.1}$$

Letting ℓ go to infinity again, it follows from Lemma 4.1 that the expression returned on Line 2 converges to

$$(\boldsymbol{v}\boldsymbol{F_m})(\boldsymbol{F_m^{-1}})^{(\boldsymbol{t})} = \boldsymbol{v}(\boldsymbol{F_m^{-1}}\boldsymbol{F_m})^{(\boldsymbol{t})} = \boldsymbol{v}\boldsymbol{I}^{(\boldsymbol{t})} = v_{\boldsymbol{t}}.$$

Hence for proving the correctness of Algorithm `extractFourier`, it remains to show that ℓ is chosen large enough in order to make sure that the estimate it close enough. Since we know that $v_{\boldsymbol{t}}$ is integer, we can round the estimate to the closest integer in the end. Hence it suffices to show that the error is smaller than $\frac{1}{2}$. This straightforwardly implies that it is sufficient that for every summand, the estimation error of the quantity returned by $\mathtt{trun}_2$ is smaller than $\frac{1}{2}$, or even stronger, that the estimation error of the input given to the $\mathtt{trun}_2$ operation is at most $\frac{1}{4}$. The latter follows from (4.5) and (5.1).

For the implementation and running time of Algorithm 2 we first remark that for Line 4, $\mathtt{trun}_\ell(v\cdot\prod_{d=1}^{\delta}\omega_{m_d}^{e_d})$ can be computed in time $\tilde{\mathcal{O}}((\ell+\lg M)\cdot\lg|\mathcal{A}|)$ and $\mathcal{O}(\ell+\lg M)$ space using floating point arithmetic and standard Taylor approximations of the basic functions $\cos, \sin$ and π to compute $\omega_{m_d}^{e_d}$, and similarly the quantity $\prod_{d=1}^{\delta}\omega_{m_d}^{e_d}$ on Line 2 can be computed in the same resource bounds. Due to Lemma 4.3, it follows that Line 5 takes $|C|(\mathtt{mpl}(C)\mathrm{polylog}(M\cdot\mathtt{msl}(C)\cdot|C|)+\tilde{\mathcal{O}}(\ell))$ time and uses space proportional to $|C|(\ell+\mathrm{polylog}(\mathtt{msl}\cdot M))$. Then it follows from straightforward analysis that the algorithm takes

$$\begin{aligned}
&|\mathcal{A}|\cdot|C|\Big(\mathtt{mpl}(C)\mathrm{polylog}(M\cdot\mathtt{msl}(C)\cdot|C|)+\tilde{\mathcal{O}}((\ell+\lg M)\cdot\lg|\mathcal{A}|)\Big)\\
&\leq|\mathcal{A}|\cdot\tilde{\mathcal{O}}(\lg|\mathcal{A}|\cdot|C|\cdot\mathtt{mpl}(C)\cdot\mathrm{polylog}(M\cdot\mathtt{msl}(C)\cdot|C|\cdot)\lg M) \text{ time, and}\\
&|C|(\ell+\mathrm{polylog}(\mathtt{msl}\cdot M))+\lg|\mathcal{A}|\\
&\leq 2|C|(\mathtt{mpl}(C)\lg(M\cdot\mathtt{msl}(C)\cdot|C|)+\lg|\mathcal{A}|) \text{ space.}
\end{aligned}$$

■

It is worth mentioning that a result similar to Theorem 5.1 can be obtained using modular arithmetic instead of complex numbers (for a similar approach see [DKSS08]). In another variant, if one only is interested in determining whether $v_t = 0$, then it is possible to use randomization in the spirit of Section 4.5 to get rid of the $\mathtt{mpl}(C)$ in the exponent and replace it with a multiplicative term polynomial in $|C|$.

5.1.1 Applications

The Subset Sum Problem

We now show how to use Theorem 5.1 to give a pseudo-polynomial-time, polynomial-space algorithm for the Subset Sum problem, defined below.

Subset Sum	**Parameter:** t
Input: Set $S = \{1, \ldots, n\}$ and function $\omega : S \to \mathbb{N}$ and and integer t	
Question: Does there exist a subset $X \subseteq S$ such that $\omega(X) = t$?	

The Subset Sum can trivially be solved in $\mathcal{O}(2^n)$ time and polynomial space. This has been improved to $\mathcal{O}^*(2^{n/2})$ time and space in [HS74] and to $\mathcal{O}^*(2^{n/2})$ time and $\mathcal{O}^*(2^{n/4})$ space in [SS79]. Since an element $e \in S$ with $w(e) > t$ cannot participate in any solution we will assume that $w(e) \leq t$ for every $e \in S$. Also, if $t > 2^n$ then the trivial $\mathcal{O}(2^n)$ time algorithm runs in $\mathcal{O}(t)$ time and polynomial space. For every $1 \leq i \leq n$ and $x \leq nt$ let $\boldsymbol{s}(i) \in \mathbb{N}^{nt}$ be the vector such that for every j, $s(i)_j$ is the number of subsets $S \subseteq \{1, \ldots, i\}$ such that $\sum_{j \in S} w_j = x$. Recall the $\mathcal{O}(nt)$ time, $\mathcal{O}(t)$ space dynamic programming algorithm by Bellman, also discussed in Section 2.1, which can be formulated as the following recurrence:

$$s(i)_j = \begin{cases} [j = 0] & \text{if } i = 0, \\ s(i-1)_j + s(i-1)_{j-w_i} & \text{otherwise.} \end{cases}$$

Now we will reformulate this recurrence into a recurrence over the ring $\mathbb{Z}[\mathbb{Z}_{nt}]$ in order to be able to apply Theorem 5.1. To this end, we interpret $\boldsymbol{s}(i)$ as an element of $\mathbb{Z}[\mathbb{Z}_{nt}]$, obtaining the following:

$$\boldsymbol{s}(i) = \begin{cases} \langle 1, 0 \rangle & \text{if } i = 0, \\ \boldsymbol{s}(i-1) * (\langle 1, 0 \rangle + \langle 1, w_i \rangle) & \text{otherwise.} \end{cases} \tag{5.3}$$

We are interested to know whether $s(n)_t \neq 0$, or for the counting variant of Subset Sum, even to compute $s(n)_t$ itself. A recurrence like (5.3) can naturally be seen as a circuit C over the ring $\mathbb{Z}[\mathbb{Z}_{nt}]$. Moreover trivially, $|C|$ is $\mathcal{O}(n)$ and $\lambda(g) \leq 2$ for every input gate g, and using Observation 4.2, it is easy to see that $\mathtt{msl}(C) = 2$ and $\mathtt{mpl}(C) = n - 1$. Then applying Theorem 5.1, setting $\mathcal{A} = \mathbb{Z}_{nt}$, $M = 2$ and using the fact that we can assume without loss of generality that $t < 2^n$, we obtain the following result:

Theorem 5.2 Subset Sum can be solved in $\tilde{\mathcal{O}}(n^3 t)$ time and $\tilde{\mathcal{O}}(n^2 \lg t)$ space.

It is also worth mentioning that, if we use the unit cost model instead of the log-cost model (see Section 1.2), the running time of the above algorithm is $\tilde{\mathcal{O}}(n^2 t)$.

Knapsack

Now we show that essentially the same approach as in the previous section can be taken to solve the Knapsack problem in pseudo-polynomial time using polynomial space. The Knapsack problem is formally defined as follows:

Knapsack	**Parameter:** v, w
Input: Set $S = \{1, \ldots, n\}$, functions $\nu, \omega : S \to \mathbb{N}$ and integers v, w.	
Question: Is there a subset $X \subseteq S$ such that $\nu(X) \geq v$ and $\omega(X) \leq w$?	

Note that the problem boils down to the Subset Sum problem in the special case where $v = w$ and it holds for every $e \in S$ that $\nu(e) = \omega(e)$. Similarly to the Subset Sum problem, there is a trivial brute force algorithm running in $\mathcal{O}(2^n)$ time and polynomial space, and the $\mathcal{O}^*(2^{n/2})$ time and space in [HS74] and $\mathcal{O}^*(2^{n/2})$ time and $\mathcal{O}^*(2^{n/4})$ space in [SS79] apply as well to this more general problem.

Also like before, any item with weight more than w cannot participate in the solution, and any item with weight at most w and value at least v constitutes a solution by itself. Hence we can assume without loss of generality that $\log w \leq n$, $\log v \leq n$ and that $\omega(e) \leq w$ and $\nu(e) \leq v$ for every $e \in S$. For every $1 \leq i \leq n$ we define $s(i) \in \mathcal{M}_{-}^{nv+1}$, where

$$s(i)_y = \min\{\omega(X) : X \subseteq S \wedge \nu(X) \geq y\}.$$

Then the dynamic programming algorithm of Bellman [Bel54] uses the following recurrence:

$$s(i)_y = \begin{cases} [y = 0] & \text{if } i = 0, \\ \min\{s(i-1)_y, \omega(i) + s(i-1)_{y-\nu(i)}\} & \text{if } i < 0. \end{cases}$$

It is easy to see that this is indeed correct. Using evaluation of this recurrence in a straightforward manner, Bellman obtained a $\mathcal{O}^*(v)$ time and space algorithm (an $\mathcal{O}^*(w)$ time and space algorithm could also be obtained in a very similar manner). In order to apply Theorem 5.1 to obtain a space-efficient algorithm, we get rid of the minimization operator by embedding it into a polynomial ring like discussed in Section 4.1. Instead of the above, we define for every $1 \leq i \leq n$, $s(i) \in \mathbb{N}[\mathbb{Z}_{nw+1} \times \mathbb{Z}_{nv+1}]$, where $s(i)_{x,y} = |\{X \subseteq S : \omega(X) = x \wedge \nu(X) = y\}|$. Then, the recurrence formulated in the ring $\mathbb{N}[\mathbb{Z}_{nw+1} \times \mathbb{Z}_{nv+1}]$ for $\boldsymbol{s}(i)$ is as follows:

$$\boldsymbol{s}(i) = \begin{cases} \langle 1, (0,0) \rangle & \text{if } i = 0, \\ \Big(\big\langle 1, \big(\omega(e), \nu(e)\big) \big\rangle + \langle 1, 0 \rangle \Big) * \boldsymbol{s}(i-1) & \text{if } i < 0. \end{cases} \tag{5.4}$$

Note that we are interested in whether there exists $x \leq v, y \geq w$ such that $s(n)_{x,y} \neq 0$. We could have used Theorem 5.1 with circuit implied by (5.4) to

check $s(n)_{x,y} \neq 0$ for each $x \leq v, y \leq w$, but that creates an unnecessary overhead of $\mathcal{O}(vw)$ in the running time. To overcome this issue, instead we work with a circuit C obtained by multiplying the outcome of the circuit implied by (5.4) with the vector $\boldsymbol{c} \in \mathbb{N}[\mathbb{Z}_{nw+1} \times \mathbb{Z}_{nv+1}]$, defined by $c_{x,y} = [x \leq v][y \geq w]$ for every x, y. Then it is easy to see that if the output of C is $\boldsymbol{s}$, then $s_{v,w} > 0$ if and only if the given instance is a YES-instance. To implement this, we need a (small) circuit $\hat{C}$ that outputs $\boldsymbol{c}$. For that we will use the following observation.

Observation 5.1 For every integer p and $p < q$ there are circuits $C^{\leq}$ and $C^{\geq}$ over $(\mathbb{Z}[\mathbb{Z}_q]; \oplus, \otimes)$ of size $\mathcal{O}(\log(p))$ outputting $a(p)$ and $b(p)$ respectively, such that for every $x \in \mathbb{Z}_q$ it holds that $a(p)_x = [0 \leq x \leq p]$ and $b(p)_x = [p \leq x < q]$. Moreover, $\mathtt{mpl}(C^{\leq}), \mathtt{mpl}(C^{\geq}), \mathtt{msl}(C^{\leq})$ and $\mathtt{msl}(C^{\geq})$ are $\mathcal{O}(\lg p)$.

Proof It is easy to see that the observation can be implemented by constructing circuits according to the following recurrences:

$$\boldsymbol{a}(p) = \begin{cases} \langle 1, 0 \rangle & \text{if } p = 0, \\ \boldsymbol{a}(p/2) * (\langle 1, p/2 \rangle + \langle 1, 0 \rangle) & \text{if } p \text{ is even}, \\ \boldsymbol{a}(p-1) + \langle 1, p \rangle & \text{if } p \text{ is odd}, \end{cases}$$

$$\boldsymbol{b}(p) = \begin{cases} \langle 1, q-1 \rangle & \text{if } p = q-1, \\ \boldsymbol{b}(p/2) * (\langle 1, -p/2 \rangle + \langle 1, 0 \rangle) & \text{if } p \text{ is even}, \\ \boldsymbol{b}(p-1) + \langle 1, -p \rangle & \text{if } p \text{ is odd}. \end{cases}$$

Also, the bounds on the maximum product and sum length claimed are easy to verify. ■

Of course, Observation 5.1 can also be used for circuits over $\mathbb{N}[\mathbb{Z}_{nv+1} \times \mathbb{Z}_{nw+1}]$. Hence, we can obtain the aforementioned circuit C by multiplying the output of the circuit implied by (5.4) by $a(v)$ and $b(w)$ and let the output of C be the result. Then, it is easy to see that we have a circuit C with $|C|$ being $\mathcal{O}(n + \lg v + \lg w) \in \mathcal{O}(n)$, and $\mathtt{mpl}(\hat{C})$ and $\mathtt{msl}(\hat{C})$ being $\mathcal{O}(n)$. Moreover, as discussed above Observation 5.1, $s_{v,w} > 0$ if and only if the current instance is a YES-instance. Applying Theorem 5.1 with $M = 2, |C| \in \mathcal{O}(n)$, $\mathtt{mpl}(C), \mathtt{msl}(C) \in \mathcal{O}(n)$ and $|\mathcal{A}| = n^2vw$ gives:

Theorem 5.3 The KNAPSACK problem can be solved in $\tilde{\mathcal{O}}(n^4vw)$ time and $\tilde{\mathcal{O}}(n^2 \log(vw))$ space.

5.2 Saving Space by Möbius Inversion

In this section we will prove a general theorem that identifies sufficient conditions for turning exponential space dynamic programming algorithms over the subset lattice into polynomial space algorithms based on Möbius inversion. While the theorem does not yield new polynomial space algorithms, it unifies and generalizes several well-known results, such as the Unweighted Set Cover algorithm by Björklund et al [BHK09], and the Hamiltonian Path algorithm by Karp [Kar82]. This includes most of the algorithms discussed in Chapter 3, in the sense that they follow from relatively straightforward dynamic programming algorithms that were previously known (see Subsection 5.2.2 for an example with Unweighted Steiner Tree).

5.2.1 Dynamic Programming over Subsets without Tables

Lemma 5.1 Let $\mathcal{R}$ be a ring and $C = (D, \lambda, \mathcal{R}[\mathbb{U}^n])$ with singleton inputs and output $\boldsymbol{v}$. Then, there is an algorithm running in time polynomial in $|C|$ and $\lg|\mathcal{R}|$ that, given $Y \subseteq \{1, \ldots, n\}$, creates a circuit $C^Y = (D, \lambda^Y, \mathcal{R}^{\mathbb{U}^n})$, such that C^Y outputs $(\boldsymbol{a\zeta})_Y$.

Proof Let $\lambda^Y(a) = \lambda(a)$ if a is not an input gate, and $\lambda^Y(a)_X = [X \subseteq Y]e$ if a is an input gate and $\lambda(a) = \langle e, X\rangle$, where $e \in \mathcal{R}$. It is easy to see that this can be done in time polynomial in $|C|$ and $\lg|\mathcal{R}|$. By Theorem 4.5 we know that $\boldsymbol{\zeta}$ is a homomorphism from $\mathcal{R}[\mathbb{U}^n]$ to $\mathcal{R}^{\mathbb{U}^n}$ and since multiplication in $\mathcal{R}^{\mathbb{U}^n}$ is done coordinate-wise, it follows from Observation 4.1 that C^Y outputs $(\boldsymbol{a\zeta})_Y$. ∎

Theorem 5.4 There exists an algorithm that given a circuit $C = (D, \lambda, \mathbb{R}[\mathbb{U}^n])$ with output $\boldsymbol{v}$ and $|\lambda(a)| \leq M$ for every input gate a, determines $v_{\{1,\ldots,n\}}$ in $\mathcal{O}^\star(2^n \mathtt{mpl}(C))$ time and $\mathcal{O}^\star(\mathtt{mpl}(C))$ space.

Proof The algorithm simply evaluates the inclusion-exclusion formula (see (3.1)):

$$v_{\{1,\ldots,n\}} = \sum_{Y \subseteq \{1,\ldots,n\}} (-1)^{n-|Y|} (\boldsymbol{v\zeta})_Y,$$

where for every $Y \subseteq \{1, \ldots, n\}$ the quantity $(\boldsymbol{v\zeta})_Y$ is computed by first straightforwardly evaluating the circuit C^Y using Lemma 5.1. By Observation 4.3 we know that all integers involved in that computation are bounded from above by $(M \cdot \mathtt{msl}(C[g]))^{\mathtt{mpl}(C[g])}$ and hence this can be done in $\mathcal{O}^\star(\mathtt{mpl}(C))$ time and space using fast integer multiplication (for example [Für09]). The claimed bounds follow. ∎

5.2.2 An Application of Theorem 5.4

Now we will give an application of Theorem 5.4. In particular we will show that the polynomial space algorithm as discussed in Section 3.2.1 for Unweighted Steiner Tree follows directly from the Dreyfus-Wagner [DW72] recurrence. Recall that Unweighted Steiner Tree is defined as follows:

Unweighted Steiner Tree	**Parameter:** k
Input: A graph $G = (V, E)$, $K \subseteq V$ with $\lvert K\rvert = k$ and an integer $t \leq \lvert V\rvert$.	
Question: Does there exist a subtree (V', E') of G such that $V' \leq t$ and $K \subseteq V'$?	

Recall that a dynamic programming algorithm for this problem was given already in 1972 by Dreyfus and Wagner [DW72]. This algorithm uses $\mathcal{O}^*(3^k)$ time and $\mathcal{O}^*(2^k)$ space. In 2007, Björklund et al. [BHKK07] gave an improved algorithm using $\mathcal{O}^*(2^k)$ time and space and in Section 3.2.1, we gave an $\mathcal{O}^\star(2^k)$-time and polynomial-space algorithm. We now show how the last result can be obtained from the first result using Theorem 5.4. For every $1 \leq i \leq n$ and $v \in V$ define $f(i, v) \in \mathbb{B}^{\mathcal{P}(K)}$ as

$$f(i,v)_X = \begin{cases} \texttt{true} & \text{if } \exists \text{subtree } (V', E') \text{ of } G : (V' \cap (K \cup \{v\})) = X \cup \{v\} \wedge \lvert E'\rvert \leq k, \\ \texttt{false} & \text{otherwise.} \end{cases}$$

The instance of Unweighted Steiner Tree is a YES-instance if and only if $f(i, v)_K = \texttt{true}$ for some $i \leq t$, where $v \in K$. Thus, it remains to determine $f(t, v)_K$. The recurrence of the dynamic programming algorithm from [DW72] to do this is as follows:

$$f(i,v)_X = \begin{cases} [X = \varnothing] & \text{if } i = 0 \wedge v \notin K, \quad (5.5a) \\ [X = \{v\}] & \text{if } i = 0 \wedge v \in K, \quad (5.5b) \\ \bigvee\limits_{1 \leq j < i} \bigvee\limits_{w \in N(v)} \bigvee\limits_{Y \cup Z = X} f(j,w)_Y \wedge f(i-j,v)_Z & \text{if } i > 1. \quad (5.5c) \end{cases}$$

Cases (5.5a) and (5.5b) follow directly from the definition of $f(i, v)$. For Case (5.5c), note that every subtree (V', E') of G with $X \cup \{v\} \subseteq V'$ and $\lvert E'\rvert \leq i$ can be split into two subtrees that cause $f(j, w)_Y$ and $f(i - j, v)_Z$ to be $\texttt{true}$ for some j, w, Y and Z with the imposed restrictions and that the union of two such subtrees always gives a subtree causing $\boldsymbol{f}(i, v)$ to be $\texttt{true}$ (since we can assume without loss of generality that they are disjunct). Now let us rewrite the recurrence in the ring $\mathbb{Z}[\mathbb{U}^k]$:

$$\boldsymbol{f}(i,v) = \begin{cases} \langle 1, \varnothing \rangle & \text{if } i = 0 \wedge v \notin K, \\ \langle 1, \{v\} \rangle & \text{if } i = 0 \wedge v \in K, \\ \sum\limits_{1 \leq j < i} \sum\limits_{w \in N(v)} \boldsymbol{f}(j,w) * \boldsymbol{f}(i-j,v) & \text{if } i > 1. \end{cases}$$

And apply the zeta-transform on both sides (recall that $\boldsymbol{f}(i,v) \in \mathbb{Z}[\mathbb{U}^k]$ is a vector):

$$(\boldsymbol{f}(i,v)\boldsymbol{\zeta})_X = \begin{cases} 1 & \text{if } i = 0 \wedge v \notin K, \text{(5.6a)} \\ [v \in X] & \text{if } i = 0 \wedge v \in K, \text{(5.6b)} \\ \sum\limits_{1 \le j < i} \sum\limits_{w \in N(v)} (\boldsymbol{f}(j,w)\boldsymbol{\zeta})_X \cdot (\boldsymbol{f}(i-j,v)\boldsymbol{\zeta})_X & \text{if } i > 1, \text{(5.6c)} \end{cases}$$

where Case 5.6c follows from the linearity of the zeta-transform $\boldsymbol{\zeta}$ and the fact that it diagonalizes convolution from Observation 4.5. Now note that in fact, $\boldsymbol{f}(i,v)$ is the number of branching walks (refer to Section 3.2) in $G[(V \setminus K) \cup X \cup \{v\}]$ from v of size i. Hence applying Theorem 5.4 to (5.6) results in the algorithm implied by Theorem 3.4.

5.2.3 Subset Convolution

We consider circuits over $\mathcal{R}[\mathbb{D}^n]$. One notices quickly that since $y \cup z$ is not defined for every $y, z \in \mathbb{D}^n$, it is hard to directly find a transformation that diagonalizes the multiplication in this ring (the so-called 'subset convolution' from [BHKK07]). Therefore, we first use a natural embedding into $\mathcal{R}[\mathbb{U}^n \times \mathbb{Z}]$ or $\mathcal{R}[\mathbb{Z}_2^n \times \mathbb{Z}]$. It should be noted that both are well-known and for example appear in [BHKK07] and [KW09]. If $\boldsymbol{a} \in \mathcal{R}[\mathbb{D}^n]$, we let $\eta : \mathcal{R}[\mathbb{D}^n] \to \mathcal{R}[\mathbb{U}^n \times \mathbb{Z}]$ or $\eta : \mathcal{R}[\mathbb{D}^n] \to \mathcal{R}[\mathbb{Z}_2^n \times \mathbb{Z}]$ be such that for every $Y \in \mathbb{D}^n$ and integer i it holds that $\eta(\boldsymbol{a})_{Y,i} = a_Y[|Y| = i]$. Then for both cases, it is easy to see that $(\boldsymbol{a} * \boldsymbol{b})_Y = (\eta(\boldsymbol{a}) * \eta(\boldsymbol{b}))_{Y,|Y|}$ since on the right hand side only pairs of disjoint indices W, X can contribute since otherwise $|W \cup X| < |W| + |Y|$ and $|W \Delta X| < |W| + |X|$. Implementing this in a circuit, we obtain the following easy lemma:

Lemma 5.2 There exists an algorithm that, given a circuit $C = (D, \lambda, \mathcal{R}[\mathbb{D}^n])$ where $\mathcal{R}$ is a semiring with output $\boldsymbol{v}$ and singleton inputs, outputs a circuit C' over $\mathcal{R}[\mathbb{U}^n]$ with output $\boldsymbol{v}$ such that $|C'|$ is $\mathcal{O}(|C| n \lg n)$, $\texttt{mpl}(C') = \texttt{mpl}(C)$ and $\texttt{msl}(C')$ is $\mathcal{O}(n \cdot \texttt{msl}(C))$.

To assist the formal proof, we make the following definition:

Definition 5.1 A *relaxation* of a vector $f \in \mathcal{R}^{\mathbb{U}^n}$ is a sequence of functions $\{f^i : f^i \in \mathcal{R}^{\mathbb{U}^n}\}$, for $0 \le i \le |V|$, such that for every $0 \le i \le |V|$, $Y \subseteq V$:

$$f^i_Y = \begin{cases} f_Y & \text{if } i = |Y| \\ 0 & \text{if } i < |Y| \end{cases}$$

Proof (of Lemma 5.2) We construct C' by replacing every gate of $a \in C$ by n gates $a^1, \ldots, a^n \in C'$. For every input gate a with $\lambda(a) = \langle a_S, S \rangle$, we set $a^i = 0$ if $i < |S|$

and $a^i = a$ otherwise. Notice that this means that all constant gates of C' are singletons. If a is not a input gate, let b and c be the in-neighbors of a in C. Note it could be that $b = c$. If a is an addition gate, then for every i we set $a^i = b^i + c^i$. Otherwise, if a is a multiplication gate, we add gates to C' such that $a^i = \sum_{j=0}^{i}(b^i * c^{i-j})$. Clearly, the above procedure can be executed in polynomial time, and C' has the claimed properties.

We prove that for every gate a, $\{a^i\}$ is a relaxation of a by induction along a topological order of C. If a is a constant gate this follows by construction. Otherwise, let b and c be the in-neighbors of a. Then $\{b^i\}$ and $\{c^i\}$ are relaxations of b and c respectively by the induction hypothesis. If a is an addition gate ($a = b + c$), then $\{a^i\}$ is a relaxation of a since

- for $i = |X|$, $a^i_X = b^i_X + c^i_X = b_X + c_X$ since $b^i_X = b_X$ and $c^i_X = c_X$ by the induction hypothesis.
- for $i < |X|$, $a^i_X = b^i_X + c^i_X = b_X + c_X = 0$ since both b^i_X and c^i_X are 0 by the induction hypothesis.

On the other hand if a is a subset convolution gate ($a = b * c$), We have that for each $0 \le i \le n$,

$$a^i_X = \sum_{j=0}^{i} \sum_{Y \cup Z = X} b^j_Y c^{i-j}_Z.$$

Consider a summand $b^j_Y c^{i-j}_Z$. There are two cases:

- ($|X| > i$) Since $Y \cup Z = X$ we have that $|Y| + |Z| \ge |X| > i$. Now either $|Y| > j$ and $b^j_Y = 0$, or $|Z| > i - j$ and $c^{i-j}_Z = 0$, because $\{b^i\}$ and $\{c^i\}$ both are relaxations of b and c respectively. Then we have that $a^i_X = 0$.
- ($|X| = i$) If $|Y| + |Z| > i$, either $|Y| > j$ or $|Z| > i - j$ and $a^j_A b^{i-j}_B = 0$ analogously to the first case. If $|Y| + |Z| = i$ then Y and Z are disjoint. Since only these pairs Y, Z will contribute to the sum, we match the definition of subset convolution and hence $a^i_X = a_X$.

Thus $\{a^i\}$ is a relaxation of a for all gates a, concluding the proof. ∎

Also, the following variation of Lemma 5.2 that we state just for the sake of completeness, has a proof (almost) identical to Lemma 5.2:

Lemma 5.3 There exists an algorithm that, given a circuit $C = (D, \lambda, \mathcal{R}[\mathbb{D}^n])$ where $\mathcal{R}$ is a semiring with output $\boldsymbol{v}$ and singleton inputs, outputs a circuit C' over $\mathcal{R}[\mathbb{Z}_2^n]$ with output $\boldsymbol{v}$ such that $|C'|$ is $\mathcal{O}(|C| n \lg n)$, $\mathtt{mpl}(C') = \mathtt{mpl}(C)$ and $\mathtt{msl}(C')$ is $\mathcal{O}(n \cdot \mathtt{msl}(C))$.

5.3 Combining DFT and Möbius to Save Space

In this section we will combine the tools introduced in the previous two sections to obtain new polynomial space algorithms for several minimization problems. We obtain an analogue of Theorem 5.4 in the case where $\mathcal{R}$ is the min-sum semiring $\mathcal{M}_-$ (recall that the identity of min is ∞ and the identity of + is 0).

Evidently, Möbius Inversion is not applicable in this case since the addition operator of $\mathcal{M}$, minimization, does not have the inverse which is required for Möbius inversion. Therefore, we follow the commonly used solution, as already discussed in Section 4.1 and in 5.1.1 for the KNAPSACK problem, namely embedding $\mathcal{M}_-$ in the ring $\mathbb{Z}[\mathbb{Z}_N]$.

5.3.1 Minimization Dynamic Programming Without Tables

Theorem 5.5 Let n, t be integers, $U = \{1, \ldots, n\}$, and $C = (D, \lambda, \mathcal{M}_-[\mathbb{D}^n])$ with output $\boldsymbol{v}$. Then there exists an algorithm deciding whether $v_U \leq t$ in $\mathcal{O}^\star(2^n t \cdot \mathtt{mpl}(C))$ time and $\mathcal{O}^\star(1)$ space.

Before proving Theorem 5.5 let us remark that there is an algorithm deciding whether $v_U \leq w$ in $\mathcal{O}^\star(3^n)$ time and $\mathcal{O}^\star(2^n)$ space, by storing the elements of $\mathcal{M}_-$ for each subset and each gate and using standard dynamic programming (we omit the details). Not surprisingly, our proof instead relies on the Möbius and Discrete Fourier transformations. It is perhaps best summarized graphically as in Figure 5.1.

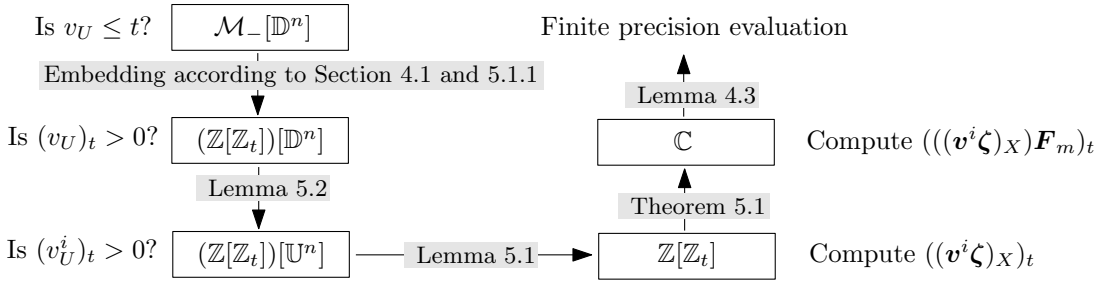

Figure 5.1 – Outline of the proof of Theorem 5.5.

Proof (of Theorem 5.5) Let M be an integer such that for every input gate a of C, $\lambda(a) = \langle i, X\rangle$ where $i = \infty$ or $0 \leq i \leq M$. First notice that $M \cdot \mathtt{mpl}(C)$ is an upper bound for every coefficient of a gate of C. That is, for every gate a of C and $X \subseteq U$ it holds that $a_X \leq M \cdot \mathtt{mpl}(C)$. Also notice that we can assume that $M \leq t$ since if $\lambda(a) = \langle i, X\rangle$ with $i > M$, replacing it with $\lambda(a) = \langle \infty, X\rangle$ does not make a difference.

Now, use Lemma 5.2 to obtain a circuit $C^{(1)}$ over the semiring $\mathcal{M}_-[\mathbb{U}^n]$ with output $\boldsymbol{v}$. Also, according to the discussion from Section 4.1 and similarly to

what we did for KNAPSACK in Subsection 5.1.1, we can safely embed the min-sum semiring into $\mathbb{Z}[\mathbb{Z}_{t+M\cdot\mathtt{mpl}(C)}]$. More specifically, we obtain a circuit $C^{(2)} = (D, \lambda', \mathbb{Z}[\mathbb{Z}_{t+M\cdot\mathtt{mpl}(C)}])$ by letting $\lambda'(a) = \lambda(a)$ for a being not an input gate of $C^{(1)}$ and otherwise $\lambda'(a) = \langle i, 1\rangle$ where $\lambda(a) = i$. Next, we obtain a circuit $C^{(3)}$ by adding the circuit $C^{\leq}(t)$ outputting $\boldsymbol{a}(p)$ as described in Subsection 5.1.1 and let the output of $C^{(3)}$ be the product of $\boldsymbol{a}(p)$ and the output of $C^{(2)}$. If we denote $\boldsymbol{v}^{(3)}$ for the output of $C^{(3)}$, it follows that $(v_U^{(3)})_t > 0$ if and only if $v_U \leq t$. So it remains to compute $(v_U^{(3)})_t$. To this end we straightforwardly evaluate the Möbius inversion/Inclusion-Exclusion formula (see Theorem 3.1 and 4.4):

$$(v_U^{(3)})_t = \sum_{X\subseteq U} (-1)^{n-|X|}((\boldsymbol{v}^{(3)}\boldsymbol{\zeta})_X)_t, \tag{5.7}$$

so it remains to show how to compute $((\boldsymbol{v}^{(3)}\boldsymbol{\zeta})_X)_t$ for an arbitrary fixed $X \subseteq V$ in $\mathcal{O}^\star(M\cdot\mathtt{mpl}(C))$ time and $\mathcal{O}^\star(1)$ space. To do this, use Lemma 5.1 to obtain from $C^{(3)}$ a circuit C^X over the ring $\mathbb{Z}[\mathbb{Z}_{t+M\cdot\mathtt{mpl}(C)}]$ that outputs the vector $((\boldsymbol{v}^{(3)}\boldsymbol{\zeta})_X)$. Finally, use Theorem 5.1 to determine $((\boldsymbol{v}^{(3)}\boldsymbol{\zeta})_X)_t$. It is easy to see that all steps take time polynomial in the input, except that we have to sum over 2^n summand in (5.7) and that applying Theorem 5.1 takes $\mathcal{O}^\star(M\cdot\mathtt{mpl}(C))$ time since $\mathtt{mpl}(C^{(3)})$ is easily seen to be $\mathcal{O}^\star(\mathtt{mpl}(|C|))$. ■

5.3.2 Applications

We will now give a few applications of Theorem 5.5, hence, considering circuits using pointwise addition and the 'min-sum subset convolution'. Note that in the context of the min-sum semiring, Iverson's bracket notation works as follows; $[b]_X = 0$ for every X if $b = \mathtt{true}$ and $[b]_x = \infty$ for every X otherwise. Thus, a constant $[X = S]v$ is v if $X = S$ and $[X = S]v$ is $[\mathtt{false}]$ otherwise. We will only consider decision variants, but it should be noted that using binary search and standard self-reduction it is possible to extend these algorithms to construct an optimal solution, at the cost of a polynomial factor in the input size in the running time.

Traveling Salesman Problem

Recall that a Hamiltonian path of a graph is a path visiting all vertices exactly once. We study the following generalization of Hamiltonian path:

TRAVELING SALESMAN PROBLEM (TSP) **Parameter:** $|V|, t$
Input: A graph $G = (V, E)$, an integer t, a vertex s and a function $\omega : V \times V \to \{1, \ldots, t\}$.
Question: Is there a Hamiltonian path $E' \in E$ with $\omega(E') \leq t$?

Denote $n = |V|$. Early $\mathcal{O}^*(2^n)$ time and space dynamic programming algorithms are given in [Bel62, HK62]. Later, an algorithm running in time $\mathcal{O}^*(2^n t)$

using $O^*(t)$ space was given by Karp [Kar82]. TSP can also be solved in $\mathcal{O}^*(4^n)$ time and polynomial space [GS87]. Recently, [KP10] proposed a combination of these two approaches to obtain a space-time trade off. It is an interesting open problem ([Woe04]) whether TSP can be solved in $\mathcal{O}(2^n)$ time and polynomial space.

For $v \in V$ and $X \subseteq V \setminus \{s, v\}$, define $f(v)_X$ as the minimum weight of a Hamiltonian path in $G[X \cup \{s, v\}]$ starting in s and ending in v. The Bellman-Held-Karp recurrence [Bel62, HK62] is:

$$f(v)_X = \begin{cases} \omega(s, v) & \text{if } X = \varnothing, \\ \min_{u \in N(v) \cap X} f(u)_{X \setminus \{v\}} + \omega(u, v) & \text{otherwise.} \end{cases} \tag{5.8}$$

To see that the above equation holds, note that there only is one Hamiltonian path to consider if $X = \varnothing$ and its weight is $\omega(s, v)$. If X is not empty, any Hamiltonian path of $G[X \cup \{s, v\}]$ starting in s and ending in v consists of a Hamiltonian path of $G[X \cup \{s, u\}]$ starting in s and ending in u, and the edge (u, v). Hence, we can minimize over all the last edges a Hamiltonian path can have, and find the minimum weight such a Hamiltonian path can have using previously computed values and the value $\omega(u, v)$.

In order to turn (5.8) into a circuit, we formulate it in the semiring $\mathcal{M}_-[\mathbb{D}^n]$. For every v, let $f(v) \in \mathcal{M}_-[\mathbb{D}^n]$ be as defined above, then we claim that

$$f(v) = \langle \omega(s, v), \varnothing \rangle + \sum_{u \in N(v)} f(u) * \langle \omega(u, v), \{u\} \rangle . \tag{5.9}$$

The best way to see that (5.9) holds, is perhaps to straightforwardly expand the definition of addition and multiplication in the semiring $\mathcal{M}_-[\mathbb{D}^n]$:

$$f(v)_X = \min \Big\{ [X = \varnothing] \omega(s, v), \min_{u \in N(v)} \min_{A \cup B = X} f(u)_A + [B = \{u\}] \omega(u, v) \Big\}. \tag{5.10}$$

note that here $\min, +$ refer to the standard integer minimization and addition and that $[p]$ denotes 0 is $p = \texttt{true}$ and ∞ otherwise. At first sight, (5.9) looks like an infinite recurrence (that is, $f(v)$ seems to depend on $f(v)$), and hence it can not be translated into a circuit. However, from the expanded version (5.10) we can see that this not the case since B has to be non-empty in order to influence the minimization. So, this bounds the number of recursive calls and it is easy to see we can instead use the following recurrence in the semiring $\mathcal{M}_-[\mathbb{D}^n]$:

$$f(i, v) = \begin{cases} \omega(s, v) & \text{if } i = 1, \\ \sum_{u \in N(v)} (f(i-1, u) * \langle 1, v \rangle) + \omega(u, v) & \text{if } i > 1. \end{cases} \tag{5.11}$$

since $f(n, v)_V = f(v)_V$ for every $v \in V \setminus \{s\}$. In order to apply 5.5, it remains to bound the maximum product length of the circuit C implied by (5.11). Although it is quite easy to see $\texttt{mpl}(C)$ is $\mathcal{O}^*(1)$, we apply Observation 4.2 for

clarity. The circuit C'' defined in Observation 4.2, obtained by replacing addition with maximization and multiplication by addition, then is given by the following recurrence:

$$f''(i,v) = \begin{cases} 1 & \text{if } i = 1, \\ \max\limits_{u \in N(v)} \max\{f''(i-1,u)+1, 1\} & \text{if } i > 1. \end{cases}$$

Then Observation 4.2 implies that $\texttt{mpl}(C)$ is $\mathcal{O}^\star(1)$ and applying Theorem 5.5 we obtain:

Theorem 5.6 Traveling Salesman Problem can be solved in $\mathcal{O}^\star(2^n d)$ time and polynomial space.

Steiner Tree

In this section, we will extend the example of Section 5.2.2. First, recall the Steiner Tree problem from Section 3.2.1:

Steiner Tree **Parameter:** k
Input: A graph $G = (V, E)$, an integer t, a weight function $\omega : E \to \{1, \ldots, t\}$, and a terminal set $K \subseteq V$ with $|K| = k$.
Question: Does there exist a subtree (V', E') of G such that $\omega(E') \leq t$ and $K \subseteq V'$?

The current fastest algorithm for this problem is due to [FKM+07], and uses $\mathcal{O}((2+\epsilon)^k n^{h(\epsilon)})$ time and space for some function h. We first again consider the algorithm of Dreyfus and Wagner [DW72], but now for the weighted case. In the semiring $\mathcal{M}_-[\mathbb{D}^n]$, the recurrence used can be written as follows: $f(i,v)_X =$

$$\begin{cases} [X = \varnothing] & \text{if } i = 0 \wedge v \notin K, \quad (5.12a) \\ [X = \{v\}] & \text{if } i = 0 \wedge v \in K, \quad (5.12b) \\ \min\limits_{1 \leq j < i} \min\limits_{w \in N(v)} \min\limits_{Y \cup Z = X} f(j,w)_Y + f(i-j,v)_Z + \omega(v,w) & \text{if } i > 1. \quad (5.12c) \end{cases}$$

Following the argumentation of Subsection 5.2.2, it is easy to see that (5.12) holds. To make it more clear that this is a recurrence in the semiring $\mathcal{M}_-[\mathbb{D}^n]$, let us rewrite (5.12) again in a more implicit way.

$$\boldsymbol{f}(i,v) = \begin{cases} \langle 0, \varnothing \rangle & \text{if } i = 0 \wedge v \notin K, \quad (5.13a) \\ \langle 0, \{v\} \rangle & \text{if } i = 0 \wedge v \in K, \quad (5.13b) \\ \sum\limits_{1 \leq j < i} \sum\limits_{w \in N(v)} \boldsymbol{f}(j,w) * \boldsymbol{f}(i-j,v) * \langle \omega(v,w), \varnothing \rangle & \text{if } i > 1. \quad (5.13c) \end{cases}$$

Now we can apply Theorem 5.5 by considering (5.13) as a circuit C. Then, $\texttt{mpl}(C)$ is easily seen to be $\mathcal{O}^\star(1)$ and the following result follows:

Theorem 5.7 The Weighted Steiner Tree problem can be solved in $\mathcal{O}^*(2^k t)$ time and polynomial space.

5.4 Concluding Remarks

First, it should be noted that a variant of Section 5.2 also appeared implicitly as a subroutine of the Koutis-Williams approach [KW09], using the Fourier transform rather than Möbius inversion (unfortunately, the authors were not sufficiently aware of this connection at the time of writing [LN10]). Moreover, in [Kan10] an algorithm for Subset Sum similar to the one given in this chapter was given.

It should be noted that the Koutis-Williams approach (see Section 4.7) can also be used to solve reasonable variants of the Steiner Tree problem. Consider for example

Small Steiner Tree **Parameter:** ℓ
Input: A graph $G = (V, E)$, a terminal set $K \subseteq V$ and integers $1 \leq t \leq n, 1 \leq \ell \leq |K|$.
Question: Does there exist a subtree (V', E') of G such that $|E'| \leq t$ and $|T \cap V'| \geq \ell$?

Then, applying the Koutis-Williams approach directly to the circuit implied by (5.6) gives the following theorem

Theorem 5.8 The Small Steiner Tree problem can be solved by a Monte-Carlo algorithm in $\mathcal{O}^*(2^\ell)$ time and $\mathcal{O}^*(1)$ space.

An elegant application of Section 5.2 has already been found in [Bjö11]. Here a non-trivial recurrence is given for the number of perfect matchings in graph. Since the recurrence is over the ring $\mathbb{Z}[\mathbb{D}^n]$, Section 5.2 can be used to evaluate the recurrence in a space-efficient manner.

The min-sum semiring embedding into the ring $\mathbb{Z}[\mathbb{Z}_m]$ seems very inefficient. It should also be noticed that for many purposes, an algebraic structure with less structure than a ring is sufficient for $\mathcal{M}_-$ to be embedded in. For example, using the Isolation lemma, any semiring of characteristic two suffices for using inclusion-exclusion with a randomized embedding for a decision problem. On the other hand, perhaps it is possible that any algebraic object that allows a useful embedding must actually have operations that are as costly to perform as for $\mathbb{Z}[\mathbb{Z}_m]$.

Let us continue with some concrete open questions:

Open Question 2 Can KNAPSACK be solved in $\mathcal{O}^\star(v+w)$ time and $\mathcal{O}^\star(1)$ space?

The following so-called *fully polynomial approximation scheme* (although its running time is pseudo-polynomial in ϵ since we represent in binary) for the KNAPSACK problem is well known (see for example [KT06]):

Theorem 5.9 There exist an algorithm that given an instance of KNAPSACK and $\epsilon > 0$, runs in $\mathcal{O}^\star(1/\epsilon)$ time and space and returns

- `YES`, if there exists $X \subseteq S$ such that $\nu(X) \leq v$ and $\omega(X) \geq (1+\epsilon)w$,
- `NO`, if there does not exist $X \subseteq S$ such that $\nu(X) \leq v$ and $\omega(X) \geq w$.

We ask for a space-efficient analogue. It should be noted that since Theorem 5.9 depends uses the dynamic programming algorithm for KNAPSACK as a subroutine, an affirmative answer to Open Question 2 would imply an affirmative answer for the following question:

Open Question 3 Does there exist an algorithm that given an instance of KNAPSACK and $\epsilon > 0$, runs in $\mathcal{O}^\star(1/\epsilon)$ time and $\mathcal{O}^\star(1)$ space and returns

- `YES`, if there exists $X \subseteq S$ such that $\nu(X) \leq v$ and $\omega(X) \geq (1+\epsilon)w$,
- `NO`, if there does not exist $X \subseteq S$ such that $\nu(X) \leq v$ and $\omega(X) \geq w$.

We also ask for the complexity of two weighted variants studied in this chapter. The following open question was already asked in [Woe04].

Open Question 4 — (Woe04) Can the TRAVELING SALESMAN PROBLEM be solved in $\mathcal{O}^\star(2^n)$ time and $\mathcal{O}^\star(1)$ space?

Even an $\mathcal{O}^\star(2^n + t)$ time and $\mathcal{O}^\star(1)$ space would be interesting. When we allow running times linear in t, the current fastest polynomial space follows by combining the algorithm of [Bjö10b] with Theorem 5.1 (as noted by Björklund [Bjö10a]). It should be mentioned that in [KP10], space/time tradeoffs between the known $\mathcal{O}^\star(2^n)$ time and space algorithm from [Bel62] and the $\mathcal{O}^\star(4^n)$ time and $\mathcal{O}^\star(1)$ space algorithm from [GS87].

Open Question 5 Can STEINER TREE be solved in $\mathcal{O}^\star(c^k)$ time and $\mathcal{O}^\star(1)$ space?

Chapter 6

On Homomorphic Hashing for Coefficient Extraction

This chapter is based on the following paper:

[KKN11]. Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. On homomorphic hashing for coefficient extraction. 2011. Manuscript

In this chapter we will continue were we left off in the previous chapter. The method used continuously in that chapter could be named *coefficient extraction* (see also [Lip10]). Coefficient extraction can be seen as a general method for designing algorithms. The most successful and famous instantiation is clearly the Discrete Fourier Transform for fast polynomial multiplication (usually credited to [CT65]). Recently many new results have been using variants of coefficient extraction as already discussed in the previous two chapters with applications to, only to name a few, k-Path, Set Cover, Set Partition, Steiner Tree, Subset Sum, Traveling Salesman (all being in the area of exact exponential algorithms). The general approach of the method is the following:

1. Define a variable (the so-called coefficient) whose value (almost) immediately gives the solution of the problem to be solved,
2. Show that the variable can be expressed by a relatively easy formula or circuit (see Definition 4.7) in terms of a (cleverly chosen) algebraic object like a ring or field,
3. Show how to perform the operations in the algebraic object relatively efficiently.

For some applications of the method, the first and second step can be implemented by using the recurrence of previously known dynamic programming recurrences (as already seen in Chapter 5). But how does the coefficient extraction method

compare here to the straightforward alternative for computing all table entries? In this chapter we try to settle this question. Mainly, it is well-known that if the DP table is sparse, DP can be adjusted easily to give an improved algorithm. But how does coefficient extraction perform in this setting? Interestingly it appears, informally stated, that even if only the last segment of the DP table (or equivalently, the output of the circuit) is guaranteed to be sparse (in the sense that many entries are zero), coefficient extraction can be improved in several cases.

The subject of sparse instances is a highly motivated and well-studied subject of study: for example, several sequence analysis problems [EGGI92a, EGGI92b] and matrix multiplication where the input matrices or their product is assumed to be sparse [Lin11, YZ05]. Perhaps the most related to our work is the result of [Man95] for interpolating sparse polynomials; here the main tool is Chernoff bounds and the given algorithm obtains all non-zero coefficients in time polynomial in the number of non-zero coefficients. We will only interpolate one requested coefficient, but will do so a in space efficient and faster way.

Our main tool is hashing in a way consistent with the constructed circuit: homomorphic hashing. Before the actual coefficient extraction is performed, the circuit is transformed to another one over a simpler ring or field by a homomorphic hash function. Because the function is homomorphic, hashing the input gates results in a circuit outputting the hash of the original output gate. Since the function is a hash-function, the coefficient to be extracted does not collide with other coefficients and hence coefficient extraction on the new circuit can be used instead. It should be noted that this framework is not new. For example, the Koutis-Williams and Color-Coding approach (see Section 4.7) could also be seen as instantiations of this approach.

We study three types of algebraic objects: the polynomial ring, the group algebra $\mathbb{F}[\mathbb{Z}_2^n]$, and the Möbius algebra. For the first two types we show that existing algorithms can indeed be improved when sparsity is promised. As applications we show that the Subset Sum can be solved in polynomial space and linear in the number of distinct sums of subsets of the given integers, and that Set Partition can be solved in $\mathcal{O}^\star(2^{\mathrm{rk}(A)})$ time and $\mathcal{O}^\star(1)$ space, where $\mathrm{rk}(A)$ is the rank of the incidence matrix of the given set system. We also give a so-called 'Win/Win approach'[(1)] that solves the Linear Sat problem in $\mathcal{O}^\star(2^{n/2})$ time and $\mathcal{O}^\star(1)$ space. For studying the Möbius algebra, we propose a hashing scheme based on an old result of Solomon and give a possible application to CNF-Sat with the promise that the number of projections of variable assignments to the CNF-formula is small (see Section 6.3 for a more formal definition).

Coefficient extraction via the Fourier Transform is a very well-known method

[(1)] Here we distinguish two cases that are solved by different algorithms/arguments. An instantion of this was already encountered in Section 3.5. The term is usually used when describing 'bidimensionality theory' (see for example [DFHT05, DH08]) that distinguish between instances of small and high treewidth (see also Subsection 7.0.3.)

(see for example [CLRS01]). We already saw that it can be used to solve the Subset Sum problem in a space efficient manner (see Section 5.1). Sparse coefficient extraction in the polynomial ring is a well-studied problem [Man95, Zip79], even for polynomials in $\mathbb{F}[\mathbb{Z}_2^n]$ (see for example [BO88]). However, in the field of Theoretical Computer Science, the Möbius algebra has still found relatively few applications: for the specific case of the Möbius algebra of the subset lattice a few are listed in Chapters 3 and 5.

This chapter is organized as follows: in Section 6.1 we study numerical dynamic programming and the Subset Sum problem, in Section 6.2 we study the Set Partition and Linear Sat problems and in Section 6.3 we study the CNF-Sat problem. Finally we will give some concluding remarks in Section 6.4.

6.1 Homomorphic Hashing for Subset Sum

Let us start by again recalling the Subset Sum problem:

Subset Sum	**Parameter:** t
Input: Integer n, function $\omega : \{1,\dots,n\} \to \mathbb{N}$ and integer t.	
Question: Does there exist a subset $X \subseteq \{1,\dots,n\}$ such that $\omega(X) = t$?	

Also, recall the dynamic programming algorithm as described in Section 2.1. It used the following observation: for $1 \le i \le n$ and $0 \le j \le t$, define $s(i-1,j)$ to be `true` if and only if there exists $X \subseteq \{1,\dots,i\}$ such that $\omega(X) = j$, then:

$$s(i,j) = \begin{cases} \texttt{true} & \text{if } i = 0, \quad (6.1a) \\ s(i-1,j) \vee s(i-1,j-a_i) & \text{otherwise.} \quad (6.1b) \end{cases}$$

Theorem 6.1 — Folklore, (Bel54) There exists an algorithm that solves any given instance (n,ω,t) of the Subset Sum problem in $\mathcal{O}(n \cdot |S| \lg |S|)$ time and $\mathcal{O}(|S|)$ space, where $S = |\{\omega(X) : X \subseteq \{1,\dots,n\}\}|$.

Before proceeding to the proof let us note that the theorem is stated in the unit-cost model. In the log-cost model the resource bounds would be $\mathcal{O}(n \cdot b \cdot |S| \lg |S|)$ time and $\mathcal{O}(|S|((b + \lg n)))$ space. We give the proof of Theorem 6.1 for completeness

Proof We use what could be called 'Sparse Dynamic Programming'. For every $0 \le i \le n$ we create a binary search tree B_i to which we will add all values j such that $s(i,j) = \texttt{true}$. Implementing this with an AVL tree (see for example [CLRS01]), this requires $\mathcal{O}(S)$ space and queries, removals and insertions, and all of this can be performed in $\lg |S|$ time. First, B_0 only contains 0. Then B_i is created by creating a copy of B_{i-1} and for all integers j in B_{i-1} adding $\omega(i) + j$ to B_i if it does not exist yet and after this B_{i-1} can be removed from the working memory. In the end the algorithm return `YES` if and only if t is in B_n. ∎

As mentioned in the introduction of this chapter, after considering Theorem 5.2, a natural question is whether we can obtain a space efficient analogue of Theorem 6.1. In this section we will show that this is possible and prove it is in the same generic setting as in Section 5.1. The main observation in the generic setting is that if the last row of the dynamic programming table contains many zero's, then we can work with a hashed version of the table to speed up the computation since collisions are less probable.

Theorem 6.2 Given a circuit $C = (D, \lambda, \mathbb{Z}[\mathbb{N}])$ with singleton inputs output gate $\boldsymbol{v}$ and integers S, t with $|\mathrm{supp}(\boldsymbol{v})| \leq S$,

1. an integer $\widetilde{v_t}$ such that $\mathrm{prob}[\widetilde{v_t} = v_t] \geq \frac{1}{2}$ can be computed using $\mathcal{O}^\star(\mathtt{mpl}(C))$ space and $\mathcal{O}^\star(S \cdot \mathtt{mpl}(C))$ expected time, and
2. v_t can be computed using at most $\mathcal{O}^\star(\mathtt{mpl}(C))$ space and $\mathcal{O}^\star(S^2 \cdot \mathtt{mpl}(C))$ time.

In order to prove Theorem 6.2, we will need the following number-theoretic results:

Lemma 6.1 — (Ros41) If $u > 55$, then the number of primes at most u is at least $\frac{u}{\ln u + 2}$.

The next lemma follows almost directly:

Lemma 6.2 — Folklore There exists an algorithm $\mathtt{pickprime}(u)$ running in $\mathcal{O}^\star(1)$ expected time that, given integer $u \geq 2$ as input, outputs a prime chosen uniformly at random from the set of primes at most u.

Proof Take an integer $i \leq u$ uniformly at random, and check whether it is a prime using the polynomial time algorithm of [AKS04]. If i is prime, then output i and halt; otherwise repeat. This clearly meets the running time due to Lemma 6.1.■

Proof (of Theorem 6.2) In the following, let M be the largest integer such that there exists an input gate a of C with $\lambda(a) = \langle e, M \rangle$ for some integer e and $Z = M \cdot \mathtt{mpl}(C)$. Without loss of generality let us assume that $S \cdot \lg Z > 55$. It is easy to see that for every gate a of C and $i > Z$, $i \notin \mathrm{supp}(\boldsymbol{a})$. The algorithms are given in Algorithm 3.

Let us start by analyzing `extractWithModulus`. Line 16 can be performed in polynomial time since all elements of $\mathbb{Z}[\mathbb{N}]$ are singleton vectors. Line 17 takes $\mathcal{O}^\star(p \cdot \mathtt{mpl}(C) \cdot \lg(p + \mathtt{msl}(C))) = \mathcal{O}^\star(p \cdot \mathtt{mpl}(C))$ time[(2)] and $\mathtt{mpl}(C)$ space due

[(2)] Note that the integers $\lambda(a)$ do not show up in the runtime since `extractWithModulus` runs in time polynomial in the size of their binary representation.

Function `hashExtractFourier1`$(C = (D, \mathbb{Z}[\mathbb{N}], \lambda), S, t)$
1: $p \leftarrow$ `pickprime`$(2S \cdot \lg Z \cdot \ln^2(2S \cdot \lg Z))$. Using Lemma 6.2.
2: **return** `extractWithModulus`(C, t, p).

Function `hashExtractFourier2`$(C = (D, \mathbb{Z}[\mathbb{N}], \lambda), S, t)$
3: $mult, res, i, j \leftarrow 0$. res is result candidate; $mult$ is multiplicity of res
4: **while** $i \le 2S \cdot \lg Z$ **do**
5: $j \leftarrow j + 1$
6: **if** the deterministic primality testing from [AKS04] returns j is prime **then**
7: $i \leftarrow i + 1$; higher prime counter
8: $a =$ `extractWithModulus`(C, t, p).
9: **if** $res = 0$ **then**
10: $res \leftarrow a$
11: **else if** $a = R$ **then**
12: $mult \leftarrow mult + 1$
13: **else**
14: $mult \leftarrow mult - 1$
15: **return** res

Function `extractWithModulus`$(C = (D, \mathbb{Z}[\mathbb{N}], \lambda), t, p)$
16: $C' \leftarrow (D, \mathbb{Z}[\mathbb{Z}_p], \lambda')$ where $\lambda'(a) = \langle e, i \bmod p \rangle$ if $\lambda(a) = \langle e, i \rangle$ for an input gate a of C.
17: **return** `extractFourier`$(C, t \bmod p)$. see Algorithm 2

Algorithm 3 – Implementing Theorem 6.2.

to Theorem 5.1. Note that C' is the circuit obtained by applying h to C (see Definition 4.10), where $h : \mathbb{Z}[\mathbb{N}] \to \mathbb{Z}[\mathbb{Z}_p]$ is defined by $h(\boldsymbol{v})_j = \sum_{i:i \equiv j \pmod p} v_i$ for any $\boldsymbol{v} \in \mathbb{Z}_{\mathbb{N}}$. Then by Observation 4.1 and Theorem 5.1 we have that

$$\Big(\texttt{extractWithModulus}(C, t, p)\Big)_t = \sum_{i:i \equiv t \pmod p} v_i, \tag{6.2}$$

and hence it follows that if there exists no $i \in \mathrm{supp}(v)$ with $i \neq t$ and $i \equiv t \pmod p$, then the left-hand side of (6.2) is v_t.

Next we consider `hashExtractFourier1`. It follows from Lemma 6.2 that Line 1 can be performed in $\mathcal{O}^\star(1)$ expected time. Line 2 is performed using $\mathcal{O}^\star(S \cdot \texttt{mpl}(C))$ time and $\mathcal{O}^\star(\texttt{mpl}(C))$ space according to the above discussion on `extractWithModulus`. Denote $\gamma = 2S \cdot \lg Z \cdot \ln^2(2S \cdot \lg Z)$ and note that

$$\frac{\gamma}{\ln \gamma + 2} = \frac{2S \cdot \lg Z \cdot \ln^2(2S \cdot \lg Z)}{\ln\big(2S \cdot \lg Z \cdot \ln^2(2S \cdot \lg Z)\big) + 2} \ge 2S \cdot \lg Z. \tag{6.3}$$

Then, for correctness of the probability bound of Item 1, notice that

$$\text{prob}[\widetilde{v_t} \neq v_t] \leq \sum_{i \in \text{supp}(\boldsymbol{v})} \text{prob}[p \text{ divides } |i-t|] \leq \frac{|\text{supp}(\boldsymbol{v})| \lg Z}{\frac{\gamma}{\ln \gamma + 2}} \leq \frac{S \lg Z}{2S \cdot \lg Z} \leq \frac{1}{2}.$$

To see that this holds, note that the first inequality follows from (6.2) and the union bound; the second inequality follows from Lemma 6.1 and the fact that for every $i \in \text{supp}(\boldsymbol{a})$ it holds that $i \leq Z$ and hence i has at most $\lg Z$ prime divisors; the third inequality follows from (6.3). This concludes the proof of Item 1 of the theorem.

Finally let us consider `hashExtractFourier2`. For $1 \leq i \leq 2S \cdot \lg Z + 1$, denote the values a has on iteration i. Because of (6.2) and the properties of `extractWithModulus`, we know that for at most $S \lg Z$ values of i, $a_i \neq v_t$. Hence the majority of the set $a_1, \ldots, a_{2S \cdot \lg Z + 1}$ is v_t. The remaining part of `hashExtractFourier2` implements the classic majority voting algorithm (see [BM91]). The easiest way to see that this indeed returns the majority is perhaps to note that if we remove any pair a_i, a_{i+1} with $a_i \neq a_{i+1}$, then the majority is the same in the remaining sequence so at iteration i the algorithm (implicitly) replaces $a_1, \ldots, a_i$ with M occurrences of the integer C not cancelled out yet. This concludes the proof of (Item 2 of) the theorem. ■

Application to Subset Sum

Theorem 6.3 Given an instance (n, ω, t) of Subset Sum and an integer S as input such that $|\{\omega(X) : X \subseteq \{1, \ldots, n\}\}| \leq S$,

1. an integer y such that $\text{prob}[x = y] \geq \frac{1}{2}$ can be computed in $\mathcal{O}^\star(S)$ expected time and $\mathcal{O}^\star(1)$ space, and
2. x can be computed in $\mathcal{O}^\star(S^2)$ time and $\mathcal{O}^\star(1)$ space.

Where $x = |\{X \subseteq \{1, \ldots, n\} : \omega(X) = t\}|$.

Proof Consider the recurrence (5.3). Although it was originally formulated over the ring $\mathbb{Z}[\mathbb{Z}_{nt}]$, of course it is equivalent when formulated in the ring $\mathbb{Z}[\mathbb{N}]$ since then all elements indexed by integers at least nt are zero. So let C be the circuit over $\mathbb{Z}[\mathbb{N}]$ implementing recurrence (5.3), and apply Theorem 6.2 to C. It is easy to see that $\text{supp}(\boldsymbol{s}(n)) = \{\omega(X) : X \subseteq \{1, \ldots, n\}\}$ (see (5.3)) and as explained also in Section 5.1.1, $\texttt{mpl}(C) = n - 1$. Then both items follow from Theorem 6.2. ■

Also, it follows straightforwardly from Algorithm 6.2 that the algorithms of Theorem 6.3 return 0 only if the given instance of Subset Sum is a `NO`-instance. Then, using standard self-reduction to detect false positives in combination with Theorem 6.3 and iteratively doubling an estimate of S, we can also obtain algorithms that do not require an upper bound on the number of non-zero entries in the DP table as input:

Corollary 6.1 Given an instance (n, ω, t) of Subset Sum it can be decided whether it is a YES-instance in (i) $\mathcal{O}^\star(S)$ expected time, or in (ii) $\mathcal{O}^\star(S^2)$ time, where $S = |\{\omega(X) : X \subseteq \{1, \ldots, n\}\}|$.

6.2 Homomorphic Hashing for Linear Satisfiability

In this section, we let $\mathbb{F}$ be an arbitrary field of non-even characteristic[3] and the standard addition and multiplication refer to operations in $\mathbb{F}$. We first revisit the Walsh-Hadamard transform.

For a non-negative integer s, the *Walsh-Hadamard matrix* is the matrix $\mathbf{\Phi} \in \mathbb{F}^{\mathbb{Z}_2^s \times \mathbb{Z}_2^s}$, defined for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_2^s$ by $\Phi_{\boldsymbol{x},\boldsymbol{y}} = (-1)^{\boldsymbol{x}\boldsymbol{y}^T}$. It follows that $\mathbf{\Phi} = \boldsymbol{F}_{\boldsymbol{m}}$, where $m = (2, 2, \ldots, 2)$, where $\boldsymbol{F}$ is the *Fourier transform* (see Section 4.3). It inherits the following properties from Theorem 4.1 and Lemma 4.1:

Lemma 6.3 — Folklore The Walsh-Hadamard matrix satisfies $\mathbf{\Phi}\mathbf{\Phi} = 2^s \boldsymbol{I}$ and, for every $\boldsymbol{f}, \boldsymbol{g} \in \mathbb{F}[\mathbb{Z}_2^s]$, it holds that $(\boldsymbol{f} * \boldsymbol{g})\mathbf{\Phi} = \boldsymbol{f}\mathbf{\Phi} \circ \boldsymbol{g}\mathbf{\Phi}$.

Proof For the first equality, let $\boldsymbol{A} = \mathbf{\Phi}\mathbf{\Phi}$. Then for every $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_2^s$ we have

$$\begin{aligned} A_{\boldsymbol{x},\boldsymbol{y}} &= \sum_{\boldsymbol{z} \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{x}\boldsymbol{z}^T + \boldsymbol{y}\boldsymbol{z}^T} \\ &= \sum_{\boldsymbol{z} \in \mathbb{Z}_2^s} (-1)^{(\boldsymbol{x}+\boldsymbol{y})\boldsymbol{z}^T} \\ &\quad \left(\text{using that } \textstyle\sum_{\boldsymbol{p} \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{q}\boldsymbol{p}^T} = 2^s[\boldsymbol{q} = \boldsymbol{0}]\right) \\ &= 2^s[\boldsymbol{x} = \boldsymbol{y}]. \end{aligned}$$

For the second equality, let $\boldsymbol{z} \in \mathbb{Z}_2^s$ and observe that

$$\begin{aligned} (\boldsymbol{f}\mathbf{\Phi} \circ \boldsymbol{g}\mathbf{\Phi})_{\boldsymbol{z}} &= \sum_{\boldsymbol{x} \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{z}\boldsymbol{x}^T} f_{\boldsymbol{x}} \sum_{\boldsymbol{y} \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{z}\boldsymbol{y}^T} g_{\boldsymbol{y}} \\ &= \sum_{\boldsymbol{x},\boldsymbol{y} \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{z}(\boldsymbol{x}+\boldsymbol{y})^T} f_{\boldsymbol{x}} g_{\boldsymbol{y}} \\ &= \sum_{\boldsymbol{w} \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{z}\boldsymbol{w}^T} (f * g)_{\boldsymbol{w}} \\ &= \mathbf{\Phi}(f * g)_{\boldsymbol{z}}. \end{aligned}$$

■

In particular, if $\mathbb{F}$ is a field of non-even characteristic, then $\mathbf{\Phi}$ is a bijective homomorphism (or equivalently, isomorphism) from $\mathbb{F}[\mathbb{Z}_2^s]$ to $\mathbb{F}^{\mathbb{Z}_2^s}$.

[3] This is to make sure the Walsh-Hadamard transform does not make non-zero entries vanish.

Theorem 6.4 Let $\mathbb{F}$ be a field of non-even characteristic. There exists a randomized algorithm that, given as input (i) a circuit C with singleton inputs over $\mathbb{F}[\mathbb{Z}_2^n]$, (ii) an integer $S \geq |\mathrm{supp}(\boldsymbol{v})|$, and (iii) an element $\boldsymbol{t} \in \mathbb{Z}_2^n$, outputs the coefficient $\boldsymbol{v}_{\boldsymbol{t}} \in \mathbb{F}$ with probability at least $\frac{1}{2}$, where $\boldsymbol{v} \in \mathbb{F}[\mathbb{Z}_2^n]$ is the output of C. The algorithm (a) runs in time $\mathcal{O}^\star(S)$ and uses $\mathcal{O}^\star(S)$ arithmetic operations in $\mathbb{F}$, and (b) requires storage for $\mathcal{O}^\star(1)$ bits and elements of $\mathbb{F}$.

Function `hashZ2`

1: Let $s = \lceil \log S \rceil + 1$.
2: Choose a matrix $\boldsymbol{H} \in \mathbb{Z}_2^{s \times n}$ uniformly at random from the set of all $s \times n$ matrices with binary entries.
3: Let $h : \mathbb{F}[\mathbb{Z}_2^n] \to \mathbb{F}[\mathbb{Z}_2^s]$ be the homomorphism defined by $h(\boldsymbol{a}) = \boldsymbol{b}$ where $\boldsymbol{b}_{\boldsymbol{x}} = \sum_{\boldsymbol{y} \in \mathbb{Z}_2^n : \boldsymbol{y}\boldsymbol{H} = \boldsymbol{x}} \boldsymbol{a}_{\boldsymbol{y}}$ for all $\boldsymbol{x} \in \mathbb{Z}_2^s$. Apply h to C to obtain the circuit C_1.
4: **return** $\dfrac{1}{2^s} \displaystyle\sum_{\boldsymbol{x} \in \mathbb{Z}_2^s} (-1)^{(\boldsymbol{t}\boldsymbol{H})\boldsymbol{x}^T}\ \mathtt{sub}(C_1, \boldsymbol{x})$.

Function $\mathtt{sub}(C_1, \boldsymbol{x})$

5: Let $\varphi : \mathbb{F}[\mathbb{Z}_2^s] \to \mathbb{F}$ be the homomorphism defined by $\varphi(\boldsymbol{w}) = \sum_{\boldsymbol{y} \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{x}\boldsymbol{y}^T}\ w_{\boldsymbol{y}}$ for all $\boldsymbol{w} \in \mathbb{F}[\mathbb{Z}_2^s]$. Apply φ to C_1 to obtain the circuit C_2.
6: Evaluate C_2 and return the output.

Algorithm 4 – Implementing Theorem 6.4.

Proof (of Theorem 6.4) The algorithm is given in Algorithm 4. Let us first analyse the complexity of this algorithm: Steps 1 and 2 can be done in time polynomial in the input. Step 3 can as well be done in polynomial time since it amounts to relabeling all input gates with $h(\boldsymbol{e})$ where $\boldsymbol{e}$ was the old label. Indeed, we know that $\boldsymbol{e} \in \mathbb{F}[\mathbb{Z}_2^n]$ is a singleton $\langle v, \boldsymbol{y} \rangle$, so $h(\boldsymbol{e})$ is the singleton $\langle v, \boldsymbol{H}\boldsymbol{y} \rangle$ and this can be computed in polynomial time. Step 4 takes $\mathcal{O}^\star(S)$ operations and calls to `sub`, so for the complexity bound it remains to show that each call to `sub` runs in polynomial time. Step 5 can be implemented in polynomial time similar to Step 3 since the singleton $\boldsymbol{e} = \langle v, \boldsymbol{y} \rangle$ is mapped to $(-1)^{\boldsymbol{x}\boldsymbol{y}^T} v$. Finally, the direct evaluation of C_2 uses $|C_2|$ operations in $\mathbb{F}$. Hence the algorithm meets the time bound, and also the space bound is immediate.

The correctness of `hashZ2` is a consequence of the following two claims. Let $\boldsymbol{w}$ be the output of C_1.

Claim 6.4.1 $\mathrm{prob}_{\boldsymbol{H}}[\boldsymbol{v}_{\boldsymbol{t}} = \boldsymbol{w}_{\boldsymbol{t}\boldsymbol{H}}] \geq \frac{1}{2}$.

Proof For every $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}[\mathbb{Z}_2^n]$ we have $(\boldsymbol{a} + \boldsymbol{b})\boldsymbol{H} = \boldsymbol{a}\boldsymbol{H} + \boldsymbol{b}\boldsymbol{H}$ and hence h is easily seen to be a homomorphism by Observation 4.1. Thus, by Observation 4.1 we

know that $\boldsymbol{w} = h(\boldsymbol{v})$, that is, for every $\boldsymbol{z} \in \mathbb{Z}_2^s$

$$\boldsymbol{w}_{\boldsymbol{z}} = \sum_{\boldsymbol{y} \in \mathbb{Z}_2^n : \boldsymbol{y}\boldsymbol{H} = \boldsymbol{z}} \boldsymbol{v}_{\boldsymbol{z}}.$$

Hence, if $\boldsymbol{v}_{\boldsymbol{t}} \neq \boldsymbol{w}_{\boldsymbol{t}\boldsymbol{H}}$, there must exist $\boldsymbol{y} \in \mathrm{supp}(v)$ such that $\boldsymbol{y} \neq \boldsymbol{t}$ and $\boldsymbol{y}\boldsymbol{H} = \boldsymbol{t}\boldsymbol{H}$. Equivalently, $(\boldsymbol{y} - \boldsymbol{t})\boldsymbol{H} = \boldsymbol{0}$. For any $\boldsymbol{x} \in \mathbb{Z}_2^n$ with $\boldsymbol{x} \neq \boldsymbol{0}$ we have

$$\mathrm{prob}_{\boldsymbol{H}}[\boldsymbol{x}\boldsymbol{H} = 0] = \prod_{i=1}^{s} \mathrm{prob}_{\boldsymbol{H}}[(\boldsymbol{x}\boldsymbol{H})_i = 0] = 2^{-s},$$

where the probability is taken uniform over all binary $s \times n$ matrices, and the two equalities follow from the fact that the random variables $(\boldsymbol{x}\boldsymbol{H})_i$ for $i = 1, \ldots, s$ are independent and uniformly distributed. Now the claim follows by taking the union bound over all elements in the support:

$$\mathrm{prob}[\boldsymbol{v}_{\boldsymbol{t}} \neq \boldsymbol{w}_{\boldsymbol{t}\boldsymbol{H}}] \leq \mathrm{prob}[\exists \boldsymbol{x} \in \mathrm{supp}(\boldsymbol{v}) \text{ with } \boldsymbol{x} \neq \boldsymbol{t} \text{ and } \boldsymbol{x}\boldsymbol{H} = \boldsymbol{e}\boldsymbol{H}] \leq |S| 2^{-s} \leq \frac{1}{2} \quad \blacksquare$$

Claim 6.4.2 Algorithm `hashZ2` returns $\boldsymbol{w}_{\boldsymbol{t}\boldsymbol{H}}$.

Proof Let $\boldsymbol{b} \in \mathbb{F}[\mathbb{Z}_2^s]$ such that $\boldsymbol{b}_{\boldsymbol{x}} = \mathtt{sub}(C_1, \boldsymbol{x})$ for every $\boldsymbol{x} \in \mathbb{Z}_2^s$. It suffices to show that $\boldsymbol{b} = \boldsymbol{w}\boldsymbol{\Phi}$ since `hashZ2` returns $\frac{1}{2^s}(\boldsymbol{b}\boldsymbol{\Phi})_{\boldsymbol{t}\boldsymbol{H}}$ as can bee seen from Line 4, and this is equal to $\boldsymbol{w}_{\boldsymbol{H}\boldsymbol{t}}$ by Lemma 6.3. For proving that $\boldsymbol{b} = \boldsymbol{w}\boldsymbol{\Phi}$, we first claim that φ is a homomorphism from $\mathbb{F}[\mathbb{Z}_2^s]$ to $\mathbb{F}$ since, for $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}[\mathbb{Z}_2^s]$, we have

$$\varphi(\boldsymbol{a} + \boldsymbol{b}) = \sum_{\boldsymbol{y} \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{x}\boldsymbol{y}^T} (a_{\boldsymbol{y}} + b_{\boldsymbol{y}}) = \sum_{\boldsymbol{y} \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{x}\boldsymbol{y}^T} a_{\boldsymbol{y}} + \sum_{\boldsymbol{y} \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{x}\boldsymbol{y}^T} b_{\boldsymbol{y}} = \varphi(\boldsymbol{a}) + \varphi(\boldsymbol{b}),$$

and $\varphi(\boldsymbol{a} * \boldsymbol{b})$ is equal to

$$\sum_{\boldsymbol{y} \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{x}\boldsymbol{y}^T} \sum_{\boldsymbol{y}_1 + \boldsymbol{y}_2 = \boldsymbol{y}} a_{\boldsymbol{y}_1} + b_{\boldsymbol{y}_2} = \sum_{\boldsymbol{y}_1 \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{x}\boldsymbol{y}_1^T} a_{\boldsymbol{y}_1} \sum_{\boldsymbol{y}_2 \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{x}\boldsymbol{y}_2^T} b_{\boldsymbol{y}_2} = \varphi(\boldsymbol{a}) \circ \varphi(\boldsymbol{b}).$$

Then, by Observation 4.1, $\mathtt{sub}(C_1, \boldsymbol{x})$ returns $\varphi(\boldsymbol{w})$. For $\boldsymbol{a} \in \mathbb{Z}[\mathbb{Z}_2^s]$ we have $\varphi(\boldsymbol{a}) = (\boldsymbol{a}\boldsymbol{\Phi})_{\boldsymbol{x}}$ so $\mathtt{sub}(C_1, \boldsymbol{x}) = (\boldsymbol{w}\boldsymbol{\Phi})_{\boldsymbol{x}}$ and hence $\boldsymbol{b} = \boldsymbol{w}\boldsymbol{\Phi}$. $\blacksquare$

Now Claims 6.4.1 and 6.4.2 combined imply directly that `hashZ2` returns $\boldsymbol{v}_{\boldsymbol{t}}$ with probability at least $\frac{1}{2}$. $\blacksquare$

6.2.1 Application to Linear Satisfiability

To motivate Theorem 6.4, we give some consequences of it for the following problem:

LINEAR SAT **Parameter:** n, m
Input: A matrix $\boldsymbol{A} \in \mathbb{Z}_2^{n \times m}$, vectors $\boldsymbol{b} \in \mathbb{Z}_2^m$ and $\boldsymbol{\omega} \in \mathbb{N}^n$ and integer $t = n^{O(1)}$.
Question: Is there a vector $\boldsymbol{x} \in \mathbb{Z}_2^n$ such that $\boldsymbol{xA} = \boldsymbol{b}$ and $\boldsymbol{\omega x}^T \leq t$?

Variants of LINEAR SAT have been studied, perhaps most notably in [Hås01], where approximability was studied. In [AGK+10, CGJY11] the Fixed Parameter Tractability (see Section 1.3) was studied for parameterizations above guarantees. There it was quoted from of [Hås01] that (variants of) LINEAR SAT are "as basic as satisfiability". Let us first mention that a first non-trivial algorithm is easily obtained using the approach from [HS74]. The idea is to obtain the iterate over all assignments of $\boldsymbol{x'} = (x_1, \ldots, x_{\lfloor n/2 \rfloor})$, and create a set $\mathcal{B}$ that contains all vectors $\boldsymbol{b'} = \boldsymbol{x'A'}$ where $\boldsymbol{A'}$ is the matrix consisting of the first $m/2$ columns of $\boldsymbol{A}$. Then sort $\mathcal{B}$ in lexicographical order and iterate over all assignments of the variables $x_{\lceil n/2 \rceil}, \ldots, x_n$ and find whether a matching assignment of $\boldsymbol{x'}$ exists using binary search on $\mathcal{B}$. Then, the following is obtained:

Observation 6.1 — (HS74) LINEAR SAT can be solved in $\tilde{\mathcal{O}}(2^{n/2}m)$ time and $\tilde{\mathcal{O}}(2^{n/2}m)$ space.

Also, using 'sparse dynamic programming', as in the proof of Theorem 6.1, the following can easily be obtained:

Observation 6.2 — (HS74) LINEAR SAT can be solved in $\tilde{\mathcal{O}}(2^{\mathrm{rk}(\boldsymbol{A})}(n+m))$ time and $\tilde{\mathcal{O}}(2^{\mathrm{rk}(\boldsymbol{A})}(n+m))$ space.

Using Theorem 6.4, we obtain the following:

Theorem 6.5 There exists an algorithm that, given an instance $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{\omega}, t)$ of LINEAR SAT, computes the number of $\boldsymbol{x} \in \mathbb{Z}_2^n$ with $\boldsymbol{xA} = \boldsymbol{b}$ and $\boldsymbol{x\omega}^T \leq t$ with probability at least $\frac{1}{2}$ in $\mathcal{O}^\star(2^{\mathrm{rk}(A)})$ time and $\mathcal{O}^\star(1)$ space.

Proof For $1 \leq i \leq n$ and $w \leq t$ define $f[i, w] \in \mathbb{Q}[\mathbb{Z}_2^n]$ as

$$\boldsymbol{f}[i,w] = \begin{cases} \langle 1, \mathbf{0} \rangle & \text{if } i = w = 0, \\ 0 & \text{if } i = 0 \wedge w \neq 0, \\ f[i-1, w + f[i-1, j-\omega_i] * \langle 1, \boldsymbol{A}^{(i)} \rangle & \text{otherwise.} \end{cases} \tag{6.4}$$

It is easy to see that for every $1 \leq i, 0 \leq w$, and $\boldsymbol{y} \in \mathbb{Z}_2^n$, $f[i, w]_{\boldsymbol{y}}$ is the number of $\boldsymbol{x} \in \mathbb{Z}_2^i$ such that $\boldsymbol{\omega' x}^T = t$ and $\boldsymbol{xA'} = \boldsymbol{y}$ where $\boldsymbol{\omega'}$ and $\boldsymbol{A'}$ are obtained by truncating $\boldsymbol{\omega}$ and $\boldsymbol{A}$ respectively to the first i columns. Hence, we let C be the

circuit implementing Recurrence 6.4 and let its output be $\sum_{w=0}^{t} \boldsymbol{f}[n,w]$. Thus, if $\boldsymbol{v}$ is the output of C, $v_{\mathbf{1}}$ is the number of $\boldsymbol{x} \in \mathbb{Z}_2^n$ with $\boldsymbol{xA} = \boldsymbol{b}$ and $\boldsymbol{x\omega}^T \leq t$.

Also, $|\mathrm{supp}(\boldsymbol{v})| \leq 2^{\mathrm{rk}(A)}$ since any element of the support of $\boldsymbol{v}$ is a sum of rows of A and hence in the column-space of A, which has size at most $2^{\mathrm{rk}(A)}$. To apply Theorem 6.4, let $\mathbb{F} = \mathbb{Q}$ and observe that the computations are in fact carried out over integers bounded in absolute value poly-exponentially in n and hence the operations in the base field can also be executed poly-logarithmically in n. The theorem follows from Theorem 6.4. ■

Now we give a so-called Win/Win approach for Linear Sat. First, let us see how to exploit that in an instance of Linear Sat, $\mathrm{rk}(A)$ is relatively high: by Theorem 1.3 we know that if $\boldsymbol{A}$ has full rank (that is, $\mathrm{rk}(\boldsymbol{A}) = \min\{n,m\}$) then there exists at most one vector $\boldsymbol{x}$ such that $\boldsymbol{xA} = \boldsymbol{b}$ and it can be found in polynomial time using Gauss elimination. Hence, we first find $\mathrm{rk}(\boldsymbol{A})$ linearly independent columns (which can as well be done in polynomial time using elementary linear algebra) and iterate over all subsets of the remaining columns, for every subset we determine whether it can be extended by picking a subset of the linearly independent columns in polynomial time (using Gauss elimination). If this is possible we check whether we found a solution, and if we haven't found any in the end we can safely return `NO`.

Observation 6.3 Linear Sat can be solved in $\mathcal{O}^\star(2^{n-\mathrm{rk}(\boldsymbol{A})})$ time and $\mathcal{O}^\star(1)$ space.

Now the idea of the Win/Win approach is to distinguish two scenarios which are dealt with by different arguments.

Theorem 6.6 Linear Sat can be solved in $\mathcal{O}^\star(2^{n/2})$ time and $\mathcal{O}^\star(1)$ space with constant one-sided error probability.

Proof Compute $\mathrm{rk}(\boldsymbol{A})$. If $\mathrm{rk}(\boldsymbol{A}) \geq n/2$, run the algorithm implied by Observation 6.3. Otherwise, run the algorithm implied by Theorem 6.5. ■

Set Partition

Set Partition **Parameter:** n
Input: An integer t and a set system $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = m, |U| = n$.
Question: Is there a subset $\mathcal{C} \subseteq \mathcal{F}$ with $|\mathcal{C}| \leq t$ such that $\bigcup_{S \in \mathcal{C}} S = U$ and for every $\sum_{S \in \mathcal{C}} |S| = |U|$?

We will refer to a subset $\mathcal{X} \subseteq S$ with $\sum_{S \in \mathcal{X}} |S| = n$ and $\cup_{S \in \mathcal{X}} S = U$ as a *set partition of size* $|\mathcal{X}|$. Given a set system $(\mathcal{F}, U)$ we let its *Incidence matrix* be the $|U| \times |\mathcal{F}|$ matrix $\mathcal{A}$ with 0/1 entries and index by elements of U and $\mathcal{F}$ where $A_{e,S} = [e \in S]$.

Theorem 6.7 There exists an algorithm that, given an instance $(\mathcal{F}, U, t)$ of Set Partition, computes the number of set partitions of size at most t with probability at least $\frac{1}{2}$ in $2^{\mathrm{rk}(A)} n^{\mathcal{O}(1)} m$ time and $\mathcal{O}^\star(1)$ space, where $\boldsymbol{A}$ is the incidence matrix of the set system $(\mathcal{F}, U)$.

Proof We use Theorem 6.5. Assume $\mathcal{F} = \{S_1, \ldots, S_m\}$ and create the instance $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{\omega}, t')$ of Linear Sat where $\boldsymbol{A}$ is the incidence matrix of the set system $(\mathcal{F}, U)$, $\boldsymbol{b} = 1$, $\omega_i = |S_i|m + 1$ and $t' = nm + t$. It is easy to see that the algorithm of Theorem 6.5 returns exactly the number of set partitions of size at most t ■

We can also obtain a faster exponential space variant, but for this we need the following special case of Theorem 4.2:

Theorem 6.8 — Fast Walsh-Hadamard transform, Folklore Given a vector $\boldsymbol{a} \in \mathbb{F}[\mathbb{Z}_2^s]$, $\boldsymbol{a\Phi}$ can be computed in time $\mathcal{O}(2^s s)$ and using $\mathcal{O}(2^s s)$ operations in $\mathbb{F}$.

Theorem 6.9 There exists an algorithm that, given an instance $(\mathcal{F}, U, t)$ of Set Partition, computes the number of set partitions with probability at least $\frac{1}{2}$ in $(2^{\mathrm{rk}(A)} + m) n^{\mathcal{O}(1)}$ time and $\mathcal{O}^\star(1)$ space, where $\boldsymbol{A}$ is the incidence matrix of the set system $(\mathcal{F}, U)$.

Proof Consider the following circuit C over $\mathbb{Q}[\mathbb{Z}_2^n]$:

$$f[i,j] = \begin{cases} \langle 1, \mathbf{0} \rangle & \text{if } i = j = 0 \\ 0 & \text{if } i = 0 \text{ and } j \neq 0 \\ \sum_{h=0}^{j} f[i-1,h] g[j-h] & \text{otherwise, where} \end{cases} \tag{6.5}$$

$$g[j] = \sum_{i=1}^{m} \left[|S_i| = j \right] \langle 1, S_i \rangle \tag{6.6}$$

For every $\boldsymbol{x} \in \mathbb{Z}_2^n$ and non-negative integers i and j, the coefficient $f[i,j]_{\boldsymbol{x}}$ counts the number of ways to choose an i-tuple of sets in $\mathcal{S}$ such that their sizes sum up to j and their characteristic vectors sum to $\boldsymbol{x}$ in $\mathbb{Z}_2^n$. Thus $\mathrm{supp}(f[k,n]) \leq 2^{\mathrm{rk}(A)}$. Furthermore, $f[k,n]_{\mathbf{1}}$ is the number of set partitions of size k times $k!$. Indeed, if a k-tuple of sets from $\mathcal{S}$ contributes to $f[k,n]_{\mathbf{1}}$, each element of U must occur in a unique set in the k-tuple. It remains to compute $\boldsymbol{f}[k,n]$. For this we will use algorithm `hashZ2` with $s = \mathrm{rk}(A)$, except that we replace Line 4 with the following to compute $\boldsymbol{w_{tH}}$, where $\boldsymbol{w}$ is the output of C_1:

4: **for** every $0 \le j \le n$ **do**
5: Compute and store $h(\boldsymbol{g}[j])$ using (6.6)
6: Compute and store $h(\boldsymbol{g}[j])\boldsymbol{\Phi}$ using Theorem 6.8
7: **return** $\frac{1}{2^s}\sum_{\boldsymbol{x}\in\mathbb{Z}_2^s}(-1)^{(\mathbf{1}\boldsymbol{H})\boldsymbol{x}^T}(\boldsymbol{w}\boldsymbol{\Phi})_{\boldsymbol{x}}$, using (6.5) and the stored values to compute the vector $\boldsymbol{w}\boldsymbol{\Phi}$.

Algorithm 5 – Changes to Algorithm 4 to implement Theorem 6.9.

The correctness follows from Claim 6.4.1 and the observation that the inversion formula from Theorem 6.3 is returned on Line 7. Indeed, h is a homomorphism and Φ is a bijective homomorphism, so Observation 4.1 enables us to compute $\boldsymbol{w}\boldsymbol{\Phi}$ using (6.6).

To establish the time and space complexity, we observe that Steps 5 and 6 take $2^s n^{\mathcal{O}(1)}$ time by Theorem 6.8, and Step 7 takes also $2^s n^{\mathcal{O}(1)}$ time since we can compute $\boldsymbol{w}\boldsymbol{\Phi}$ via (6.5) in $O(n^2)$ operations in $\mathbb{Q}^{\mathbb{Z}_2^s}$ by relying on the stored values $h(\boldsymbol{g}[j])\boldsymbol{\Phi}$. ∎

6.3 Towards Homomorphic Hashing for CNF-Sat

In this section our objective is to mimic the approach of the previous sections for the semigroup algebra $\mathbb{F}[\mathbb{U}^n]$, where $\mathbb{F}$ is a field and $\mathbb{U}^n$ is the semigroup as defined in Section 1.3. For clarity, we use some alternative notation: instead of $\mathbb{U}^n$, we write $(2^U, \cup)$ so its elements are subsets of U and the group operator is the union.

The direct attempt to mimic, unfortunately, fails. Indeed, let h be an arbitrary homomorphism from $(2^U, \cup)$ to $(2^V, \cup)$ with $|V| < |U|$. Let $U = \{e_1, e_2, \ldots, e_n\}$ and consider the minimum value $1 \le j \le n-1$ with $h(\{e_1, \ldots, e_j\}) = \cup_{i=1}^{j} h(\{e_i\}) = \cup_{i=1}^{j+1} h(\{e_i\}) = h(\{e_1, \ldots, e_{j+1}\})$; in particular, for $X = \{e_1, \ldots, e_j, e_{j+2}, \ldots, e_n\} \neq U$ we have $h(X) = h(U)$, which signals failure since we cannot isolate $X \neq U$[(4)].

Our main contribution in this section is that we introduce a more promising homomorphism from the semigroup algebra $\mathbb{F}[(2^U, \cup)]$ to the Solomon algebra $\mathbb{F}[P]$ of a partially ordered set P. Unfortunately, we only achieved partial progress towards obtaining results similar to the ones of the previous sections. Our main result here is formulated as follows (See Subsection 6.3.1 for the additional definitions used.):

[(4)] let us emphasize that this only excludes the most naive approach as candidate.

Theorem 6.10 Let C be a circuit over $\mathbb{F}[(2^U, \cup)]$ with singleton inputs outputting $\boldsymbol{v} \in \mathbb{F}[(2^U, \cup)]$. Let $(P, \leq)$ be a poset that has a maximum element $\hat{1}$ and that satisfies

1. $U \subseteq P$,
2. for every $X \in \operatorname{supp}(\boldsymbol{v})$, X has a join in P, and
3. for every $X \in \operatorname{supp}(\boldsymbol{v})$, the join of X in P is $\hat{1}$ if and only if $X = U$.

Then $\boldsymbol{v}_U$ can be computed (a) in $\tilde{\mathcal{O}}(|P|^2|C|^{\mathcal{O}(1)})$ time and arithmetic operations in $\mathbb{F}$, and (b) using storage for $|P||C|^{\mathcal{O}(1)}$ elements of $\mathbb{F}$.

One possible application is the CNF-Sat problem: recall that here we are given a CNF-formula $\mathcal{C} = \{C_1, \ldots, C_m\}$ over the variables $v_1, \ldots, v_n$ as input, the task is to determine whether there exists an assignment of $0/1$ values to the variables such that every clause C_j contains at least one literal (a variable or its negation) that is satisfied by the assignment.

A *prefix assignment* is an assignment of $0/1$ values to the variables $v_1, v_2, \ldots, v_i$ for some $1 \leq i \leq n$. A *projection* (*prefix projection*) of a CNF-formula is a subset $\pi \subseteq \mathcal{C}$ such that there exists an assignment (prefix assignment) of the variables that satisfies a clause if and only if it is in π. Considering the previous section, it is sensible to ask about the complexity of CNF-Sat if we are given an upper bound on the number of projections. It should be noted that an algorithm running in time linear in the number of *partial* projections can be obtained by standard dynamic programming. A corollary of Theorem 6.10 that is possibly useful for resolving this question is:

Corollary 6.2 Let $\mathcal{C}$ be a CNF-formula and let $(P, \leq)$ be a poset that has a maximum element $\hat{1}$ and that satisfies

1. $\mathcal{C} \subseteq P$,
2. for every projection $\pi \subseteq \mathcal{C}$, π has a join in P, and
3. for every $\pi \in \operatorname{supp}(v)$, the join of π is $\hat{1}$ iff $\pi = \mathcal{C}$.

Then the number of satisfying assignments of $\mathcal{C}$ can be counted $|P|^2(mn)^{\mathcal{O}(1)}$ time and $|P|(mn)^{\mathcal{O}(1)}$ space.

Proof Use the circuit over $\mathbb{Q}[(2^{\mathcal{C}}, \cup)]$ that implements the expression

$$\boldsymbol{f} = (\langle 1, V_1\rangle + \langle 1, \overline{V_1}\rangle) * (\langle 1, V_2\rangle + \langle 1, \overline{V_2}\rangle) * \ldots * (\langle 1, V_m\rangle + \langle 1, \overline{V_m}\rangle),$$

where $V_i \subseteq \mathcal{C}$ ($\overline{V_i} \subseteq \mathcal{C}$) is the set of all clauses that contain a positive (negative) literal of v_i. Then use Theorem 6.10 to determine $f_{\mathcal{C}}$, the number of satisfying

assignments of $\mathcal{C}$; Item 1 and 2 of Theorem 6.10 are clearly satisfied, and for item 2 it is easy to see that $\mathrm{supp}(\boldsymbol{f})$ is exactly the set of projections of $\mathcal{C}$. ■

It remains to prove Theorem 6.10. The next section reviews the necessary background on the Solomon algebra. Section 6.3.2 introduces the homomorphic hash function and proves Theorem 6.10.

6.3.1 Möbius Inversion and the Solomon Algebra

Recall the standard poset terminology from Section 1.3 and recall Möbius inversion and the Möbius function μ from Section 6.3.1.

Definition 6.1 — (Sol67) Let $(P, \leq)$ be a poset. The *Solomon algebra* $\mathbb{F}[P]$ is the set $\mathbb{F}^P$ equipped with coordinate-wise addition $\oplus$ and the *Solomon product* $\otimes$ defined for all $\boldsymbol{f}, \boldsymbol{g} \in \mathbb{F}[P]$ and $z \in P$ by

$$(\boldsymbol{f} \oplus \boldsymbol{g})_z = f_z + g_z \qquad (\boldsymbol{f} \otimes \boldsymbol{g})_z = \sum_{x,y \in P} \left(\sum_{x,y \leq q \leq z} \mu(q, z) \right) f_x g_y$$

It is easy to see that, in the special case that P is the subset lattice on 2^n elements, $\mathbb{F}[P] = \mathbb{F}[\mathbb{U}^n]$. In the general case, the Solomon product is easily verified to be associative:

Lemma 6.4 — (Sol67) For every $\boldsymbol{f}, \boldsymbol{g}, \boldsymbol{h} \in \mathbb{F}[P]$ and $s \in P$ we have

$$((\boldsymbol{f} \otimes \boldsymbol{g}) \otimes \boldsymbol{h})_s = \sum_{x,y,z \leq s} \sum_{x,y,z \leq q} \mu(q, s) f_x g_y h_z$$

Proof

$$
\begin{aligned}
((\boldsymbol{f}\otimes\boldsymbol{g})\otimes\boldsymbol{h})_s &= \sum_{w,z\in P}\left(\sum_{w,z\le r\le s}\mu(r,s)\right)(\boldsymbol{f}\otimes\boldsymbol{g})_w h_z \\
&\quad \text{(Definition of the Solomon product)} \\
&= \sum_{w,z\in P}\left(\sum_{w,z\le r\le s}\mu(r,s)\right)\left(\sum_{x,y\in P}\sum_{x,y\le q\le w}\mu(q,w)f_x g_y\right)h_z \\
&\quad \text{(Reordering summations)} \\
&= \sum_{w,x,y,z\in P}\left(\sum_{\substack{w,z\le r\le s\\ x,y\le q\le w}}\mu(r,s)\mu(q,w)\right)f_x g_y h_z \\
&\quad \text{(Reordering summation)} \\
&= \sum_{x,y,z\le s}\sum_{\substack{x,y\le q\\ z\le r}}\left(\sum_{q\le w\le r}\mu(q,w)\right)\mu(r,s)f_x g_y h_z \\
&\quad \text{(Möbius inversion (Theorem 4.4))} \\
&= \sum_{x,y,z\le s}\sum_{\substack{x,y\le q\\ z\le r}}[q=r]\mu(r,s)f_x g_y h_z \\
&\quad \text{(Variable identification)} \\
&= \sum_{x,y,z\le s}\sum_{x,y,z\le q}\mu(q,s)f_x g_y h_z. \qquad \blacksquare
\end{aligned}
$$

Associativity then follows from Lemma 6.4 since $((\boldsymbol{f}\otimes\boldsymbol{g})\otimes\boldsymbol{h})_s = (\boldsymbol{f}\otimes(\boldsymbol{g}\otimes\boldsymbol{h}))_s$. The Solomon product is particularly useful since it is diagonalized by the zeta-transformation:

Theorem 6.11 — (Sol67) The zeta transform is a homomorphism from $\mathbb{F}[P]$ to $\mathbb{F}^P$

Proof For every $w \in P$ we have

$$((\boldsymbol{f} \oplus \boldsymbol{g})\boldsymbol{\zeta})_w = \sum_{z \le w} (f_z + g_z) = \sum_{z \le w} f_z + \sum_{z \le w} g_z = (\boldsymbol{f}\boldsymbol{\zeta})_w + (\boldsymbol{g}\boldsymbol{\zeta})_w, \quad \text{and}$$

$$\begin{aligned}
((\boldsymbol{f} \otimes \boldsymbol{g})\boldsymbol{\zeta})_w &= \sum_{z \le w} \sum_{x,y \in P} \left(\sum_{x,y \le q \le z} \mu(q,z) \right) f_x g_y \\
&= \sum_{x,y,q,z \in P} [x, y \le q \le z \le w] \mu(q,z) f_x g_y \\
&\quad \big(\text{Reordering summation}\big) \\
&= \sum_{x,y,q \in P} [x, y \le q \le w] \left(\sum_{q \le z \le w} \mu(q,z) \right) f_x g_y \\
&\quad \big(\text{Möbius inversion (Theorem 4.4)}\big) \\
&= \sum_{x,y,q \in P} [x, y \le q \le w][q = w] f_x g_y \\
&= \sum_{x,y \in P} [x, y \le w] f_x g_y \\
&= \left(\sum_{x \le w} f_x \right) \left(\sum_{y \le w} g_y \right) \\
&= (\boldsymbol{f}\boldsymbol{\zeta})_w (\boldsymbol{g}\boldsymbol{\zeta})_w.
\end{aligned}$$

■

Lemma 6.5 — (Sol67) Let P be a poset with a minimum element $\hat{0}$. Then, $\mathbb{F}[P]$ is a commutative ring with the multiplicative identity $\langle 1, \hat{0} \rangle$.

Proof The singleton $\langle 1, \hat{0} \rangle$ is the multiplicative identity because

$$(\langle 1, \hat{0} \rangle \otimes \boldsymbol{g})_z = \sum_{y \le z} \left(\sum_{y \le q \le z} \mu(q,z) \right) g_y = g_z,$$

where the last equality follows from Möbius inversion. To see that $\mathbb{F}[P]$ is a commutative ring, note that $\boldsymbol{\zeta}$ is an isomorphism from $\mathcal{S}[P]$ to the ring $\mathbb{F}^P$. Indeed, Theorem 6.11 shows that $\boldsymbol{\zeta}$ is a homomorphism, and $\boldsymbol{\zeta}\boldsymbol{\mu} = \boldsymbol{\mu}\boldsymbol{\zeta} = \boldsymbol{I}$ shows that ζ is bijective. ■

6.3.2 Towards homomorphic hashing for the union product

Let U be an n-element set and let $(P, \le)$ be a poset that satisfies $U \subseteq P$ and has a minimum element $\hat{0}$. Define the function $h : \mathbb{F}[(2^U, \cup)] \to \mathbb{F}[P]$ by setting, for all $\boldsymbol{a} \in \mathbb{F}[(2^U, \cup)]$,

$$h(\boldsymbol{a}) = \bigoplus_{X \subseteq U} \langle a_X, \hat{0} \rangle \otimes \bigotimes_{e \in X} \langle 1, e \rangle . \tag{6.7}$$

Lemma 6.6 For singletons $\langle c, x\rangle, \langle d, x\rangle \in \mathbb{F}[P]$ it holds that $\langle c, x\rangle \otimes \langle d, x\rangle = \langle cd, x\rangle$.

Proof For all $z \in P$, we have $(\langle c, x\rangle \otimes \langle d, x\rangle)_z =$

$$\sum_{w,y\in P}\left(\sum_{w,y\le q\le z}\mu(q,z)\right)[w = x]c[y = x]d = \sum_{x\le q\le z}\mu(q,z)cd = [x = z]cd.$$

■

Lemma 6.7 The mapping h is a homomorphism from $\mathbb{F}[(2^U, \cup)]$ to $\mathbb{F}[P]$.

Proof For every $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{Z}[(2^U, \cup)]$ we have

$$\begin{aligned}
h(\boldsymbol{v} + \boldsymbol{w}) &= \bigoplus_{X\subseteq U}\left\langle v_X + w_X, \hat{0}\right\rangle \otimes \bigotimes_{e\in X}\langle 1, e\rangle \\
&= \bigoplus_{X\subseteq U}(\left\langle v_X, \hat{0}\right\rangle \oplus \left\langle w_X, \hat{0}\right\rangle) \otimes \bigotimes_{e\in X}\langle 1, e\rangle) \\
&= \bigoplus_{X\subseteq U}\left(\left\langle v_X, \hat{0}\right\rangle \otimes \bigotimes_{e\in X}\langle 1, e\rangle \oplus \left\langle w_X, \hat{0}\right\rangle \otimes \bigotimes_{e\in X}\langle 1, e\rangle\right) \\
&= \bigoplus_{X\subseteq U}\left\langle v_X, \hat{0}\right\rangle \otimes \bigotimes_{e\in X}\langle 1, e\rangle \oplus \bigoplus_{X\subseteq U}\left\langle w_X, \hat{0}\right\rangle \otimes \bigotimes_{e\in X}\langle 1, e\rangle = h(\boldsymbol{v}) \oplus h(\boldsymbol{w}), \\
h(\boldsymbol{v} * \boldsymbol{w}) &= \bigoplus_{X\subseteq U}\left\langle (\boldsymbol{v} * \boldsymbol{w})_X, \hat{0}\right\rangle \otimes \bigotimes_{e\in X}\langle 1, e\rangle \\
&\quad \big(\text{Definition of multiplication in } \mathbb{F}[(2^U, \cup)]\big) \\
&= \bigoplus_{X\subseteq U}\left\langle \sum_{V\cup W=X} v_V w_W, \hat{0}\right\rangle \otimes \bigotimes_{e\in X}\langle 1, e\rangle \\
&\quad \big(\text{By definition of } \oplus\big) \\
&= \bigoplus_{X\subseteq U}\bigoplus_{V\cup W=X}\left\langle v_V w_W, \hat{0}\right\rangle \otimes \bigotimes_{e\in X}\langle 1, e\rangle \\
&\quad \big(\text{By Lemma 6.6 and commutativity of } \otimes \text{ implied by Lemma 6.5}\big) \\
&= \bigoplus_{X\subseteq U}\bigoplus_{V\cup W=X}\left\langle v_V w_W, \hat{0}\right\rangle \otimes \bigotimes_{e\in V}\langle 1, e\rangle \otimes \bigotimes_{e\in W}\langle 1, e\rangle \\
&\quad \big(\text{We have that } V \text{ and } W \text{ determine } X\big) \\
&= \bigoplus_{V,W\subseteq U}\left\langle v_V w_W, \hat{0}\right\rangle \otimes \bigotimes_{e\in V}\langle 1, e\rangle \otimes \bigotimes_{e\in W}\langle 1, e\rangle \\
&\quad \big(\text{By Lemma 6.6 }\big) \\
&= \bigoplus_{V,W\subseteq U}(\left\langle v_V, \hat{0}\right\rangle \otimes \left\langle w_W, \hat{0}\right\rangle) \otimes \bigotimes_{e\in V}\langle 1, e\rangle \otimes \bigotimes_{e\in W}\langle 1, e\rangle \\
&\quad \big(\text{By distributivity from Lemma 6.5}\big) \\
&= \left(\bigoplus_{X\subseteq U}\left\langle v_X, \hat{0}\right\rangle \otimes \bigotimes_{e\in X}\langle 1, e\rangle\right) \otimes \left(\bigoplus_{X\subseteq U}\left\langle v_X, \hat{0}\right\rangle \otimes \bigotimes_{e\in X}\langle 1, e\rangle\right) = h(\boldsymbol{v}) \otimes h(\boldsymbol{w}).
\end{aligned}$$

■

Proof (of Theorem 6.10) The algorithm is given in Algorithm 6. Note that in the following, we can without loss of generality assume that P has a minimum element $\hat{0}$, since otherwise we can introduce a minimum element.

Function `hashU`
1: For every $x \in P$ compute the Möbius function $M[x] = \mu(x, \hat{1})$ using `mobius`$(P, \leq)$.
2: Construct circuit C_1 over $\mathbb{F}[P]$ obtained from C by applying h.
3: **return** $\sum_{x \in P} M[x]$`sub`(C_1, x)

Function `sub`(C_1, x)
4: Construct circuit C_2 over $\mathbb{F}$ obtained from C_1 by applying $\zeta_x : \boldsymbol{e} \mapsto \sum_{y \leq x} e_y$.
5: Evaluate C_2 and return the output.

Function `mobius`$(P, \leq)$
6: Let $P = \{v_1, v_2, \ldots, v_{|P|}\}$ such that $v_i \leq v_j$ implies $i \geq j$.
7: $M[v_1] \leftarrow 1$.
8: **for** $i = 1, 2, \ldots, |P|$ **do**
9: **for** every $v_j \geq v_i$ **do**
10: $M[v_i] \leftarrow M[v_i] - M[v_j]$

Algorithm 6 – Implementing Theorem 6.10

Let us first analyse the complexity. To establish the claimed bounds we may assume that each operation in $\mathbb{F}$ takes one time unit, and each element of $\mathbb{F}$ requires one storage unit. Step 1 can be implemented in $O(|P|^2)$ time and $O(|P|)$ storage. Step 2 expands every singleton (of $\mathbb{F}[(2^U, \cup)]$) in C according to (6.7) into a product of at most $n+1$ singletons of $\mathbb{F}[P]$, and thus can be implemented in time and storage $O(n|C|)$. Step 3 makes $|P|$ calls to `sub`. Step 4 can be implemented in time and space $O(|C_1|)$ because the singleton $\boldsymbol{e} = \langle v, y \rangle$ is mapped to $[y \leq x]v$. Finally, the evaluation of C_2 over $\mathbb{F}$ uses $|C_2|$ operations. Hence by the assumption $|C| \geq n$ the algorithm meets the claimed time and space bounds.

The proof of correctness of `hashU` is divided into the following two Claims. Denote by $\boldsymbol{w} \in \mathbb{F}[P]$ the output of circuit C_1.

Claim 6.11.1 Algorithm `hashU` returns $w_{\hat{1}}$.

Proof Algorithm `mobius` follows the recurrence (4.3) and thus $M[x]$ is $\mu(x, \hat{1})$. Let $\boldsymbol{b} \in \mathbb{F}[P]$ such that for every $x \in P$, we have $b_x =$ `sub`(C_1, x). Note that algorithm `find2` returns $(\boldsymbol{b}\boldsymbol{\mu})_{\hat{1}}$ in Step 3, so by Möbius inversion it is sufficient to show that $\boldsymbol{b} = \boldsymbol{w}\boldsymbol{\zeta}$, which follows from Observation 4.1 since the zeta transform is applied to circuit C_1, which outputs $\boldsymbol{w}$. ∎

Claim 6.11.2 $w_{\hat{1}} = v_U$.

Proof Since h is a homomorphism by Lemma 6.7, $\boldsymbol{w} = h(\boldsymbol{v})$ by Observation 4.1. So it remains to show that $(h(\boldsymbol{v}))_{\hat{1}} = v_U$.

$$
\begin{aligned}
(h(\boldsymbol{v}))_{\hat{1}} &= \left(\bigoplus_{X \subseteq U} \langle v_X, \hat{0} \rangle \otimes \bigotimes_{e \in X} \langle 1, e \rangle \right)_{\hat{1}} \\
&\quad \big(\text{for } X \notin \operatorname{supp}(\boldsymbol{v}) \text{ we have } v_X = 0\big) \\
&= \left(\bigoplus_{X \in \operatorname{supp}(\boldsymbol{v})} \langle v_X, \hat{0} \rangle \otimes \bigotimes_{e \in X} \langle 1, e \rangle \right)_{\hat{1}} \\
&\quad \big(\text{Definition of } \oplus\big) \\
&= \sum_{X \in \operatorname{supp}(\boldsymbol{v})} \left(\langle v_X, \hat{0} \rangle \otimes \bigotimes_{e \in X} \langle 1, e \rangle \right)_{\hat{1}} \\
&\quad \big(\text{Iteratively applying Lemma 6.4, denoting } X = \{e_1, \ldots, e_{|X|}\}\big) \\
&= \sum_{X \in \operatorname{supp}(\boldsymbol{v})} \sum_{a_0, \ldots, a_{|X|} \in P} \left(\sum_{a_0, \ldots, a_{|X|} \leq q \leq \hat{1}} \mu(q, \hat{1}) \right) v_X [a_0 = \hat{0}] \prod_{i=1}^{|X|} [a_i = e_i] \\
&\quad \big(X \text{ has a join by Assumption 2}\big) \\
&= \sum_{X \in \operatorname{supp}(\boldsymbol{v})} \left(\sum_{e_1 \vee \ldots \vee e_{|X|} \leq q \leq \hat{1}} \mu(q, \hat{1}) \right) v_X \\
&\quad \big(\text{Möbius inversion}\big) \\
&= \sum_{X \in \operatorname{supp}(\boldsymbol{v})} \sum_{e_1, \ldots, e_{|X|} \in P} [e_1 \vee \ldots \vee e_{|X|} = \hat{1}] v_X \\
&\quad \big(\text{Assumption 3}\big) \\
&= v_U. \qquad \blacksquare
\end{aligned}
$$

Combining the above claims yields the theorem. ■

6.4 Concluding Remarks

Let us first emphasize that the hash functions used in the first two sections are commonly used; for example Color-Coding also uses (much more involved) hashing with prime numbers (see for example [FG06, Section 13.3] for a proof) and a slightly related approached was used by Kane [Kan10]. The hashing in

Section 6.2 is similar to the randomized algorithm for matrix product verification from Section 1.1 and a variant is for example also used in [PP10].

As mentioned in the introduction, one property of homomorphic hashing that is particularly appealing is that it can be used to obtain running times depending on the structure of *total candidate solutions* rather than *partial candidate solutions*. For example, if we are looking for a set partition in Section 6.2 consisting of k sets, the studied algorithm runs in time proportional to the number of distinct sums of the characteristic vectors in $\mathbb{F}[\mathbb{Z}_2^n]$ of *exactly* k sets, rather then *at most* k sets for dynamic programming.

For problems like CNF-Sat it might be interesting whether a similar construction is possible: an example question in this direction could be, given a CNF-formula ϕ and an integer S such that ϕ has at most S projections, can its satisfiability be determined in $\mathcal{O}^\star(S^c)$ for some constant c? Can we given faster-than-brute-force algorithm if the number of projections is small?

It should be noted that the three hashing approaches are somewhat complementary; for example, the Traveling Salesman Problem could potentially benefit from each approach because it has a numeric aspect (the cost of a Hamiltonian path) and as well a "partition" or "covering" aspect (the fact that the path is Hamiltonian). Here we have focused on sparse instances, but a dual line of study would be to investigate what happens when the corresponding DP tables are as *dense* as possible.

Open Question 6 — (Woe08) Can Subset Sum be solved in $\mathcal{O}^\star(2-\epsilon)^n$ time and $\mathcal{O}^\star(1)$ space, for some $\epsilon > 0$?

Perhaps a Win/Win approach could be used to answer Open Question 6 similarly to the algorithm for Linear Sat. For example, what happens when the dynamic programming is "dense"? Consider the following problem:

All distinct Subset Sum **Parameter:** t
Input: Integer n, function $\omega : \{1,\ldots,n\} \to \mathbb{N}$ such that for every $X, Y \subseteq \{1,\ldots,n\}$ it holds that $\omega(X) = \omega(Y) \to X = Y$, and integer t.
Question: Does there exist a subset $X \subseteq \{1,\ldots,n\}$ such that $\omega(X) = t$?

Can the "all distinct" property be exploited and can we get a $\mathcal{O}^\star(2-\epsilon)^n$-time and $\mathcal{O}^\star(1)$-space algorithm for this problem, for some $\epsilon > 0$? And if so, does it help to resolve Open Question 6? Note that similar questions and a similar problem called Equal Subset Sum was already introduced and studied in [Woe08, WY92].

Chapter 7

Solving Connectivity Problems Parameterized by Treewidth in Single Exponential Time

This chapter is based on the following paper:

[CNP+11a]. 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 *FOCS*, 2011. To appear. Full version: [CNP+11b]

The notion of *treewidth* (recall the definition from Section 1.3) was introduced independently by Rose in 1974 [Ros74] (under the name of partial k-tree) and in 1984 by Robertson and Seymour [RS84], and in many cases proved to be a good measure of the intrinsic difficulty of various NP-hard problems on graphs, and a useful tool for attacking those problems. Many of them can be efficiently solved through dynamic programming if we assume the input graph to have bounded treewidth. For example, an expository algorithm to solve Vertex Cover and Independent Set running in time[(1)] $\mathcal{O}^\star(4^{\mathbf{tw}(G)})$ is described in the algorithms textbook by Kleinberg and Tardos [KT06], while the book of Niedermeier [Nie06] on fixed-parameter algorithms presents an algorithm with running time $\mathcal{O}^\star(2^{\mathbf{tw}(G)})$.

The interest in algorithms for graphs of bounded treewidth stems from their usefulness: such algorithms are used as sub-routines in a variety of settings. Amongst them prominent are approximation algorithms [BHM10, BKMK07, DH08, Epp00] and Fixed Parameter Tractable (see Section 1.3) algorithms [DFT06,

[(1)] When stating running times involving $\mathbf{tw}(G)$ we assume the algorithm is given a treedecomposition of G of width at most $\mathbf{tw}(G)$.

FGSS09] 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 parameterized algorithms in general graphs [MRR08, TSB05] for problems like Connected Vertex Cover and Cutwidth; and exact algorithms [FGSS09, vRNvD09] 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 neighborhood of each vertex), matching upper and lower bounds for the runtime of the optimal solution are known. For instance, for the aforementioned $\mathcal{O}^\star(2^{\mathbf{tw}(G)})$ algorithm for Vertex Cover there is a matching lower bound — unless the Strong Exponential Time Hypothesis (see Chapter 2) fails, there is no algorithm for Vertex Cover running faster than $(2-\varepsilon)^{\mathbf{tw}(G)}$ for any $\varepsilon > 0$ (see [LMS11a]).

On the other hand, when the problem involves some sort of a "global" constraint — e.g., connectivity — the best known algorithms usually have a runtime on the order of $\mathcal{O}^\star(2^{\mathcal{O}(\mathbf{tw}(G)\log\mathbf{tw}(G))})$. In these cases the typical dynamic programming routine has to keep track of all the ways in which the solution can traverse the corresponding separator of the tree decomposition, that is $\Omega(l^l)$ on the size l of the separator, and therefore of treewidth. This obviously implies weaker results in the applications mentioned above. This problem was observed, for instance, by Dorn, Fomin and Thilikos [DFT06, DFT08] and by Dorn et al. in [DPBF10], and the question whether the known $\mathcal{O}^\star(2^{\mathcal{O}(\mathbf{tw}(G)\log\mathbf{tw}(G))})$-time parameterized algorithms for Hamiltonian Path, Connected Vertex Cover and Connected Dominating Set are optimal was for the first time explicitly asked by Lokshtanov, Marx and Saurabh [LMS11b].

The $\mathcal{O}^\star(2^{\mathcal{O}(\mathbf{tw}(G)\log\mathbf{tw}(G))})$ 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 [HMU01], 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, the results of this chapter come as a significant surprise.

7.0.1 Our Results

In this chapter we introduce a technique we name "Cut&Count". Briefly stated, we first reduce the original problem to the task of counting possibly disconnected "cut solutions" modulo 2 by (i) making sure that the number of disconnected cut solutions is always even and (ii) using randomization to make sure that the

number of connected cut solutions is odd iff there is a solution. The reduction is performed in such a way that counting cut solutions is a local problem and can be done sufficiently fast by standard dynamic programming.

For most problems involving a global constraint, our technique gives a randomized algorithm with runtime $\mathcal{O}^\star(c^{\mathbf{tw}(G)})$. In particular we are able to give such algorithms for the three problems mentioned in [LMS11b], as well as for all the other sample problems mentioned in [DFT08]. Moreover, the constant c is in all cases well defined and small. The randomization we mention comes from the usage of the Isolation Lemma (see Section 4.5). This gives us Monte Carlo algorithms with one-sided error. The formal statement of a typical result is as follows:

Theorem 7.1 — (CNP+11b) There exists a randomized algorithm that, given a graph $G = (V, E)$, a tree decomposition of G of width t and a number k in $\mathcal{O}^\star(3^t)$ 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.

Less elaborately, we call an algorithm as in Theorem 7.1 an algorithm with *constant error probability*. We see similar results for a number of other global problems. As the exact value of c in the $c^{\mathbf{tw}(G)}$ expression is often important and highly non-trivial to obtain, we gather the results in the second column of Table 7.1.

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

Theorem 7.2 Unless the Strong Exponential Time Hypothesis is false, there do not exist a constant $\varepsilon > 0$ and an algorithm that given an instance $(G = (V, E), T, k)$ together with a path decomposition of the graph G of width p solves the Connected Dominating Set problem in $\mathcal{O}^\star((4 - \varepsilon)^p)$ time.

Since each path decomposition is also a tree decomposition, a lower bound for pathwidth is at least as strong as for treewidth (see also Observation 1.1). We have such matching lower bounds for several other problems presented in the third column of Table 7.1. 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.

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

Problem name	A	B	C	D
Steiner Tree[2]	$3^{\mathbf{tw}(G)}$•	$3^{\mathbf{pw}(G)}$		
Feedback Vertex Set	$3^{\mathbf{tw}(G)}$	$3^{\mathbf{pw}(G)}$	3^k•	3.83^k [CCL10]
Connected Vertex Cover	$3^{\mathbf{tw}(G)}$	$3^{\mathbf{pw}(G)}$	2^k•	2.4882^k [BR10]
Connected Dominating Set	$4^{\mathbf{tw}(G)}$	$4^{\mathbf{pw}(G)}$		
Connected Feedback Vertex Set	$4^{\mathbf{tw}(G)}$	$4^{\mathbf{pw}(G)}$	3^k	46.2^k [MPR+10]
Connected Odd Cycle Transversal	$4^{\mathbf{tw}(G)}$	$4^{\mathbf{pw}(G)}$		
Undirected/Directed Min Cycle Cover	$\frac{4^{\mathbf{tw}(G)}}{6^{\mathbf{tw}(G)}}$•			
Undirected/Directed Longest Path (Cycle)	$\frac{4^{\mathbf{tw}(G)}}{6^{\mathbf{tw}(G)}}$			
Exact k-Leaf Spanning Tree	$4^{\mathbf{tw}(G)}$	$4^{\mathbf{pw}(G)}$		
Exact k-Leaf Outbranching	$6^{\mathbf{tw}(G)}$			
Maximum Full Degree Spanning Tree	$4^{\mathbf{tw}(G)}$			
Graph Metric Traveling Salesman Problem	$4^{\mathbf{tw}(G)}$			
(Directed) Cycle Packing		$2^{\Omega(\mathbf{pw}(G)\log\mathbf{pw}(G))}$		
(Directed) Max Cycle Cover		$2^{\Omega(\mathbf{pw}(G)\log\mathbf{pw}(G))}$		
Maximally Disconnected Dominating Set		$2^{\Omega(\mathbf{pw}(G)\log\mathbf{pw}(G))}$•		

Table 7.1 – 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 $\mathbf{tw}(G)$) parametrization. For example, the entry at the third row represents Theorem 7.1. Column B contains lower bounds showing problems that are not solvable in $\mathcal{O}^\star(c^{\mathbf{pw}(G)})$ time for any constant c unless the Strong Exponential Time Hypothesis fails, and in $2^{\Omega(\mathbf{pw}(G)\lg\mathbf{pw}(G))}$ time unless the Exponential Time Hypothesis fails, where $\mathbf{pw}(G)$ is the pathwidth. Columns C is devoted to time complexities of our algorithm for solution size parameterization, whereas Column D contains previously best known time complexities. All problems statements can be found in [CNP+11b]; entries marked with a • are proved in this thesis while all remaining proofs can be found in the full version [CNP+11b].

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 evidence, lower bounds for the time complexity of solutions to such problems are given in [CNP+11b], proving that $c^{\mathbf{tw}(G)}$ solutions of these problems are unlikely:

[2] In this chapter, Steiner Tree will denote the variant with unit weights (as opposed to Chapter 3).

Theorem 7.3 — (CNP+11b) Unless the Exponential Time Hypothesis is false, there does not exist an $\mathcal{O}^\star(2^{o(p \log p)})$-time algorithm solving Cycle Packing or Max Cycle Cover (both in the directed and the undirected setting). The parameter p denotes the width of a given path decomposition of the input graph.

To further verify this intuition, we also investigate the (artificial) Maximally Disconnected Dominating Set problem, in which we ask for a dominating set with the largest possible number of connected components, and indeed we find a similar phenomenon, which we will explain in Section 7.3.

7.0.2 Previous Work

The Cut&Count technique has two main ingredients. The first step we assure that objects we are not interested in are counted an even number of times, and then do the calculations in $\mathbb{Z}_2$ (or for example any other field of characteristic 2), which causes them to disappear. This line of reasoning goes back to Tutte [Tut47], and was recently used by Björklund [Bjö10b] and Björklund et. al [BHKK10a].

The second step 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 for example the Held and Karp relaxation for the Traveling Salesman Problem problem from 1970 [HK70, HK71] 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. [BGRS10]. To the best of our knowledge the idea of defining problems through cuts has not been used in the exact and parameterized settings.

A number of papers circumvent the problems stemming from the lack of single exponential algorithms parameterized by treewidth for connectivity–type problems. For instance in the case of parameterized algorithms, sphere cuts [DFT06, DPBF10] (for planar and bounded genus graphs) and Catalan structures [DFT08] (for H-minor-free graphs) were used to obtain $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$-time 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 $\mathcal{O}^\star(2^{o(\mathbf{tw}(G) \log \mathbf{tw}(G))})$-time algorithms was deemed to be more likely.

7.0.3 Consequences of the Cut&Count Technique

As already mentioned, algorithms for graphs with 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. In this thesis we do not explore all possible applications, but only give sample applications in various directions.

We would like to emphasize that the strength of the Cut&Count technique

shows not only in the quality of the results obtained in various fields, which are frequently better than the previously best known ones, achieved through a plethora of techniques and approaches, but also in the ease in which new strong results can be obtained.

Consequences for FPT Algorithms

Let us recall the definition of the Feedback Vertex Set problem:

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

This problem is on Karp's original list of 21 NP-complete problems [Kar72]. It has also been extensively studied from the parameterized 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 $\mathcal{O}^\star(f(k))$, where n is the input size and f is some function (usually exponential in k) (see also Section 1.3). 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. [BBYG00, Bod94, CFL+08, DFL+05, DF06, DF99, GGH+06, KPS04, RSS02, RSS06]. The best — so far — result in this series is the $\mathcal{O}^\star(3.83^k)$ result of Cao, Chen and Liu [CCL10]. Our contribution is:

Theorem 7.4 There exists a Monte Carlo algorithm with constant one-sided error probability that solves the Feedback Vertex Set problem in a graph $G = (V, E)$ in $\mathcal{O}^\star(3^k)$ time and polynomial space.

We also give similar improvements for Connected Vertex Cover (from $\mathcal{O}^\star(2.4882^k)$ time of [BR10] to $\mathcal{O}^\star(2^k)$ time) and Connected Feedback Vertex Set (from $\mathcal{O}^\star(46.2^k)$ time of [MPR+10] to $\mathcal{O}^\star(3^k)$ time).

Parameterized Algorithms for H-minor-free graphs

A large branch of applications of algorithms parameterized by treewidth is the bidimensionality theory, used to find subexponential time algorithms for various problems in H-minor-free graphs. In this theory we use the theorem of Demaine et al. [DHiK05], which ensures that any H-minor-free graph either has treewidth bounded by $\mathcal{O}(\sqrt{k})$, or a $2\sqrt{k} \times 2\sqrt{k}$ grid as a minor. In the latter case we are assumed to be able to answer the problem in question (for instance a $2\sqrt{k} \times 2\sqrt{k}$ grid as a minor guarantees that the graph does not have a Vertex Cover or Connected Vertex Cover smaller than k). Thus, we are left with solving

the problem with the assumption of bounded treewidth. In the case of for instance Vertex Cover, a standard dynamic programming algorithm suffices, thus giving us a $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ 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 $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k}\log k)})$ complexity — thus, we lose a logarithmic factor in the exponent.

There have been 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 [DFT08], 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-$\mathcal{O}$ notation these are rarely considered to be important in the bidimensionality theory), but rather in simplifying the proof — instead of delving into the combinatorial structure of each particular problem, we are back to a simple framework of applying the Robertson-Seymour theorem and then following up with a dynamic programming algorithm on the obtained tree decomposition.

Consequences for Exact Algorithms for Graphs of Bounded Degree

Another application of our methods can be found in the field of solving problems with a global constraint in graphs of bounded degree. The problems that have been studied in this setting are mostly local in nature (such as Vertex Cover, see, e.g., [BEPvR10]); however global problems such as, for example, the Traveling Salesman Problem (TSP) and Hamiltonian Cycle have also received considerable attention [BHKK10b, Epp07, Geb08, IN07].

Throughout the following, we let n denote the number of vertices of the given graph. The starting point is the following theorem by Fomin et al. [FGSS09]:

Theorem 7.5 — (FGSS09) 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 $\mathcal{O}^\star(3^{\mathbf{tw}(G)})$ and

$\mathcal{O}^\star(4^{\mathbf{tw}(G)})$-time algorithms) 5. Furthermore in the [CNP+11b], we improve the general $\mathcal{O}^\star(4^{\mathbf{tw}(G)})$-time algorithm for Hamiltonian Cycle to $\mathcal{O}^\star(3^{\mathbf{pw}(G)})$ time in case of a path decomposition of cubic graphs[(3)]. Consequently we prove the following theorem, which improves over previously best results for maximum degree three $\mathcal{O}(1.251^n)$-time algorithm of Iwama and Nakashima [IN07], and for degree four $\mathcal{O}(1.657^n)$-time, which is the algorithm of Björklund [Bjö10b].

Corollary 7.1 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 in $O(1.588^n)$ time for graphs of maximum degree 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 [FT04]:

Proposition 7.1 — (FT04) 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 $\mathcal{O}(c^{\sqrt{n}})$-time 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 7.2 There exists a Monte Carlo algorithm with constant one-sided error probability that solves the Hamiltonian Cycle problem on planar graphs in $\mathcal{O}(4^{3.183\sqrt{n}}) = \mathcal{O}(2^{6.366\sqrt{n}})$ time.

To the best of our knowledge the best algorithm known so far was the $\mathcal{O}(2^{6.903\sqrt{n}})$-time algorithm of Bodlaender et al. [DPBF10].

Similarly, we obtain an $\mathcal{O}(2^{6.366\sqrt{n}})$-time algorithm for Longest Cycle on planar graphs (compare to the $\mathcal{O}(2^{7.223\sqrt{n}})$ time of [DPBF10]), and — as in the previous subsections — well-behaved $c^{\sqrt{n}}$-time algorithms for all mentioned problems.

7.0.4 Organization of this Chapter

This chapter is organised as follows. In Section 7.1 we present the Cut&Count technique on two examples: the Steiner Tree problem and the Directed

(3) The observation, which was observed and shared with us by Andreas Björklund, is that the dynamic programming subroutine used by our algorithm can be improved because many values of the dynamic programming will always be zero.

Min Cycle Cover problem. In Section 7.2 we give the $\mathcal{O}^\star(3^k)$-time algorithm for Feedback Vertex Set and the $\mathcal{O}^\star(2^k)$-time algorithm for Connected Vertex Cover when parameterized by the solution size, whereas in Section 7.3 we move to lower bounds. We end this chapter with a number of conclusions and open problems in Section 7.4.

As the reader might have already noticed, there is a quite a large amount of material covered in the paper [CNP+11b] on which this chapter is based. In order to prevent that this thesis loses focus, we will only include and discuss a subset of the results from [CNP+11b].

7.0.5 Notation

Here we introduce some notation we will use on top of the notation already introduced in Section 1.3. Just as $G[X]$ for $X \subseteq V(G)$ denotes a subgraph induced by the set X of vertices, we use $G[X]$ for $X \subseteq E(G)$ for the graph (V, X), where $G = (V, E)$. Note that in the graph $G[X]$ for an edge set X the set of vertices remains the same as in the graph G.

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

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

We denote the symmetric difference of two sets A and B by $A \triangle B$. For a function $s : A \to B$ and $a \in A$,$b \in B$, we denote $s[a \to b]$ be the function equal to s except that $s(a) = b$. Also, recall the following from Section 1.3. For two integers a, b we use $a \equiv b$ to indicate that a is even if and only if b is even. If $\omega : U \to \{1, \ldots, N\}$, we shorthand $\omega(S) := \sum_{e \in S} \omega(e)$ for $S \subseteq U$.

7.1 Cut&Count: Illustration of the Technique

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

The Cut&Count technique applies to problems with certain connectivity requirements. Let $\mathcal{S} \subseteq 2^U$ be a set of solutions; we aim to decide whether it is empty. Conceptually, Cut&Count can naturally be split in two parts:

- **The Cut part**: Relax the connectivity requirement by considering the set $\mathcal{R} \supseteq \mathcal{S}$ of possibly connected candidate solutions. Furthermore, consider the set $\mathcal{C}$ of pairs (X, C) where $X \in \mathcal{R}$ and C is a consistent cut (to be defined later) of X.

- **The Count part**: Compute $|\mathcal{C}|$ modulo 2 using a sub-procedure. Non-connected candidate solutions $X \in \mathcal{R} \setminus \mathcal{S}$ cancel since they are consistent with an even number of cuts. Connected candidates $x \in \mathcal{S}$ remain.

Note that we need the number of solutions to be odd in order to make the counting part work. For this we use the Isolation Lemma (Lemma 4.2): we introduce uniformly and independently chosen weights $\omega(v)$ for every $v \in U$ and compute $|\mathcal{C}_W|$ modulo 2 for every integer W, where $\mathcal{C}_W = \{(X, C) \in \mathcal{C} \mid \omega(X) = W\}$. Let us recall that for two integers a, b we use $a \equiv b$ to indicate that a is even if and only if b is even. The general setup can thus be summarized as in Algorithm 7.

Function `cutandcount`$(U, \mathbb{T}, \texttt{CountC})$
Input Set U; tree decomposition $\mathbb{T}$; Procedure `CountC` accepting a weight function $\omega : U \to \{1, \ldots, N\}$, integer $W \in \mathbb{Z}$ and $\mathbb{T}$.
1: **for** every $v \in U$ **do**
2: Choose $\omega(v) \in \{1, \ldots, 2|U|\}$ uniformly at random.
3: **for** every $0 \leq W \leq 2|U|^2$ **do**
4: **if** `CountC`$(\omega, W, \mathbb{T}) \equiv 1$ **then return** `YES`
5: **return** `NO`

Algorithm 7 – `cutandcount`$(U, \mathbb{T}, \texttt{CountC})$

The following corollary that we use throughout the paper follows from Lemma 4.2 by setting $\mathcal{F} = \mathcal{S}$ and $N = 2|U|$:

Corollary 7.3 Let $\mathcal{S} \subseteq 2^U$ and $\mathcal{C} \subseteq 2^U \times (2^V \times 2^V)$. Suppose that for every $W \in \mathbb{Z}$:

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

Then Algorithm 7 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 counted is even for each W and provide an algorithm `CountC`. In the next two subsections, we illustrate both parts by giving two specific applications.

7.1.1 Steiner Tree

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

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

Definition 7.1 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 $G[X]$.

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

Let v_1 be an arbitrary terminal. Define $\mathcal{C}_W$ to be the set of all consistently cut subgraphs $(X, (X_1, X_2))$ such that $X \in \mathcal{R}_W$ and $v_1 \in X_1$. Before we proceed with the Count part, let us state the following easy combinatorial identity:

Lemma 7.1 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 $2^{\mathsf{cc}(G[X])-1}$.

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 $\mathsf{cc}(G[X]) - 1$ other connected components we are free to choose a side of a cut, which gives $2^{\mathsf{cc}(G[X])-1}$ possibilities leading to different consistently cut subgraphs. ■

The Count part. The following lemma shows that the first condition of Corollary 7.3 is indeed met:

Lemma 7.2 Let $G, \omega, \mathcal{C}_W$, $\mathcal{S}_W$, and $\mathcal{R}_W$ be as above. Then for all W, $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$.

Proof By Lemma 7.1, we know that $|\mathcal{C}_W| = \sum_{X \in \mathcal{R}_W} 2^{\mathtt{cc}(G[X])-1}$. Thus $|\mathcal{C}_W| \equiv \left|\{X \in \mathcal{R}_W | \mathtt{cc}(G[X]) = 1\}\right| = |\mathcal{S}_W|$. ■

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

Lemma 7.3 Given $G = (V, E)$, $T \subseteq V$, an integer k, $\omega : V \to \{1, \ldots, N\}$ and a tree decomposition $\mathbb{T}$ of G of width t, there exists an algorithm that can determine $|\mathcal{C}_W|$ modulo 2 for every $0 \leq W \leq kN$ in $\mathcal{O}^\star(3^t N^2)$ time.

Proof We start with transforming $\mathbb{T}$ to a nice tree decomposition of the same width which can be done in polynomial time. We will use dynamic programming, but we first need some preliminary definitions. Recall that for a bag $x \in \mathbb{T}$ we denoted by V_x the set of vertices of all descendants of x, while by G_x we denoted the graph composed of vertices V_x and the edges E_x introduced by the descendants of x. We now define "partial solutions": For every bag $x \in \mathbb{T}$, integers $0 \leq i \leq k$, $0 \leq w \leq kN$ and $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^{B_x}$ define

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

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

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

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 colors) 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 $A_r(k, W, \varnothing)$ for all W (recall that r is the root of the tree decomposition), because $|\mathcal{C}_W| = |\mathcal{C}_r(k, W)|$.

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

- **Leaf bag** x:

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

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

- **Introduce vertex** v **bag** x:

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

  For the first case, note that by definition, v can not be colored $\mathbf{0}$ if it is a terminal. For the other cases, the accumulators have to be updated and we have to make sure we do not put $s(v_1) = \mathbf{1}_2$.

- **Introduce edge** uv **bag** x:

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

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

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

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

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

- **Join bag**:

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

The only valid combinations to achieve the coloring s is to have the same coloring in both children. Since vertices colored $\mathbf{1}_j$ 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 by obtaining the following theorem.

Theorem 7.6 There exists a Monte-Carlo algorithm that given a tree decomposition of width t solves Steiner Tree in $\mathcal{O}^\star(3^t)$ time. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.

Proof Run Algorithm 7 by setting $U = V$, and `CountC` to be the algorithm implied by Lemma 7.3. The correctness follows from Corollary 7.3 by setting $\mathcal{S} = \bigcup_W \mathcal{S}_W$ and $\mathcal{C} = \bigcup_W \mathcal{C}_W$ and Lemma 7.2. It is easy to see that the timebound follows from Lemma 7.3. ■

7.1.2 Directed Min 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?

In this problem the aim is to maximize connectivity in a more flexible way than in the previous section: in the previous section the solution induced one connected component, while it may induce at most k weakly connected components in the context of this 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. Formal definition follows.

The Cut part. As said before, we assume that we are given a weight function $\omega : U = A \times \{\mathbf{X}\} \cup A \times \{\mathbf{M}\} \to \{1, \ldots, N\}$, where $N = 2|U| = 4|A|$. 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 belong to M as well.

Definition 7.2 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 7.4 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 7.1) 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:

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

which equals $|\mathcal{S}_W|$ by the definition. Now, it suffices to present a dynamic programming routine counting $|\mathcal{C}_W|$ modulo 2 in a bottom-up fashion. Before we give this, let us give the following definition and theorem that we will use:

Definition 7.3 — (CP10, vRBR09). Let $p \geq 2$ be an integer constant and let B be a finite set. For $t_1, t_2, t \in \{0, 1, \ldots, p-1\}^B$ we say that $t_1 + t_2 = t$ if and only if $t_1(b) + t_2(b) = t(b)$ for all $b \in B$. For functions $f, g : \{0, 1, \ldots, p-1\}^B \to \mathbb{B}$ define their generalized subset convolution $*^p$ as

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

Note that here the addition **is not** evaluated in $\mathbb{Z}_p$ but in $\mathbb{Z}$. The following can be proved using Theorem 4.2 in combination with a technique similar to the one of Subsection 5.2.3.

Theorem 7.7 — Generalized Subset Convolution (CP10, vRBR09) Given $f, g : \{0, \ldots, p-1\}^B \to \mathbb{B}$, their generalized subset convolution $f *^p g$ can be computed in $p^{|B|}|B|^{O(1)}$ time.

Lemma 7.5 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 D 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 $\mathcal{O}^\star(6^t N^2)$ time.

Proof Again, we start with transforming $\mathbb{T}$ to a nice tree decomposition and use dynamic programming. We follow the notation from the Steiner Tree example (see Section 7.1.1). Let $\Sigma = \{\mathbf{00}, \mathbf{01}_1, \mathbf{01}_2, \mathbf{10}_1, \mathbf{10}_2, \mathbf{11}\}$. For every bag $x \in \mathbb{T}$ of the tree decomposition, integers $0 \leq i \leq |V|$, $0 \leq w \leq 2N|V|$ and $s \in \Sigma^{B_x}$ (called the *coloring*) define

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

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

The algorithm computes $A_x(i, w, s)$ for all bags $x \in \mathbb{T}$ 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, 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**:

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

- **Introduce vertex bag**:

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

  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 $\text{insubs}, \text{outsubs} : \Sigma \to 2^{\Sigma}$.

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

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

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

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

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

- **Forget vertex v bag x**:

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

The forgotten vertex must have degree two.

- **Join bag**: We have two children y and z. Figure 7.1 shows how two individual states of a vertex in y and z combine to a state of x. XX indicates that two states do not combine. The correctness of the table is easy to check.

	$\mathbf{00}$	$\mathbf{01}_1$	$\mathbf{01}_2$	$\mathbf{10}_2$	$\mathbf{10}_1$	$\mathbf{11}$
$\mathbf{00}$	$\mathbf{00}$	$\mathbf{01}_1$	$\mathbf{01}_2$	$\mathbf{10}_2$	$\mathbf{10}_1$	$\mathbf{11}$
$\mathbf{01}_1$	$\mathbf{01}_1$	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 7.1 – The join table of Directed Min Cycle Cover indicating which states combine to which other states.

For colorings $s_1, s_2, s \in \Sigma^{B_x}$ 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 7.1. We can now write the recurrence formula for join bags.

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

A straightforward computation of the above formula leads to $36^t|V|^{O(1)}$ time complexity. We now introduce a definition and Theorem that can be used 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 **1**'s in a state s, i.e., the sum of all degrees of vertices in B_x. Then, define

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

We claim that

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

To see this, first notice that the values of accumulators are divided among the children, and that no vertex or edge is accounted for twice by the definition of A_x. Hence, it suffices to prove that exactly all combinations of table entries from A_y and A_z that combine to state s acccording to Table 7.1 contribute to $A_x(i,w,s)$. 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 7.1 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 7.7, the values $A_x(i,w,s)$ for a join bag x can be computed in time $\mathcal{O}^\star(6^t N^2)$.

It is easy to see that the above recurrence leads to a dynamic programming algorithm that computes the parity of $|\mathcal{C}_W| = A_r(k,W,\varnothing)$ for all values of W in $\mathcal{O}^\star(6^t N^2)$ time. Note that we count the parities and not the numbers A_x themselves, hence all arithmetical operations are done in constant time each. ■

Combining all the observations in the same way as in the proof of Theorem 7.6, we can conclude the following:

Theorem 7.8 There exists a Monte-Carlo algorithm that, given a tree decomposition of width t, solves Directed Min Cycle Cover in $\mathcal{O}^\star(6^t)$ time. The algorithm cannot give false positives and may give false negatives with probability at most 1/2.

7.2 Parameterizations by Solution Size

7.2.1 Feedback Vertex Set Parameterized by Solution Size

In this subsection we provide a proof of Theorem 7.4. Recall the definition of Feedback Vertex Set from Section 7.0.3. We will first give some intuition before proceeding to the formal proof.

Defining a solution candidate with a relaxed connectivity condition to work with our technique is somewhat more tricky here, as there is no explicit connectivity requirement in the problem. To overcome this, we reformulate the problem using the following simple lemma:

Lemma 7.6 A graph $G = (V, E)$ with n vertices and m edges is a forest if and only if G has at most $n - m$ connected components.

Now we use the Cut&Count technique. Instead of looking for the feedback vertex set we look for its complement, being a forest. Let $U = V \times \{\mathbf{F}, \mathbf{M}\}$, where $V \times \{\mathbf{F}\}$ is used to assign weights to vertices from the chosen forest and $V \times \{\mathbf{M}\}$ for markers. We also 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, W, we define:

1. $\mathcal{R}_W^{A,B}$ to be the family of pairs (X, M), where $X \subseteq V$, $|X| = A$, $G[X]$ contains exactly B edges, $M \subseteq X$, $|M| = n-k-B$, and $\omega(X \times \{\mathbf{F}\}) + \omega(M \times \{\mathbf{M}\}) = W$;

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

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

Observe that by Lemma 7.6 the graph G admits a feedback vertex set of size k if and only if there exist integers B, W such that the set $\mathcal{S}_W^{n-k,B}$ is nonempty.

The Count part. Similarly as in the case of Min Cycle Cover (analogously to Lemma 7.4), note that, for any A, B, $(X, M) \in \mathcal{R}_W^{A,B}$, there are $2^{\mathtt{cc}(M,G[X])}$ cuts (X_1, X_2) such that $((X, M), (X_1, X_2)) \in \mathcal{C}_W^{A,B}$, where $\mathtt{cc}(M, G[X])$ denotes the number of connected components of $G[X]$ which do not contain any marker from the set M. Hence, we have $|\mathcal{S}_W^{A,B}| \equiv |\mathcal{C}_W^{A,B}|$ for every A, B and W by Lemma 7.6.

It remains to show how to count $|\mathcal{C}_W^{n-k,B}|$ modulo 2 for every W and B in $3^k|V|^{\mathcal{O}(1)}$ time and polynomial space. For this we will combine the Cut&Count approach with *iterative compression* [RSV04]. The idea of *iterative compression* is to break the given task down in *compression steps*: we assume we are given a solution of size at most $k + 1$ and use it to find a solution of size at most k, or conclude that none exists. Here, we apply iterative compression by showing that, given a feedback vertex set of size $k + 1$, we can compute $|\mathcal{C}_W^{n-k,B}|$ modulo 2 in $3^k|V|^{\mathcal{O}(1)}$ time and polynomial space.

Proof (of Theorem 7.4) We will show that the procedure $\mathtt{fvs}(G, k)$ from Algorithm 8 requires at most $\mathcal{O}^\star(3^k)$ time and polynomial space and either returns a feedback vertex set in the graph G of size at most k or returns `NO`. We also show that if

Function `fvs`$(G = (V, E), k)$

$V_0 \leftarrow \varnothing$; $S_0 \leftarrow \varnothing$
for every $1 \le i \le |V|$ **do** — invariant: S_0 is a fvs of $G[V_0]$
  $V_i \leftarrow V_{i-1} \cup \{v_i\}$; $G_i = G[V_i]$; $S_i \leftarrow S_{i-1} \cup \{v_i\}$
  `found` $\leftarrow$ `false`; `repeat` $\leftarrow 0$; $p \leftarrow \varnothing$
  while `repeat` $< |V| \wedge$ `found` $=$ `false` **do** — boost error probability
    `repeat` $\leftarrow$ `repeat` $+ 1$
    for every $e \in (V \times \{\mathbf{M}, \mathbf{F}\})$ **do**
      Choose $\omega(e) \in \{1, \ldots, 4|V|\}$ at random
    if $\exists W, B$: `count`$(G_i, S_i, p, k, \omega, W, B) \equiv 1$ **then**
      `found` $\leftarrow$ `true`
      for every $0 \le j \le |V_i|$ **do** — write solution to p using self-reduction
        $c_{\mathbf{X}} \leftarrow$ `count`$(G_i, S_i, p[v_j \to \mathbf{X}], k, \omega, W, B)$
        $c_{\mathbf{O}} \leftarrow$ `count`$(G_i, S_i, p[v_j \to \mathbf{O}], k, \omega, W, B)$
        Let $\alpha \in \{\mathbf{X}, \mathbf{O}\}$ s.t. $c_\alpha \equiv 1$
        $p \leftarrow p[v_j \to \alpha]$
  if `found` $=$ `false` **then return no**
  $S_i \leftarrow \{v \in V_i : p(v) = \mathbf{O}\}$
return $S_{|V|}$

Function `count`$(G = (V, E), S, p, k, \omega, W, B)$

$r \leftarrow 0$
for every $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1}_2\}^S$ *consistent** with p **do**
  for every tree T_i in $G[V \setminus S]$, $1 \le i \le t$ **do**
    Arbitrarily fix a root r_i of T_i
    Compute all values $A_{r_i}(a, b, c, w, \alpha)$ by DP using Equations 7.1-7.2
  Compute all values $A_s(a, b, c, w)$ by DP using Equations 7.3-7.6.
  $r \leftarrow r + \sum_{c,w} q_w^c A_s(n - k, B, n - k - B - c, W - w)$
return r

* $s \in \{\mathbf{0}, \mathbf{1}_1, \mathbf{1_2}\}^S$ is *consistent* with p if:

$p^{-1}(\{\mathbf{X}\}) \subseteq s^{-1}(\{\mathbf{1}_1, \mathbf{1}_2\})$

$p^{-1}(\{\mathbf{O}\}) \subseteq s^{-1}(\{\mathbf{0}\})$

$(s^{-1}(\{\mathbf{1}_1\}) \times s^{-1}(\{\mathbf{1}_2\})) \cap E = \varnothing$ — the cut is consistent

Algorithm 8 – Implementing Theorem 7.4

the algorithm returns NO, then the probability that there exists a feedback vertex set of size at most k in G is at most $1/2$.

The proof has two parts: we first consider the procedure $\texttt{fvs}(G, k)$ and then consider the procedure $\texttt{count}(G, S, p, k, \omega, W, B)$. The second procedure is given a graph G, a feedback vertex set S in G, a set of requirements on the vertices V represented by p, the parameter k, a weight function $\omega : V \times \{\mathbf{M}, \mathbf{F}\} \to \{1, \ldots, 4|V|\}$, a target weight W, and an integer B. The procedure $\texttt{count}(G, S, p, k, \omega, W, B)$ computes the number of $((X, M), (X_1, X_2)) \in \mathcal{C}_W^{n-k,B}$ that are consistent with the requirements on the vertices V given by p, that is, the number of $((X, M), (X_1, X_2)) \in \mathcal{C}_W^{n-k,B}$ such that: the vertices $v \in V$ with $p(v) = \mathbf{X}$ are in the set X and the vertices $v \in V$ with $p(v) = \mathbf{O}$ are in the feedback vertex set (not in X).

Part 1. Consider the procedure $\texttt{fvs}(G, k)$. The main loop of this procedure starting at Line 2 implements the n compression steps discussed above. That is, the body of this loop computes a feedback vertex set in $G_i = G[V_i]$ of size at most k using the given feedback vertex set S_i of size at most $k + 1$, or it outputs **no** if no such set is found. The while-loop starting at Line 4 repeats the search for such a feedback vertex set at most n times. Each try uses a new weight function, which is initialised at Lines 6-7 by choosing each weight uniformly independently at random. Recall that if there exist B, W such that $|\mathcal{C}_W^{n-k,B}| \equiv 1$ (evaluated using G_i), then such a feedback vertex set of size at most k exists in G_i, as $|\mathcal{C}_W^{n-k,B}| \equiv |\mathcal{S}_W^{n-k,B}|$. The algorithm tests this by trying all possible B, W at Line 9 for which it calls the procedure $\texttt{count}(G_i, S_i, p, k, \omega, W, B)$ without imposing any requirements ($p = \varnothing$). If B, W such that $|\mathcal{C}_W^{n-k,B}| \equiv 1$ are found, then the algorithm constructs the corresponding feedback vertex set by a standard self-reduction: for each vertex $v \in V_i$ it guesses whether v should be in X or in the complementing feedback vertex set, and checks which guess is correct by testing whether $|\mathcal{C}_W^{n-k,B}| \equiv 1$ while keeping track of the guesses in the set of requirements p. Justifying Line 14, note that for one of the two guesses the counted quantity $|\mathcal{C}_W^{n-k,B}|$ indeed has to be odd, since the two guesses partition an odd-sized set. After completing the self-reduction, the feedback vertex set S_i of size at most k in G_i is extracted at Line 18.

In the second part of the proof below, we show that $\texttt{count}(G, S, p, k, \omega, W, B)$ can be implemented in $\mathcal{O}^\star(3^{|S|})$ time and polynomial space. It is not hard to see that, as a result, the procedure $\texttt{fvs}(G, k)$ runs in $\mathcal{O}^\star(3^k)$ time and polynomial space since the loops require $|V|^{\mathcal{O}(1)}$ repetitions, at most $|V|^{\mathcal{O}(1)}$ values of B and W need to be tested at each iteration, and $S_i \leq k+1$ at each call to the procedure $\texttt{count}(G_i, S_i, p, k, \omega, W, B)$.

If $\texttt{fvs}(G, k)$ returns a feedback vertex set, then it is clearly correct since $|\mathcal{S}_W^{n-k,B}| \equiv |\mathcal{C}_W^{n-k,B}| \equiv 1$, as argued in Section 7.2. We now analyse the error probability: the probability that the algorithm returns **no** while there exists a feedback

vertex set of size at most k. For each instantiation of the weight function ω, the probability that $\mathcal{C}_W^{n-k,B} \neq \varnothing$ while $|\mathcal{C}_W^{n-k,B}| \equiv 0$ is at most $1/2$. Since we try n independently instantiated weight functions at each iteration of the main loop, the probability that the algorithm incorrectly outputs **no** after n tries is at most 2^{-n}. We conclude that the total error probability is at most $n2^{-n}$ (which is less than $1/2$ if $n > 2$), as the body of the main loop is executed at most n times.

Part 2. Now, consider the procedure $\texttt{count}(G, S, p, k, \omega, W, B)$. We will show that it computes the number of pairs $((X, M), (X_1, X_2)) \in \mathcal{C}_W^{n-k,B}$ that are consistent with p, where p specifies that some vertices must be in X or must be in the complement of X. Note that $G = (V, E)$ is the graph of the current compression step of whole algorithm (i.e., $G = G_i$) and not the original input graph of the procedure $\texttt{fvs}(G, k)$. For this computation, the algorithm distinguishes between whether a vertex is put in X_1, in X_2, or not in X at all, for every vertex in V. Similar to the algorithm for Steiner Tree, it uses the colors $\mathbf{1}_1$, $\mathbf{1}_2$, and $\mathbf{0}$, respectively, to represent these possibilities. We note that we will also use the colors $\mathbf{1}_1^{\mathbf{M}}$ and $\mathbf{1}_1^{\mathbf{F}}$ to additionally represent that a vertex is in M, or in $X_1 \setminus M$, respectively.

The procedure considers all $3^{|S|}$ colorings s of S with the colors $\mathbf{0}$, $\mathbf{1}_1$, and $\mathbf{1}_2$ at Line 2. For each such coloring s, it computes the number of $((X, M), (X_1, X_2)) \in \mathcal{C}_W^{n-k,B}$ that are consistent with p and such that $s^{-1}(\{\mathbf{0}\}) \cap X = \varnothing$, $s^{-1}(\{\mathbf{1}_1\}) \subseteq X_1$, and $s^{-1}(\{\mathbf{1}_2\}) \subseteq X_2$. Note that the algorithm only considers consistent colorings s, i.e., colorings that are consistent with p and that do not contain vertices u, v with $uv \in E$ with $s(u) = \mathbf{1}_1$ and $s(v) = \mathbf{1}_2$, as these do represent the consistent cuts (X_1, X_2). It computes the required numbers by using the fact that $G[V \setminus S]$ is a forest: it applies dynamic programming on each of the trees in this forest.

Given a tree T_i in $G[V \setminus S]$ with root r_i, let T_i^x be the subtree of T_i rooted at x, and let V_i^x be the vertices of T_i^x. By dynamic programming, the algorithm computes the values $A_x(a, b, c, w, \alpha)$ for each $x \in T_i$. Here, $A_x(a, b, c, w, \alpha)$ is the number of ways to divide the vertices of V_i^x over the sets M, $X_1 \setminus M$, X_2, and $V \setminus X$ in a way consistent with p and the coloring s of S such that exactly a vertices are assigned to X, there are exactly b edges in total between the vertices in X and between the vertices in X and the vertices in S with color $\mathbf{1}_1$ or $\mathbf{1}_2$, there are exactly c vertices in M, the total weight of these assignments is w (using weight function ω), and the vertex x corresponds to the color $\alpha \in \{\mathbf{0}, \mathbf{1}_1^{\mathbf{F}}, \mathbf{1}_1^{\mathbf{M}}, \mathbf{1}_2\}$. For a vertex v in a tree T_i we say that a coloring α of this vertex is *consistent*, if it is consistent with p, and there does not exist an edge vu with $u \in S$ such that either, $\alpha \in \{\mathbf{1}_1^{\mathbf{F}}, \mathbf{1}_1^{\mathbf{M}}\}$ and $s(u) = \mathbf{1}_2$, or $\alpha = \mathbf{1}_2$ and $s(u) = \mathbf{1}_1$.

For a leaf x of T_i, one can see that $A_x(a, b, c, w, \alpha)$ can be computed using the following formulas, where the given colorings must be interpreted as colorings of the vertex x.

We have that $A_x(a, b, c, w, \mathbf{0}) = [\sigma \text{ is consistent}][a = b = c = w = 0]$, and

$A_x(a,b,c,w,\sigma) = [\sigma \text{ is consistent}][a=1]\cdot$

$$\begin{cases} [\mathrm{b} = |\{\ \mathrm{xv} \in E| : v \in S, s(v) = \mathbf{1}_1\}|][c=0][w=\omega(x\times\{\mathbf{F}\})] & \text{if } \sigma = \mathbf{1}_1^{\mathbf{F}}, \text{ (7.1a)} \\ [\mathrm{b} = |\{\ \mathrm{xv} \in E| : v \in S, s(v) = \mathbf{1}_1\}|][c=1][w=\omega(x\times\{\mathbf{M}\})] & \text{if } \sigma = \mathbf{1}_1^{\mathbf{M}}, \text{(7.1b)} \\ [\mathrm{b} = |\{\ \mathrm{xv} \in E| : v \in S, s(v) = \mathbf{1}_2\}|][c=0][w=\omega(x\times\{\mathbf{F}\})] & \text{if } \sigma = \mathbf{1}_2. \text{ (7.1c)} \end{cases}$$

For internal nodes x of T_i with children $y_1,\ldots,y_l$, we compute the values $A_x^j(a,b,c,w,\alpha)$ for $0 \le j \le l$, where $A_x^j(a,b,c,w,\alpha)$ has the same meaning as $A_x(a,b,c,w,\alpha)$ but considers partitions of the vertices of T_i^x restricted to the subtree of T_i^x in which x only has its first j children. In this way, we obtain the values $A_x(a,b,c,w,\alpha)$ in $l+1$ steps as $A_x(a,b,c,w,\alpha) = A_x^l(a,b,c,w,\alpha)$. Notice that the values $A_x^0(a,b,c,w,\alpha)$ can be computed by using the formula for a leaf, as x becomes a leaf after removing all its children. For $j \ge 1$, one can see that $A_x^j(a,b,c,w,\alpha) =$

$$\sum_{\beta\in\{\mathbf{0},\mathbf{1}_1^{\mathbf{F}},\mathbf{1}_1^{\mathbf{M}},\mathbf{1}_2\}} [\beta \text{ is consistent}] \sum_{\substack{a_x+a_y=a \\ b_x+b_y+[\mathbf{0}\notin\{\alpha,\beta\}]=b \\ c_x+c_y=c \\ w_x+w_y=w}} A_x^{j-1}(a_x,b_x,c_x,w_x,\alpha)\cdot A_{y_j}(a_y,b_y,c_y,w_y,\beta). \quad (7.2)$$

This formula considers all possible consistent colorings of the j-th child of x and combines this with the already computed values while updating the accumulators a, b, c, and w. Notice that the equation adds one to the accumulator b if this edge is in $G[X]$, i.e., if $\alpha,\beta \in \{\mathbf{1}_1^{\mathbf{F}},\mathbf{1}_1^{\mathbf{M}}\}$ or $\alpha=\beta=\mathbf{1}_2$.

Next, the algorithm combines the values computed for each of the trees T_i. This results in the values $A_s(a,b,c,w)$ that represent the total number of ways to divide the vertices of $\cup_{i=1}^t V_i^{r_i}$ over the sets M, $X_1\setminus M$, X_2, and $V\setminus X$ in a way consistent with p and the coloring s of S such that exactly a vertices are assigned to X, there are exactly b edges in total between the vertices in X and between the vertices in X and the vertices in S with color $\mathbf{1}_1$ or $\mathbf{1}_2$, there are exactly c vertices in M, and the total weight of these assignments is w (using weight function ω). It does so in a way similar to how it combined the values for multiple children of an internal node of a tree: it computes the values $A_s^j(a,b,c,w)$ that represent the same values only restricted to the vertices in $\cup_{i=1}^j V_i^{r_i}$.

This results in the following equation for $A_s^1(a,b,c,w)$ that simply sums over all colorings of r_1:

$$A_s^1(a,b,c,w) = \sum_{\alpha\in\{\mathbf{0},\mathbf{1}_1^{\mathbf{F}},\mathbf{1}_1^{\mathbf{M}},\mathbf{1}_2\}} A_{r_1}(a,b,c,w,\alpha). \quad (7.3)$$

Also, computing the values $A_s^i(a,b,c,w)$ from the values $A_s^{i-1}(a,b,c,w)$ and the values $A_{r_i}(a_i,b_i,c_i,w_i,\alpha)$ can be done using the following equation that is similar and sums over all values of the accumulators:

$$A_s^i(a,b,c,w) = \sum_{\substack{a_i+a_{i-1}=a \\ b_i+b_{i-1}=b \\ c_i+c_{i-1}=c \\ w_i+w_{i-1}=w}} A_s^{i-1}(a_{i-1},b_{i-1},c_{i-1},w_{i-1}) \sum_{\alpha\in\{\mathbf{0},\mathbf{1}_1^{\mathbf{F}},\mathbf{1}_1^{\mathbf{M}},\mathbf{1}_2\}} A_{r_i}(a_i,b_i,c_i,w_i,\alpha). \quad (7.4)$$

Finally, we let $A_s(a,b,c,w)$ be the following values in which the accumulators a and b also account for vertices in S that are in X, and edges between vertices in S that are in $G[X]$. We note that the effect of the vertices in S are not taken into account in the accumulators c and w: this is done later at Line 7 of the procedure. It holds that $A_s(a,b,c,w) =$

$$A_s^t(a-|\{v \in S : s(v) \in \{\mathbf{1}_1^{\mathbf{F}}, \mathbf{1}_1^{\mathbf{M}}, \mathbf{1}_2\}\}|, b-|\{uv \in E : u,v \in \{\mathbf{1}_1^{\mathbf{F}}, \mathbf{1}_1^{\mathbf{M}}, \mathbf{1}_2\}\}|, c, w, \alpha). \quad (7.5)$$

As a last step of the main loop, the algorithm computes the total contribution of the values computed for the current choice of s at Line 7 and adds this to the intermediate sum r. Here, q_w^c stands for the number of ways to pick c markers to be in M from the vertices in S (more specifically, the vertices in $s^{-1}(\{\mathbf{1}_1\})$) such that the total weight of the vertices in S is w. Notice that this weight depends on the choice of the markers. The algorithm sums over all possible number of markers c that can be chosen and all possible weight sums w. As the procedure $\texttt{count}(G,S,p,k,\omega,W,B)$ computes the number of $((X,M),(X_1,X_2)) \in \mathcal{C}_W^{n-k,B}$ that are consistent with p, the values of the accumulators can now be fixed: the number of vertices in X should be $n-k$, the number of edges in $G[X]$ should be B, the number of markers should be $n-k-B$ (where we subtract an additional c as we choose the many markers in S), and the total weight should be W (where subtract w also to account for the vertices in S).

We note that the values q_w^c can be computed in the following way. Let $S = \{v_1,\ldots,v_{|S|}\}$, and let $Q(c,w,i)$ be the number of ways to pick c markers from the vertices in $\{v_j : 1 \le j \le i \wedge s(v_j) = \mathbf{1}_1\}$ such that the total weight of the vertices in $\{v_1,\ldots,v_i\}$ equals w. The values $Q(c,w,i)$ can be computed by the following recurrence from which the require values follow as $q_w^c = Q(c,w,|S|)$: $Q(c,w,0) = 0$, and for $i > 0, Q(c,w,i) =$

$$\begin{cases} Q(c,w,i-1) & \text{if } s(v_i) = \mathbf{0}, \\ Q(c-1,w-\omega(v_i \times \{\mathbf{M}\}),i-1) + Q(c,w-\omega(v_i \times \{\mathbf{F}\}),i-1) & \text{if } s(v_i) = \mathbf{1}_1, \\ Q(c,w-\omega(v_i \times \{\mathbf{F}\}),i-1) & \text{if } s(v_i) = \mathbf{1}_2. \end{cases} \quad (7.6)$$

Note that the case where $s(v_i) = \mathbf{1}_1$ here represents the choice of picking the marker or not, and that the weights are updated in the two cases that correspond to that $v_i \in X$.

We conclude by considering the running time and space requirements of the procedure $\texttt{count}(G,S,p,k,\omega,W,B)$. This is $\mathcal{O}^\star(3^{|S|})$ time and polynomial space because $3^{|S|}$ coloring s of S are considered and all other operations can be performed in polynomial time and space as all accumulators used in the computations can only have $|V|^{\mathcal{O}(1)}$ different values. ■

As a side remark, let us briefly note that without using the isolation lemma at the last step, the provided algorithm can be easily adjusted to count the number of connected vertex covers MODULO 2. In Chapter 8 we will connect this to the Strong Exponential Time Hypothesis.

7.2.2 Connected Vertex Cover Parameterized by Solution Size

Connected Vertex Cover	**Parameter:** k
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 every edge is incident with at least one vertex from X?	

We will denote $|V| = n$. In this section we will prove the following.

Theorem 7.9 There exists a Monte Carlo algorithm with constant one-sided error probability that solves the Connected Vertex Cover problem in $\mathcal{O}^\star(2^k)$ time and polynomial space.

It is worth mentioning that lower bounds for (the parity version of) this problem will be studied in Chapter 8. Similarly to Subsection 7.2.1, our algorithm uses a combination of iterative compression and the Cut&Count technique. As the universe for Algorithm 7 we take the vertex set $U = V$. Recall that we generate a random weight function $\omega : U \to \{1, 2, \ldots, N\}$, taking $N = 2|U| = 2|V|$.

The Cut part. For an integer W we define:

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

The Count part. Similarly as in the case of Steiner Tree we note that by Lemma 7.1 for each $X \in \mathcal{R}_W$ there exist $2^{\mathsf{cc}(G[X])-1}$ consistent cuts of $G[X]$, thus for any W we have $|\mathcal{S}_W| \equiv |\mathcal{C}_W|$.

It remains to show how to count $|\mathcal{C}_W|$ modulo 2 for every W in $\mathcal{O}^\star(2^k)$ time and $\mathcal{O}^\star(1)$ space. For this we will combine the Cut&Count approach with iterative compression in a way very similar to Subsection 7.2.1. We will often be less implicit in the description and refer to Subsection 7.2.1. It may be surprising that we obtain a $\mathcal{O}^\star(2^k)$ time algorithm while we also know that Connected Vertex Cover cannot be solved in $\mathcal{O}^\star(3^{\mathbf{pw}(G)})$ unless the Strong Exponential Time Hypothesis fails. The reason is that using iterative compression, we obtain a special type of path decomposition with only one bag of size $\omega(1)$, and we can make sure that this bag always induces a connected graph. Informally stated, the latter allows us to reduce the number of states to be considered in this bag.

Function `cvc`$(G=(V,E),k)$

Choose a spanning tree T arbitrarily; denote its edges by edges $e_1\dots,e_{n-1}$.
$G_i \leftarrow$ the graph obtained by contracting $e_{n-1},\dots,e_i$.
$w_i \leftarrow$ the result of contracting $e_i=(u_i,v_i)$ at step i.
$S_0 \leftarrow \varnothing$
for every $1 \le i \le |V|$ **do** — invariant: S_0 is a cvc of $G[V_0]$
  $V_i \leftarrow V(G_i); S_i \leftarrow S_{i-1} \cup \{u_i,v_i\} \setminus \{w_i\}$
  `found` $\leftarrow$ `false`; `repeat` $\leftarrow 0$; $p \leftarrow \varnothing$
  while `repeat` $< |V| \wedge$ `found` $=$ `false` **do** — boost error probability
    `repeat` $\leftarrow$ `repeat` $+1$
    for every $v \in V$ **do**
      Choose $\omega(v) \in \{1,\dots,2|V|\}$ at random
    if $\exists W,B$:`count`$(G_i,S_i,p,k,\omega,W,B) \equiv 1$ **then**
      `found` $\leftarrow$ `true`
      for every $0 \le j \le |V_i|$ **do** — write solution to p using self-reduction
        $c_{\mathbf{X}} \leftarrow$ `count`$(G_i,S_i,p[v_j \to \mathbf{X}],k,\omega,W)$
        $c_{\mathbf{O}} \leftarrow$ `count`$(G_i,S_i,p[v_j \to \mathbf{O}],k,\omega,W)$
        Let $\alpha \in \{\mathbf{X},\mathbf{O}\}$ s.t. $c_\alpha \equiv 1$
        $p \leftarrow p[v_j \to \alpha]$
  if `found` $=$ `false` **then return no**
  $S_i \leftarrow \{v \in V_i : p(v) = \mathbf{O}\}$
return $S_{|V|}$

Function `count`$(G=(V,E),S,p,k,\omega,W)$

$r \leftarrow 0$
for every $s \in \{\mathbf{0},\mathbf{1}_1,\mathbf{1}_2\}^S$ *consistent** with p **do**
  $X'_1 \leftarrow s(\{\mathbf{1}_1\}); X'_2 \leftarrow s(\{\mathbf{1}_2\}); X' = X'_1 \cup X'_2$.
  $s \leftarrow \left|\left\{(Y,(Y_1,Y_2)) : Y_1 \uplus Y_2 = Y \subseteq V \setminus S \wedge \left(X' \cup Y, (X'_1 \cup Y_1, X'_2 \cup Y_2)\right) \in \mathcal{C}_W\right\}\right|$
  $r \leftarrow r + s$.
return r

* $s \in \{\mathbf{0},\mathbf{1}_1,\mathbf{1}_2\}^S$ is *consistent* with p if:

$\forall e \in E \cap S \times S : s^{-1}(\{\mathbf{1}_1,\mathbf{1}_2\}) \cap e \neq \varnothing$ — s can be extended to vertex cover

$p^{-1}(\{\mathbf{X}\}) \subseteq s^{-1}(\{\mathbf{1}_1,\mathbf{1}_2\})$

$p^{-1}(\{\mathbf{O}\}) \subseteq s^{-1}(\{\mathbf{0}\})$

$(s^{-1}(\{\mathbf{1}_1\}) \times s^{-1}(\{\mathbf{1}_2\})) \cap E = \varnothing$ — the cut is consistent

Algorithm 9 – Connected Vertex Cover

Proof (of Theorem 8.13) We will show that the procedure $\texttt{cvc}(G, k)$ from Algorithm 9 requires at most $\mathcal{O}^\star(2^k)$ time and polynomial space and either returns a connected vertex cover in the graph G of size at most k or returns `NO`. We also show that if the algorithm returns `NO`, then the probability that there exists a connected vertex cover of size at most k in G is at most $1/2$.

As mentioned before, the proof will be very similar to the one of Theorem 7.4, so we will skip arguments that evidently carry over. The proof has two parts: we first consider the procedure $\texttt{cvc}(G, k)$ and then consider the procedure $\texttt{count}(G, S, p, k, \omega, W)$. The second procedure is given a graph G, a connected vertex cover S in G, a set of requirements on the vertices V represented by p, the parameter k, a weight function $\omega : V \to \{1, \ldots, 2|V|\}$, and a target weight W, and it computes the number of pairs $(X, (X_1, X_2)) \in \mathcal{C}_W$ that are consistent with the requirements on the vertices V given by p, that is, the number of pairs $(X, (X_1, X_2)) \in \mathcal{C}_W$ such that: the vertices $v \in V$ with $p(v) = \mathbf{X}$ are in the set X and the vertices $v \in V$ with $p(v) = \mathbf{O}$ not in X.

Part 1. Consider the procedure $\texttt{cvc}(G, k)$. The only difference with $\texttt{fvs}(G, k)$ from Subsection 7.2.1 is that the graphs $G_1, \ldots, G_n$ are obtained by *edge contraction* (that is, replace two adjacent vertices with one vertex and let the neighborhood of the new vertex be the union of the two old neighborhoods) rather than *vertex deletion*. Note that like in Section 7.2.1, a solution of the bigger graph can still be obtained from a solutions of the smaller graph: G_i is obtained from G_{i+1} by contracting an edge $e_i = (u_i, v_i)$ resulting into a vertex w_i, and if S_i is a connected vertex cover of G_i, than $S_i \cup \{u_i, v_i\} \setminus w_i$ is a connected vertex cover of G_{i+1}.

Part 2. Now, consider the procedure $\texttt{count}(G, S, p, k, \omega, W)$. Let us start with analysing its running time. Because of the discussion above, we know that S is a connected vertex cover of G, and that $|S| \leq k + 1$. On Line 2, we iterate over all consistent colorings. First observe that since $G[S]$ is connected, the number of consistent colorings is at most $2^{|S|+1}$ and they can be enumerated with polynomial. To see this, note that once the state of a vertex is determined, all its neighbors have only two states: if a vertex has state $\mathbf{0}\}$, all its neighbors must have state either $\mathbf{1}_1$ or $\mathbf{1}_2\}$, if a vertex has state $\{\mathbf{1}_1\}$ all its neighbors must have state either $\mathbf{0}$ or $\mathbf{1}_2\}$, and if a vertex has state $\{\mathbf{1}_2\}$ all its neighbors must have state either $\mathbf{0}$ or $\mathbf{1}_1\}$. Hence for the running time it suffices to show that Line 4 can be performed in polynomial time. This can be done using standard dynamic programming, since $V \setminus S$ is an independent set (because S is a connected vertex cover) and hence it can chosen independently whether it's elements are in X_1 and X_2. Then, Line 4 can be implemented in polynomial time using standard dynamic programming as in Section 2.1 with accumulators for the total weight and size that have to sum up to W and k.

To prove the theorem, it suffices to show that $\texttt{count}(G, S, p, k, \omega, W)$ computes

the number of pairs $(X, (X_1, X_2)) \in \mathcal{C}_W$ that are consistent with the requirements on the vertices V given by p, that is, the number of $(X, (X_1, X_2)) \in \mathcal{C}_W$ such that: the vertices $v \in V$ with $p(v) = \mathbf{X}$ are in the set X and the vertices $v \in V$ with $p(v) = \mathbf{O}$ not in X (note that we slightly abuse notation here since we refer $\mathcal{C}_W$ as being defined by the G given as argument). It is easy to see that for every object $(X, (X_1, X_2)) \in \mathcal{C}_W$, there is exactly one coloring s where it is accounted for so the Theorem follows. ■

7.3 Lower Bounds

In this section we briefly describe a collection of negative results concerning the possible time complexities for algorithms for connectivity problems parameterized by treewidth or pathwidth. In order to prevent that this thesis looses focus, we will only describe one lower bound of each type, and refer to [CNP+11b] for the other ones mentioned in Table 7.1. Our goal is to complement our positive results by showing that in some situations the known algorithms (including ours) probably cannot be further improved.

The complexity theoretic assumptions used are the *Exponential Time Hypothesis* (ETH) and the *Strong Exponential Time Hypothesis* (SETH) (see Chapter 2).

The lower bounds presented below are of two different types. In Section 7.3.1 we discuss several problems that, assuming the Exponential Time Hypothesis (refer to Chapter 2), do not admit an algorithm running in $\mathcal{O}^\star(2^{o(p \log p)})$ time, where p denotes the pathwidth of the input graph. In Section 7.3.2 we state that, assuming the Strong Exponential Time Hypothesis (refer to Chapter 2), the base of the exponent in some of our algorithms cannot be improved further.

7.3.1 Lower bounds assuming ETH: Maximally Disconnected Dominating Set

We have shown that a lot of well-known algorithms running in $\mathcal{O}^\star(2^{\mathcal{O}(\mathbf{tw}(G))})$ 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 7.1.2 we can solve similarly the following artificial generalizations: given a graph G and an integer r, what is the minimum size of a vertex cover (dominating set, feedback vertex set, odd cycle transversal) that induces **at most** r connected components?

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

An overview of all our results can be found in Table 7.1, and all proofs and

problem definitions can be bound in [CNP+11b]. Here we will only prove the following:

Theorem 7.10 Assuming ETH, there is no $\mathcal{O}^\star(2^{o(p \log p)})$ time algorithm for Cycle Packing, Max Cycle Cover (both in the directed and the undirected setting) or for Maximally Disconnected Dominating Set. The parameter p denotes the width of a given path decomposition of the input graph.

The proof goes along the framework introduced by Lokshtanov et al. [LMS11b]. We start our reduction from the $k \times k$ Hitting Set problem. By $[k]$ we denote $\{1, 2, \ldots, k\}$. In the set $[k] \times [k]$ *a row* is a set $\{i\} \times [k]$ and *a column* is a set $[k] \times \{i\}$ (for some $i \in [k]$).

$k \times k$ Hitting Set **Parameter:** k
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 $S \cap S_i \neq \varnothing$ for any $1 \leq i \leq m$?

Theorem 7.11 — (LMS11b), Theorem 2.4 Assuming ETH, there is no $\mathcal{O}^\star(2^{o(k \log k)})$ time algorithm for $k \times k$ Hitting Set.

In this subsection we provide a reduction from $k \times k$ Hitting Set to Maximally Disconnected Dominating Set. We are given an instance $(k, S_1, \ldots, S_m)$ of $k \times k$ Hitting Set, called the initial instance, and we are to construct an equivalent instance (G, ℓ, r) of Maximally Disconnected Dominating Set.

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

Gadgets

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

Definition 7.4 By adding a *force gadget* for vertex $v \in V$ we mean the following construction: we introduce $\ell + 1$ new vertices of degree one, connected to v.

Lemma 7.7 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 7.5 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 $|X|$, connected to all vertices in X.

Lemma 7.8 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 $X \cap D = \varnothing$, 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 7.9 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 decomposition is a indeed a 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 $p+1$. ■

Construction

Let $S_i^{\texttt{row}} = \{i\} \times [k]$ be a set containing all elements in the i-th row in the set $[k] \times [k]$. We denote $\mathcal{S} = \{S_s : 1 \leq s \leq m\} \cup \{S_i^{\texttt{row}} : 1 \leq i \leq k\}$. 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 7.10 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 new decomposition of H consists of all bags V_A for $A \in \mathcal{S}$ in an arbitrary order. Note that the above decomposition is indeed a path decomposition of H of width at most $3k-1$ (as $|A| \leq k$ for each $A \in \mathcal{S}$) and it satisfies conditions for Lemma 7.9. ■

From Hitting Set to Dominating Set

Lemma 7.11 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, $D \cap X^A = \{x_{i_A,j_A}^A\}$ for each $A \in \mathcal{S}$, thus D dominates all vertices added in one-in-many gadgets attached to sets X^A.

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

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

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

From Dominating Set to Hitting Set

Lemma 7.12 If there exists a dominating set D in the graph G, such that $|D| \leq \ell$ and D induces at least r connected components, then the initial $k \times k$ Hitting Set instance was a YES-instance.

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

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

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

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

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

7.3.2 Lower Bounds Assuming SETH: Connected Dominating Set

Following the framework introduced by Lokshtanov et al. [LMS11a], we proved in [CNP+11b] that an improvement in the base of the exponent in a number of our algorithms would contradict SETH. Formally, we prove the following type of a theorem for problems marked in the third column of Table 7.1.

Theorem 7.12 — (CNP+11b) Unless the Strong Exponential Time Hypothesis is false, there do not exist a constant $\varepsilon > 0$ and an algorithm that given an instance $(G = (V, E), k)$ together with a path decomposition of the graph G of width p solves Connected Vertex Cover in $\mathcal{O}^\star((3 - \varepsilon)^p)$ time.

Note that Vertex Cover (without a connectivity requirement) admits a $\mathcal{O}^\star(2^{\mathbf{tw}(G)})$ algorithm whereas Dominating Set, Feedback Vertex Set and Odd Cycle Transversal admit $\mathcal{O}^\star(3^{\mathbf{tw}(G)})$ algorithms and those algorithms are optimal (assuming SETH) [LMS11a]. 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 results show that this is not an artifact of the Cut&Count technique, but rather an intrinsic characteristic of these problems. An overview of all our results can be found in Table 7.1

Theorem 7.13 Assuming SETH, there cannot exist a constant $\varepsilon > 0$ and an algorithm that, given an instance $(G = (V, E), k)$ together with a path decomposition of the graph G of width p, solves the Connected Dominating Set problem in $\mathcal{O}^\star((4-\varepsilon)^p)$ 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 $1 \le t \le n'$ we create a path $\mathcal{P}_t$ consisting of $4a$ vertices $v^\alpha_{t,k}$ and $h^\alpha_{t,k}$, $0 \le k < a$ and $1 \le \alpha \le 2$. On the path $\mathcal{P}_t$ the vertices are arranged in the following order:

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

Let $\mathcal{V}$ and $\mathcal{H}$ be the sets of all vertices $v^\alpha_{t,k}$ and $h^\alpha_{t,k}$ ($1 \le t \le n'$, $0 \le k < a$, $1 \le \alpha \le 2$), 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 $1 \le t \le n'$ and $0 \le k < a$ 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 neighbors 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 $0 \le j < n+1$. 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

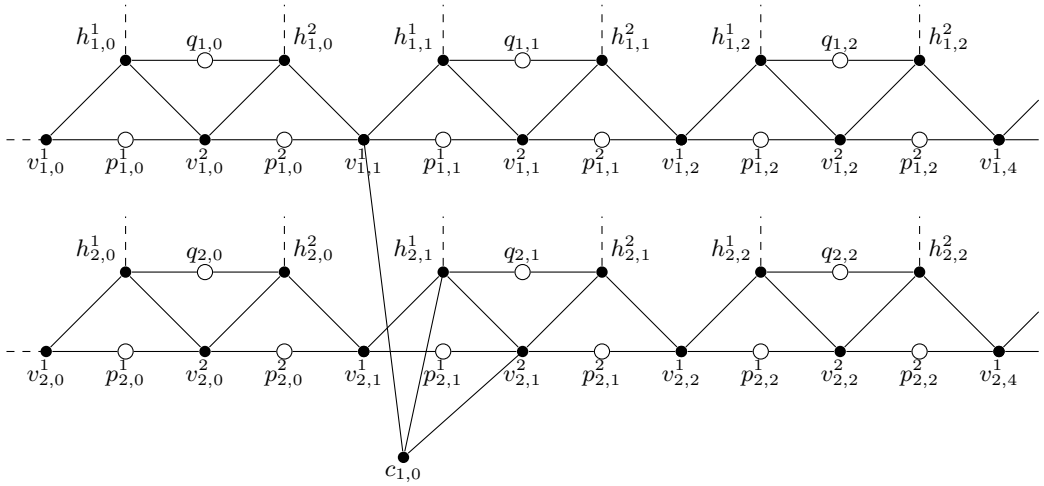

Figure 7.2 – Part of the construction for Connected Dominating Set. Dashed edges are connecting a vertex with the root r. Empty circles represent guard vertices.

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 7.13 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.

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 7.14 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 neighborhoods, 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 assigment $\phi = \phi_{mj}$ satisfies Φ. Consider a clause C_i and focus on the vertex $c_{i,j}$. It is not contained in X, thus one of its neighbor is contained in X. As this neighbor 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 complete. ■

Pathwidth Bound

In order to establish the bound on pathwidth of the obtained graph, let us recall the notion of mixed search game. The technique was used in a similar manner already in [LMS11a].

Definition 7.6 — (TUK95, LMS11a) In a *mixed search game*, a graph H 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 $\mathrm{ms}(H)$, is the minimum number of searchers required for a winning strategy of mixed searching on H.

Proposition 7.2 — (TUK95) For a graph G, $\mathrm{pw}(H) \leq \mathrm{ms}(H) \leq \mathrm{pw}(H) + 1$.

Moreover, in our case the presented cleaning strategy easily yields a polynomial time algorithm that constructs a path decomposition of G of width not greater than the number of searchers used.

Lemma 7.15 The 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 until 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 (Proof of Theorem 7.13) 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 $\mathcal{O}^\star((4-\varepsilon)^p)$ time algorithm. Lemma's 7.13, 7.14 ensure the correctness, whereas Lemma 7.15 implies that the running time of our algorithm is $\mathcal{O}^\star((4-\varepsilon)^{n/2})$, however we have $(4-\varepsilon)^{n/2} = (\sqrt{4-\varepsilon})^n$ and $\sqrt{4-\varepsilon} < 2$. This concludes the proof. ■

7.4 Concluding Remarks

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

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

To clarify, note that a standard dynamic programming for determining whether $\mathcal{F}$ is empty or not runs in $\mathcal{O}^\star(\theta^{O(1)})$ time, and usually even in $\mathcal{O}^\star(\theta)$ time. In fact, the dominant term $\theta^{O(1)}$ in the claimed running time is a rather cruel upper bound for the Cut&Count technique as well, as for many problems we can do a lot better. If $\theta = c^t|V|^{O(1)}$ with t being the width of $\mathbb{T}$, then for many instances of the above we have shown solutions running in $\theta = (c + c')^t|V|^{O(1)}$ time, where c' is, intuitively, the number of states affected by the cuts of the Cut&Count technique. See also [Pil11] for a further exploration.

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 neighborhood of each vertex), standard dynamic programming techniques give $\mathcal{O}^\star(c^{\mathbf{tw}})$ time algorithms for a constant c. The main consequence of the Cut&Count technique as presented in this work is that problems which can be formulated as a local constraint with an additional upper bound on the number of connected components also admit $\mathcal{O}^\star(c^{\mathbf{tw}})$ 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 $\mathcal{O}^\star(2^{o(p \log p)})$ 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.

Typically, our results give rise to more new questions than it solves old ones:

Open Question 7 Can Cut&Count be derandomized? For example, can Connected Vertex Cover be solved deterministically in $\mathcal{O}^\star(c^t)$ on graphs of treewidth t for some constant c?

Open Question 8 Since general derandomization seems hard, we ask whether it is possible to derandomize the presented FPT algorithms parameterized by the solution size for Feedback Vertex Set, Connected Vertex Cover or Connected Feedback Vertex Set. Note that the tree decomposition considered in these algorithms is of a very specific type, which could potentially make this problem easier than the previous one.

Open Question 9 Do there exist algorithms running in time $\mathcal{O}^\star(c^{\mathbf{tw}(G)})$ that solve counting or weighted variants? For example can the number of Hamiltonian paths be determined, or the Traveling Salesman Problem solved in $\mathcal{O}^\star(c^{\mathbf{tw}(G)})$?

Open Question 10 Can exact exponential time algorithms be improved using Cut&Count (for example for Connected Dominating Set, Steiner Tree and Feedback Vertex Set)?

Open Question 11 All our algorithms for directed graphs run in time $\mathcal{O}^\star(6^t)$. Can the constant 6 be improved? Or could it be that it is optimal (again, assuming SETH)?

Chapter 8

On Problems as Hard as CNF-Sat

This chapter is based on the following paper:

[CDL+11]. Marek Cygan, Holger Dell, Daniel Lokshtahnov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-Sat. 2011. Manuscript

What the field of Exact Algorithms sorely lacks is a complexity-theoretic framework for showing running time lower bounds. Some problems, such as Independent Set and Dominating Set have seen a chain of improvements [FGK09, NvR10, vRNvD09, Rob86, KLR09a], each new improvement being smaller than the previous. For these problems the running time of the algorithms seems to converge towards $\mathcal{O}^\star(c^n)$ for some unknown c. For other problems, such as Graph Coloring or Steiner Tree, non-trivial solutions have been found (see [BHK09] and Chapter 3), but improving these algorithms further seems to be out of reach. For other problems yet, such as CNF-Sat or Hitting Set, no algorithms faster than brute force have been discovered. Currently, there are no tools available that would allow us to explain why the different problems exhibit the behavior they do, and developing such tools is of utmost importance for the field of Exact Algorithms. Preferably, such a tool would allow us to rule out $\mathcal{O}^\star(c^n)$ time algorithms for concrete problems, and concrete constants c, under plausible complexity assumptions.

At this point it is necessary to recall that the Exponential Time Hypothesis (ETH) (see Chapter 2) is considered a plausible complexity assumption, and assuming the ETH a number of running time lower bounds have been proved for exponential time algorithms [IPZ01], parameterized algorithms [CCF+05, LMS11b] and approximation schemes [Mar07]. However, currently it does not seem possible to show lower bounds under the ETH on the form $f(n)$ for problems that do admit $f(n)^c$ time algorithms for some constant $c > 1$.

In this chapter we show a number of tight running time lower bounds for basic problems assuming the Strong Exponential Time Hypothesis (SETH) (see Chapter 2). Recall that the SETH states that k-CNF-Sat requires $\mathcal{O}^\star(2^n)$ time when k goes to infinity. While the SETH is not as well believed as the ETH, our lower bounds demonstrate that a faster satisfiability algorithm is a barrier to improving the currently best known algorithms for several well-known problems, or equivalently, instead of trying to find an improved SAT algorithms, we can also focus on improving other well-known algorithms. Our most interesting result[(1)] is a connection between the possible optimality of several well-known dynamic programming based algorithms and brute-force algorithms. As we have already seen in Chapters 2 and 7 such a type of connection has been made already in [LMS11b, LMS11a] concerning algorithms on graphs of bounded treewidth, but the challenge of improving these algorithms seems very different from improving the algorithms studied in this chapter.

Our starting point is our result proved in Section 8.1 that the following statements are equivalent (confer Section 8.1 for the problem definitions):

1. $\exists \delta < 1$ such that k-CNF-Sat is solvable in $\mathcal{O}^\star(2^{\delta n})$ time for all k,
2. $\exists \delta < 1$ such that Hitting Set for set systems with sets of size at most k is solvable in $\mathcal{O}^\star(2^{\delta n})$ time,
3. $\exists \delta < 1$ such that Set Splitting for set systems with sets of size at most k is solvable in $\mathcal{O}^\star(2^{\delta n})$ time,
4. $\exists \delta < 1$ such that k-NAE-Sat is solvable in $\mathcal{O}^\star(2^{\delta n})$ time for all k,
5. $\exists \delta < 1$ such that satisfiability of cn size series-parallel circuits is solvable in $\mathcal{O}^\star(2^{\delta n})$ time for all c.

This immediately implies that a $2^{\delta n}$ time algorithm for any of the above problems without the restrictions on clause width or set size would violate SETH. All of the problems above have $\mathcal{O}^\star(2^n)$ time brute force algorithms, and hence our bounds are tight. The equivalences $1-4$ are obtained by a number of novel non-trivial gadgets, while the equivalence with 5 is obtained by combining a number of known results.

Then in Section 8.2, we show the aforementioned connection of the possible optimality of several well-known dynamic programming algorithms and brute-force algorithms. More specifically, any of the following would imply that SETH fails (confer Section 8.2 for the problem definitions):

- $\exists \delta < 1$ such that $\oplus$Set Cover is solvable in $\mathcal{O}^\star(2^{\delta n})$ time,
- $\exists \delta < 1$ such that $\oplus$Steiner Tree is solvable in $\mathcal{O}^\star(2^{\delta k})$ time ,

[(1)] at least, in the context of this thesis

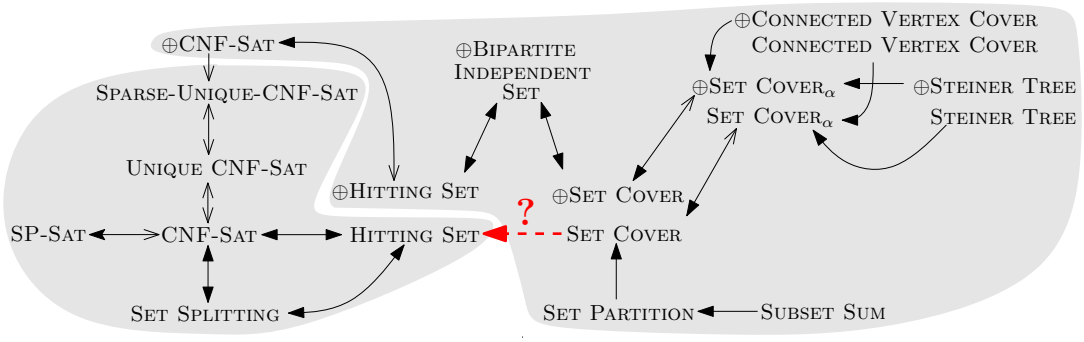

Figure 8.1 – Overview of all reductions. An arrow denotes that improving the best-known algorithm for the problem mentioned at the source implies that the best-known algorithm at the sink can be improved as well. Fat arrowheads indicate our results; other arrowheads were known before or trivial. The left grey marked area contains all results discussed in Section 8.1 while the right area contains all results discussed in Section 8.2. The red arrow indicates an open problem.

- $\exists \delta < 1$ such that ⊕Connected Vertex Cover is solvable in $\mathcal{O}^\star(2^{\delta k})$ time [CNP+11a] (see also Section 7.2).

In order to obtain these results we (i) note that it follows from previous results that significantly improving over brute-force for the ⊕CNF-Sat problem implies that SETH fails (ii) extend a reduction from Section 8.1 to be parity preserving (iii) use an elegant, and to our knowledge, new lemma on symmetries of bipartite graphs to show that ⊕Hitting Set, ⊕Set Cover and Bipartite Independent Set are actually the same problem.

Motivated by these results, we furthermore use the hypothesis that Set Cover cannot be solved in time $\mathcal{O}^\star(2^{\delta n})$ time for $\delta < 1$ unless SETH fails. In particular, we show that unless Set Cover can be solved faster than $\mathcal{O}^\star(2^{\delta n})$ for $\delta < 1$ the following algorithms can not be improved.

- The $\mathcal{O}^\star(2^n)$ time algorithm for Set Partition(folklore),
- The $\mathcal{O}^\star(2^k)$ time algorithm for Steiner Tree [BHKK07, DW72, Ned09] (see also Chapter 3),
- The $\mathcal{O}^\star(2^k)$ time algorithm for Connected Vertex Cover [CNP+11a] (see also Section 7.2), and
- The $\mathcal{O}^\star(Wn)$ time algorithm for Subset Sum [Bel54].

The lower bounds for Steiner Tree and Connected Vertex Cover are particularly interesting, because the currently best algorithms for these problems were obtained after a chain of improvements. The lower bound for Subset Sum

concerns an algorithm stemming from the 1950's taught in most undergraduate algorithms classes (see also Section 2.1 for more details), but interestingly further improvements have not been discussed in the literature.

The currently best known algorithms for *solving* the Steiner Tree and Connected Vertex Cover problems in fact count the number of solutions modulo two. Hence our results also indicate that improving over these algorithms requires completely different methods. On the other hand, the ⊕Set Cover is very much different from the CNF-sat problem, so our reduction could also be seen as providing another possible route to improved algorithms for the CNF-sat problem.

Preliminaries and Notation

We use the notation introduced in Section 1.3. In particular, recall the following notation: for a set U and positive integer $i \leq U$, $\binom{U}{i}$ denotes the family of all subsets of U of size i. An assignment $\alpha\colon \{v_1, \ldots, v_n\} \to \{0,1\}$ to n boolean variables $v_1, \ldots, v_n$ is identified with a set $A \subseteq \{v_1, \ldots, v_n\}$ as $A = \{v_i \mid \alpha(v_i) = 1\}$. We will define the problems studied in such a way that we are looking for an $\mathcal{O}^\star(2^{\delta n})$-time algorithm (and a $\mathcal{O}^\star(2^n)$-time algorithm is known), m is a parameter such that the input representation consists of at most m^c bits for some constant c and k is some structural parameter. Then we use the following notation:

Definition 8.1 For a problem Π with complexity parameter n and second parameter k, define $\sigma_k(\Pi)$ as

$$\inf\{\delta > 0 | \exists \text{a } \mathcal{O}^\star(2^{\delta n})\text{-time Monte Carlo algorithm for } k\text{-}\Pi.\}$$

Also, define $\sigma(\Pi) = \sigma_\infty(\Pi) = \lim_{k\to\infty} \sigma_k(k\text{-}\Pi)$.

8.1 On Improving Branching Algorithms

In this section we show that obtaining improved algorithms for combinatorial problems like Hitting Set, and Set Splitting will imply an improved algorithm for CNF-Sat. In fact, we show that these problems are equivalent, that is, obtaining an improved algorithm for any one of them would imply an improved algorithm for the others. First recall the CNF-sat problem from Chapter 2:

k-CNF-Sat **Parameter:** n
Input: A CNF formula consisting of m clauses of size at most k and on n variables.
Question: Is there a satisfying assignment?

HITTING SET **Parameter:** n
Input: An integer t and a set system $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = m$, $|U| = n$ and for every $S \in \mathcal{F}$, $|S| \leq k$
Question: Is there a subset $H \subseteq U$ with $|H| \leq t$ such that $H \cap S \neq \emptyset$ for every $S \in \mathcal{F}$?

NAE-SAT **Parameter:** n
Input: A CNF formula consisting of m clauses of size at most k and on n variables.
Question: Is there an assignment so that each clause contains literal set to true and a literal set to false?

SET SPLITTING **Parameter:** n
Input: A set system $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = m$, $|U| = n$, for every $S \in \mathcal{F}$, $|S| \leq k$ and $m \leq f(k)n$.
Question: Is there a partition $U = U_1 \uplus U_2$ so that each set $S \in \mathcal{F}$ satisfies $S \cap U_1 \neq \emptyset$ and $S \cap U_2 \neq \emptyset$?

SPARSE-UNIQUE-CNF-SAT **Parameter:** n
Input: A CNF formula consisting of m clauses of size at most k and on n variables having at most one satisfying assignment, where $m \leq f(k)n$.
Question: Is there a satisfying assignment?

UNIQUE-CNF-SAT **Parameter:** n
Input: A CNF formula of length m, consisting of clauses of size at most k and on n variables that has either a unique or no satisfying solutions.
Question: Is there a satisfying assignment?

8.1.1 Preliminary Results on CNF-Sat

The following theorem is known.

Theorem 8.1 — (CIKP03),(Tra08) For every constant positive integer k,

$$\sigma_k(\text{CNF-SAT}) \leq \sigma_k(\text{UNIQUE-CNF-SAT}) + \mathcal{O}\left(\frac{\log k}{k}\right).$$

Calabro et al. [CIKP03] proved the above theorem with an additive term of $\mathcal{O}(\log^2 k/k)$ and Traxler [Tra08] generalized it to constraint satisfaction problems with an improved bound of $O(\log k/k)$ in the case of CNF-SAT. Moreover, the following is a simple corollary of Lemma 2.1.

Theorem 8.2 For every constant positive integer k and real $\epsilon > 0$,

$$\sigma_k(\text{Unique-CNF-Sat}) \leq \sigma_k(\text{Sparse-Unique-CNF-Sat}) + \epsilon.$$

Moreover, $\sigma_k(\text{Sparse-Unique-CNF-Sat}) \leq \sigma_k(\oplus\text{CNF-Sat})$ since the number of satisfying assignments of the Sparse-Unique-CNF-Sat problem instance is odd if and only if it is satisfiable.

8.1.2 From CNF-Sat to Hitting Set

Given a CNF formula $\varphi = C_1 \wedge \ldots \wedge C_m$ over n variables $v_1, \ldots, v_n$ where the C_i are clauses of size at most k and an odd integer $p > 2$ that divides n, create the set system $\mathcal{F}_{\varphi,p} \subseteq 2^U$ as follows.

1. Let $p' = p + 2\lceil \log p \rceil$ be an odd integer, let $U = \{u_1, \ldots, u_{n'}\}$ with $n' = p' \cdot n/p$.
2. Partition the set of variables $\{v_1, \ldots, v_n\}$ of φ into blocks V_i of size p, i.e. $V_i = \{v_{pi+1}, \ldots, v_{p(i+1)}\}$.
3. Partition the set U into blocks U_i of size p', i.e. $U_i = \{u_{p'i+1}, \ldots, u_{p'(i+1)}\}$.
4. Arbitrarily choose an injective function $\psi_i : 2^{V_i} \to \binom{U_i}{\lceil p'/2 \rceil}$. 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 every $X \in \binom{U_i}{\lceil p'/2 \rceil}$ for some i, add the set X to $\mathcal{F}$.
6. For every $X \in \binom{U_i}{\lfloor p'/2 \rfloor}$ for some i such that $\psi_i^{-1}(\{U_i \setminus X\}) = \varnothing$, add the set X to $\mathcal{F}$.
7. For every clause C of φ, do the following:
   - Let $I = \{1 \leq j \leq \frac{n}{p} \mid C \text{ contains a variable of block } V_j\}$;
   - 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 $\mathcal{F}$.

Lemma 8.1 For every p, the number of satisfying assignments of φ is equal to the number of hitting sets of size $\lceil \frac{p'}{2} \rceil \frac{n}{p}$ of the set system $\mathcal{F}_{\varphi,p}$.

Proof For convenience denote $g = \frac{n}{p}$. Define $\psi: 2^V \to 2^U$ as $\psi(A) = \bigcup_{i=1}^{g} \psi_i(A \cap V_i)$. Note that ψ is injective, since for every i, ψ_i is injective. Hence to prove the lemma, it is sufficient to prove that (1) A is a satisfying assignment if and only if $\psi(A)$ is a hitting set of size $\lceil \frac{p'}{2} \rceil g$, and (2) no set $H \subseteq U$ of size $\lceil \frac{p'}{2} \rceil g$ is a hitting set of $\mathcal{F}_{\varphi,p}$ if there is no assignment $A \subseteq V$ such that $\psi(A) = H$.

For the forward direction of (1), note that the sets added in Step 5 are hit by the pigeon-hole 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, $\psi_i^{-1}(\{U_i \setminus X\}) = \varnothing$. 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). It is easy to check that $\psi(A)$ has size $\lceil \frac{p'}{2} \rceil g$ since there are g blocks.

For the reverse direction of (1), let A be an assignment such that $\psi(A)$ is a hitting set of size $\lceil \frac{p'}{2} \rceil g$. 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, $|\psi(A) \cap U_i| \geq \frac{p'}{2}$ 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 $|H| = \lceil \frac{p'}{2} \rceil g$ and there are g disjoint blocks $U_1, \ldots, U_g$. 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 $H \subseteq U$ be a set of size $\lceil \frac{p'}{2} \rceil g$ and assume that there is no assignment $A \subseteq V$ such that $\psi(A) = H$. We show that H is not a hitting set of $\mathcal{F}_{\varphi,p}$. For the sake of contradiction, suppose that H is a hitting set. Then, as in the proof of the reverse direction of (1), we obtain $|H \cap U_i| = \lceil \frac{p'}{2} \rceil$ for every i. Since it hits all sets added in Step 6, we also know that $\psi_i^{-1}(\{H \cap U_i\}) \neq \varnothing$ for every i. However, this contradicts the non-existence of $A \subseteq V$ such that $\psi(A) = H$. ∎

Theorem 8.3 For every positive integer p,

$$\sigma_k(\text{CNF-Sat}) \leq \sigma_{pk}(\text{Hitting Set}) + \mathcal{O}\left(\frac{\log p}{p}\right).$$

Proof Create the set system $\mathcal{F}_{\varphi,p}$ as described above. For a constant p, this clearly can be done in polynomial time. It is easy to see that the maximum size of a set of $\mathcal{F}_{\varphi,p}$ is at most pk, and thus we can determine the minimum hitting set of $\mathcal{F}_{\varphi,p}$ in $\mathcal{O}^\star(2^{\sigma_{pk}(\text{Hitting Set})n'})$ time, where n' is the size of the universe of

$\mathcal{F}_{\varphi,p}$. By Lemma 8.1, φ is satisfiable if and only if this quantity is $\lceil \frac{p'}{2} \rceil \frac{n}{p}$. Since $n' = \lceil \frac{n}{p} \rceil (p + 2\lceil \log p \rceil) = n(1 + \mathcal{O}(\frac{\log p}{p}))$ the theorem follows. ■

8.1.3 From Hitting Set to Set Splitting to CNF-Sat

Theorem 8.4 For every positive integer p:

$$\sigma_k(\text{Hitting Set}) \leq \sigma_{\max\{p,k\}+1}(\text{Set Splitting}) + \mathcal{O}\left(\frac{\log p}{p}\right).$$

Proof Let $(\mathcal{F}, t)$ be an instance of Hitting Set. We can assume that the universe U of $\mathcal{F}$ has n elements and that p divides n. Let $U = U_1 \dot\cup \ldots \dot\cup U_{n/p}$ be a partition in which each part has exactly $|U_i| = p$ elements of the universe U. Let $t_1, \ldots, t_{n/p}$ be non-negative integers such that $\sum_{i=1}^{n/p} t_i = t$. The t_i's are our current guess for how many elements of a t-element hitting set will intersect with the U_i's. The number of ways to write t as the ordered sum of n/p non-negative integers is exactly $\binom{t+n/p-1}{n/p-1}$, which can be bounded by $2^{\mathcal{O}(\log p/p)\cdot n}$ since $t \leq n$. For each choice of the t_i's, we construct an instance $\mathcal{F}'$ of Set Splitting as follows.

1. Let R (red) and B (blue) be two special elements and add the set $\{R, B\}$ to $\mathcal{F}'$.
2. For all i with $t_i < p$ and for all $X \in \binom{U_i}{t_i+1}$, add $X \cup \{R\}$ to $\mathcal{F}'$.
3. For every $Y \in \mathcal{F}$, add $Y \cup \{B\}$ to $\mathcal{F}'$.

Clearly $\mathcal{F}'$ can be computed in polynomial time and its universe has $n+2$ elements. The sets added in step 2 have size at most $p+1$ and the sets added in step 3 have size at most $k+1$. Given an algorithm for Set Splitting, we compute $\mathcal{F}'$ for every choice of the t_i's and we decide Hitting Set in time $\mathcal{O}^*(2^{\mathcal{O}(\log p/p)\cdot n} \cdot 2^{\sigma_{\max\{p,k\}+1}(\text{Set Splitting})\cdot n})$. It remains to show that $\mathcal{F}$ has a hitting set of size at most t if and only if $\mathcal{F}'$ has a set splitting for some choice of $t_1, \ldots, t_{n/p}$.

For the completeness of the reduction, let H be a hitting set of size t and set $t_i = |U_i \cap H|$ for all i. Then $H \cup \{R\}$ and its complement $(U - H) \cup \{B\}$ are a set splitting of $\mathcal{F}'$: The set $\{R, B\}$ added in step 1 is split. The sets $X \cup \{R\}$ added in step 2 are split since at least one of the $t_i + 1$ elements of $X \subseteq U_i$ is not contained in H. Finally, the sets $Y \cup \{B\}$ added in step 3 are split since each $Y \in \mathcal{F}$ has a non-empty intersection with H.

For the soundness of the reduction, let $(S, \overline{S})$ be a set splitting of $\mathcal{F}'$. Without loss of generality, assume that $R \in S$. By the set added in step 1, this means that $B \in \overline{S}$. The sets added in step 2 guarantee that $U_i \cap S$ contains at most t_i elements for all i. Finally, the sets added in step 3 make sure that each set $Y \in \mathcal{F}$ has a

non-empty intersection with S. Thus, $S - \{R\}$ is a hitting set of $\mathcal{F}$ and has size at most $\sum_i t_i = t$. ■

Observation 8.1 $\sigma_k(\text{Set Splitting}) \leq \sigma_k(\text{NAE-Sat}) \leq \sigma_k(\text{CNF-Sat})$.

Proof For the first reduction, let $\mathcal{F}$ be an instance of Set Splitting. We construct an equivalent CNF formula φ as follows: For each element in the universe of $\mathcal{F}$, we add a variable, and for each set $X \in \mathcal{F}$ we add a clause in which each variable occurs positively. A characteristic function of a set splitting $U = U_1 \dot\cup U_2$ is one that assigns 1 to the elements in U_1 and 0 to the elements of U_2. Observe that the characteristic functions of set splittings of $\mathcal{F}$ stand in one-to-one correspondence to variable assignments that satisfy the NAE-Sat constraints of φ. Thus, any algorithm for NAE-Sat works for Set Splitting, too.

For the second reduction, let φ be a NAE-Sat-formula. The standard reduction to CNF-Sat creates two copies of every clause of φ and flips the sign of all literals in the second copies. Then any NAE-Sat-assignment of φ satisfies both copies of the clauses of φ'. On the other hand, any satisfying assignment of φ' sets a literal to true and a literal to false in each clause of φ. Thus any algorithm for CNF-Sat works for NAE-Sat, too. ■

8.1.4 On Satisfiability for Series-Parallel Circuits

In this section, we show that the satisfiability of *series-parallel* circuits of size cn can be decided in time time $2^{\delta n}$ for $\delta < 1$ independent of c if and only if SETH is not true. Here the size of a circuit is the number of gates. However we will only consider circuits of fan-in 2, and if the fan-in of the circuit is bounded by 2, then the number of the wires and the number of gates are within a factor 2 of each other. Our proof is based on a result of Valiant regarding paths in sparse graphs [Val77]. Calabro [Cal08] discusses various notions of series-parallel graphs and provides a more complete proof of Valiant's lower bound on the size of series-parallel graphs (which he calls Valiant series-parallel graphs) that have "many" long paths. We remark that the class of Valiant series-parallel graphs is not the same as the notion of series-parallel graphs [dF97] used most commonly in graph theory.

In this section a *multidag* $G = (V, E)$ is a directed acyclic multigraph. Let $\text{input}(G)$ denote the set of vertices $v \in V$ such that the indegree of v in G is zero. Similarly, let $\mathbf{O}(G)$ denote the set of vertices $v \in V$ such that the outdegree of v in G is zero. A *labeling* of G is a function $l : V \to \mathbb{N}$ such that $\forall (u, v) \in E$, $l(a) < l(b)$. A labeling l is *normal* if for all $v \in \text{input}(G)$, $l(a) = 0$ and there exists an integer $d \in \mathbb{N}$ such that for all $v \in \mathbf{O}(G) \setminus \text{input}(G)$, $l(v) = d$. A multidag G is Valiant series-parallel (VSP) if it has a normal labeling l such that

$$\nexists\, (u, v), (u', v') \in E, l(u) < l(u') < l(v) < l(v')$$

We are interested in lower bounding the size of graphs that contain paths of length at least m even after removing any set of r edges. For a class $\mathcal{C}$ of multidags, we define

$$R_{\mathcal{C}}(r,m) = \{G = (V,E) \in \mathcal{C} \mid \forall E' \subseteq E, |E'| = r \ \exists \text{ a path of length } m \text{ in } (V, E - E')\}$$
$$S_{\mathcal{C}}(r,m) = \min\{|E| \mid \exists G = (V,E) \in R_{\mathcal{C}}(r,m)\}$$

Valiant proves the following lower bound on $S_{\text{VSP}}(r,m)$.

Theorem 8.5 — (Val77, Cal08) There exists an integer c_0 such that for all integers $m \geq 1, r \geq 0$, the following holds: if $S_{\text{VSP}}(r,m)$ exists, then $S_{\text{VSP}}(r,m) \geq c_0 r \lg m$.

We say that a boolean circuit C is a VSP circuit if the underlying multidag of C is a VSP graph. Using the depth-reduction result of the previous theorem, Valiant showed C can be converted to an equivalent depth-3 unbounded fan-in OR-AND-OR circuit of size less than 2^n. More precisely,

Theorem 8.6 — (Val77) Let C be a VSP circuit of size cn with n input variables. There is an algorithm A which on input C and a parameter $d \geq 1$ outputs an equivalent depth-3 unbounded fan-in OR-AND-OR circuit C' with the following properties:

1. Fan-in of the top OR gate in C' is bounded by $2^{n/d}$
2. Fan-in of the bottom OR gates is bounded by $2.2^{2^{\mu cd}}$ where μ is an absolute constant.
3. A runs in time $\mathcal{O}^\star(\text{size}(C'))$.

In other words, $\forall d \geq 1$, Theorem 8.6 reduces the satisfiability of a cn size VSP circuit to that of the satisfiability of a disjunction of $2^{n/d}$ k-CNFs where $k \leq 2.2^{2^{\mu cd}}$. If SETH does not hold, it follows that there exists $\delta < 1$ such that $\forall \varepsilon > 0$, k-SAT can be decided in time $\mathcal{O}^\star(2^{\delta+\varepsilon n})$ time. Combining these observations, we get the following theorem.

Theorem 8.7 If SETH does not hold, then $\forall \epsilon > 0$ the satisfiability of cn size VSP circuits can be decided in time $\mathcal{O}^\star(2^{(\delta+\varepsilon)n})$ for some $\delta < 1$.

For the reverse direction, observe that a CNF formula with cn clauses, all of size at most k can be written as a linear size VSP circuit. Suppose now that there exists a $\delta < 1$ such that for all c, satisfiability of VSP circuits can be done in time $\mathcal{O}(2^{\delta n})$. Then, given an a k-CNF-SAT instance ϕ we can use Lemma 2.1 to

reduce ϕ to the satisfiability of the disjunction of $2^{\epsilon n}$ k-CNFs, each with at most $(\frac{k}{\epsilon})^{3k}n$ clauses. By writing each of these smaller k-CNFs as a cn size VSP circuit and solving the satisfiability of each circuit in time $\mathcal{O}(2^{\delta n})$, we can solve the original CNF-Sat instance in time $2^{(\delta+\epsilon)n}$. Choosing ϵ sufficiently small implies that SETH fails.

Theorem 8.8 If the satisfiability of cn size VSP circuits can be decided in time $\text{poly}(n)2^{\delta n}$ for $\delta < 1$ independent of c, then SETH fails.

It is also worth noting that an efficient deterministic satisfiability algorithm for VSP circuits would also imply nonlinear VSP circuit lower bounds by following Williams's argument [Wil10].

8.2 On Improving Dynamic Programming Based Algorithms

Let us start with the following problem definitions:

⊕Bipartite Independent Set **Parameter:** n
Input: A bipartite graph $(A \cup B, E)$ where $|A| = n$, $|B| = m$, for every $v \in B$, $d(v) \leq k$ and $m \leq f(k)n$.
Question: Is the number of independent sets odd?

Set Cover **Parameter:** n
Input: An integer t and a set system $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = m$, $|U| = n$ and for every $S \in \mathcal{F}$, $|S| \leq k$.
Question: Is there a subset $\mathcal{C} \subseteq \mathcal{F}$ with $|\mathcal{C}| \leq t$ such that $\bigcup_{S \in \mathcal{C}} S = U$?

⊕Set Cover **Parameter:** n
Input: A set system $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = m$, $|U| = n$, for every $S \in \mathcal{F}$, $|S| \leq k$ and $m \leq f(k)n$.
Question: Is the number of $\mathcal{C} \subseteq \mathcal{F}$ with $\bigcup_{S \in \mathcal{C}} S = U$ odd?

⊕Steiner Tree **Parameter:** n
Input: An integer n, graph $G = (V, E)$ of maximum degree at most k with terminals $T \subseteq V$ and $m = |V| \leq f(k)n$.
Question: Is the number of subsets $T \subseteq X \subseteq V$ with $|X| \leq n$ and $G[X]$ connected odd?

⊕Set Cover$_\alpha$ **Parameter:** n
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 $S \in \mathcal{F}$, $|S| \leq k$ and $n \leq f(k)n$.
Question: Is the number of $\mathcal{C} \subseteq \mathcal{F}$ with $|\mathcal{C}| = t$ such that $\bigcup_{S \in \mathcal{C}} S = U$ odd?

⊕Connected Vertex Cover **Parameter:** n
Input: An integer n and a graph $G = (V, E)$ with $|V| = m$ of maximum degree at most k and $m \leq f(k)n$.
Question: Is the number of sets $X \subseteq V$ such that $|X| = n$, $X \cap e \neq \emptyset$ for every $e \in E$ and $G[X]$ is connected odd?

In this section we give a reduction that gives some evidence that Set Cover may not be solvable in time better than $2^{|U|} = 2^n$ unless CNF-Sat can be. In particular, we show that ⊕CNF-Sat, ⊕Hitting Set and ⊕Set Cover are equivalent using an interesting property of the number of independent sets in a bipartite graph. These equivalences have some surprising corollaries. For example, this allows us to conclude that the current best parameterized algorithm for Connected Vertex Cover, Steiner Tree, presented in [CNP+11a] are optimal unless there is an improved algorithm for ⊕CNF-Sat.

8.2.1 From ⊕Set Cover to ⊕Hitting Set

Given a CNF formula φ over n variables and clauses of size at most k and an odd integer $p > 2$ that divides n, we first create the set system $\mathcal{F}_{\varphi,p} \subseteq 2^U$ as described in Section 8.1.2. Given the set system $\mathcal{F}_{\varphi,p} \subseteq 2^U$, create the set system $\mathcal{F}'_{\varphi,p}$ as follows:

8. For every block U_i:
   - add a special element e_i to the universe,
   - for every $X \in \binom{U_i}{\lfloor p'/2 \rfloor}$, add the set $X \cup \{e_i\}$ to the set family.

Lemma 8.2 The number of hitting sets of the instance $\mathcal{F}_{\varphi,p}$ of size $\lceil p'/2 \rceil \frac{n}{p}$ is odd iff the number of hitting sets of the instance $\mathcal{F}'_{\varphi,p}$ is odd.

Proof Let $g = \frac{n}{p}$. We first prove that the number of hitting sets of $\mathcal{F}_{\varphi,p}$ of size $\lceil p'/2 \rceil g$ is equal to the number of hitting sets H' of $\mathcal{F}'_{\varphi,p}$ such that $|H' \cap U_i| = \lceil \frac{p'}{2} \rceil$ for every $1 \leq i \leq g$. Suppose that H is a hitting set of $\mathcal{F}_{\varphi,p}$ of size $\lceil p'/2 \rceil g$, then it is easy to see that $H \cup \{e_1, \ldots, e_g\}$ is a hitting set of $\mathcal{F}'_{\varphi,p}$ since all the sets added in Step 8 are hit by some e_i, and indeed $|H' \cap U_i| = \lceil \frac{p'}{2} \rceil$ for every $1 \leq i \leq g$. For the reverse direction, suppose H' is a hitting set of $\mathcal{F}'_{\varphi,p}$ such that $|H' \cap U_i| = \lceil \frac{p'}{2} \rceil$

for every $1 \leq i \leq g$, then $\{e_1, \ldots, e_g\} \subseteq H'$ since all the sets added in Step 8 are hit by H'. And hence we have a bijection between the two families of hitting sets.

For every hitting set H' of $\mathcal{F}'_{\varphi,p}$ and block U_i, we know that $|H' \cap U_i| \geq \lceil p'/2 \rceil$ since otherwise the set $U_i \setminus H'$ added in Step 5 is not hit by H'. So it remains to show that the number of hitting sets H' of $\mathcal{F}'_{\varphi,p}$ such that there is an $1 \leq i \leq g$ with $|H' \cap U_i| > \lceil \frac{p'}{2} \rceil$ is even. Given such a hitting set H', let $\gamma(H') = H' \Delta \{e_i\}$ where i is the smallest integer such that $|H' \cap U_i| > \lceil \frac{p'}{2} \rceil$. Obviously γ is its own inverse and $|\gamma(H') \cap U_i| > \lceil \frac{p'}{2} \rceil$ so now it remains to show that $\gamma(H')$ is also a hitting set of $\mathcal{F}'_{\varphi,p}$. To see this, notice that all sets $X \cup \{e_i\}$ added in Step 8 where $X \in \binom{U_i}{\lfloor p'/2 \rfloor}$ are hit since $|\gamma(H') \cap U_i| > \lceil \frac{p'}{2} \rceil$ and that those are the only sets containing e_i. ■

Theorem 8.9 For every positive integer p:

$$\sigma_k(\oplus\text{CNF-Sat}) \leq \sigma_{pk}(\oplus\text{Hitting Set}) + \mathcal{O}\left(\frac{\log p}{p}\right).$$

Proof Create the set system $\mathcal{F}'_{\varphi,p}$ as described above. For a constant p, this clearly can be done in polynomial time. Also, from the construction of $\mathcal{F}'$ it follows that the number of sets of $\mathcal{F}'$ is bounded by $f(k)n$ for constant p. Also it is easy to see that the maximum size of a set of $\mathcal{F}'_{\varphi,p}$ is at most pk, and thus we can determine the number of hitting sets modulo 2 of $\mathcal{F}'_{\varphi,p}$ in $\mathcal{O}^\star(2^{\sigma_{pk}(\oplus\text{Hitting Set})n'})$ time where n' is the size of the universe of $\mathcal{F}_{\varphi,p}$. Since $n' = \lceil \frac{n}{p} \rceil (p + 2\lceil \log p \rceil) = n(1 + \mathcal{O}(\frac{\log p}{p}))$ the theorem follows. ■

8.2.2 The Flip: ⊕Hitting Set Equals ⊕Set Cover

Lemma 8.3 Let $G = (A \cup B, E)$ be a bipartite graph, then the number of independent sets of G is congruent to

$$|\{X \subseteq A : N(X) = B\}| \text{ modulo } 2.$$

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. ■

It is worth mentioning that this lemma was inspired by a non-modular variant from [NvR10, Lemma 2] that we will also discuss in Section 9.1 (see also [vR11, Proposition 9.1]).

Theorem 8.10 For every positive integer k,

$$\sigma_k(\oplus\textsc{Hitting Set}) = \sigma_k(\oplus\textsc{Bipartite Independent Set}) = \sigma_k(\oplus\textsc{Set Cover}).$$

Proof Given a set system $\mathcal{F} \subseteq 2^U$, let $G = (\mathcal{F}, U, E)$ be the bipartite graph where $(S, e) \in E$ iff $e \in S$. Note that the number of hitting sets of $\mathcal{F}$ is equal to $|\{X \subseteq U : N(X) = \mathcal{F}\}|$. Then by Lemma 8.3, the number of hitting sets is equal 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 equal to the number of independent sets of G modulo 2. Thus the problems are equivalent. ■

8.2.3 From ⊕Set Cover to ⊕Steiner Tree and ⊕Connected Vertex Cover

Here, we give reductions from Set Cover to Steiner Tree and the Connected Vertex Cover problem, and then from ⊕Set Cover to ⊕Steiner Tree and the ⊕Connected Vertex Cover problem. To do this, we will first need an intermediate result, showing that Set Cover$_\alpha$, that is, Set Cover with the solution size bounded by αn, is as hard as Set Cover for every $\alpha > 0$ (and similarly for ⊕Set Cover and ⊕Set Cover$_\alpha$). Given this intermediate result, the reductions to ⊕Steiner Tree and ⊕Connected Vertex Cover follow easily.

Theorem 8.11 For any $\alpha, \varepsilon > 0$ we have:

$$\sigma(\oplus\textsc{Set Cover}) \leq \sigma(\oplus\textsc{Set Cover}_\alpha) + \varepsilon\,.$$

Proof The reduction transforms the instance $(\mathcal{F}'_{\varphi_i,p}, U')$ of ⊕Set Cover into polynomially many instances of the $(pkq + z)$-⊕Set Cover$_\alpha$ problem, for some positive integers q, z.

In order to find the parity of the number of set covers of the instance $(\mathcal{F}'_{\varphi_i,p}, U')$ 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 $j = 1, \ldots, |\mathcal{F}'_{\varphi_i,p}|$. 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 $(\mathcal{F}'_{\varphi_i,p}, U')$ of size j'. Let q be the smallest integer which is a power of two satisfying $\frac{|\mathcal{F}'_{\varphi_i,p}|}{q} + 2 \leq \alpha|U'|$. 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 constant). Observe that q is upper bounded by a constant independent of n since $|\mathcal{F}'_{\varphi_i,p}| \leq c_1 n$ and $|U'| \geq n$.

We create a temporary set system $(\mathcal{F}_0, U_0)$ to ensure that the size of the set cover 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 $(\mathcal{F}'_{\varphi_i,p}, U')$ 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 size of $(\mathcal{F}'_{\varphi_i,p}, U')$ of size $j' - (q-r) < j$ which we already know.

To obtain a $\oplus$Set Cover$_\alpha$ instance we set $U^* = U_0$ and to a family $\mathcal{F}^*$ we add all unions of exactly q sets from $\mathcal{F}_0$. 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 pqk-$\oplus$Set Cover$_\alpha$, since $\mathcal{F}^*$ could be a multiset. Note that each subset of U^* appears in $\mathcal{F}^*$ at most $(2^{pqk})^q = 2^{pq^2k}$ 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 $z = 1+pq^2k$ 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 $(pqk+z)$-$\oplus$Set Cover$_\alpha$ instance since $t = t^* + 1 \leq \alpha|U|^* \leq \alpha|U|$. Observe that in the final instance we have $|U| \leq n(1 + 2\frac{\log p}{p}) + q + z$ and $|\mathcal{F}| \leq (c_1 n + q)^q + 1$.

To summarize the reduction, we took an instance of k-Unique-CNF-Sat, and after sparsification, each sparsified formula was transformed into an instance of pk-$\oplus$Hitting Set, which was equivalent to pk-$\oplus$Set Cover because of Lemma 8.3. Next, in each pk-$\oplus$Set Cover instance we iterated over the size of solution, 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 initial formula of k-Unique-CNF-Sat instance has a satisfying assignment, 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 claims. 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.

Claim 8.11.1 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.

Claim 8.11.2 — Folklore For any non-negative integers a, b such that $b \leq a$ the binomial coefficient $\binom{a}{b}$ is odd iff $\text{ones}(b) \subseteq \text{ones}(a)$, where $\text{ones}(x)$ is the set of indices containing ones in the binary representation of x.

Proof Since $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ we infer that $\binom{a}{b}$ is odd iff

$$\sum_{i=1}^{\infty} \left\lfloor \frac{a}{2^i} \right\rfloor = \sum_{i=1}^{\infty} \left\lfloor \frac{b}{2^i} \right\rfloor + \sum_{i=1}^{\infty} \left\lfloor \frac{a-b}{2^i} \right\rfloor.$$

Since for each i we have

$$\left\lfloor \frac{a}{2^i} \right\rfloor \geq \left\lfloor \frac{b}{2^i} \right\rfloor + \left\lfloor \frac{a-b}{2^i} \right\rfloor$$

the binomial coefficient $\binom{a}{b}$ is odd iff for each positive integer i we have $\left\lfloor \frac{a}{2^i} \right\rfloor = \left\lfloor \frac{b}{2^i} \right\rfloor + \left\lfloor \frac{a-b}{2^i} \right\rfloor$. This means that $\binom{a}{b}$ is odd iff for each i either $\left\lfloor \frac{a}{2^{i-1}} \right\rfloor$ is odd or $\left\lfloor \frac{b}{2^{i-1}} \right\rfloor$ is even. ■

Claim 8.11.3 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 2, using Lemma 8.11.2 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 Claim 8.11.1 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 Claim 8.11.3 we know that d_{j_0} equals one. Thus modulo 2 we have the following congruence.

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

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 claim.

Claim 8.11.4 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 $|\Phi^{-1}(A_0)|$ 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 2 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 $\oplus$Set Cover$_\alpha$ with universe size bounded by $n(1 + 2\frac{\log p}{p}) + q + z$ and set family size bounded by $(c_1 n + q)^q + 1$ we verify whether the initial formula ϕ has a satisfying assignment, which finishes the prove of Theorem 8.1. ■

Corollary 8.1 For any $\alpha > 0$ we have:

$$\sigma(\oplus\text{Set Cover}) \leq \sigma(\oplus\text{Set Cover}_\alpha)\,.$$

We can now obtain the following two results by standard Karp-reductions.

Theorem 8.12 For every $\alpha > 0$,

$$\sigma(\text{Set Cover}_\alpha) \leq \sigma(\text{Steiner Tree}) + \alpha, \text{and}$$
$$\sigma(\oplus\text{Set Cover}_\alpha) \leq \sigma(\oplus\text{Steiner Tree}) + \alpha.$$

Proof Given an instance of Set Cover$_\alpha$ consisting of a set system $(\mathcal{F}, U)$ and integer i, let G' be the graph obtained from the incidence graph of $(\mathcal{F}, U)$ by adding a vertex s universal to $\mathcal{F}$ with a pendant vertex u, and define the terminal set to be $U \cup \{u\}$. It is easy to see that the number of Steiner trees of size $|U|+i+1$ is equal to the number of set covers of $(\mathcal{F}, U)$ of size i. Hence the theorem follows. ■

Theorem 8.13 For every $\alpha > 0$,

$$\sigma(\text{Set Cover}_\alpha) \leq \sigma(\text{Connected Vertex Cover}) + \alpha, \text{ and}$$
$$\sigma(\oplus\text{Set Cover}_\alpha) \leq \sigma(\oplus\text{Connected Vertex Cover}) + \alpha.$$

Proof Given an instance $(\mathcal{F}, U)$ of Set Cover$_\alpha$, we create an instance of Connected Vertex Cover where G is 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 we can take without loss of optimality all vertices having a pendant vertex, and then connecting these vertices is equivalent to covering all elements of U with sets in $\mathcal{F}$. Hence, an $\mathcal{O}^\star(2^{\sigma(\text{Connected Vertex Cover})n})$ time algorithm for Connected Vertex Cover implies an $\mathcal{O}^\star(2^{(\sigma(\text{Connected Vertex Cover})+\alpha)n})$ time algorithm for Set Cover$_\alpha$.

For the parity case, let us study the number of connected vertex covers of size j of G for every j. Similar to the previous case, 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, since for $i = j - |U| - 1$ we have $\binom{|U|+1}{j-i-|U|-1} = 1$.

Thus, if in time $\mathcal{O}^\star(2^{\sigma(\text{Connected Vertex Cover})n})$ we can compute the number of connected vertex covers of size n modulo 2, we can compute the parity of all $(c_1, \ldots, c_{i+|U|+1})$ and hence the parity of s_i in $\mathcal{O}^\star(2^{(\sigma(\text{Connected Vertex Cover})+\alpha)n})$.■

8.2.4 From Set Cover to Set Partition and Subset Sum

Connected Vertex Cover **Parameter:** n
Input: An integer n and a graph $G = (V, E)$ with $|V| = m$ of maximum degree at most k and $m \leq f(k)n$.
Question: Is there a subset $X \subseteq V$ such that $|X| \leq n$, $X \cap e \neq \emptyset$ for every $e \in E$ and $G[X]$ is connected?

Set Partition **Parameter:** n
Input: An integer t and a set system $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = m, |U| = n$, for every $S \in \mathcal{F}$, $|S| \leq k$ and $m \leq f(k)n$.
Question: Is there a subset $\mathcal{C} \subseteq \mathcal{F}$ with $|\mathcal{C}| \leq t$ such that $\bigcup_{S \in \mathcal{C}} S = U$ and for every $S, S' \in \mathcal{F}$ with $S \neq S'$, $S \cap S' = \emptyset$?

Subset Sum **Parameter:** n
Input: Integers $a_1, \ldots, a_m \in \mathbb{Z}_+$ such that the number of 1's in the bit representation of every a_i is at most k and a target integer t on n bits.
Question: Is there a subset $X \subseteq \{1, \ldots, m\}$ with $\sum_{i \in X} a_i = t$?

Theorem 8.14 For every positive integer k,

$$\sigma_k(\text{Set Cover}) \leq \sigma_k(\text{Set Partition}).$$

Proof Let $(\mathcal{F}, t)$ be an instance of Set Cover. Create an instance $(\mathcal{F}', t)$ of Set Partition by for every $S \in \mathcal{F}$ adding all subsets of S to $\mathcal{F}'$. Clearly $(\mathcal{F}', t)$ has a set partitioning of size at most t if and only if $(\mathcal{F}, t)$ has a set cover of size at most t. Since the size of the sets in $\mathcal{F}$ is bounded by k, the reduction runs in polynomial time. ■

Theorem 8.15 For every positive integer k,

$$\sigma_k(\text{Set Partition}) \leq \sigma_{k+\log k+1}(\text{Subset Sum}).$$

Proof Let $(\mathcal{F}, t)$ be an instance of Set Partition, $\mathcal{F} \subseteq 2^U$. We create an instance of Subset Sum as follows. Let the target integer t' for Subset Sum have a bit expansion consisting of three fields: First, as the most significant bits, a field coding the value of t, to check the cardinality of the solution $\mathcal{C} \subseteq \mathcal{F}$; second, a field of length $\log t + \log n$ containing the value n, to check the total size of all sets in $\mathcal{C}$; finally, a field of length $\log t + n$ containing n ones. The paddings of length $\log t$ serve to isolate the fields from each other. For every $S_i \in \mathcal{F}$, we create an integer a_i with the same field division as t', where the first field encodes 1, the second field encodes $|S_i|$, and the third field contains a one in position j if and only if $u_j \in S_i$. We argue that the resulting Subset Sum instance is positive if and only if $\mathcal{F}$ contains a partitioning of U using exactly t sets.

Clearly, if $\mathcal{C} \subseteq \mathcal{F}$ partitions U and $|\mathcal{C}| = t$, then the integers a_i corresponding to $S_i \in \mathcal{C}$ sum to t'. The first field sums to t by cardinality of $\mathcal{C}$, the second sums to n, and in the third field the non-zero digits are simply partitioned between the a_i.

So let A be a collection of integers a_i that sum to t'. By the first field, we have $|A| \leq t$; thus the padding of length $\log t$ is enough to isolate the fields, and we have $|A| = t$. By the same argument on the second field, the sum over all $a_i \in A$ of the number of non-zero bits in the third field is exactly n. Now, the only way that the third field can actually contain n true bits is if the true bits in the third field are partitioned among the a_i. Thus, $\mathcal{C} = \{S_i : a_i \in A\}$ is a set partitioning of U of cardinality exactly t.

By looping over all $t_0 \leq t$ for the Set Partition instance, this solves the problem. Note that the length of the bit string t' is $n+\mathcal{O}(\log n)$, which disappears into the asymptotics, and the number of set bits in the a_i is bounded by $k+\log k+1$. ■

8.3 Conclusions and Further Work

In this paper we showed that for several problems on covering and packing, the current known algorithms cannot be improved unless we get a faster algorithm for either CNF-Sat or ⊕CNF-Sat. Thus, we have created a class of combinatorial problems that are equivalent to CNF-Sat. The obvious open question is the following:

Open Question 12 Is $\sigma(\text{CNF-Sat}) \leq \sigma(\text{Set Cover})$?

In Figure 8.1, the fundamental difference between the two grey areas is that in the left area the known efficient algorithms are **OPP** algorithms and in the other area it is not known:

Open Question 13 — Paturi(Pat10), orginally asked for Hamiltonian Path Is there an OPP algorithm with one-sided error probability at most $1-2^{-\mathcal{O}(n)}$ for any problem in the right grey area of Figure 8.1?

See Chapter 2 for the definition of an OPP algorithm.

It would be nice to obtain lower bounds on the running time of some graph problems like Dominating Set or Independent Set. In Chapter 9 we will give a observation in this direction. It is worth mentioning that related to Section 8.1.4, very recently [SS11] proved that if SETH is false, then there exists a $\mathcal{O}^\star(2^{\delta n})$-time algorithm for the Circuit Sat problem restricted to circuits on n variables of constant depth and size linear in n.

Part IV

Epilogue

Chapter 9

Concluding Remarks

Here we will, on a very high level, relate the chapters of this thesis to each other and propose some further research. Before this, we will elaborate on a result published in [NvR10] and give a couple of small observations related to Section 8.2.2.

9.1 More on Symmetry of Bipartite Graphs

In Section 8.2.2 we have seen that by a simple combinatorial property of symmetry of bipartite graphs (refer to Lemma 8.3), it followed that ⊕Hitting Set, ⊕Bipartite Independent Set, and ⊕Set Cover are equivalent (refer to Theorem 8.10). This was especially interesting since for ⊕Hitting Set the fastest known algorithm is exhaustive search while for Set Cover this is naive dynamic programming (and inclusion-exclusion), so in this case, by Lemma 8.3 improving over exhaustive search for ⊕Hitting Set is equivalent to improving over naive dynamic programming for ⊕Set Cover, so the duality suggested in Section 2.2 is in this case very concrete.

Here, we would like to state a variant of Lemma 8.2.2, not exploiting the properties of parity versions that appeared in [NvR10], actually inspired Lemma 8.2.2, and may be useful for further research. Also we would like to mention one more consequence of Lemma 8.3 concerning branching algorithms for Independent Set and Dominating Set.

Given a graph $G = (V, E)$, and $A, B \subseteq V$ with $A \cap B = \emptyset$, and integers $0 \leq i \leq |A|$ and $0 \leq j \leq |B|$, we define $b_{i,j}(A, B)$ as follows:

$$b_{i,j}(A, B) = |\{X \subseteq A : |X| = i \wedge |N(X) \cap B| = j\}|$$

Consider the following computation task: we are given a bipartite graph $G = (V = A \cup B, E)$ and are asked to compute $b_{i,j}(A, B)$ for every $0 \leq i \leq |A|$ and $0 \leq j \leq |B|$. In [vR11] this is called the #Partial Red Blue Dominating Set problem and is easy to see that if we take $n = |A|$ as complexity parameter,

this generalizes the (counting variant of) the Hitting Set problem ($b_{i,|B|}(A,B)$ is the number of 'hitting sets' of size i). On the other hand, if we are asked to compute the number of $b_{j,i}(B,A)$ for every $0 \leq i \leq |A|$ and $0 \leq j \leq |B|$ (and again take the complexity parameter to be $n = |A|$), this generalizes the (counting variant of) the Set Cover problem ($b_{|B|,i}(B,A)$ is the number of 'set covers' of size i). Similarly to Theorem 8.10 we show that the two problems solved by entirely different techniques are intimately connected:

Lemma 9.1 Given a graph $G = (V,E)$, and $A, B \subseteq V$ with $A \cap B = \varnothing$, the following holds:

$$b_{i,j}(A,B) = \sum_{k=0}^{|A|} \sum_{l=0}^{|B|} (-1)^{l+j-|B|} \binom{l}{|B|-j} \binom{|A|-k}{i} b_{l,k}(B,A)$$

This lemma also appears in [NvR10] and [vR11, Subsection 9.2.1.]. Using Lemma 9.1 we can always choose to compute the values $b_{i,j}(A,B)$ or $b_{i,j}(B,A)$, since given either one we can compute the other in polynomial time. Hence, we obtain the following corollary:

Corollary 9.1 Let $G = (A \uplus B, E)$. For every $\delta < 1$, there exists an $\mathcal{O}^\star(2^{\delta|A|})$-time algorithm that computing $b_{ij}(A,B)$ for every i,j if and only if there exists an $\mathcal{O}^\star(2^{\delta|B|})$ algorithm computing $b_{ij}(A,B)$ for every i,j.

Let U be a set and let $P_1, P_2 \ldots, P_n$ be subsets of U. The sets $P_1, P_2, \ldots, P_n$ can be thought of as properties. We define for a partitioning $R \uplus O \uplus F = \{1,2,\ldots,n\}$ and integer $0 \leq t \leq n$:

$$a_t(R,O,F) = |\{e \in U \mid |R[e]| = t \wedge |F[e]| = 0\}|$$

where we denote $R[e] = \{i \in R \mid e \in P_i\}$ and $F[e] = \{i \in F \mid e \in P_i\}$. We can say that $e \in U$ is counted in $a_t(R,O,F)$ if it satisfies (i.e., is contained in) exactly t of the properties in R (the *required properties*) and none of the properties in F (the *forbidden properties*). Now, the following holds for every $R \uplus O \uplus F = \{1,2,\ldots,n\}$ and integers $0 \leq i,t \leq n$:

$$a_t(R \cup \{i\}, O, F) = a_t(R, O, F \cup \{i\}) + (a_{t-1}(R, O \cup \{i\}, F) - a_{t-1}(R, O, F \cup \{i\})) \quad (9.1)$$

To see this holds recall that the set of elements $e \in U$ accounted for on the left-hand side is the set all elements contained in exactly t *required properties* and none of the forbidden properties. This set can be partitioned into the elements $e \notin P_i$ which is accounted for at $a_t(R,O,F \cup \{i\})$ and the elements $e \in P_i$. The elements $e \in P_i$ accounted for on the left-hand are exactly the elements that are accounted for $a_{t-1}(R, O \cup \{i\}, F)$ but are not accounted for in $a_{t-1}(R,O,F \cup \{i\}))$.

If one expands the recurrence (9.1), then the following natural variation on the inclusion/exclusion formula can be obtained for computing $a_t(R,O,F)$ similarly to the proof of (3.1):

$$a_t(R,O,F) = \sum_{X\subseteq R}(-1)^{|X|-|R|+t}\binom{|X|}{|R|-t}a_0(\varnothing, O\cup(R\setminus X), F\cup X)$$

To see this, consider the branching tree after exhaustively applying the recurrence (9.1). Let X be a set of forbidden properties, i.e., the set of properties that go into the first or third branch when branching. We consider the set of leaves of this tree where X is the set of forbidden properties. For each such leaf, we have lowered the parameter t exactly t times, thus we have taken $|R|-t$ times the first branch and $|X|-|R|+t$ times the second branch. As a result, we have $\binom{|X|}{|R|-t}$ such leaves, and the contribution of each leaf to the sum in the root is multiplied exactly $|X|-|R|+t$ times by -1.

Proof (of Lemma 9.1) Now we will give a similar formula, where $a_t(R,O,F)$ is replaced by $b_{i,j}(A,B)$. We substitute U with the set of all subsets of A of size i and for a vertex $v\in B$, P_v are all elements of U that contain a neighbor of v. In this way, we obtain the following formula summing over all sets of vertices $X\subseteq B$ that correspond to the blue vertices that we forbid to be dominated:

$$\begin{aligned} b_{i,j}(A,B) &= \sum_{X\subseteq B}(-1)^{|X|-|B|+j}\binom{|X|}{|B|-j}b_{i,0}(A\setminus N(X),\varnothing)\\ &= \sum_{X\subseteq B}(-1)^{|X|-|B|+j}\binom{|X|}{|B|-j}\binom{|A\setminus N(X)|}{i}\end{aligned}$$

The second equality here follows from the definition of $b_{i,j}(A,B)$: $\binom{|A\setminus N(X)|}{i}$ equals the number of ways to choose i red vertices that have no forbidden neighbors.

If we now group the summands of the summation by the size of $A\cap N(X)$ and the size of X, then we obtain the following equation:

$$\begin{aligned} b_{i,j}(A,B) &= \sum_{k=0}^{|A|}\sum_{l=0}^{|B|}\sum_{\substack{X\subseteq B\\ |A\cap N(X)|=k,|X|=l}}(-1)^{l-|B|+j}\binom{l}{|B|-j}\binom{|A|-k}{i}\\ &= \sum_{k=0}^{|A|}\sum_{l=0}^{|B|}(-1)^{l-|B|+j}\binom{l}{|B|-j}\binom{|A|-k}{i}\sum_{\substack{X\subseteq B\\ |A\cap N(X)|=k,|X|=l}}1\\ &\qquad \big(\text{applying the definition of } b_{j,i}(B,A)\big)\\ &= \sum_{k=0}^{|A|}\sum_{l=0}^{|B|}(-1)^{l+j-|B|}\binom{l}{|B|-j}\binom{|A|-k}{i}b_{l,k}(B,A).\end{aligned}$$

■

We would also like to mention some consequences of Lemma 8.3 for branching algorithms. In a graph $G = (V, E)$, and *independent set* is a subset X such that $u, v \in X \to (u, v) \notin E$ and a *dominating set* is a set X such that for every $v \in V$ it holds that $N[v] \cap X \neq \varnothing$. Consider the following problems:

DOMINATING SET **Parameter:** n
Input: A graph $G = (V, E)$ with $|V| = n$ and target weight t.
Question: Is there a dominating set $X \subseteq V$ with $|X| \leq t$?

⊕WEIGHTED DOMINATING SET **Parameter:** n
Input: A graph $G = (V, E)$ with $|V| = n$, weight function $\omega : V \to \{0, \ldots, N\}$, and target weight t with N polynomial in n..
Question: Is the number of dominating sets $X \subseteq V$ with $\omega(X) = t$ odd?

⊕WEIGHTED INDEPENDENT SET **Parameter:** n
Input: A graph $G = (V, E)$ with $|V| = n$, weight function $\omega : V \to \{0, \ldots, N\}$, and target weight t with N polynomial in n.
Question: Is the number of independent sets $X \subseteq V$ with $\omega(X) = t$ odd?

It is known that DOMINATING SET can be solved in $\mathcal{O}^\star(1.4969^n)$ [vRB11], with slightly slower algorithms for counting extensions [vRNvD09, NvR10]. The currently fastest algorithm for the INDEPENDENT SET problem is $\mathcal{O}^\star(1.2132^n)$ [KLR09b]. Also there are algorithms known for counting the number of *maximum weight* independent sets (to the author's knowledge, the algorithm given in [Wah08] running in $\mathcal{O}^\star(1.2377)$ time is currently the fastest known). Unfortunately, the author is not aware of any work on ⊕WEIGHTED INDEPENDENT SET, even with leaving out the 'parity' part (that is, counting the number of independent sets of exactly a given weight).

Theorem 9.1 If ⊕WEIGHTED INDEPENDENT SET can be solved in $\mathcal{O}^\star(\alpha^n)$ time then ⊕WEIGHTED DOMINATING SET can be solved in $\mathcal{O}^\star(\alpha^{2n})$ time.

Proof Suppose we are given a graph $G = (V, E)$ with $|V| = n$, a weight function $\omega : V \to \{0, \ldots, N\}$ and a target weight t. Create the graph $I = (V \cup V', E')$ where V' is a copy of V (and the copy of v is denoted by v') and $(v, w') \in E'$ if and only if $w \in N[v]$. Let $\omega(v') = 0$ for every $v \in V$. Then count the number of independent sets of weight t in I modulo 2 in $\mathcal{O}^\star(\alpha^{2n})$ time. By the same arguments as in the proof of Lemma 8.3, this is equal to the number of dominating sets of G of weight exactly t modulo 2. ■

Theorem 9.2 If ⊕Weighted Dominating Set can be solved in $\mathcal{O}^\star(\alpha^n)$ time then there exists a Monte-Carlo algorithm with constant one-sided error probability that solves Dominating Set in $\mathcal{O}^\star(\alpha^n)$ time.

Proof The algorithm for Dominating Set is the following: given a graph $G = (V, E)$ and an integer t, choose $\omega(v) \in \{1, \ldots, 2|V|\}$ uniformly and independently at random for every $v \in V$. Let $\omega'(v) = \omega(v) + (2|V|)^2 + 1$. Use the algorithm for ⊕Weighted Dominating Set to compute for every weight $1 \leq w \leq t(2|V| + (2|V|)^2 + 1)$ the parity of the number of dominating sets of weight w with respect to ω'. Return `YES` if for some w the number of dominating sets of weight w is odd, and `NO` otherwise.

Let us analyse the correctness of the above algorithm. Note that for every $X \subseteq V$,

$$|X|((2|V|)^2 + 1) < \omega'(X) \leq |X|((2|V|)^2 + 1) + |X|2|V|.$$

Hence there exists a dominating set X of G such that $\omega'(X) \leq t(2|V| + (2|V|)^2 + 1)$ if and only if G has a dominating set of size at most t. Consequently, if the algorithm returns `YES` then G has a dominating set of size at most t. In the other direction let us assume that G has a dominating set of size at most t. Let $\mathcal{F}$ be the family of all dominating sets of G of minimum size. By Lemma 4.2, there will be a unique dominating set X_0 in $\mathcal{F}$ minimizing ω with probability at least $\frac{1}{2}$ and hence for $w = \omega(X_0) + |X_0|((2|V|)^2 + 1)$ the algorithm detects an odd number of dominating sets and thus returns `YES`. ■

9.2 Conclusion

Upper bounds

In this thesis, we have seen that transformations are very useful to improve over the naive way of evaluating a dynamic programming recurrence. The transformations we have encountered are naturally divided into two classes:

First are the invertible transformations that we used to save *space*. We first briefly discussed the well-known fact that, informally stated, if we implicitly have a small number of exponentially[(1)] large matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_l$ and want to compute one specific entry of their product (or another formula/circuit on the matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_l$), a matrix that diagonalizes all $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_l$ simultaneously would be very helpful for obtaining space efficient algorithms. We saw several examples where the diagonalizing matrix is a Fourier-transformation or Möbius-transformation matrix. These kind of algorithms transform the dynamic programming table into one of same size, but with easier logical dependence between the

[(1)] that is, exponential in the input size

table entries. It would be interesting to see how far this type of approach can bring us towards resolving Open Question 1.

Second are hashing methods. Here we implicitly transform the dynamic programming table into a smaller one using varying arguments in order to save *time*. Then we can use naive dynamic programming or sometimes apply transformations of the first class discussed here to compute the appropriate answer. The most notable example, the Color-coding technique has been proven already very powerful in history of theoretical computer science. Besides Color-coding, there are the approaches that rely on cancellation modulo 2:

- the Koutis-Williams approach: hash the subset lattice to find a non-zero entry in the dynamic programming table indexed with a subset of size at least k, and
- Björklund's cycle reversal technique (not explained in this thesis, see [Bjö10b]) and the Cut&Count approach: hash the set partition lattice to find non-zero entries in the dynamic programming table indexed with a partition consisting of at least k partition classes.

Could it be that these approaches can be unified by a single hash function on lattices?

A main advantage of the Color-coding approach is that it works equally well for cardinality problems (for example Longest Path) as for problems with exponentially large weights (for example, Weighted Longest Path). See also Open Question 9. Are there different kind of hash functions that have these properties and can be used as alternatives to the cancellation modulo 2 techniques? We also saw that the elementary hash functions of working modulo a randomly chosen prime or hashing by a randomly chosen matrix could be helpful to speedup sparse dynamic programming in some settings. Here we are stuck at the moment for problems such as CNF-Sat, but we did see a promising ring that could be used to hash to in Section 6.3.1.

A very nice open question we would like mention here is the following:

Open Question 14 Is there an algorithm that given a graph G and a tree decomposition of G of width at most t, solves the Independent Set problem in $\mathcal{O}^\star(2^{ct})$ time and $\mathcal{O}^\star(1)$ space? Or, would such a result have surprising complexity theoretic consequences?

Even a connection with Open Question 1 would be interesting. This question was explicitly asked in [LMS11c], and also in [Ned10]. See [LMS11c] for related observations and open questions.

Open Question 15 Is one of the algorithms for Subset Sum presented in Chapters 5 and 6 practical? Could it compete, or even be improved to compete with Bellman's algorithm?

Open Question 16 Do other useful embeddings of the min-sum semiring more efficient than the embedding in the polynomial ring exist? Or are there other techniques for applying transformations to dynamic programming algorithms over the min-sum semiring?

Lower Bounds

For some upper bounds given in this thesis, we gave a matching lower bound, thus completely determining the complexity of the problem assuming the SETH. For some other upper bounds, such as for the Steiner Tree parameterized by the number of terminals and Connected Vertex Cover parameterized by the solutions size we gave some indication that these might be optimal under the SETH (at least, we showed that giving improved algorithms for their counting versions is not possible under the SETH).

An interesting consequence of the techniques of Section 8.2.2 is that improving over exhaustive search for parity problems is equivalent to improving over exhaustive search for dynamic programming problems. On the other hand, for the problems studied in Chapter 8, no relation between exhaustive search and dynamic programming is known for the decision problems. How can we establish such a connection? One possible approach (but perhaps not the easiest one) to find such connections could be by attacking the following open question:

Open Question 17 Is $\sigma(\oplus\text{Set Cover}) \leq \sigma(\text{Set Cover})$? Is $\sigma(\oplus\text{CNF-Sat}) \leq \sigma(\text{CNF-Sat})$?

Also, we can try to attack the following question from both the lower bound and upper bound perspective:

Open Question 18 Can it be determined in $\mathcal{O}^\star((3-\epsilon)^k)$ whether a graph has a Feedback Vertex Set of size at most k? Or can we find a connection with the SETH (perhaps using the tools of Chapter 8)?

Bibliography

[AB03] Manindra Agrawal and Somenath Biswas. Primality and identity testing via chinese remaindering. *J. ACM*, 50:429–443, July 2003.

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Citeseer, 2009.

[ADF95] Karl R. Abrahamson, Rodney G. Downey, and Michael R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogues. *Ann. Pure Appl. Logic*, 73(3):235–276, 1995.

[Adl78] Leonard M. Adleman. Two theorems on random polynomial time. In *FOCS*, pages 75–83. IEEE, 1978.

[AFS09] Omid Amini, Fedor V. Fomin, and Saket Saurabh. Counting subgraphs via homomorphisms. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 2009.

[AGK+10] Noga Alon, Gregory Gutin, Eun Jung Kim, Stefan Szeider, and Anders Yeo. Solving MAX-r-SAT above a tight lower bound. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*,

SODA '10, pages 511–517, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[AKL+79] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, Laszlo Lovasz, and Charles Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, pages 218–223, Washington, DC, USA, 1979. IEEE Computer Society.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in $\mathcal{P}$. *The Annals of Mathematics*, 160(2):781–793, 2004.

[AM08] Vikraman Arvind and Partha Mukhopadhyay. Derandomizing the isolation lemma and lower bounds for circuit size. In *APPROX-RANDOM*, pages 276–289, 2008.

[Ant94] Howard Anton. *Elementary linear algebra*. Wiley, 7th edition, 1994.

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[BBYG00] 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.

[BCT90] Ulrich Baum, Michael Clausen, and Benno Tietz. Improved upper complexity bounds for the discrete Fourier transform. Technical report, 1990.

[Bel54] Richard E. Bellman. *Dynamic Programming (reprint 2003)*. Dover Publications, Incorporated, 1954.

[Bel62] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, 1962.

[BEPvR10] 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 *Proc. of SWAT 2010*, volume 6139 of *LNCS*, pages 62–73, 2010.

[BGRS10] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved LP-based approximation for Steiner tree. In *Proc. of STOC 2010*, pages 583–592, Proc. of STOC 2010.

[BH08] Andreas Björklund and Thore Husfeldt. Exact algorithms for exact satisfiability and number of perfect matchings. *Algorithmica*, 52(2):226–249, 2008.

[BHK09] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.

[BHK+11] Andreas Björklund, Thore Husfeldt, Petteri Kaski, Mikko Koivisto, Jesper Nederlof, and Pekka Parviainen. Fast zeta transforms for lattices with few irreducibles. 2011.

[BHKK07] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *STOC*, pages 67–74, 2007.

[BHKK08] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Computing the Tutte polynomial in vertex-exponential time. In *FOCS*, pages 677–686, 2008.

[BHKK10a] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *CoRR*, abs/1007.1161, 2010.

[BHKK10b] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Trimmed Möbius inversion and graphs of bounded degree. *Theory Comput. Syst.*, 47(3):637–654, 2010.

[BHM10] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Dániel Marx. Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. In *Proc. of STOC 2010*, pages 211–220, 2010.

[Bjö10a] Andreas Björklund. Determinant sums (and labeled walks) for undirected Hamiltonicity. 2010. Dagstuhl 'Exact Complexity', slides: http://www.dagstuhl.de/Materials/Files/10/10441/10441.BjoerklundAndreas.Slides.pptx.

[Bjö10b] Andreas Björklund. Determinant sums for undirected Hamiltonicity. In *FOCS*, pages 173–182, 2010.

[Bjö11] Andreas Björklund. Counting perfect matchings as fast as Ryser. 2011. http://arxiv.org/abs/1107.4466.

[BK06] Hans L. Bodlaender and Dieter Kratsch. An exact algorithm for graph coloring with polynomial memory. Technical Report UU-CS-2006-015, Department of Information and Computing Sciences, Utrecht University, 2006.

[BK08] Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.

[BKMK07] Glencora Borradaile, Claire Kenyon-Mathieu, and Philip N. Klein. A polynomial-time approximation scheme for Steiner tree in planar graphs. In *Proc. of SODA 2007*, pages 1285–1294, 2007.

[BM91] Robert S. Boyer and J. Strother Moore. Mjrty: A fast majority vote algorithm. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 105–118, 1991.

[BO88] Michael Ben-Or. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 301–309, New York, NY, USA, 1988. ACM.

[Bod94] Hans L. Bodlaender. On disjoint cycles. *International Journal of Foundations of Computer Science*, 5(1):59–68, 1994.

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

[BODI11] Joshua Buresh-Oppenheim, Sashka Davis, and Russell Impagliazzo. A stronger model of dynamic programming algorithms. *Algorithmica*, 60(4):938–968, 2011.

[BR10] 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.

[Cal08] Chris Calabro. A lower bound on the size of series-parallel graphs dense in long paths. *Electronic Colloquium on Computational Complexity (ECCC)*, 15(110), 2008.

[CCF+05] Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David W. Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized -hard problems. *Inf. Comput.*, 201(2):216–231, 2005.

[CCL10] Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set new measure and new structures. In *Proc. of SWAT 2010*, volume 6139 of *LNCS*, pages 93–104, 2010.

[CDL+11] Marek Cygan, Holger Dell, Daniel Lokstahnov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-Sat. 2011. Manuscript.

[CFL+08] 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.

[CG95] Fan R. K. Chung and Ronald L. Graham. On the cover polynomial of a digraph. *J. Comb. Theory, Ser. B*, 65(2):273–290, 1995.

[CGJY11] Robert Crowston, Gregory Gutin, Mark Jones, and Anders Yeo. Lower bound for Max-r-Lin2 and its applications in algorithmics and graph theory. *CoRR*, abs/1104.1135, 2011.

[CIKP03] Chris Calabro, Russell Impagliazzo, Valentine Kabanets, and Ramamohan Paturi. The complexity of unique k-SAT: An isolation lemma for k-CNFs. In *Proceedings of the 18th IEEE Conference on Computational Complexity, CCC 2003*, page 135. IEEE Computer Society, 2003.

[CIP06] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *IEEE Conference on Computational Complexity*, pages 252–260. IEEE Computer Society, 2006.

[CKL+09] Jianer Chen, Joachim Kneis, Songjian Lu, Daniel Mölle, Stefan Richter, Peter Rossmanith, Sing-Hoi Sze, and Fenghui Zhang. Randomized divide-and-conquer: Improved path, matching, and packing algorithms. 38(6):2526–2547, 2009.

[CKSU05] Henry Cohn, Robert D. Kleinberg, Balázs Szegedy, and Christopher Umans. Group-theoretic algorithms for matrix multiplication. In *FOCS*, pages 379–388. IEEE Computer Society, 2005.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to algorithms, 2001.

[CNP+11a] 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 *FOCS*, 2011. To appear. Full version: [CNP+11b].

[CNP+11b] 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. *CoRR*, abs/1103.0534, 2011.

[Cob65] Alan Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1965.

[Coo] Stephen A. Cook. The $\mathcal{P}$ versus $\mathcal{NP}$ problem. *The millennium prize problems*, pages 87–106.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[CP10] Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. *Theoretical Computer Science*, 411(40-42):3701–3713, 2010.

[CRS95] Suresh Chari, Pankaj Rohatgi, and Aravind Srinivasan. Randomness-optimal unique element isolation with applications to perfect matching and related problems. *SIAM J. Comput.*, 24(5):1036–1050, 1995.

[CS93] Donald Chinn and Rakesh K. Sinha. Bounds on sample space size for matrix product verification. *Inform. Proc. Letters*, 48:87–91, 1993.

[CT65] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput*, 19(90):297–301, 1965.

[CU03] Henry Cohn and Christopher Umans. A group-theoretic approach to fast matrix multiplication. In *FOCS*, pages 438–449. IEEE Computer Society, 2003.

[CW82] Don Coppersmith and Shmuel Winograd. On the asymptotic complexity of matrix multiplication. *SIAM J. Comput.*, 11(3):472–492, 1982.

[dF97] Babette de Fluiter. *Algorithms for Graphs of Small Treewidth*. PhD thesis, Utrecht University, 1997.

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.

[DF06] 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, 2006.

[DFHT05] Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-

genus graphs and h-minor-free graphs. *J. ACM*, 52:866–893, November 2005.

[DFL+05] 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 *Proc. of COCOON 2005*, volume 3595 of *LNCS*, pages 859–869, 2005.

[DFT06] Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Fast subexponential algorithm for non-local problems on graphs of bounded genus. In *Proc. of SWAT 2006*, volume 4059 of *LNCS*, pages 172–183, 2006.

[DFT08] Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Catalan structures and dynamic programming in H-minor-free graphs. In *Proc. of SODA 2008*, pages 631–640, 2008.

[DH08] Erik D. Demaine and Mohammad T. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *The Computer Journal*, 51(3):292–302, 2008.

[DHiK05] Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Ken ichi Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *Proc. of FOCS 2005*, pages 637–646, 2005.

[DKSS08] Anindya De, Piyush P. Kurur, Chandan Saha, and Ramprasad Saptharishi. Fast integer multiplication using modular arithmetic. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 499–506, New York, NY, USA, 2008. ACM.

[DL78] Richard A. DeMillo and Richard J. Lipton. A Probabilistic Remark on Algebraic Program Testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

[DPBF10] 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.

[Dre02] Stuart Dreyfus. Richard Bellman on the birth of dynamic programming. *Operations Research*, pages 48–51, 2002.

[DW72] Stuart E. Dreyfus and Robert A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17:449–467, 1965.

[EGGI92a] David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming I: linear cost functions. *J. ACM*, 39(3):519–545, July 1992.

[EGGI92b] David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming II: convex and concave cost functions. *J. ACM*, 39(3):546–567, July 1992.

[EJT10] Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *FOCS*, pages 143–152. IEEE Computer Society, 2010.

[Epp00] David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000.

[Epp07] David Eppstein. The traveling salesman problem for cubic graphs. *Journal of Graph Algorithms and Applications*, 11(1):61–81, 2007.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[FGK08] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Faster Steiner tree computation in polynomial-space. In *ESA*, pages 430–441, 2008.

[FGK09] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5), 2009.

[FGR09] Henning Fernau, Serge Gaspers, and Daniel Raible. Exact and parameterized algorithms for max internal spanning tree. In Christophe Paul and Michel Habib, editors, *WG*, volume 5911 of *Lecture Notes in Computer Science*, pages 100–111, 2009.

[FGSS09] 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.

[FK10] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[FKM+07] Bernhard Fuchs, Walter Kern, Daniel Mölle, Stefan Richter, Peter Rossmanith, and Xinhui Wang. Dynamic programming for minimum Steiner trees. *Theory Comput. Syst.*, 41(3):493–500, 2007.

[FKVV98] Joan Feigenbaum, Sampath Kannan, Moshe Y. Vardi, and Mahesh Viswanathan. Complexity of problems on graphs represented as obdds (extended abstract). In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '98, pages 216–226, London, UK, 1998. Springer-Verlag.

[FLGS10] Fedor V. Fomin, Daniel Lokshtanov, Fabrizio Grandoni, and Saket Saurabh. Sharp separation and applications to exact and parameterized algorithms. In Alejandro López-Ortiz, editor, *LATIN*, volume 6034 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2010.

[For09] Lance Fortnow. The status of the $\mathcal{P}$ versus $\mathcal{NP}$ problem. *Commun. ACM*, 52:78–86, September 2009.

[FT04] Fedor V. Fomin and Dimitrios M. Thilikos. A simple and fast approach for solving problems on planar graphs. In *Proc. of STACS 2004*, volume 2996 of *LNCS*, pages 56–67, 2004.

[Für09] Martin Fürer. Faster integer multiplication. *SIAM J. Comput.*, 39(3):979–1005, 2009.

[Gas08] Serge Gaspers. *Exponential-Time Algorithms: Structures, Measures, and Bounds*. PhD thesis, University of Bergen, 2008.

[Geb08] Heidi Gebauer. On the number of Hamilton cycles in bounded degree graphs. In *Proc. of ANALCO 2008*, pages 241–248, 2008.

[GGH+06] 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.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GKM+11] Parikshit Gopalan, Adam Klivans, Raghu Meka, Daniel Štefankovič, Santosh Vempala, and Eric Vigoda. An fptas for knapsack and related counting problems. In *FOCS*, 2011. To appear.

[GS87] Yuri Gurevich and Saharon Shelah. Expected computation time for hamiltonian path problem. *SIAM J. Comput.*, 16(3):486–502, 1987.

[GSS08] Serge Gaspers, Saket Saurabh, and Alexey A. Stepanov. A moderately exponential time algorithm for full degree spanning tree. In *TAMC*, pages 479–489, 2008.

[GW84] Hana Galperin and Avi Wigderson. Succinct representations of graphs. *Inf. Control*, 56:183–198, April 1984.

[Har89] Juris Hartmanis. 'Gödel, von Neumann and the P=?NP Problem. *Bulletin of the European Association for Theoretical Computer Science*, 38:101–107, 1989.

[Har03] Juris Hartmanis. Separation of complexity classes. *J. ACM*, 50:58–62, January 2003.

[Hås01] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48:798–859, July 2001.

[Hel89] Paul Helman. A common schema for dynamic programming and branch and bound algorithms. *J. ACM*, 36:97–128, January 1989.

[Hem02] Lane A. Hemaspaandra. Sigact news complexity theory column 36. *SIGACT News*, 33:34–47, June 2002.

[HJB84] M. Heideman, D. Johnson, and C. Burrus. Gauss and the history of the fast fourier transform. *ASSP Magazine, IEEE*, 1(4):14 –21, october 1984.

[HK62] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[HK70] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[HK71] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1(1):6–25, 1971.

[HKK05] Illya V. Hicks, Arie M. C. A. Koster, and Elif Kolotoğlu. Branch and tree decomposition techniques for discrete optimization. *Tutorials in Operations Research 2005*, pages 1–19, 2005.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley-Longman, 2nd edition, 2001.

[HS74] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21:277–292, April 1974.

[Imp95] Russel Impagliazzo. A personal view of average-case complexity. In *Proceedings of the 10th Annual Structure in Complexity Theory Conference*

(SCT'95), pages 134–, Washington, DC, USA, 1995. IEEE Computer Society.

[IN07] Kazuo Iwama and Takuya Nakashima. An improved exact algorithm for cubic graph TSP. In *Proc. of COCOON 2007*, volume 4598 of *LNCS*, pages 108–117, 2007.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[Ive62] Kenneth E. Iverson. *A programming language*. John Wiley & Sons, Inc., New York, NY, USA, 1962.

[KAF+10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen Lenstra, Emmanuel Thomé, Joppe Bos, Pierrick Gaudry, Alexander Kruppa, Peter Montgomery, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit rsa modulus. Cryptology ePrint Archive, Report 2010/006, 2010. `http://eprint.iacr.org/`.

[Kan10] Daniel M. Kane. Unary subset-sum is in logspace. *CoRR*, abs/1012.1336, 2010.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[Kar82] Richard M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Oper. Res. Lett.*, 1:49–51, 1982.

[KGGK94] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing*. Benjamin/Cummings, 1994.

[KGK77] Samuel Kohn, Allan Gottlieb, and Meryle Kohn. A generating function approach to the traveling salesman problem. In *ACM '77: Proceedings of the 1977 annual conference*, pages 294–300, New York, NY, USA, 1977. ACM.

[KH67] Richard M. Karp and Michael Held. Finite-state processes and dynamic programming. *SIAM Journal on Applied Mathematics*, 15(3):pp. 693–718, 1967.

[KI04] Valentine Kabanets and Russell Impagliazzo. Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[Kin10] Shiva Kintali. Realizable paths and the nl vs l problem. *CoRR*, abs/1011.3840, 2010.

[KKN11] Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. On homomorphic hashing for coefficient extraction. 2011. Manuscript.

[Klo94] Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[KLR09a] Joachim Kneis, Alexander Langer, and Peter Rossmanith. A fine-grained analysis of a simple independent set algorithm. In *FSTTCS*, volume 4 of *LIPIcs*, pages 287–298, 2009.

[KLR09b] Joachim Kneis, Alexander Langer, and Peter Rossmanith. A fine-grained analysis of a simple independent set algorithm. In Ravi Kannan and K Narayan Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009)*, volume 4 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 287–298, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[KLS96] Jeff Kahn, Nathan Linial, and Alex Samorodnitsky. Inclusion-exclusion: Exact and approximate. *Combinatorica*, 16(4):465–477, 1996.

[Knu69] Donald E. Knuth. The art of computer programming, vol. 2, seminumerical algorithms, 1969.

[KO63] Anatolii A. Karatsuba and Yuri P. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7:595–+, January 1963.

[Koi09] Mikko Koivisto. Partitioning into sets of bounded cardinality. In Jianer Chen and Fedor V. Fomin, editors, *IWPEC*, volume 5917 of *Lecture Notes in Computer Science*, pages 258–263. Springer, 2009.

[Kou08] Ioannis Koutis. Faster Algebraic Algorithms for Path and Packing Problems. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2008.

[KP10] Mikko Koivisto and Pekka Parviainen. A space-time tradeoff for permutation problems. In Moses Charikar, editor, *SODA*, pages 484–492. SIAM, 2010.

[KPS04] 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 *LNCS*, pages 235–247. Springer, 2004.

[KS93] Tracy Kimbrel and Rakesh Kumar Sinha. A probabilistic algorithm for verifying matrix products using $\mathcal{O}(n^2)$ time and $\log_2 n + o(1)$ random bits. *Inf. Process. Lett.*, 45:107–110, February 1993.

[KT06] Jon M. Kleinberg and Éva Tardos. *Algorithm design*. Addison-Wesley, 2006.

[KW09] Ioannis Koutis and Ryan Williams. Limits and Applications of Group Algebras for Parameterized Problems. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 653–664. Springer, 2009.

[Lev73] Leonid A. Levin. Universal sorting problems. *Problemy Peredaci Informacii*, 9:115–116, 1973.

[Lev03] Anany V. Levitin. *Introduction to the design & analysis of algorithms*. Addison-Wesley Reading, MA, 2003.

[Lin11] Andrzej Lingas. A fast output-sensitive algorithm for boolean matrix multiplication. *Algorithmica*, 61(1):36–50, 2011.

[Lip10] Richard Lipton. Beating bellman for the knapsack problem. 2010. http://rjlipton.wordpress.com/2010/03/03/beating-bellman-for-the-knapsack-problem/.

[LMS11a] Daniel Lokshtanov, Daniel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proc. of SODA 2011*, pages 777–789, 2011.

[LMS11b] Daniel Lokshtanov, Daniel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. In *Proc. of SODA 2011*, pages 760–776, 2011.

[LMS11c] Daniel Lokshtanov, Matthias Mnich, and Saket Saurabh. Planar k-Path in subexponential time and polynomial space. In *WG*, 2011.

[LN10] Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In Leonard J. Schulman, editor, *STOC*, pages 321–330. ACM, 2010.

[Lov79] László Lovász. On determinants, matchings, and random algorithms. In *FCT*, pages 565–574, 1979.

[Man95] Yishay Mansour. Randomized interpolation and approximation of sparse polynomials. *SIAM J. Comput.*, 24(2):357–368, 1995.

[Mar07] Dániel Marx. On the optimality of planar and geometric approximation schemes. In *FOCS*, pages 338–348, 2007.

[Mni10] Matthias Mnich. *Algorithms in Moderately Exponential Time*. PhD thesis, Eindhoven University, 2010.

[MPR+10] Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. FPT algorithms for connected feedback vertex set. In *Proc. of WALCOM 2010*, volume 5942 of *LNCS*, pages 269–280, 2010.

[MR96] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. *ACM Comput. Surv.*, 28:33–37, March 1996.

[MR97] David K. Maslen and Daniel N. Rockmore. Generalized ffts-a survey of some recent results. *Amer Mathematical Society*, 1997.

[MRR08] 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.

[MVV87] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[Ned08] Jesper Nederlof. Inclusion exclusion for hard problems. Master's thesis, Utrecht University, August 2008.

[Ned09] Jesper Nederlof. Fast polynomial-space algorithms using Möbius inversion: Improving on Steiner tree and related problems. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 713–725. Springer, 2009.

[Ned10] Jesper Nederlof. Saving space by algebraization. 2010. Dagstuhl 'Exact Complexity', http://www.dagstuhl.de/Materials/Files/10/10441/10441.NederlofJesper.Slides.pdf.

[Nie06] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2006.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput*, 22:838–856, 1993.

[NSS95] Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 182–, Washington, DC, USA, 1995. IEEE Computer Society.

[NvR10] Jesper Nederlof and Johan M. M. van Rooij. Inclusion/exclusion branching for partial dominating set and set splitting. In Venkatesh Raman and Saket Saurabh, editors, *IPEC*, volume 6478 of *Lecture Notes in Computer Science*, pages 204–215. Springer, 2010.

[Pan85] Victor Pan. Fast and efficient parallel algorithms for the exact inversion of integer matrices. In S. Maheshwari, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 206 of *Lecture Notes in Computer Science*, pages 504–521. Springer Berlin / Heidelberg, 1985. 10.1007/3-540-16042-6_29.

[Pap79] Christos H. Papadimitriou. Optimality of the fast fourier transform. *J. ACM*, 26:95–102, January 1979.

[Pap98] Christos H. Papadimitriou. Np-completeness: A retrospective. In *In Proc. of ICALP 98*, pages 2–6. Springer, 1998.

[Pat10] Ramamohan Paturi. Exact algorithms and complexity. 2010. Dagstuhl 'Exact Complexity', slides: http://www.dagstuhl.de/Materials/Files/10/10441/10441.PaturiRamamohan.Slides.pdf.

[PHN08] Oriana Ponta, Falk Hüffner, and Rolf Niedermeier. Speeding up dynamic programming for some $\mathcal{NP}$-hard graph recoloring problems. In *TAMC'08: Proceedings of the 5th international conference on Theory and applications of models of computation*, pages 490–501, Berlin, Heidelberg, 2008. Springer-Verlag.

[Pil11] Michał Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In Filip Murlak and Piotr Sankowski, editors, *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2011.

[PP10] Ramamohan Paturi and Pavel Pudlák. On the complexity of circuit satisfiability. In Leonard J. Schulman, editor, *STOC*, pages 241–250. ACM, 2010.

[PY86] Christos H. Papadimitriou and Mihalis Yannakakis. A note on succinct representations of graphs. *Inf. Control*, 71:181–185, December 1986.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.

[Rob86] John M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7(3):425–440, 1986.

[Ros41] Barkley Rosser. Explicit bounds for some functions of prime numbers. *American Journal of Mathematics*, 63(1):211–232, 1941.

[Ros74] Donald J. Rose. On simple characterizations of k-trees. *Discrete Mathematics*, 7(3-4):317–322, 1974.

[RS84] Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory*, 36(1):49–64, 1984.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21:120–126, February 1978.

[RSS02] 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 *LNCS*, pages 241–248. Springer, 2002.

[RSS06] 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.

[RSV04] Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

[RW11] Liam Roditty and Virginia Vassilevska Williams. Minimum weight cycles and triangles: Equivalences and algorithms. *CoRR*, abs/1104.2882, 2011.

[Sau08] Saket Saurabh. *Exact Algorithms for Optimization and Parameterized Versions of some Graph Theoretic Problems*. PhD thesis, The Institute of Mathematical Sciences, Chennai, India, 2008.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[Sch80] Jack. T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27:701–717, October 1980.

[Sch99] Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 410–, Washington, DC, USA, 1999. IEEE Computer Society.

[Sch05] Uwe Schöning. Algorithmics in exponential time. In Volker Diekert and Bruno Durand, editors, *STACS 2005*, volume 3404 of *Lecture Notes in Computer Science*, pages 36–43. Springer Berlin / Heidelberg, 2005.

[Sch11] Dominik A. Scheder. *Algorithms and Extremal Properties of SAT and CSP*. PhD thesis, Swiss Federal Institute of Technology Zurich, 2011.

[Sha79] Adi Shamir. Factoring numbers in o(logn) arithmetic steps. *Information Processing Letters*, 8(1):28 – 31, 1979.

[Sip92] Michael Sipser. The history and status of the $\mathcal{P}$ versus $\mathcal{NP}$ question. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, STOC '92, pages 603–618, New York, NY, USA, 1992. ACM.

[Sip97] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[Sol67] Louis Solomon. The burnside algebra of a finite group. *Journal of Combinatorial Theory*, 2(4):603 – 615, 1967.

[SS79] Richard Schroeppel and Adi Shamir. A tcs2 = 0 (2n) time/space tradeoff for certain np-complete problems. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, SFCS '79, pages 328–336, Washington, DC, USA, 1979. IEEE Computer Society.

[SS11] Rahul Santhanam and Srikanth Srinivasan. On the limits of sparsification. 2011. http://eccc.hpi-web.de/report/2011/131/.

[Sta11] Richard P. Stanley. Enumerative combinatorics volume 1 second edition. 2011. http://www-math.mit.edu/ rstan/ec/ec1/.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[SZ99] Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *FOCS*, pages 605–615, 1999.

[Ter99] Audrey Terras. *Fourier analysis on finite groups and applications*, volume 43. Cambridge Univ Pr, 1999.

[Tra08] Patrick Traxler. The time complexity of constraint satisfaction. In *Proceedings of the 3rd International Workshop on Parameterized and Exact Computation, IWPEC 2008*, pages 190–201. Springer, 2008.

[Tra10] Patrick Traxler. *Exponential Time Complexity of SAT and Related Problems*. PhD thesis, Swiss Federal Institute of Technology Zurich, 2010.

[TSB05] 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.

[TUK95] Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Mixed searching and proper-path-width. *Theoretical Computer Science*, 137(2):253–268, 1995.

[Tut47] William T. Tutte. The factorization of linear graphs. *J. London Math. Soc.*, 22:107–111, 1947.

[Val77] Leslie Valiant. Graph-theoretic arguments in low level complexity. In *Proceedings of the 6th Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 53*, 1977.

[von51] John von Neumann. Various techniques used in connection with random digits. In A. S. Householder et al., editor, *The Monte Carlo Method*, volume 12, pages 36–38. National Bureau of Standards, Applied Mathematics Series, 1951.

[vR11] Johan M. M. van Rooij. *Exact Exponential-Time Algorithms for Domination Problems in Graphs*. PhD thesis, Utrecht University, June 2011.

[vRB11] Johan M.M. van Rooij and Hans L. Bodlaender. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17):2147 – 2164, 2011.

[vRBR09] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Proc. of ESA 2009*, volume 5757 of *LNCS*, pages 566–577, 2009.

[vRNvD09] Johan M. M. van Rooij, Jesper Nederlof, and Thomas C. van Dijk. Inclusion/exclusion meets measure and conquer. In *ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 554–565, 2009.

[Wag86] Klaus W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23:325–356, 1986. 10.1007/BF00289117.

[Wah08] Magnus Wahlström. A tighter bound for counting max-weight solutions to 2sat instances. In Martin Grohe and Rolf Niedermeier, editors, *IWPEC*, volume 5018 of *Lecture Notes in Computer Science*, pages 202–213. Springer, 2008.

[Wig09] Avi Wigderson. Knowledge, creativity, and $\mathcal{P}$ versus $\mathcal{NP}$. 2009. www.math.ias.edu/ avi/PUBLICATIONS/MYPAPERS/AW09/AW09.pdf.

[Wil09] Ryan Williams. Finding paths of length k in $\mathcal{O}^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.

[Wil10] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. In Leonard J. Schulman, editor, *STOC*, pages 231–240. ACM, 2010.

[Wil11] Ryan Williams. Non-uniform acc circuit lower bounds. In *IEEE Conference on Computational Complexity*, pages 115–125. IEEE Computer Society, 2011.

[Woe01a] Gerhard J. Woeginger. Exact algorithms for np-hard problems: A survey. In Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi, editors, *Combinatorial Optimization*, volume 2570 of *Lecture Notes in Computer Science*, pages 185–208. Springer, 2001.

[Woe01b] Gerhard J. Woeginger. When does a dynamic programming formulation guarantee the existence of an fptas? *Electronic Colloquium on Computational Complexity (ECCC)*, (084), 2001.

[Woe04] Gerhard J. Woeginger. Space and time complexity of exact algorithms: Some open problems (invited talk). In *IWPEC*, pages 281–290, 2004.

[Woe08] Gerhard J. Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics*, 156(3):397–405, 2008.

[WW10] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *FOCS*, pages 645–654. IEEE Computer Society, 2010.

[WWY10] Virginia Vassilevska Williams, Ryan Williams, and Raphael Yuster. Finding heaviest -subgraphs in real weighted graphs, with applications. *ACM Transactions on Algorithms*, 6(3), 2010.

[WY92] Gerhard J. Woeginger and Zhongliang Yu. On the equal-subset-sum problem. *Inf. Process. Lett.*, 42:299–302, July 1992.

[Yat37] Frank Yates. The design and analysis of factorial experiments. 1937.

[YZ05] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Transactions on Algorithms*, 1(1):2–13, 2005.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proc. of EUROSAM 1979*, volume 72 of *LNCS*, pages 216–226, 1979.

Acknowledgements

First and foremost I want to thank my PhD advisor, Pinar Heggernes. What I really appreciated was that she always made me feel like an equal even though she was my boss. For example, instead of fixing some subjects to work on, she not only let me free to choose my own quite different research directions but even enthusiastically encouraged me and gave many thorough comments.

Besides that, whenever I asked for it, she also gave me as much support as possible during many non-scientific activities such as, to name just a few, dealing with tax offices, finding appartments and losing backpacks with passports and laptops. Pinar, thanks a lot!!

In advance, and independently of the sneakiness of the questions to be asked at the defense, I already want to thank my evaluation committee (consisting of Peter Rossmanith, Fabrizio Grandoni and Marc Bezem). Thanks also for being willing to review this work, especially in the relatively short period and since many of the results are very recent!

I also want to thank all my coauthors, especially Marek Cygan, Petteri Kaski, Mikko Koivisto, Marcin Pilipczuk, Michał Pilipczuk, Saket Saurabh and Jakub Onufry Wojtaszczyk for hosting me at their respective institutes.

I also want to thank my (ex-)colleagues/fellow students, both from UiB and UU. A special thanks goes to Rémy Belmonte, Pim van 't Hof, Erik-Jan van Leeuwen and Ruben van der Zwaan for being willing to listen to early explanations of some parts of this thesis.

I would also like to thank the Norwegian Research Council for funding this thesis and the many trips during the last three years. I'm sorry I forgot to thank you in some published papers, I hope it's ok.

Last but not least, I thank my family and friends. If you are in the Netherlands: I'll see you soon! If you're elsewhere, I hope to see you soon!

Jesper Nederlof,
November 2011.