# Why are certain polynomials hard?

A look at non-commutative, parameterized and homomorphism polynomials

**Christian Engels**

**Dissertation**

zur Erlangung des Grades des
Doktors der Naturwissenschaften

an der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

September, 2015

|  |  |
|---|---|
| Tag des Kolloquiums: | Wednesday 27th January, 2016 |
| Dekan: | Prof. Dr. Markus Bläser |
| Vorsitzender | Prof. Dr. Holger Hermanns<br>Universität des Saarlandes, Saarbrücken |
| Gutachter : | Prof. Dr. Markus Bläser<br>Universität des Saarlandes, Saarbrücken |
|  | Prof. Dr. Raghavendra Rao B. V.<br>Indian Institute of Technology Madras, Chennai |
| Akademischer Mitarbeiter: | Dr. Karteek Sreenivasaiah |

# Abstract

In this thesis we will try to answer the question why specific polynomials have no small suspected arithmetic circuits. We will look at this general problem in three different ways.

First, we study non-commutative computation. Here we show matching upper and lower bounds for the non-commutative permanent for various restricted graph classes. Our main result gives algebraic branching program upper and lower bounds for graphs with connected component size 6 as well as a #P hardness result. We introduce a measure that characterizes this complexity on these instances.

Secondly, we introduce a new framework for arithmetic circuits, similar to fixed parameter tractability in the boolean setting. This framework shows that specific polynomials based on graph problems have the expected complexity as in the counting FPT case. We introduce classes $\mathsf{BVW}[t]$ which are close to the boolean setting but hardly use the power of arithmetic circuits. We then introduce $\mathsf{VW}[t]$ with modified problems to remedy this situation.

Thirdly, we study polynomials defined by graph homomorphisms and show various dichotomy theorems. This shows that even restrictions on the graphs can already give us hard instances.

Finally, we stray from our main continuous thread and handle simple heuristics for metric graphs. Instead of studying specific metrics we look at a randomized process giving us shortest path metric instances.

# Zusammenfassung

In dieser Thesis versuchen wir die Frage zu beantworten, warum allgemein vermutet wird dass bestimmte Polynome keine kleinen arithmetischen Schaltkreise haben. Wir untersuchen dieses Problem in drei verschiedene Richtungen.

Erstens, untersuchen wir nicht kommutative Berechnungen. Wir zeigen hier, obere und untere Schranken für die nicht kommutative Permanente auf verschiedenen eingeschränkten Graphen. Unser Hauptresultat zeigt Schranken für Graphen mit Komponenten der Größe höchstens 6 sowie ein #P-härte Resultat. Wir führen ein Maß ein, dass die Komplexität solcher Instanzen charakterisiert.

Zweitens, führen wir ein neue theoretischen Rahmen für arithmetische Schaltkreise ein, ähnlich der parametrisierten Komplexität. Dieses zeigt das bestimmte Polynome die auf Graphproblemen basieren, die erwartete Komplexität haben. Die von uns eingeführten Klassen sind ähnlich zu der boolschen Situation und benutzen die Kraft der arithmetischen Schaltkreise kaum. Um dies zu beheben führen wir Klassen $\mathsf{VW}[t]$ ein, die modifizierte Probleme beinhalten.

Drittens, untersuchen wir Polynome definiert durch Graphhomomorphismen und beweisen Dichotomie Theoreme die zeigen, dass selbst Restriktionen auf den Graphen uns harte Instanzen geben.

Zum Schluss weichen wir von unserem roten Faden ab und untersuchen einfache Heuristiken auf metrischen Graphen. Anstatt eine bestimmte Metrik zu untersuchen, schauen wir uns einen randomisierten Prozess an der uns eine kürzeste Pfade Metrik erzeugt.

# Acknowledgements

In no particular order I want to thanks the following people.

First I want to thank my family because without their emotional and financial support I would be a very different person. I also want to thank some brilliant minds. Markus Bläser, my advisor who was always open to listen to my half cooked ideas and ramblings. Rahgavendra Rao who brought me many interesting problems as well as give me confidence to tackle these problems together with him. Karteek Sreenivasaiah who after subsequent visits turned into a friend and a good colleague. And finally Radu Curticapean with whom I shared an office space. While he gave me many headaches he was still a good office mate with many discussions about various fixed parameter tractable algorithms, immanants and many more topics. I do not want to forget to mention all the friends I made while studying in Saarbrücken who for their sake shall stay anonymous. They gave me the strength to follow the path I wanted to follow. And they were always there if i needed to vent about various topics. I also want to mention the various colleagues at the chair with whom I had many interesting discussion either about theoretical computer science or the world in general.

Finally, I want to thank Jessica Schulze who not only listened to my rants all the time but who also was always there with good advice, I declined to follow, as well as show me that if life gives you lemons you should make life take the lemons back. And sue life.

*Nothing can happen till you swing the bat.*

# Contents

# 1 Introduction

With the introduction of computer science and the general study of boolean functions and their complexity, one of the most studied question is “Which problems are hard to compute?” This is evident in the classification of different problems into complexity classes like the time hierarchy theorems or the famous $\mathsf{P}$ vs $\mathsf{NP}$ distinction. To solve the question whether $\mathsf{P} = \mathsf{NP}$ or in general if specific classes fall together we study upper and lower bounds for specific complete problems. Both, finding upper and lower bounds, is highly linked with the following intuitive question: “Why are certain problems hard to compute?”

Despite this question seldom being stated directly they are at the heart of most discoveries in theoretical computer science. In algorithms design, we ask why a problem should be easy to solve and these answers then give us insight in how to construct the algorithms. Of course, more often than not we need multiple iterations to have a final answer. A noteworthy example is the case for primality testing. One of the big result in the last century was by Agrawal, Kayal, and Saxena [AKS04] that testing for primality is in $\mathsf{P}$. Previously, the wisdom was that there will not be a non-randomized deterministic polynomial time algorithm without relying on major mathematical assumptions. However, with the invention of new techniques such as by Agrawal and Biswas [AB03], the complexity and hence the fundamental question what makes this problem hard could be reevaluated. This is such a well known pattern in algorithms design such that if we find a polynomial time algorithm, even with a running time of e.g. $n^{100}$, we will often find an algorithm with a running time given by a small degree polynomial after some additional studying of the problem.

In the view of upper bounds, we can also see that sometimes “obvious” hard problems might not be hard at all. A good example for this is the determinant. After all, the determinant is an exponentially large sum over all cycle covers of a graph where every cycle cover is weighted by 1 or $-1$. Surely, computing the determinant must be hard. But as it turns out, it is widely known that computing the determinant is easy. We can even chose from multiple algorithms such as gaussian elimination or the combinatorial algorithm from Mahajan and Vinay [MV99]. As it turns out the weights actually give us the power to compute the determinant in polynomial time while a version where the cycle covers are not weighted with 1 and $-1$, called the permanent, is hard to compute[1]. The gap between the determinant and permanent is assumed to be exponential, as the best known algorithm for the permanent has a running time of $O(2^n n^2)$. The difference of complexity between the determinant and

[1] At least that is the general suspected complexity of the permanent. As always, proving unconditional lower bounds is difficult.

the permanent is simply astonishing. Asking the question why the permanent is hard is a central question.

Even fields such as average case analysis use the given meta question. But instead of asking in general what makes certain problems hard to compute they ask what makes specific instances for a given problem hard to solve. Average case analysis looks at some distribution on instances and hence asks the fundamental question if the hardness of the known computational expensive instances are widespread or if they are only a few of them. This can even be seen in newly developed theory such as Smoothed Analysis, developed by Spielman and Teng [ST04]. Here they look explicitly onto hard instances but permute them slightly with a gaussian distribution. In essence, they ask the question if the hard instances have to be constructed very precisely, by sharp "peaks" if you will, or if they are more fundamental throughout the problem instances. Of course, both these reasons for hardness are in some sense fundamental on the problem we study but the first case seems much more important than the second case.

In the field of finding lower bounds, we, of course, ask the question more directly. Starting from beautiful undergraduate courses lower bounds such as the $n \log n$ bound for the number of comparisons for any sorting algorithms on an arbitrary set of elements, up to the recent separation by Williams [Wil14] of non-deterministic exponential time versus constant depth polynomial size circuits with modulo gates. All these proofs were developed with this question in mind. What makes these problems hard and how can we exploit this to actually prove this hardness.

A good example are various lower bounds based on shifted partial derivatives [Kay12, GKKS14, KSS14, FLMS14]. We will not introduce the method in this thesis but state later some important results achieved by it in Chapter 2. The basic idea of any lower bound proof using a specific measure can be summed up as follows. We prove that specific circuits of size $s$ can only construct polynomials with a measure related to $s$. Then if we want to prove a lower bound for a polynomial, we just have to show that this polynomial has large measure and hence $s$ has to be large for the circuit computing our target polynomial. Of course, this needs fundamental understanding of the circuits and the target polynomial. To find lower bounds, we need to cleverly chose a polynomial that will be hard to compute with the structure of the circuit and we need to be able to prove this with the help of our measure. We see again that our question is important here.

Of course, there are other lower bounds not based on a specific measure, mostly related to specific subclasses of circuits [JS82, RY09, LMS15, Raz09]. Here we restrict the class of circuits we look at, e.g. multilinear circuits, to try to find a lower bound using the specific property. Again our question is related in a slightly different formulation "What makes these restricted circuits weak for a specific polynomial?"

We see that the question of why certain things are hard is at the core of upper and lower bounds. We will try to answer this question partially for the case of arithmetic circuits and hence the question transforms to "Why are certain polynomials hard?" Arithmetic circuits complexity tries to distinguish the complexity of polynomials and answer the question $\mathsf{VP} =? \mathsf{VNP}$ for the main classes in this field given by Valiant

[Val79a]. However, an extensive amount of research was already done dealing with lower bounds for arithmetic circuits. We, however, by asking this question directly, can see a continuous thread throughout this thesis and our major motivation for the very diverse topics we will cover in arithmetic circuit complexity.

## 1.1 Outline

We will give a basic introduction to arithmetic circuits in Chapter 2. This will show the basic model and some tools we need. We will also give a short introduction to the immanant in Section 2.6 which is a certain polynomial we study to some extend. To give basic context we also include some important results as well as some of the recent new techniques for proving lower bounds. The following chapters will be self contained if Chapter 2 is understood.

In Chapter 3 we will study certain non-commutative polynomials. The non-commutative setting is especially interesting as certain polynomials which can be easy in the commutative case, can already be hard in the non-commutative case. We will prove several lower bounds and show that under certain orderings of variables some polynomials change their hardness characteristics. Hence, we can answer our continuous thread throughout this thesis with this insight that some non-commutative polynomials have inherit in them the complexity related to the order of the variables. In detail, we show algebraic branching program (ABP) lower bounds for computing the permanent on graphs of connected component size two with an almost matching upper bound. The difference between a large and small ABP here depends on the specific ordering of the vertices. We also show for connected component of size 6 the #P hardness of the non-commutative Cayley permanent which is a permanent with an order of the vertices specified. This bound can be transferred to the Cayley determinant and the Cayley immanant. We will construct a polynomial for which the commutative variant is easy but a certain non-commmutative variant is VNP hard. Finally, we prove upper bounds on some computational problems on non-commutative circuits such as computing coefficients of a given monomial.

In Chapter 4 we develop a version of Fixed Parameter Tractability, a notion which essentially says that specific problems can be easy to compute if a certain parameter is small. We will give a small recap of parameterized complexity theory in Section 4.2. We transfer this to an arithmetic circuit setting in two different ways. The first way (Section 4.5) is heavily based on the fixed parameter counting complexity and mostly ignores the power of arithmetic circuits. In the second way, we try to remedy this situation and define new hardness classes as well as new complete problems (Section 4.6). We believe that such a theory give us a good way to handle parameterizing arithmetic circuit complexity. With this theory we can hope to find certain parameters that shed light on how the complexity changes with changes in the parameter. We give a first clue about a special polynomial in Section 4.7 which will be related to the immanant. We can see this as starting a study in a more formal framework for fixed parameter tractability for arithmetic circuits which directly corresponds to our question.

In Chapter 5 we give dichotomies for certain polynomials on graph classes. These will show us that even simply to define polynomials can already exhibit a dichotomy. Additionally, if we see this from the viewpoint of studying what makes polynomials hard, our results show that polynomials on graph homomorphisms almost always explode in the complexity. This is related to the fact that homomorphisms are invariant under isomorphisms which we especially make use of. This is especially interesting with the recent result that restricted homomorphisms polynomials can characterize VP. In detail, we study the homomorphisms from a class of graphs to a specific given graph, similar to the original question about deciding if homomorphisms exist by Hell and Nešetřil [HN90]. These homomorphisms are given by polynomials which are the sum over all graphs that are homomorphic to the given graph from our graph class where a graph is represented by the product of its edge variables. For this model we show dichotomies for cycles, cliques, outerplanar graphs, planar graphs and graphs for genus bounded by a constant $k$.

We shift our gaze away from the main theme of arithmetic circuits of the thesis in Chapter 6. We will study certain graph problems in the optimization setting on general metric graphs. We rediscover a framework for studying these graphs, and show multiple result for simple approximation algorithms. On these certain randomized metrics almost all algorithms we study have a constant approximation ratio or better. In our main theme of asking why certain problems are hard, we answer the question wether just having a metric graph makes the problem easy or if there is something more special about specific metrics such as the euclidean metric. We show that nearest neighbour and insertion heuristics for the traveling salesman problem (TSP) have constant approximation ratio even under shortest path metric graphs. In addition, we show that for the $k$-median a trivial algorithm has already a $1 + \epsilon$ approximation ratio and a bound on the expected running time of the 2-Opt algorithm.

## 1.2 Preliminaries

We generally keep relevant definitions inside their respective chapters but we want to state some general definitions now.

We start with general definitions on graphs. A graph is a set of vertices $V$ and a set of edges $E \subseteq V \times V$. We denote a graph by $G = (V, E)$. The edges can either be directed which we denote by $(u, v)$ or undirected which we denote by $\{u, v\}$. For a given graph $G = (V, E)$ we define $V(G) = V$ the vertex set and $E(G) = E$ the edge set. We will also denote by $V(S)$ where $S \subseteq E$ a set of edges, the vertices given by the set $\{u, v \mid (u, v) \in S \text{ or } \{u, v\} \in S\}$, all vertices which occur in the edges. We will denote by $N(v)$ the *neighbourhood of* $v$ for a given graph, meaning the set $\{u \mid (u, v) \in E \text{ or } \{u, v\} \in E\}$.

We will sometimes use the special graphs denoted by $K_n$, the complete graph, without self-loops, on $n$ vertices and $K_{n,m}$ the complete bipartite graph with one partition having $n$ vertices and another partition having $m$ vertices, again without self-loops.

We call a function $f$ polynomially bounded if there exists a polynomial $p$ such that for all $n$, $f(n) \leq p(n)$. We define the shorthand notation $\mathsf{poly}(n)$ to be the set of all polynomially bounded functions $\mathbb{N} \to \mathbb{N}$.

We denote by $\mathcal{P}(S)$ the powerset of $S$. We will denote by $\mathbb{N}$ the set of natural numbers (starting from one). We denote by $[n]$ the set $\{1, \dots, n\} \subseteq \mathbb{N}$. Let $S$ be a set. We call a mapping $f : S^n \to S^m$ for $m < n$ a projection if it is idempotent, meaning $f \circ f_{|S^m} = f$ where $f_{|S^m}$ is the function that threats $S^m$ as the subspace of $S^n$. Intuitively, a projection removes some elements from the set $S^n$ and maps them remaining elements to the same elements in $S^m$.

As we will sometimes refer to the class $\#\mathsf{P}$, which is defined on counting problems, we will introduce the class here. A counting problem is given by a function $\Sigma^* \to \mathbb{N}$.

**Definition 1.1** ([AB09])**.** *We call $\#\mathsf{P}$ the class of all functions $f : \Sigma^* \to \mathbb{N}$ such that there exists a polynomial time decidable relation $R \subseteq \Sigma^* \times \Sigma^*$ such that*

$$f(x) = |\{y \mid (x, y) \in R\}|.$$

The class itself contains the major interesting hard counting problems in this world such as counting the number of perfect matchings. Additionally, it is known that $\#\mathsf{P}$ under polynomial time oracle reductions already contains the whole polynomial hierarchy.

We can also talk about the witness function for a given problem that is given by $f : \Sigma^* \to \mathcal{P}(S)$ and is defined by

$$f(x) = \{y \mid (x, y) \in R\}$$

for a computable relation $R \subseteq \Sigma^* \times \Sigma^*$. With this we can see $\#\mathsf{P}$ as counting the size of the image of the witness function for a given input $x$.

To complete our definition for the counting class, we mention the three kinds of reduction used for counting problems.

**Definition 1.2** ([AB09])**.** *Let $f, g : \Sigma^* \to \mathbb{N}$ be two functions.*

- *We call $(s, t)$ where $s : \Sigma^* \to \Sigma^*$ and $t : \mathbb{N} \to \mathbb{N}$ a polynomial time many-one reduction from $f$ to $g$ if*
  $$f(x) = t(g(s(x)))$$
  *for all $x \in \Sigma^*$.*
- *We call $s : \Sigma^* \to \Sigma^*$ a parsimonious reduction from $f$ to $g$ if*
  $$f(x) = g(s(x))$$
  *for all $x \in \Sigma^*$.*
- *We call $M$ a polynomial time Turing reduction if $M$ with oracle access to $g$ computes $f$ and $M$ has a running time bounded by a polynomial.*

We will sometimes use the notation $C/\mathsf{poly}$ if $C$ is a given complexity class defined on Turing machines.

**Definition 1.3.** *We denote by $C/\mathsf{poly}$ the class of all languages $L$ where $L$ is given as follows: There exists a Turing machine $M \in C$ which in addition to normal operations can read an extra tape, called an advice tape. For any $L \in C/\mathsf{poly}$. The advice tape contains for any $n = |x|$, the input size, a unique string.*

With this we can define the class $\mathsf{P}/\mathsf{poly}$ which is equivalent to the class given by all families of polynomial sized circuits that decide a given language.

We will also sometimes refer to other well known complexity classes which are defined in various textbooks such as [AB09].

For the case of randomized algorithms, we need some standard notation which we clarify here. We will use the symbols $\mathbb{P}(X)$ for the probability that event $X$ occurs. We will use $\mathbb{E}(X)$ to mean the expected value of a random variable $X$.

# 2 Arithmetic Circuits and Complete Problems

The complexity of boolean functions and of decision problems is one of the fundamental research areas in theoretical computer science. For quite a while the hope was that we can find general lower bounds for various problems and for boolean circuit classes. However, the famous $\mathsf{P}$ vs $\mathsf{NP}$ problem is still unsolved and finding general circuit lower bounds seems hard. In fact, research showed fundamental walls for using some techniques to find separations [BGS75, RR97, AW09]. Only recently the separation of non-uniform constant depth polynomial size circuits and non-deterministic exponential time was found [Wil14]. Even such an "obvious" separation needed many decades of research.

Because of this inability to find lower bounds the study of arithmetic circuits was founded. The hope was that the additional structure of polynomials as well as sophisticated techniques from algebra would enable us to prove lower bounds in this model. Additionally, these lower bounds might transfer to the boolean setting or give us new insights into the original problems. Combining this reasoning with the strong result by Kabanets and Impagliazzo [KI04] that proving lower bounds for boolean circuits is essentially equivalent to showing that derandomizing polynomial identity testing is possible, shows how useful the arithmetic circuit model can be. The connection between $\mathsf{NP}$ and the algebraic classes was recently reinvigorated by Mulmuley and Sohoni [MS01] and subsequent papers where they started the study of Geometric Complexity Theory, the newest approach to settle the question of $\mathsf{P}$ vs $\mathsf{NP}$.

We give an introduction to this topic and the curious reader can dive deeper into the field with the textbook by Bürgisser [Bür00a] or the excellent survey by Shpilka and Yehudayoff [SY10].

## 2.1 Introduction to Arithmetic Circuits

Let $\mathcal{R}$ be a ring. For a monomial in multiple variables we define the degree to be the sum of the degrees of all variables. For a polynomial $p \in \mathcal{R}[x_1, \ldots, x_n]$ we denote by $\deg(p)$ the degree of $p$, namely the maximum of the degree of the monomials. This is also called the total degree of the polynomial.

The basic computational structure in this field is an arithmetic circuit.

**Definition 2.1.** *An* arithmetic circuit *over a ring $\mathcal{R}$ in $n$ variables $x_1, \ldots, x_n$ is a directed acyclic connected graph with vertices, called gates, of the following type:*

- *Vertices of in-degree two, labeled by $*$ and $+$.*

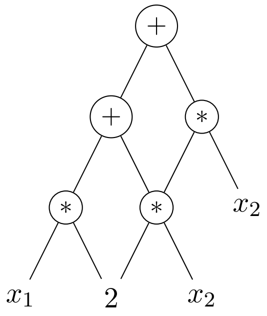


Figure 2.1: A possible circuit for the polynomial $2x_2^2 + 2x_1 + 2x_2$.

- *Vertices with in-degree zero labeled by constants from $\mathcal{R}$ or variables from $\{x_1, \dots, x_n\}$, called input gates.*
- *A unique vertex with out-degree zero and in-degree two, called the output gate.*

We can iteratively define the computation model. The polynomial computed at an input gate is the variable or constant labeled with it. Let $g$ be a gate with label $\circ$ and let the children of $g$ compute the polynomials $p_\ell$ and $p_r$. Then the polynomial computed at gate $g$ is $p_\ell \circ p_r$ where $\circ \in \{+, *\}$. We call the polynomial computed at the output gate the polynomial computed by the circuit.

The given definition is very similar to boolean circuits but instead of boolean primitives being used at a gate, we compute with the primitive operations on the ring, namely multiplication and addition. We give an example of an arithmetic circuit in Figure 2.1.

This definition can easily be extended to gates of in-degree more than two where we will talk about *unbounded fan-in* circuits. We denote by the fan-in of a gate the in-degree of the vertex. We call the fan-in of the circuit the maximum fan-in over all gates in the circuit. In the literature the form of circuits will generally be described by the layers they have. A $\sum\prod\sum$ circuit, for example, is a circuit of depth 3 where every gate has unbounded fan-in with the top gate being a summation gate, followed by a product gate, followed by a summation gate. If we have a circuit of fan-in more than two but still want to restrict the fan-in to $k$ we will denote this by $\prod^k$ or $\sum^k$. If we do not mention any bound, we will assume circuits of fan-in two.

We can now setup our computation model.

**Definition 2.2.** *We call an infinite family of polynomials, written $(f_n)$, a $p$-*family *if for all $n$, $f_n \in \mathcal{R}[x_1, \dots, x_{q'(n)}]$ and $\deg(f_n) \leq q(n)$ for some polynomially bounded function $q$ and $q'$. Additionally, $f_n$ can be computed by a circuit $C_n$ for all $n$.*

We will slightly abuse notation by writing $(f_n) \in \mathcal{R}[x_1, \dots, x_{q(n)}]$ to specify that for all $n$, $f_n \in \mathcal{R}[x_1, \dots, x_{q(n)}]$ where $q(n)$ is clear from the context. The reader should notice that our model is non-uniform.

Now we can define a complexity measure on such circuits. The obvious choice is the circuit size.

**Definition 2.3.** *Let $f$ be a polynomial. Then $L(f)$ is the minimum number of gates of all arithmetic circuit computing $f$. We define $\mathsf{size}(C)$ to be the number of gates of the circuit $C$.*

We can extend this definition to $p$-families such that $L((f_n))$ is now a function $\mathbb{N} \to \mathbb{N}$, mapping $n$ to $L(f_n)$.

The complexity of polynomials can be very different depending on the ring they are defined in. One of the most well known restriction is to fields of characteristic not equal to two. We will see why such a restriction is useful in a later section. In this thesis, we also want to distinguish between the polynomial ring being commutative, denoted by $\mathcal{R}[x_1, \dots, x_n]$, or non-commutative, denoted by $\mathcal{R}\langle x_1, \dots, x_n \rangle$. We will deal in Chapter 3 extensively with the non-commutative case. In all other chapters we will assume $\mathcal{R}$ to be commutative unless specified. Additionally, we keep the dependency on the ring implicit when defining arithmetic circuit classes. Notice that the previous definitions can be transferred to the non-commutative case.

Now we can introduce the analogues to boolean complexity classes $\mathsf{P}$ and $\mathsf{NP}$, first introduced by Valiant [Val79a].

**Definition 2.4.** *A p-family $(f_n)$ of polynomials is in $\mathsf{VP}$ if $L((f_n))$ is a polynomially bounded function.*

This gives us the analogue to boolean complexity class $\mathsf{P}$ for polynomials but can we define an analogue to $\mathsf{NP}$? Indeed we can, with the following, slightly surprising definition.

**Definition 2.5.** *Let $q(n)$, $s(n)$ be polynomially bounded functions. We say a p-family $(f_n) \in \mathcal{R}[x_1, \dots, x_{q(n)}]$ is in $\mathsf{VNP}$ if the following condition holds. There exists a p-family $(g_n) \in \mathcal{R}[x_1, \dots, x_{q(n)}, y_1, \dots, y_{s(n)}]$ in $\mathsf{VP}$ such that*

$$f(x_1, \dots, x_{q(n)}) = \sum_{e \in \{0,1\}^{s(n)}} g(x_1, \dots, x_{q(n)}, e_1, \dots, e_{s(n)}).$$

As a quick glance reveals, this corresponds more to the class $\#\mathsf{P}$ than to the class $\mathsf{NP}$. This connection goes even deeper as we will see in the next theorem.

We will sometimes use the operator semantic as in the boolean world, meaning that we define $\sum \cdot \mathcal{C}$ for an arithmetic circuit class $\mathcal{C}$ to be the set of all polynomial families $(f_n) \in \mathcal{R}[x_1, \dots, x_n]$ such that there exists polynomially bounded functions $r(n)$, $s(n)$ and a polynomial family $(g_n) \in \mathcal{C}$ such that for all $n$

$$f_n = \sum_{e \in \{0,1\}^{s(n)}} g_{r(n)+s(n)}(x_1, \dots, x_{r(n)}, e).$$

In this notation $\mathsf{VNP} = \sum \cdot \mathsf{VP}$. We can easily extend this notation to the cases where $(f_n) \in \mathcal{R}[x_1, \dots, x_{q(n)}]$.

Let us first take a look at the membership of $p$-families in $\mathsf{VNP}$. How can we prove membership in $\mathsf{VNP}$? After all the definition is rather cumbersome to use. Thankfully, there is a nice theorem by Valiant which shows another connection with $\#\mathsf{P}$.

**Theorem 2.1.1** (Valiant's Criterion)**.** *Suppose $\phi : \{0,1\}^* \to \mathbb{N}$ is a function in the class* $\#\mathsf{P}/\mathsf{poly}$*. Then the family $(f_n)$ of polynomials defined for all $n$ by*

$$f_n = \sum_{e \in \{0,1\}^n} \phi(e) x_1^{e_1} \dots x_n^{e_n}$$

*is in* $\mathsf{VNP}$*.*

We can now introduce the central notion of a reduction used in arithmetic circuit complexity. While it looks very restrictive it is still a powerful form of reduction.

**Definition 2.6.** *A polynomial family $(f_n) \in \mathcal{R}[x_1, \dots, x_{q(n)}]$ is a $p$-*projection *of a polynomial family $(g_{n'}) \in \mathcal{R}[x_1, \dots, x_{q(n')}]$ and denoted by $(f_n) \leq_p (g_{n'})$ if the following holds:*

*$f(x_1, \dots, x_{q(n)}) = g(a_1, \dots, a_{q(n')})$ where $a_i \in \mathcal{R} \cup \{x_1, \dots, x_{q(n)}\}$ and $n' \leq q(n)$ for some polynomially bounded function $q$.*

Many completeness results can be obtained even with this restricted type of reduction.

In the standard literature the notation $\leq$ is normally used but in this thesis we use the explicit notation given in the definition to avoid confusion with the second type of reduction, called *c-reduction.*

Let $(f_n) \in \mathcal{R}[x_1, \dots, x_{q(n)}], (g_{n'}) \in \mathcal{R}[x_1, \dots, x_{q'(n')}]$ be polynomial families. Let $L^{g_{n'}}(f)$ be the minimal number of gates for computing $f$ where the circuit is enhanced with oracle gates that on input $a_1, \dots, a_{q'(n'_i)}$ compute $g(a_1, \dots, a_{q'(n'_i)})$ for some natural number $n'_i \leq n'$.

**Definition 2.7.** *Let $(f_n)$, $(g_{n'})$ be as in the previous definition. We say $f$ c-reduces to $g$, written $(f_n) \leq_c (g_n)$, if there exists a polynomial $q$ such that $L^{g_{q(n)}}(f)$ is bounded by some polynomial.*

Readers familiar with Turing reductions in the boolean world will notice the similarity. As in the boolean world, they carry some problems if we define complexity classes based on them. For example, the class complete for the zero polynomial is equal to $\mathsf{VP}$ which is not quite what we expect. Hence, we will only use $c$-reductions for larger classes than $\mathsf{VP}$. We will discuss some complete problems for $\mathsf{VNP}$ in Section 2.3.

We can also define these reductions on non-commutative algebras in a similar way.

Another interesting question is how these classes relate to the boolean complexity. As these were invented to produce lower bounds, can we perhaps transfer such bounds easily? Sadly, the relation to the boolean complexity classes is not trivial. It can be shown that proving $\mathsf{P}/\mathsf{poly} \neq \mathsf{NP}/\mathsf{poly}$ implies that $\mathsf{VP} \neq \mathsf{VNP}$ but this is the only implication we know of. We will discuss the classes $\mathsf{VP}$ and $\mathsf{VNP}$ a bit more in Section 2.3. However, there exists an interesting connection with computing integers (starting from 1 and only using addition and multiplication) with related classes to $\mathsf{VP}$ and $\mathsf{VNP}$ called $\mathsf{VP}_0$ and $\mathrm{VNP}_0$. The classes can be roughly seen as their analogue $\mathsf{VP}$

and $\mathsf{VNP}$ but without being permitted to use arbitrary elements from $\mathcal{R}$. Instead we are only allowed to use 1 and $-1$ and need to compute all other constants in the circuit itself. This model was studied by Koiran [Koi05] and he showed that if computing specific numbers is hard then a separation between $\mathsf{VP}_0$ and $\mathrm{VNP}_0$ is proven. How this separation transfers to the $\mathsf{VP}$ vs $\mathsf{VNP}$ question is, however, unclear.

For completeness sake we will mention Valiant's Conjecture.

**Conjecture 2.1.** *Let $\mathcal{R}[X]$ be a commutative polynomial ring. Then* $\mathsf{VP} \neq \mathsf{VNP}$ *over this ring.*

## 2.2 Restricted Circuit Models

In the previous section we only introduced the general circuit model but there are many different restricted models known. As we will discuss and use some of these in later sections, we give an introduction here, starting with the most restricted model.

**Definition 2.8.** *We call an arithmetic circuit an* arithmetic formula *if the underlying graph of the circuit is a tree. We call the set of all p-families computable by a family of polynomial size formulas* $\mathsf{VF}$.

It is known that $\sum \cdot \mathsf{VF}$ is already equal to $\mathsf{VNP}$. This will be a part of the motivation why we study formulas instead of arithmetic circuits in Chapter 4.

Next we can define a seemingly completely different model which actually turns out to be a restriction of general circuits and a generalization of arithmetic formulas.

**Definition 2.9.** *An* algebraic branching program *(ABP) is a directed acyclic weighted graph with two special nodes $s$, $t$ and edges with weights given by variables or constants in $\mathcal{R}$. The weight of a path is the product of the weights of its edges. The polynomial computed by an ABP $P$ is the sum of the weights of all $s$ to $t$ paths in $P$ and is denoted by $p_P$.*

*We call the set of all p-families computable by a family of polynomial sized ABPs* $\mathsf{VBP}$

We will use this model extensively in Chapter 3 because of the easy to analyze structure. First notice that we can unify an ABP by layering it. For this we first ensure that every path through the ABP has the same length by adding edges of weight 1. Then we put every vertex in a layer $i$ for some $i$ and only allow connections from layer $i-1$ to layer $i$ and from layer $i$ to layer $i+1$. This shows that, in essence, an ABP can only save a number of "states" corresponding to the width. This makes ABPs simpler to analyze, especially if the width is restricted to constant size. In fact, constant width ABPs are equivalent to formulas as proven by Ben-Or and Cleve [BC92]. Their proof extends Barrington's ([Bar89]) characterization of $\mathsf{NC}_1$ by boolean bounded-width branching programs. Another extension of Barrington's proof will occur in Section 3.5.

**Definition 2.10.** *We call an arithmetic circuit a* skew circuit *if for every product gate at least one of its inputs is a variable.*

*We denote the class of all p-families computable by a family of polynomial size skew circuits by* $\mathsf{VP}_{skew}$.

Skew circuits can be seen as a simple extension of ABPs but as it turns out they are equally powerful in the commutative setting. Let us give a short intuition for this proof. Assume we already transformed for every vertex $v$ the ABP into the circuit $C_v$. We can then iteratively compute for an edge $v$ to $u$ with input $x$ the circuit $C_v \cdot x$. This is obviously a skew circuit and as we can sum over all incomming edges to $u$ we stay roughly the same size. For the other direction, we can reverse this construction.

An extension of skew circuits is given in the following definition.

**Definition 2.11.** *We call an arithmetic circuit a* weakly skew circuit *if for every product gate at least one child forms a sub circuit that has no other connection to the rest of the circuit.*

*We call the class of all p-families computable by a family of polynomial size weakly skew circuits by* $\mathsf{VP}_{ws}$.

The next theorem proven by Toda [Tod92] given with the simple to prove fact that $\mathsf{VBP} = \mathsf{VP}_{\text{skew}}$ gives us the following theorem

**Theorem 2.2.1** ([Tod92]). *Let* $\mathcal{R}[x_1, \ldots, x_n]$ *be a commutative ring. Then* $\mathsf{VP}_{skew} = \mathsf{VBP} = \mathsf{VP}_{ws}$.

Chapter 3 will extensively deal with the non-commutative case of ABPs and general circuits and Section 3.4.2 will discuss skew and weakly skew circuits in the non-commutative setting.

We can also define a class which we will only use sparingly. It is assumed to be much smaller than $\mathsf{VP}$ and was first mentioned by Mahajan and Rao [MR13] and is defined as its boolean analogue $\mathsf{AC}_0$.

**Definition 2.12.** *Let* $(f_n)$ *be a p-family. Then* $(f_n) \in \mathsf{VAC}_0$ *if there exists a family of constant depth, unbounded fanin and polynomial size arithmetic circuit.*

Now that we learned about new smaller complexity classes and the surprising result that some structural restrictions on the circuit are not restrictions on the power, we can ask ourselves what other restrictions on $\mathsf{VP}$ are possible without decreasing the power. Malod and Portier [MP08] gave a first answer. It is enough to have polynomial size circuits that have for every multiplication gate $g$ two disjoint subcircuits as children of $g$. They called these circuits multiplicatively disjoint. Notice, that on first sight this model is closer to formulas than $\mathsf{VP}$ or $\mathsf{VP}_{\text{skew}}$. However, every weakly skew circuit is also multiplicatively disjoint. Mengel [Men13] found a more restrictive model equal to $\mathsf{VP}$. In this model, called stack branching programs, we can use constant width ABPs but with a stack attached to it. Later $\mathsf{VP}$ was characterized by specific homomorphism polynomials by Durand, Mahajan, et al. [DMMRS14]. Not only does this give us another possibility of reducing VP without reducing it power but it also gives one of the few characterizations of VP in regards to the closure of a natural problem under $p$-projections.

A collection of various arithmetic circuit complexity classes and their restriction was written by Mahajan [Mah13].

## 2.3 The Defining Problems in Arithmetic Circuit Complexity

We can now finally come back to VP and VNP and state the major complete problems. The first problem is the well known permanent.

**Definition 2.13.** *Let $A$ be a matrix with entries $a_{i,j}$ in $\mathcal{R}[x_{1,1}, \ldots, x_{n,n}]$ then the permanent is defined as*

$$\operatorname{perm}_n(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}$$

*where $S_n$ is the symmetric group, the set of all permutations on $n$ elements.*

In fact, the permanent was shown to be complete for matrices with entries in $\{0,1\}$ for the related class #P by Valiant [Val79b]. While the completeness proof is more involved the membership can be easily seen even without Valiant's Criterion. Let $(y_{i,j})$ be $n^2$ new variables which will correspond to the entries of a matrix. Let us now build the polynomial from several pieces. Let $X = \{x_{1,1}, \ldots, x_{n,n}\}$.

$$\alpha_n(X,Y) \coloneqq \left( \prod_{i,j} \prod_{\substack{m=1 \\ m \neq j}}^{n} (1 - y_{i,j} y_{i,m}) \right), \quad \beta_n(X,Y) \coloneqq \left( \prod_{i,j} \prod_{\substack{m=1 \\ m \neq i}} (1 - y_{i,j} y_{m,j}) \right),$$

$$\gamma_n(X,Y) \coloneqq \prod_{i=1}^{n} \sum_{j=1}^{n} y_{i,j}, \qquad \delta_n(X,Y) \coloneqq \prod_{i=1}^{n} \sum_{j=1}^{n} x_{i,j} y_{i,j}.$$

The permanent can then be computed as

$$\operatorname{perm}_n(X) = \sum_{e \in \{0,1\}^{n^2}} \alpha_n(X,e) \beta_n(X,e) \gamma_n(X,e) \delta_n(X,e).$$

This is relatively easy to see. For the permanent we need to enforce a permutation matrix in the entries $(y_{i,j})$, meaning a matrix where every row and every column has exactly a single one. The polynomial $\alpha_n(X,Y)$ enforces that at most a single one is in every column and $\beta_n(X,Y)$ enforces the same property for every row. Then $\gamma_n(X,Y)$ enforces a lower bound of at least a single one for every row. Finally, $\delta_n(X,Y)$ just selects the correct $x_{i,j}$ for every $y_{i,j}$ with value one. In essence, we have a boolean computation on the set $\{y_{1,1}, \ldots, y_{n,n}\}$ to check a property and then select the $x_{i,j}$ corresponding to the selected $y_{i,j}$. We will see similar definitions for new circuit classes in Chapter 4.

We can easily compare the definition of the permanent to the definition of the determinant.

**Definition 2.14.** *Let $A$ be a matrix with entries $a_{i,j}$ in $\mathcal{R}[x_{1,1},\dots,x_{n,n}]$ then the determinant is defined as*

$$\det_n(A) = \sum_{\sigma\in S_n} \operatorname{sgn}(\sigma)\prod_{i=1}^{n} a_{i,\sigma(i)}$$

*where $S_n$ is the symmetric group, the set of all permutations on $n$ elements.*

A surprising belief is that these very similar definitions have vastly different complexity in the commutative setting. While the determinant has a polynomial time algorithm (for example, via Gaussian elimination or the beautiful combinatorial algorithm in [MV99]) the complexity for the permanent has a much larger upper bound. The famous Ryser's formula for the permanent is one of the most efficient algorithms we know with a circuit size of $O(2^n n^2)$. Studying the complexity of the permanent on various restricted inputs is an active field. We will skip these results as we are only interested in the permanent as a polynomial and hence cannot restrict the input.

The reader might suspect that these two problems are the general complete problems for VP and VNP, however this is not the case. While Valiant [Val79a] showed the completeness for the permanent, the determinant is only VP complete for a different reduction, so called *qp*-reductions. They roughly correspond to quasi-polynomial size projections.

**Theorem 2.3.1** ([Val79a])**.** *Let $\mathcal{R}$ be a ring of characteristic not equal to two and $X = \{x_{1,1}, x_{1,2},\dots,x_{n,n}\}$. Then $(\mathrm{perm}_n(X))$ is complete for* VNP *under p-projections.*

The restriction away from characteristic two is obviously needed as the determinant is equal to the permanent on such fields.

VP and VNP were extensively studied in relation to lower bounds with the goal of understanding the arithmetic circuit model and to eventually separate VP from VNP. Research started on finding general lower bounds for various restricted models in the commutative setting. As we have seen already in Theorem 2.2.1 many natural syntactic restrictions on the circuits are not a restriction on the power of arithmetic circuits and hence do not give us any new insight into proving lower bounds for arithmetic circuits. Jerrum and Snir [JS82] and Raz and Yehudayoff [RY11] started the study of lower bounds for restricted circuit classes by giving lower bounds for monotone circuits for the permanent. The next candidate restriction to study were multilinear formulas as both the determinant and permanent are multilinear. However, a multilinear formula requires the circuit to be multilinear at every gate which is a major restriction. The first lower bound for Iterated Matrix Multiplication was given by Nisan and Wigderson [NW97] using the powerful technique of partial derivatives. They showed that any depth 3 circuit can be transformed to a multilinear circuit without much cost and then showed a lower bound for multilinear circuits. In combination they achieved a bound of $\Omega(n^{d-1}/d!)$ for depth 3 circuits computing Iterated Matrix Multiplication. This was improved by Raz [Raz09] to show that permanent and determinant require superpolynomial size multilinear formulas. The technique of partial derivatives was later enhanced by Kayal [Kay12] to prove a lower bound for the polynomial $x_1 \cdots x_n$ for a very restricted class of circuits. He called this technique shifted partial derivatives.

Finally, research culminated in the depth reduction arguments by Agrawal and Vinay [AV08], Koiran [Koi12], and Tavenas [Tav13]. They successively proved that any circuit of size $s$ computing a polynomial of degree $d$ has a homogeneous formula of depth 4 and size $\exp(O(\sqrt{d \log n \cdot \log ds}))$ where the fan-in of the bottom product gate is bounded by $\sqrt{n}$. Plugging a circuit of polynomial size and polynomially degree $d$ into this, we get an upper bound of $\exp(\omega(\sqrt{d} \log n))$ for the equivalent depth four homogeneous formula. Gupta, Kamath, et al. [GKKS14] searched for a matching lower bound of homogeneous depth 4 circuits but they could only manage a bound of $\exp(\omega(\sqrt{d}))$ for the permanent and determinant using the shifted partial derivatives technique from [Kay12]. Additionally, they had to restrict the fan-in of the bottom product gate in a similar fashion. But still, using shifted partial derivatives and a tighter analysis of it, finding lower bounds seemed possible. Later, Kayal, Saha, and Saptharishi [KSS14] gave a matching bound for formula size of $\exp(\Omega(\sqrt{d} \log n))$ for a polynomial in $\mathsf{VNP}$ with a similar fan-in bound on the bottom product gate. Any asymptotic improvement in the exponent of either Tavenas's depth reduction or Kayal, Saha, and Saptharishi's lower bound would show a separation between $\mathsf{VP}$ and $\mathsf{VNP}$ in the following way. Either all polynomials in $\mathsf{VP}$ can be transformed to a formula of size $\exp(o(\sqrt{d} \log n))$ and the polynomial in [KSS14] separates $\mathsf{VP}$ from $\mathsf{VNP}$ or we find a new polynomial in $\mathsf{VNP}$ with an asymptotically higher homogeneous circuit lower bound.

However, this approach seems to be at its end. Recently, Fournier, Limaye, et al. [FLMS14] showed that the Iterated Matrix Multiplication polynomial, which is in $\mathsf{VP}$, has a depth 4 homogeneous formula of size at least $\exp(\Omega(\sqrt{d} \log n))$. Hence this approach cannot disprove that the permanent has polynomial size circuits even if it could be proven that a homogeneous depth 4 circuit computing the permanent has a size of at least $\exp(\Omega(\sqrt{d} \log n))$.

A good introduction to some general lower bounds and the history of shifted partial derivatives can be found in [KS14]. A survey on the power of partial derivatives and shifted partial derivatives was written by Chen, Kayal, and Wigderson [CKW11].

## 2.4 Generating Functions of Graph Properties

We have now a perfect framework to reason about polynomials and their complexity. However, in the boolean world many basic problems are graph based. We are still missing an easy way to transfer or even define polynomials based on graph problems. In fact, we cannot even simulate many of the counting problems in $\#\mathsf{P}$. To remedy this fact, we will define Graph Properties and the Generating Functions in this section. We will later use them in Chapters 4 and 5. We will only deal with the commutative setting in this section but the definitions transfer to the non-commutative case if we take some care about the ordering of variables.

For this section, let $G = (V, E)$ be a graph and let $X = \{x_e \mid e \in E\}$. We label each edge $e$ by the indeterminate $x_e$. We will generally switch freely between having the variables indexed by either edges ($x_e$) or pairs of vertices ($x_{i,j}$ for $i, j \in V$). We let $x_j$ correspond to the self-loop at vertex $j$. While we mainly only use edge weights, we sometimes will also use vertex weights.

**Definition 2.15.** *Let $V_0 \subseteq V_1 \subseteq \dots$ be an infinite ascending chain of set of vertices. We call graphs on these vertices $G_{1,1}, \dots$ where $G_{i,j}$ are all isomorphic copies of $G_{i,1}$ on $V_i$. Then a* graph property *$\mathcal{E}$ is given by $\cup_{i,j} G_{i,j}$.*

Most graph properties used in the literature have very simple natural language statements such as “all cycles of length 5” or “all matching on an infinite family of graphs”. Now we can combine graph properties with generating functions to construct polynomials.

**Definition 2.16.** *Let $X$ be a set of indeterminates. Let $\mathcal{E}$ be a graph property. Let $G = (V, E)$ be an edge weighted, undirected graph with a weight function $w : E \to \mathcal{R} \cup X$. We extend the weight function by $w(E') := \prod_{e \in E'} w(e)$ to subsets $E' \subseteq E$.*

*The* generating function $\mathrm{GF}(G, \mathcal{E})$ *of the property $\mathcal{E}$ is defined as*

$$\mathrm{GF}(G, \mathcal{E}) := \sum_{E' \subseteq E} w(E')$$

*where the sum is over all subsets $E'$ such that the subgraph $(V, E')$ of $G$ has property $\mathcal{E}$.*

We easily generalize this definition and the definition for graph properties for families to produce families of polynomials and will denote this by writing $\mathrm{GF}(G_n, \mathcal{E})$ or $\mathrm{GF}(G_n, \mathcal{E}_n)$. As we from now on deal exclusively with families this will not cause confusion. The parameter $n$ will always be clear from the context. The reader should notice that technically the subgraphs we sum over have all vertices included and only edges missing.

Let us look at a short example. Let $\mathcal{E}$, our graph property, be the set of all graphs where every connected component has two vertices. Then our graph $G$ from Figure 2.2 gives the following generating function.

$$\mathrm{GF}(G, \mathcal{E}) = x_{a,b}x_{c,d} + x_{b,d}x_{c,a} + x_{a,b} + x_{c,d} + x_{b,d} + x_{c,a}.$$

While this may look on the first glance to be a complication these generating functions are actually very useful. They subsume counting the number of graphs fulfilling the graph property by just evaluating every variable with one. We can even use this technique to ask the same question for specific subgraphs by setting specific edge variables to zero.

Readers familiar with the Holographic Framework will see the relationship of generating functions and the framework.

We conclude by stating some VNP complete problems based on generating functions. Proofs of these theorems can be found in the textbook by Bürgisser [Bür00a].

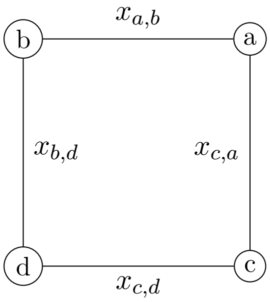


Figure 2.2: Example Graph $G$

**Theorem 2.4.1** ([Bür00a]). (GF($K_n$,$\mathcal{UHC}_n$)) *is* VNP *complete under p-projections where* $\mathcal{UHC}_n$ *is the set of all undirected Hamiltonian cycles in* $K_n$.

**Theorem 2.4.2** ([Bür00a]). *Let* $\mathcal{CL}$ *be the graph property of all cliques. Meaning, the set of all graphs, where one connected component is a complete graph and each of the remaining connected components consist of one vertex only. Then the family* GF($K_n$,$\mathcal{CL}$) *is* VNP *complete under p-projections.*

**Theorem 2.4.3** ([Bür00a]). *Let* $\mathcal{M}$ *be the set of all graphs where all connected components have exactly two vertices. Then the family* GF($K_n$, $\mathcal{M}$) *is* VNP *complete under p-projections.*

This polynomial gives us all perfect matchings in a graph which is equal to GF($K_{n,n}$, $\mathcal{M}$) for bipartite graphs which is a projection of GF($K_{n^2}$, $\mathcal{M}$).

A generalization of $\mathcal{UHC}_n$ is the cycle format polynomial. We can write a partition of $n$ in frequency notation $\rho = (\rho(1), \rho(2), \dots)$ where $\rho(i)$ denotes the number of pieces of size $i$. With this we can write the Cycle Format graph property $\mathcal{CF}_\rho$ which describes all graphs which are the union of $\rho(i)$ many $i$-cycles.

**Theorem 2.4.4** ([Bür00a]). *Let* $\rho_n$ *be a sequence of partitions of* $n$ *such that there exists some* $\epsilon > 0$ *with* $n - \rho_n(1) \geq n^\epsilon$ *for all* $n$. *Then the corresponding sequence* (GF($K_n$ $\mathcal{CF}_\rho$)) *is* VNP *complete under p-projections.*

This polynomial immediately characterized the problems of counting $k$-cycles for any $k$ and many more similar problems.

## 2.5 Tools

We gather some useful tools in this section which we will need throughout the thesis. One of the most useful one is extracting homogeneous components.

**Definition 2.17.** *Let* $\bar{x} = x_{i_1}, \dots, x_{i_l}$ *be a subset of variables and* $(f_n)$ *be a p-family over* $\mathcal{R}[x_1, \dots, x_{q(n)}]$. *We can write* $f_n$ *as*

$$f_n = \sum_{\bar{i}} \alpha_{\bar{i}} \prod_{j=1}^{q(n)} x_j^{i_j}$$

*where $\alpha_{\bar{i}} \in \mathcal{R}$. The homogeneous component of $f_n$ of degree $k$ with variables $\bar{x}$ is*

$$\mathrm{HOMC}_k^{\bar{x}}(f_n) = \sum_{\substack{i_1,\dots,i_{q(n)} \\ k=\sum_{j=1}^{q(n)} i_j}} \alpha_{i_1,\dots,i_{q(n)}} x_1^{i_1} \dots x_{q(n)}^{i_{q(n)}}.$$

We will show in the next lemma how we can extract homogeneous components with oracle reductions. This lemma was stated explicit by Rugy-Altherre [Rug12] and can also be found in [Bür00a]. It will give us a way to extract all polynomials of homogeneous degree $k$ in some set of variables in $c$-reductions.

**Lemma 2.5.1.** *Let $\mathbb{K}$ be a field of characteristic zero. Then for any sequence of integers $(k_n)$ and $\bar{x}_n$ a sequence of variables there exists a c-reduction from the homogeneous component to the polynomial itself:*

$$(\mathrm{HOMC}_{k_n}^{\bar{x}_n}(f_n)) \leq_c (f_n).$$

*The circuit for the reduction has size in $O(q(n)\delta_n)$ where $\delta_n$ is the degree of $f_n$ and $q(n)$ the number of variables $f_n$ has.*

We will give a proof based on [Rug12] for completeness sake as this is a simple interpolation argument.

*Proof.* Let us look at a polynomial $f$ from the family $(f_n)$. With an induction we can easily prove this for all $n$. Let $d = \deg(f)$. We can write

$$f(2^i x) = \sum_{k=0}^{d} (2^i)^k \,\mathrm{HOMC}_k(f).$$

We can rewrite this equation into a matrix form.

$$\begin{pmatrix} f(2^1 x) \\ \vdots \\ f(2^d x) \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 2^1 & 2^2 & \cdots & 2^d \\ \vdots & \vdots & \vdots & \vdots \\ (2^1)^d & (2^2)^d & \cdots & (2^d)^d \end{pmatrix} \begin{pmatrix} \mathrm{HOMC}_1(f) \\ \vdots \\ \mathrm{HOMC}_d(f) \end{pmatrix}.$$

It is easy to see that our matrix is a Vandermonde matrix with entries $(a_{i,j}) = \alpha_i^{j-1}$ where $\alpha_i = 2^i$. As the determinant of a general Vandermonde matrix is equal to $\prod_{1\leq i<j\leq d}(\alpha_j - \alpha_i)$ it is invertible for our choice of $\alpha_i$. Hence, from the values $f(2^1 x), \dots, f(2^d x)$ can be used to compute the vector consisting of the homogeneous components.

The size bounds follow from this computation. □

As seen in the proof this theorem works actually for any $\alpha_1, \dots, \alpha_d$ where $\alpha_j - \alpha_i \neq 0$ for all $i, j$.

The reader should note that using this theorem will blow up our circuit polynomially in size and can hence be used only a constant number of times in succession. However, we can use this lemma on subsets of vertices. We replace every variable $x_i$ in the subset by $x_i y$ for a new variable $y$ and take the homogeneous components of $y$. This then can give us polynomials where the coefficients are in $\mathcal{R}[x_1, \dots, x_{q(n)}]$.

The disadvantage of this algorithm is that we need a field of characteristic zero[1]. If we have access to the whole circuit as a whitebox we could also transform it to extract the homogeneous components. The transformation is straightforward by going bottom to top. Let us split every gate $g$ into $d$ gates $g_1, \dots, g_d$ where $g_i$ computes the homogeneous component of degree $i$ for the polynomial computed at $g$. For an addition gate $g$ with the two children $f$ and $h$ construct gate $g_i = f_i + h_i$ while for a multiplication gate we get the convolution $g_i = \sum_{j=0}^{d} f_j \cdot h_{d-j}$. It is easy to see that this just increases the size of the circuit by a factor of $d$ and as it does not use any constants it works over any field.

## 2.6 The Immanant and a Short Introduction to Characters of the Symmetric Group

As we previous learned, research focused on the determinant and permanent but there is a concept combining these two into one mathematical construct. This is called the immanant. The definition needs some mathematical background knowledge from representation theory and characters of a representation. We will give a brief overview of these concepts. For further information, the reader should consult the excellent textbook by Fulton and Harris [FH91] and the detailed theses by Tessier [Tes13] and Heide-Jørgensen [Hei12] as well as the section in [Bür00a] or the two papers [Bür00b, Bür00c].

**Definition 2.18.** *Let $V$ be a complex vector space and* $\mathrm{GL}(V)$ *the group of the linear automorphisms on $V$ and let $G$ be a group. Then a* representation *of $G$ is a group homomorphism $\rho : G \to \mathrm{GL}(V)$.*

**Definition 2.19.** *Let $\rho$ a representation of a group $G$ on $V$ where $V$ is a vector space over $\mathbb{K}$. The* character *of $\rho$ is given by the function $\chi_\rho : G \to \mathbb{K}$ such that*

$$\chi_\rho(g) = \mathrm{Tr}(\rho(g))$$

*where* Tr *is the trace.*

Some basic facts about some characters are gathered in the next proposition. We can use

**Proposition 2.1** ([FH91])**.** *Let $\rho$, $\rho'$ be a representation of $S_n$ and $\sigma$, $\pi$ elements of $S_n$. Then*

[1] We could modify the proof to work over any field with $\max_n k$ elements if this is constant.

1. $\chi_\rho(\pi) = \chi_\rho(\sigma^{-1}\pi\sigma)$.
2. $\chi_{\rho\otimes\rho'}(\pi) = \chi_\rho(\pi) + \chi_{\rho'}(\pi)$.

While the definition of the character is very general it is hard to work with in a combinatorial way. But as it turns out we can just reason about partitions of $[n]$ as we restrict ourselves to characters of the group $S_n$. For this we need to introduce a few ways to speak about partitions.

**Definition 2.20.** *A Young diagram is a set of $n$ boxes arranged in left-justified rows where the length of the rows is weakly decreasing from top to bottom.*

A Young diagram with $n$ boxes gives a set of partitions of $n$ elements where the sets are of size $\lambda_1, \dots, \lambda_m$ with $\lambda_i$ being the length of the $i$th row. As previously mentioned, this is also sometimes called the partition in frequency notation. While slightly missleading the literature generally calls the set $\lambda = (\lambda_1, \dots, \lambda_m)$ the partition and identifies partitions with Young diagrams with the obvious injection. For clarity we break with this tradition and call $\lambda = (\lambda_1, \dots, \lambda_n)$ the *type* of the partition. We will denote this by $\lambda \vdash n$.

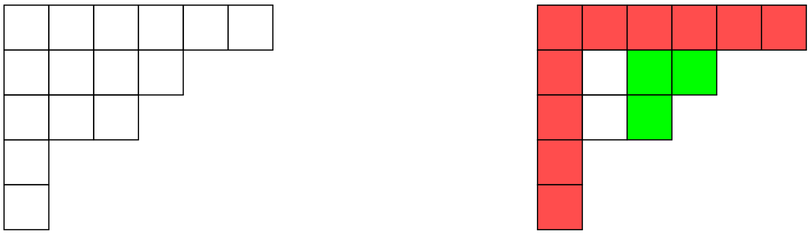
The literature has many different names for slightly different but related diagrams. For example, a Young tableaux is a Young diagram where the boxes are filled with symbols from an ordered set and hence denotes exactly one partition of this set. A Ferrers diagram is an alternative name for a Young diagram. We should also mention the french notation used in some textbooks which has the longest row on the bottom instead of the top. In general, we will switch freely between $\lambda$ denoting the Young diagram or the type of the partition.

Let us give a short example. The type of the partition given by the Young diagram in Figure 2.3a of the set $[15] = \{1, \dots, 15\}$ is $(6, 4, 3, 1, 1)$, meaning the first set has 6 elements, the second four and so forth. Let us give two other examples which will play an important role later on. We can imagine a Young diagram with only one row with $n$ boxes. This corresponds to the single partition of $n$ elements where every element is in the same set, i.e., $\{1, \dots, n\}$. On the opposite side we can imagine a Young diagram with $n$ rows, or equivalently, only one column. This then corresponds to the partition where every element is in a unique set, i.e. $\{\{i\} \mid i \in [n]\}$. See also Figure 2.4 for a graphical representation of the Young diagrams.

**Definition 2.21.** *A* skew hook *of a Young diagram $\lambda$ is a connected region of boundary boxes such that removing them from $\lambda$ leaves a smaller Young diagram.*

We call the height of a skew hook the number of vertical steps in the skew hook; one less than the number of rows.

Let $\lambda$ be a type of the partition with appropriate Young diagram and $\tau$ a skew hook. Then we denote by $\lambda \setminus \tau$ the Young diagram where we remove the skew hook $\tau$ from the diagram for $\lambda$. This is only possible if after removing the skew hook the remaining diagram is still a valid Young diagram.



(a) Example Young Diagram for 15.

(b) Skew Hook

Figure 2.3

For example, removing all the red boxes in Figure 2.3b results in an invalid Young diagram whereas removing all the green boxes is fine. We can see this as being allowed to remove a connected part of the right border.

The next theorem shows us how we can use Young diagrams and types of the partitions to compute the character of the symmetric group.

**Theorem 2.6.1** (Murnaghan-Nakayama Rule)**.** *Let $\lambda \vdash n$ and and $\sigma = \pi\psi$ be a disjoint product of $\pi \in S_{n-k}$ and $\psi$ a cycle of length $k$. Then*

$$\chi_\lambda(\sigma) = \sum_\tau (-1)^{r(\tau)} \chi_{\lambda\setminus\tau}(\pi)$$

*where the sum is over all skew hooks of length $k$ of $\lambda$ and $r(\tau)$ is the height of the skew hook $\tau$.*

We will give an example how to use this formula a bit later in this section.

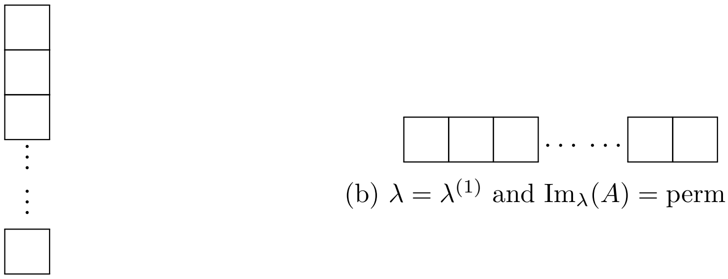
Now we can finally state the major definition for this section, given by Littlewood and Richardson [LR34].

**Definition 2.22** ([LR34])**.** *Let $\lambda \vdash n$. Then the immanant of a matrix $A \in \mathcal{R}^{n\times n}$ is defined as*

$$\mathrm{Im}_\lambda(A) := \sum_{\pi\in S_n} \chi_\lambda(\pi) \prod_{i=1}^{n} A_{i,\pi(i)}.$$

We see the immanant has a similar definition as the permanent or determinant. Indeed the following holds. Let $\lambda^{(1)}$ be the Young diagram from before with a single column and $\lambda^{(n)}$ be the Young diagram with a single row. Then for all matrices $A$, $\mathrm{Im}_{\lambda^{(1)}}(A) = \det(A)$ and $\mathrm{Im}_{\lambda^{(n)}}(A) = \mathrm{perm}(A)$. As in the determinant/permanent case we can define the matrix to consist of variables $x_{i,j}$ to define a polynomial. The corresponding Young diagrams for $\lambda^{(n)}$ and $\lambda^{(1)}$ are given in Figure 2.4.

We can use the Murnaghan-Nakayama Rule to understand computation of the immanant for $\lambda^{(1)}$ and $\lambda^{(n)}$. Let us start with $\lambda^{(n)}$. Let $\sigma = \pi\psi$ be given where $\psi$ is a cycle of length $k$. There is only one way to remove a skew hook of length $k$ from this diagram and this skew hook has height zero. Repeating this for all cycles (including self-loops) shows that the character for every permutation $\sigma$ is one.



(a) $\lambda = \lambda^{(n)}$ and $\mathrm{Im}_\lambda(A) = \det(A)$.

(b) $\lambda = \lambda^{(1)}$ and $\mathrm{Im}_\lambda(A) = \mathrm{perm}(A)$.

Figure 2.4

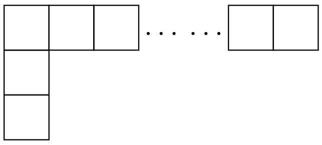


Figure 2.5: Hook Diagram with $\lambda = (n-2, 1, 1)$

Similarly, we can look at $\lambda^{(1)}$. Here there is again only one way to remove a skew hook of length $k$ but now the skew hook has height $k-1$. If we want to expand this formula it is useful to think about the permutation $\sigma$ as a series of transpositions $\sigma_1, \dots, \sigma_\ell$. With this we can now expand the recursive definition which results in

$$\underbrace{(-1)^{2-1} \dots (-1)^{2-1}}_{\ell \text{ many}}.$$

Indeed this gives us the character $\chi_{\lambda^{(1)}}(\sigma) = \mathrm{sgn}(\sigma)$.

We can see that immanant is a very general polynomial. It can “interpolate” between hard and computationally easy polynomials depending on which type of the partitions $\lambda$ we allow. Hence, the complexity of the immanant is an interesting field to study. To formulate result about the $\mathsf{VNP}$ hardness of the immanant, we have to talk about the immanant family, which we denote by $\mathrm{Im}_{n,\lambda_n}$ with families of Young diagrams. It is clear that for general $\lambda_n$ the immanant is $\#\mathsf{P}$-hard as it can compute the permanent. Let us now restrict ourselves to the commutative setting. What happens then to the complexity of the immanant if we restrict the possible types of the partitions $\lambda$? This initial question was posed by Strassen [Str90]. Hartmann [Har85] gave a first upper bound a few years earlier of $O(n^{n-s})$ for any immanant that has a family of type of the partition $\lambda_n = (\lambda_1, \dots, \lambda_s)$. He also gave lower bounds related to the computation of specific permanents. The upper bound was improved in some cases by Barvinok [Bar90]. He could show a bound of $O(n^3 d_\lambda)$ where $d_\lambda$ the number of standard tableaux of shape $\lambda$. This was followed by Bürgisser [Bür00c] with another improvement. Finally, a partial completeness result was found by Bürgisser [Bür00b].

Bürgisser conjectured that for any diagram of polynomially growing width the corresponding immanant is $\mathsf{VNP}$ complete. We call a type of the partition a hook diagram, if the type of the partition is $\lambda_n = (n-c, 1, \dots, 1)$ (cf. Figure 2.5). Bürgisser showed that for the special case of any family of hook diagrams or rectangular diagrams the immanant is $\mathsf{VNP}$ complete in the commutative setting. For the proof he used the following amazing observation by Merris [Mer83]. Let $\mathrm{HI}_{n,i}$ be the immanant for the hook diagrams of type $\lambda_n = (n-i, 1, \dots, 1)$. Then

$$\sum_{i=0}^{n-1} (-1)^i \, \mathrm{HI}_{n,i} = n \, \mathrm{HC}_n$$

where $\mathrm{HC}_n$ is the directed Hamiltonian cycle polynomial in $n$ variables[2]. With this he was able to show the completeness. He shows the membership by using Valiant's Criterion and the result by Hepler [Hep94] that computing the character of the symmetric group is $\#\mathsf{P}$ complete when the type $\lambda$ is given as input.

For the non-commutative setting we give a complete characterization of the complexity of the immanant polynomial in Theorem 3.5.7.

Bürgisser's conjecture was partially solved by Brylinski and Brylinski [BB03] with respect to diagrams who have at least an "overhang" that is polynomially growing. In contrast Mertens and Moore [MM13] showed that even non-hook diagrams, namely diagrams with constant width but polynomially growing height make the immanant $\mathsf{VNP}$ complete.

Finally, on the positive side there was the result by Grone and Merris [GM84] who showed that a $\mathrm{HI}_{2,n-2}$ is easy to compute. In essence, their proof notices that we can use the Murnaghan-Nakayama Rule to see that $\chi_{(2,1,\dots,1)}(\sigma) = \mathrm{sgn}(\sigma)(F(\sigma)-1)$ where $F(\sigma)$ is the number of fix points in $\sigma$. With this they are able to use the determinant in a clever way to compute the immanant.

We see that the complexity of the immanant is only understood in very specialized settings. One natural question is if there are better combinatorial ways to view the immanant that might help us analyzing its complexity. One of the few results on this was by Clearman, Shelton, and Skandera [CSS11] on a combinatorial way to see hook immanants. They interpret the determinant as specific paths on a graph and use this to define the immanant. However, their precise definition, while interesting, seems still to complicated to analyze and is far away from the simple definition of the permanent as the sum of all cycle covers.

We will sometimes need the hook formula given as follows which in turn needs the notion of a standard Young tableaux.

**Definition 2.23.** *A* standard Young tableaux *is a Young tableaux where the set to partition is* $\{1, \dots, n\}$ *such that each row and each column form an increasing sequence.*

For each cell we can define the hook $H_\lambda(i,j)$ the set of cells $(a,b)$ such that $a = i$ and $b \geq j$ or $a \geq i$ and $b = j$.

[2] The original paper by Merris shows that that the identity holds for undirected Hamiltonian cycle but Bürgisser uses the directed variant for the completeness proof.

**Lemma 2.6.2** (Hook Length formula)**.** *The number of standard Young tableaux is given by*

$$d_\lambda = \frac{n!}{\prod |H_\lambda(i,j)|}$$

*where the product is over all possible hooks $H_\lambda(i,j)$.*

We did not find the next lemma in the literature but include it for completeness

**Lemma 2.6.3.** *Let $M = \left(\begin{smallmatrix} A & 0 \\ 0 & B \end{smallmatrix}\right)$ be a block matrix. Then* $\mathrm{Im}(M) = \mathrm{Im}(A) \cdot \mathrm{Im}(B)$*.*

*Proof.* Let $\rho : S_n \to GL(V)$ be a representation of partition type $\lambda$. And we have two groups $G_1 = S_{n_1'}$, $G_2 = S_{n_2'}$ such that $n_1' + n_2' = n$.

Then we can build representations

$$\pi_1(g) = \begin{cases} \rho(g) & g \in G_1, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\pi_2(g) = \begin{cases} \rho(g) & g \in G_2, \\ 0 & \text{otherwise.} \end{cases}$$

First, it is clear that these are group homomorphism and hence representations of type $\lambda$. Then we can use Proposition 2.1 Item 2.

$$\begin{aligned} \mathrm{Im}_\lambda(M) &= \sum_{\sigma \in S_n} \chi_{G_1 \oplus G_2}(\sigma) \prod_{i=1}^n x_{i,\sigma(i)} \\ &= \left( \sum_{\sigma \in G_1} \chi_{G_1}(\sigma) \prod_{i=1}^n x_{i,\sigma(i)} \right) \cdot \left( \sum_{\sigma \in G_2} \chi_{G_2}(\sigma) \prod_{i=1}^n x_{i,\sigma(i)} \right) \\ &= \mathrm{Im}_\lambda(A) \cdot \mathrm{Im}_\lambda(B). \end{aligned}$$

The second to last equality holds now because $\chi_{G_2}(g)$ for $g \in G_1$ is zero and vice versa for $\chi_{G_1}$. □

# 3 On Hard Instances of Non-Commutative Problems.

This chapter is based on joint work with B. V. Raghavendra Rao and was partially published under [ER14].

## 3.1 Introduction

### Background

The underlying ring plays an important role in algebraic complexity theory. While the research focused mainly on the permanent vs determinant problem over fields and commutative rings there has also been an increasing amount of interest over non-commutative algebras. Nisan [Nis91] was the first to consider the complexity of these two polynomial families over non-commutative algebras. He showed that for any field $\mathbb{K}$, any non-commutative arithmetic formula over the free $\mathbb{K}$ algebra on $x_1, \dots, x_{n^2}$ computing the permanent or determinant of an $n \times n$ matrix requires size $2^{\Omega(n)}$. Later on, this was generalized to other classes of algebras by Chien and Sinclair [CS07]. Nisan's work left the problem of determining the arithmetic circuit complexity of the non-commutative determinant as an open question.

In a significant breakthrough, Arvind and Srinivasan [AS10] showed that computing the Cayley determinant is #P-hard over certain matrix algebras. The Cayley determinant and Cayley permanent are special non-commutative forms which we specify in the Preliminaries of this chapter. Their result was improved to other algebras by Chien, Harsha, et al. [CHSS11] and finally this question was settled by Bläser [Blä13] who classified such algebras. Further, Gentry [Gen14] simplified the reduction by Bläser considerably.

### Motivation

Though the studies in [AS10, CHSS11, Blä13] highlight the role of the underlying algebra in determining the complexity of the non-commutative determinant they do not shed much light on the combinatorial structure of non-commutative polynomials that are #P-hard. One could ask: *Does the hardness stem from the underlying algebra or are there inherent properties of polynomials that make them #P-hard in the non-commutative setting?* This chapters aims at answering this question.

As a first step, we look for polynomials that are easier to compute than the determinant in the commutative setting and whose non-commutative versions are

#P-hard. Natural candidate polynomials are the elementary symmetric polynomials and special cases of the determinant/permanent. One way to obtain special cases of determinant/permanent would be to restrict the structure of the underlying graph. For example, let $G$ be a directed graph consisting of $n$ cycles $(0,1),(2,3),\ldots,(2n-2,2n-1)$ of length two with self-loops at every node where each edge is labeled by a distinct variable. The permanent of $G$, $\mathrm{perm}(G)$, is given by $\prod_{i=0}^{n-1}(x_{2i,2i}x_{2i+1,2i+1}+x_{2i,2i+1}x_{2i+1,2i})$ where $x_{i,j}$ is the variable labeling of the edge $(i,j)$. This is one of the easiest to compute but non trivial special case of the permanent. Other candidates of special cases of graphs are outerplanar graphs, planar graphs and graphs of bounded treewidth to name a few.

### Results

In this chapter, we study the complexity of the Cayley permanent (Cayley-perm) on special classes of graphs. We exhibit a family of collections of disjoint two-cycles for which any algebraic branching program (ABP) computing the Cayley-perm should have size $2^{\Omega(n)}$ (Corollary 3.4.6) and extend it to outerplanar graphs (Corollary 3.4.7). Furthermore, we exhibit a parameter $\mathsf{cut}(G)$ (see Section 3.4.1 for the definition) for a collection $G$ of disjoint two-cycles on $n$ vertices such that any ABP computing Cayley-perm$(G)$ has size $2^{\Theta(\mathsf{cut}(G))}$ (Theorem 3.4.5). This makes the lower bound in Corollary 3.4.6 tight up to a constant factor in the exponent. It should be noted that our results also hold for the case of the Cayley determinant (Cayley-det) on such graphs. We also observe that for graphs of connected component size greater or equal to six the problem of evaluating Cayley-perm, Cayley-det and Cayley-Im (the Cayley Immanant) is #P complete (Theorems 3.5.6 and 3.5.7).

On the positive side, for graphs where each strongly connected component has at most $c$ vertices we obtain an ABP of size $n^{O(c)}c^{\mathsf{near}(G)}$ computing the Cayley-perm (Theorem 3.3.3) where $\mathsf{near}(G)$ is a parameter (see Definition 3.1) depending on the labeling of vertices on the graph.

We construct a non-commutative variant of the elementary symmetric polynomial that is #P-hard over certain algebras (Theorem 3.5.9). We show that computing Cayley-perm on rank one matrices is #P-hard.

In contrast to these hardness results, we will, as a side note, show some computational problems which are hard in the commutative case but easy in the non-commutative case (Section 3.6).

### Related Results

The study of commutative permanent on special classes of matrices was initiated by Barvinok [Bar96] who gave a polynomial time algorithm for computing the permanent of rank one matrices over a field. More recently, Flarup, Koiran, and Lyaudet [FKL07] showed that computing the permanent of bounded treewidth graphs can be done by polynomial size formulas. This was further extended by Flarup and Lyaudet [FL10] to other width measures on graphs. Datta, Kulkarni, et al. [DKLM10] showed that computing the permanent on planar graphs is as hard as the general case.

### Comparison to Other Results

Results reported in [AS10, CHSS11, Blä13, Gen14] highlight the importance of the underlying algebra and characterizes algebras for which Cayley-det is #P-hard. In contrast, our results shed light on the role played by the order in which vertices are labeled in a graph. For example, the commutative permanent of disjoint two-cycles has a depth three formula given by $\prod_{i=0}^{n-1}(x_{2i,2i}x_{2i+1,2i+1} + x_{2i,2i+1}x_{2i+1,2i})$ whereas Cayley-perm on almost all orderings of vertices requires exponential size ABPs. Independently, some lower bounds for skew circuits where recently found by Limaye, Malod, and Srinivasan [LMS15] using techniques based on Nisan's lower bounds.

## 3.2 Preliminaries

Over a non-commutative ring $\mathcal{R}$, there are many possibilities for defining the determinant/permanent of a matrix depending on the ordering of the variables (see for example [Asl96]). We will use the well known definitions of the *Cayley determinant* and the *Cayley permanent*. Let $X = (x_{i,j})_{1\leq i,j\leq n}$ be an $n \times n$ matrix with distinct variables $x_{i,j}$. Then

$$\begin{aligned} \mathsf{Cayley\text{-}det}(X) &= \sum_{\sigma\in S_n} \operatorname{sgn}(\sigma) x_{1,\sigma(1)} \cdots x_{n,\sigma(n)}, \\ \mathsf{Cayley\text{-}perm}(X) &= \sum_{\sigma\in S_n} x_{1,\sigma(1)} \cdots x_{n,\sigma(n)}. \end{aligned}$$

In the above, $S_n$ denotes the set of all permutations on $[n]$ symbols. Note that Cayley-det and Cayley-perm can also be seen as functions taking $n \times n$ matrices with entries from $\mathcal{R}$ as input.

The tensor product, or Kronecker product, of two matrices $A, B \in \mathbb{K}^{n\times n}$ with entries $a_{i,j}$, $b_{i,j}$ is denoted by $A \otimes B$ and is given by

$$A \otimes B = \begin{pmatrix} a_{1,1}B & \cdots & a_{1,n}B \\ \vdots & \ddots & \vdots \\ a_{n,1}B & \cdots & a_{n,n}B \end{pmatrix}.$$

We use the notion of *read once certificate* for ABPs as in [MR13]. Let $P$ be an ABP over disjoint sets of variables $X \cup Y$, with $|X| = n$ and $|Y| = m$. Let $p_P(X, Y)$ be the polynomial computed by $P$. $P$ is said to be read once certified in $Y$ if there are numbers $0 = i_0 < i_1 < \cdots < i_m$ where $i_m$ is at most the length of $P$ and there is a permutation $\pi \in S_m$ such that between layers from $i_j$ to $i_{j+1}$ no variable other than $y_{\pi(j+1)}$ from the set $Y$ appears as a label. We will reprove the following result from [MR13] as the original proof only works in the commutative setting.

**Proposition 3.1** ([MR13]). *Let $P$ be an ABP on $X \cup Y$ read-once certified in $Y$. Then the polynomial $\sum_{e_1,e_2,\ldots,e_m \in \{0,1\}} p_P(X, e_1, \ldots, e_m)$ can be computed by an ABP of size $\mathsf{poly}(\mathsf{size}(P))$.*

Note that the proof given in [MR13] uses the equivalence of skew circuits with ABPs, which does not hold in the non-commutative setting. We postpone a more detailed discussion about non-commutative skew circuits to Section 3.4.2.

*Proof.* Our argument is similar to the one in [MR13] except that we argue over ABPs themselves rather than skew circuits. Let $P$ be an ABP computing the non-commutative polynomial given by $p_P(x_1, \ldots, x_n, y_1, \ldots, y_m)$. Let $0 = i_0 < i_1 < \cdots < i_m$ be the layers of $P$ that witness the fact that $P$ is read-once certified in $Y = \{y_1, \ldots, y_m\}$. Without loss of generality assume that every layer of $P$ has exactly $r$ nodes. Let $g_1^j, \ldots, g_r^j$ be the nodes in the layer $i_j$. Note that variable $y_j$ is read in layers between $i_{j-1}$ to $i_j$ and is never used beyond that point and no other variable from $Y$ appears in layers between $i_{j-1}$ and $i_j$. Let $P_j$ be the portion of $P$ consisting only of layers of $P$ from $i_{j-1}$ to $i_j$. Let $p_j[\ell, k]$ be the polynomial represented as sum of weights of $g_\ell^j \rightsquigarrow g_k^{j+1}$ paths in $P$. Then

$$p_P = \sum_{k_1,\ldots,k_m} p_0[s, k_1] \cdot p_1[k_1, k_2] \cdots p_{m-2}[k_{m-2}, k_{m-1}] \cdot p_{m-1}[k_{m-1}, t].$$

For $0 \leq j < m$ and $k, k' \in \{1, \ldots r\}$ and $b \in \{0, 1\}$, let $p_j^b[k, k'] = p_j[k, k']|_{y_{j+1} \to b}$ where we substitute $y_{j+1}$ with $b$. Then

$$\sum_{e_1,\ldots,e_m \in \{0,1\}} p_P(X, e_1, \ldots, e_m)$$

is equal to

$$\begin{aligned}\sum_{k_1,\ldots,k_m} \Bigg( & \Big(p_0^0[s, k_1] + p_0^1[s, k_1]\Big) \\ & \cdot \Big(p_1^0[k_1, k_2] + p_1^1[k_1, k_2]\Big) \\ & \vdots \\ & \cdot \Big(p_{m-2}^0[k_{m-2}, k_{m-1}] + p_{m-2}^1[k_{m-2}, k_{m-1}]\Big) \\ & \cdot \Big(p_{m-1}^0[k_{m-1}, t] + p_{m-1}^1[k_{m-1}, t]\Big) \Bigg).\end{aligned} \tag{3.1}$$

Thus we can take sums of $P_j$ with $y_j = 0$ and $y_j = 1$ independent of $P_{j'}$ when $j \neq j'$. In the following we describe this construction.

We construct the new ABP inductively for every layer. We create two copies $P_j^0$ and $P_j^1$ where $P_j^b$ is obtained by setting $y_j = b$ for $b \in \{0, 1\}$. The copies are connected left as they where previously, meaning if there was an edge $g_k^{j-1}$ to a vertex $v$ there is now an edge from $g_k^{j-1}$ to both copies of $v$. For the edges to the next layer, we do the same. If there was an edge $(g_\ell^j, g_k^{j+1})$ in $P_j$ we connect this edge for $g_\ell^j$ in $P_j^0$ and $P_j^1$.

The correctness is obvious from Equation (3.1). The size is at most two times the original size.

□

Let $\mathcal{A}$ be a non-deterministic $s$-space bounded algorithm that uses non-deterministic bits in a read-once fashion and outputs a monomial on each of the accepting paths. We assume that a non-commutative monomial is output as a string on a write-only tape and non-deterministic paths are represented by binary strings $e \in \{0,1\}^m$ where $m \leq 2^{O(s)}$. The polynomial $p_{\mathcal{A}}$ computed by $\mathcal{A}$ is the sum of the monomial output on each of the accepting paths of $\mathcal{A}$, i.e., $p(x_1, \ldots, x_n) = \sum_e A(x_1, \ldots, x_n, e)$, where the sum is taken over all accepting paths $e$ of $\mathcal{A}$, and $A(x_1, \ldots, x_n, e)$ denotes the monomial output along the path represented by $e$.

**Proposition 3.2** (folklore). *Let $A(X)$ be an $s$-space bounded non-deterministic algorithm as above. There is a non-commutative ABP $P$ of size $2^{O(s)}$ that computes the polynomial $p_A(X)$.*

## 3.3 An Algorithm for the Cayley Permanent

In this section, we give an algorithm for $\mathsf{Cayley\text{-}perm}$ that is parameterized by the maximum difference between labels of vertices in individual components.

In what follows, we identify the vertices of a graph with the set $[n]$. A directed graph $G$ on $n$ vertices is said to have *component size bounded by c* if every strongly connected component of $G$ contains at most $c$ vertices where $c > 0$. We assume that edges of $G$ are labeled by distinct variables. Let us define a parameter that measures the closeness of labelings in each component.

**Definition 3.1.** *Let $G$ be a directed graph. The* nearness *parameter $\mathsf{near}(C)$ of a strongly connected component $C$ of $G$ is defined as $\mathsf{near}(C) = \max_{i,j \in C} |i - j|$. The nearness parameter of $G$ is defined as $\mathsf{near}(G) = \max_C \mathsf{near}(C)$, where the maximum is taken over the set of all strongly connected components in $G$.*

We now give Algorithm 3.1 which is a non-deterministic log-space bounded procedure $P$ that will guess a cycle cover $\gamma$ in $G$ and output the product of the weights of $\gamma$ with respect to the Cayley ordering as a monomial. Additionally, we ensure that the algorithm $P$ uses the non-deterministic bits in a read-once fashion.

We will denote by $C_1, \ldots, C_r$ the strongly connected components of $G$, sorted in the ascending order of the smallest vertex in each component. We represent a cycle cover in $G$ as a permutation $\gamma$ where $\gamma(i)$ is the successor of the vertex $i$ in the cycle cover represented by $\gamma$.

**Lemma 3.3.1.** *The Algorithm 3.1 can be implemented to use $O(\log c \cdot \mathsf{near}(G) + \log n)$ space, and is read-once on the non-deterministic bits.*

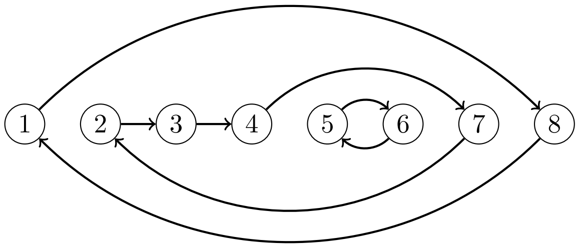


Figure 3.1: A graph with $\mathsf{near}(G) = 7$.

```
1  pos := 1
2  $T := \emptyset$
3  $\gamma :=$ the cycle cover of the empty graph
4  $f := 1$
5  for $1 \leq i \leq r$ do
6      Non-deterministically guess a cycle cover $\gamma'$ for $C_i$
7      $\gamma := \gamma \uplus \gamma'$
8      $T := T \cup V(C_i)$, where $V(C_i)$ are the vertices in $C_i$.
9      while $\exists k \in T$ with $k = pos$ do
10         $f := f \cdot x_{k,\gamma(k)}$
11         pos := pos + 1
12         $T := T \setminus \{k\}$
13 if $pos = n$ then
14     Output $f$ and accept.
```

**Algorithm 3.1:** Procedure $P$ for computing the Cayley Permanent

*Proof.* Let $T$ represent the set of vertices $v$ in the partial cover that is being built by the procedure where the weight of the edge going out of $v$ is not yet output, and pos the current position going from 1 to $n$. We have

$$T = \{k \mid \text{pos} < k \text{ and } k \text{ occurs in the components already explored}\}.$$

Firstly, we argue that at any point in time in the algorithm, $|T| \leq \mathsf{near}(G) + c$. Suppose the algorithm has processed components up to $C_i$ and is yet to process $C_{i+1}$. Let $\mu = \max_{v \in T} v$. Since the components are in ascending order with respect to the smallest vertex in them, the component $C_j$ with $\mu \in C_j$ must have $\mathsf{near}(C_j) \geq \mu - \text{pos}$. Thus $\mu - \text{pos} \leq \mathsf{near}(G)$. Also, just before Line 6 in any iteration, we have that for any $v \in T$, $\text{pos} < v \leq \mu$ and hence $|T| \leq \mu - \text{pos} + c \leq \mathsf{near}(G) + c$. Note that it is enough to store the labels of the vertices in $T$ and the choice $\gamma(v)$ made during the non-deterministic guess for each $v \in T$ and hence $O(|T| \log n)$ additional bits of information needs to be stored.

However, we will show that it is possible to implement the algorithm without explicitly remembering the vertices in $T$ and using only $O(|T|\log c)$ additional bits in memory. Suppose that the vertices in $T$ are ordered as they appear in $C_1, C_2, \ldots, C_r$ where vertices within a component are considered in the ascending order of their labels. Let $B$ be a vector of length $\mathsf{near}(G)$ where each entry $B_j$ is $\log c$ bits long which indicates the neighbour of the $j$th vertex in $T$. Now, we show how to implement Lines 9 to 12 in the procedure using $B$ as a data structure for $T$. To check if there is a $k \in T$ with $k = \text{pos}$, we can scan the components from $C_1, \ldots, C_i$ and check if the vertex assigned to pos occurs in one of the components. Remember that $\gamma(k)$ is the successor of $k$ in the cycle cover $\gamma$. To obtain $\gamma(k)$ from $B$, we need to know the number $j$ of vertices $v$ that appear in components $C_1, \ldots, C_i$ such that $v \geq \text{pos}$ and that occur before $k$. Then $\gamma(k) = B_j$. Once $B_j$ is used, we remove $B_j$ from $B$ and shift the array $B_{j+1}, \ldots, B_{\mathsf{near}(G)+c}$ by one index towards the left. Further, we can implement Line 6 by simply appending the information for $V(C_i)$ given by $\gamma'$ to the right of the array $B$. We require at most $O(c\log n)$ bits of space guessing a cycle cover $\gamma_i$ for component $C_i$ which can be re-used after the non-deterministic guessing of $\gamma_i$ is complete. Thus the overall space requirement of the algorithm is bounded by $O(\log c \cdot (\mathsf{near}(G) + c) + c\log n)$. □

**Lemma 3.3.2.** *Let $\mathsf{Acc}(G)$ be the sum of the monomials output by Algorithm 3.1 on all accepting paths. Then $\mathsf{Acc}(G) = \mathsf{Cayley\text{-}perm}(G)$.*

*Proof.* For an edge $(i,j) \in E(G)$, let $x_{i,j}$ denote the variable label on $(i,j)$. Let $A_G$ be the weighted adjacency matrix of $G$. Note that the Cayley permanent of $A_G$ equals the sum of weights of cycle covers in $G$ where the weight of a cycle cover $\gamma$ is the product of labels of edges in $\gamma$ multiplied in the Cayley order.

Recall that a permutation $\gamma \in S_n$ is a cycle cover of $G$ if and only if it can be decomposed into vertex disjoint cycle covers $\gamma_1, \ldots, \gamma_r$ of the strongly connected components $C_1, \ldots, C_r$ in $G$. Thus Line 6 enumerates all possible cycle covers in $G$. Also, the weights output at every accepting path are in the Cayley order. □

**Theorem 3.3.3.** *Let $G$ be a directed graph with component size bounded by $c$ and edges labeled by distinct variables. Then there exists an ABP of size $n^{O(c)}c^{\mathsf{near}(G)}$ computing the Cayley permanent of the adjacency matrix of $G$.*

*Proof.* By Lemma 3.3.2 and Proposition 3.2, we get an ABP $P$ computing a polynomial $p_G(X,Y)$ such that $\mathsf{Cayley\text{-}perm}(G) = \sum_{e_1,\ldots,e_m \in \{0,1\}} p_G(X,e)$ where $m = O(c\log n)$. Combining the above algorithm with the closure property of algebraic branching programs over read-once variables given by Proposition 3.1, we get a non-commutative arithmetic branching program computing $\mathsf{Cayley\text{-}perm}(G)$. It can be seen from Lemma 3.3.1 that size of the resulting branching program is at most $m2^{O((c\log c + \log c \cdot \mathsf{near}(G)) + c\log n)}$ which is equal to $n^{O(c)} \cdot c^{\mathsf{near}(G)}$ for large enough $n$. □

**Corollary 3.3.4.** *Let $G$ be as in Theorem 3.3.3. There is an ABP of size $n^{O(c)}c^{\mathsf{near}(G)}$ computing the Cayley determinant of $G$.*

*Proof.* The argument is the same as in Theorem 3.3.3 except that now the non-deterministic algorithm given in the proof of Theorem 3.3.3 also needs to compute the sign of the monomial being output. Let $C_1, \ldots, C_r$ be the strongly connected components of $G$. Then the sign of the permutation corresponding to a cycle cover $\tau$ of $G$ is the product of signs of the corresponding cycle covers of $C_i$. It can also be seen as $\mathrm{sgn}(\sigma) = (-1)^{n-\#\text{disjoint cylces in } \sigma}$. Thus it is enough to modify the algorithm given in the proof of Theorem 3.3.3 to output the sign of the cycle cover chosen for $C_i$. We do not violate the read-once restriction as the cycle cover is copied to the data structure $B$. The remaining arguments are exactly the same. □

## 3.4 Unconditional Lower Bound

### 3.4.1 ABPs

We now show that any branching program computing the non-commutative permanent of directed graphs with component size 2 must be of exponential size. This shows that the upper bound from Theorem 3.3.3 is tight up to a constant factor in the exponent, however, with a different but related parameter. All these lower bound results hold for free algebras over any field $\mathbb{K}$.

Our proof uses Nisan's ([Nis91]) partial derivative technique. We begin with some notation following his proof. Let $f$ be a non-commutative degree $d$ polynomial in $n$ variables. Let $B(f)$ denote the smallest size of a non-commutative ABP computing $f$. For $k \in \{0, \ldots, d\}$ let $M_k(f)$ be the matrix with rows indexed by all possible sequences containing $k$ variables and columns indexed by all possible sequences containing $d-k$ variables (repetitions allowed). The entry of $M_k(f)$ at $(x_{i_1} \ldots x_{i_k}, x_{j_1} \ldots x_{j_{d-k}})$ is the coefficient of the monomial $x_{i_1} \cdots x_{i_k} \cdot x_{j_1} \cdots x_{j_{d-k}}$ in $f$.

Intuitively, such a matrix shows "correlation" between the first $k$ variables and the next $n-k$ many variables. However, a polynomial size ABP can only have polynomial width and hence a low "correlation". Nisan was able to prove this with the following theorem.

**Theorem 3.4.1** ([Nis91])**.** *For any homogeneous polynomial $f$ of degree $d$*

$$B(f) = \sum_{k=0}^{d} \mathsf{rank}(M_k(f)).$$

We prove lower bounds for the Cayley permanent of graphs with every strongly connected component of size exactly 2, i.e., each strongly connected component being a two-cycle with self-loops on the vertices. Note that any collection of $n/2$ vertex disjoint two-cycles can be viewed as a permutation $\pi \in S_n$ consisting of disjoint transpositions and that $\pi$ is in fact an involution. Conversely, any involution $\pi$ on $n$ elements represents a graph $G_\pi$ with connected component size 2.

For a permutation $\pi \in S_n$ let the *cut at $i$* denoted by $C_i(\pi)$ be the set of pairs $(j, \pi(j))$ that cross $i$, i.e.,

$$C_i(\pi) = \{(j, \pi(j)) \mid i \in [j, \pi(j)] \cup [\pi(j), j]\}.$$

The *cut* parameter $\mathsf{cut}(\pi)$ of $\pi$ is defined as

$$\mathsf{cut}(\pi) = \max_{1\leq k\leq n} |C_k(\pi)|.$$

Let $G$ be a collection of vertex disjoint 2-cycles denoted by $(a_1, b_1), \ldots, (a_{n/2}, b_{n/2})$ where $n$ is even. The corresponding involution is $\pi_G = (a_1, b_1)\cdots(a_{n/2}, b_{n/2})$. By abusing the notation a bit, we let $\mathsf{cut}(G) = \mathsf{cut}(\pi_G)$. Without loss of generality, assume that $a_i < b_i$, and $a_1 < a_2 < \cdots < a_{n/2}$. Firstly, we note that $\mathsf{cut}(\pi)$ is bounded by $\mathsf{near}(G)$.

**Lemma 3.4.2.** *For any collection of disjoint 2-cycles $G$ on $n$ vertices, $\mathsf{cut}(\pi) \leq \mathsf{near}(G)$ where $\pi$ is the involution represented by $G$.*

*Proof.* Suppose $\mathsf{cut}(\pi) = r$ and $1 \leq i \leq n$ be such that $|C_i(\pi)| = r$. Let $(a, b) \in G$ where $a$ is the least value with $a < i$ and $b > i$ be the maximum such value. Then $b - a \geq \frac{2|C_i(\pi)|}{2} = r$. This concludes the proof. □

Figure 3.1 gives us an example where the equality does not hold. In this graph $\mathsf{near}(G) = 7$ but $\mathsf{cut}(G) = 3$.

Further, we note that the upper bound given in Theorem 3.3.3 holds true even if we consider $\mathsf{cut}(G)$ instead of $\mathsf{near}(G)$.

**Lemma 3.4.3.** *Let $G$ be a collection of disjoint 2-cycles and self-loops where every edge is labeled by a distinct variable or a constant from $\mathcal{R}$. Then there is an ABP of size $2^{O(\mathsf{cut}(G))}n^2$ computing the Cayley permanent of $G$.*

*Proof.* The algorithm is the same as in Theorem 3.3.3. We only need to argue the space bound as in Lemma 3.3.1. First note that either $a_i = i$, or $i$ has already occurred in one of the involutions $(a_1, b_1), \ldots, (a_{i-1}, b_{i-1})$. When the algorithm processes the component corresponding to the involution $(a_i, b_i)$, it needs to remember the outgoing edge chosen for $b_i$ (either the self-loop or the edge $b_i \to a_i$). Thus at any stage, the number of edges that needs to be stored is bounded by $t = \max_k |C_k(\sigma)|$. The rest of the arguments are exactly the same as in the original proof (Lemmas 3.3.1 and 3.3.2 and Theorem 3.3.3). □

**Lemma 3.4.4.** *Let $G$ be a collection of $\ell$ disjoint 2-cycles described by the involution $\pi$ and self loops at every vertex with edge labeled by distinct variables. Then $M_\ell(\mathsf{Cayley\text{-}perm}(G))$ contains $I_2^{\otimes t}$ as a sub-matrix where $t = \max_k |C_k(\pi)|$ and $I_2^{\otimes t}$ is the tensor product of $I_2$ with itself $t$ times and $I_2$ is the $2 \times 2$ identity matrix.*

*Proof.* Let $k \in [\ell]$, and $m = |C_k(\pi)| \leq \ell$. Let $C_k(\pi) = \{(a_{i_1}, b_{i_1}), \ldots, (a_{i_m}, b_{i_m})\}$ be such that $a_{i_j} \leq k \leq b_{i_j}$ for all $j$. Let $G_k$ be the graph restricted to involutions in $C_k(\pi)$. By induction on $m$, we argue that $M_m(\mathsf{Cayley\text{-}perm}(G_k))$ contains $I_2^{\otimes m}$ as a sub-matrix. The lemma would then follow since $M_m(\mathsf{Cayley\text{-}perm}(G_k))$ is itself a sub-matrix of $M_\ell(\mathsf{Cayley\text{-}perm}(G))$.

We begin with $m = 1$ as the base case. Consider the transposition $(a_{i_j}, b_{i_j})$, with $a_{i_j} \leq k \leq b_{i_j}$. The corresponding two cycle has four edges. Let $f_{i_j}$ be the Cayley

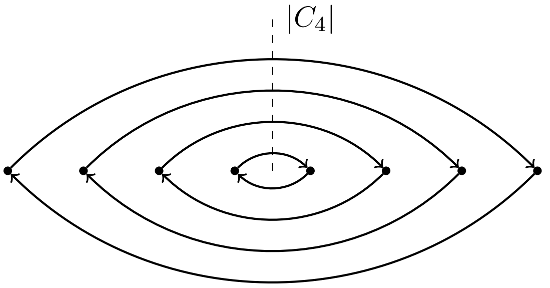


Figure 3.2: An example of a two connected component graph with high $\mathsf{cut}(G)$.

permanent of this graph then $M_1(f_{i_j})$ has the $2 \times 2$ identity matrix as a sub-matrix. Let us dwell on this simple part. For ease of notation let the variables corresponding to the self-loops be given by $x_{(a)}, x_{(b)}$ for $(a_{i_j}, a_{i_j})$ and $(b_{i_j}, b_{i_j})$ respectively and the edge $(a_{i_j}, b_{i_j})$ by $x_{(a,b)}$ and the edge $(b_{i_j}, a_{i_j})$ by $x_{(b,a)}$. Now our matrix has monomials $x_{(a)}, x_{(a,b)}$ as rows and $x_{(b)}, x_{(b,a)}$ as columns. We can ignore the other orderings as these will always be zero. As the valid cycle covers are given by $x_{(a)}x_{(b)}$ and $x_{(a,b)}x_{(b,a)}$ the base case is clear.

For the induction step, suppose $m > 1$. Suppose $a_1 < a_2 < \cdots < a_m$. Let $G'_k$ be the graph induced by $C_k(\pi) \setminus (a_1, b_1)$. Let $M' = M_{m-1}(\mathsf{Cayley\text{-}perm}(G'_k))$. The rows of $M'$ are labeled by monomials consisting of variables with first index $\leq k$ and the columns of $M'$ are labeled by monomials consisting only of variables with first index $> k$. Let $M = M_m(\mathsf{Cayley\text{-}perm}(G_k))$. $M$ can be obtained from $M'$ as follows: Make two copies of the row labels of $M'$, the first one with monomials pre-multiplied by $x_{a_1,a_1}$, and the second pre-multiplied by $x_{a_1,b_1}$. Similarly, make two copies of the columns of $M'$, the first by inserting $x_{b_1,b_1}$ to the column labels of $M'$ at appropriate position, and then inserting $x_{b_1,a_1}$ similarly. Now, the matrix $M$ can be viewed as two copies of $M'$ that are placed along the diagonal. Thus $M = M' \otimes I_2$, combining this with Induction Hypothesis completes the proof. □

**Remark 3.1.** *It should be noted that the ordering of the variables is crucial in the above argument. If $a_1, b_1 < k$ in the above, then $\mathsf{rank}(M) = \mathsf{rank}(M')$.*

As a side note, this proof above can also be visualized using basic facts from Quantum Computation. Consider a cycle $(a_{i_j}, b_{i_j})$. We can assign 2 Q-bits for edges outgoing from these vertices. A zero means the edge $(a_{i_j}, a_{i_j})$ is taken and a one that $(a_{i_j}, b_{i_j})$ is taken for the first bit. The other bit behaves similar but with the edges $(b_{i_j}, b_{i_j})$ and $(b_{i_j}, a_{i_j})$. It is clear that for valid cycle covers these pair form an entangled quantum state with two Q-bits but only two states (the $(0,0)$ state and the $(1,1)$ state). Now adding Q-bits which have no connection to the previous two cycles gives us the tensor product of the states.

We give an example of a two connected component graph along with the involution in Figure 3.2. We assume every vertex has a self-loop in this drawing. You can see that the size of $C_k$ directly corresponds to the number of decisions a cycle cover has to "keep in mind" if it is constructed iteratively from left to right.

**Theorem 3.4.5.** *Let $G$ be a collection of disjoint two cycles described by the involution $\pi$ and self-loops at every vertex, with edges labeled by distinct variables. Then any non-commutative ABP computing the Cayley permanent on $G$ has size at least $2^{\Omega(\mathsf{cut}(G))}$.*

*Proof.* It is enough to argue that for every $k$, there is an $\ell$ with $\mathsf{rank}(M_\ell(f)) \geq 2^{\Omega(|C_k(\pi)|)}$, then the claim follows from Theorem 3.4.1. Let $\ell = |C_k(\pi)|$, and suppose the transpositions crossing $k$ are given by $(a_{i_1}, b_{i_1}), \dots, (a_{i_\ell}, b_{i_\ell})$. Let $G'$ be the subgraph of $G$ induced by the vertices corresponding to the transpositions above. Let $f = \mathsf{Cayley\text{-}perm}(G')$. Applying Lemma 3.4.4 on $G'$ we conclude that $M_\ell(f)$ has $I_2^{\otimes|C_k(\pi)|}$ as a sub-matrix, i.e., the identity matrix of dimension $2^{|C_k(\pi)|} \times 2^{|C_k(\pi)|}$. Note that $f$ can be obtained by setting weights of the self-loops of vertices not in $G'$ to zero, and setting the remaining variables to one. Moreover, the matrix $M_\ell(f)$ is a sub matrix of $M_\ell(\mathsf{Cayley\text{-}perm}(G))$ obtained by relabeling the rows and columns as per the substitution mentioned above, and removing rows and columns that are zero. We conclude $\mathsf{rank}(M_\ell(\mathsf{Cayley\text{-}perm}(G))) \geq \mathsf{rank}(M_\ell(f)) \geq 2^{|C_k(\pi)|}$. □

The structural characterization above can be used to prove lower bounds for the Cayley permanent of a collection of 2-cycles. Let $\pi = (a_1, b_1) \cdots (a_{n/2}, b_{n/2})$, $a_1 < a_2 < \cdots < a_{n/2}$ be an involution. Then the graph $G_\pi$ associated with $\pi$ is the collection of 2-cycles $(a_1, b_1), \dots, (a_{n/2}, b_{n/2})$ and self-loops at every vertex.

**Corollary 3.4.6.** *Let $G$ be a collection of disjoint two cycles described by the involution $\pi$ and self-loops at every vertex, with edges labeled by distinct variables. Then $B(\mathsf{Cayley\text{-}perm}(G)) \in 2^{\Theta(\mathsf{cut}(G))}$. Further, there exists a graph $G$ with $\mathsf{cut}(G) = \Theta(n)$.*

*Proof.* The first part follows immediately from Lemma 3.4.3 and Theorem 3.4.5. For the second statement, consider the invoution $\pi$ with $\pi(i) = n/2 + i$. It can be seen that $\max_k |C_k(\pi)| = n/2$. Thus by Theorem 3.4.5 we have any ABP computing the Cayley permanent of $G$ is of size $2^{\Omega(n)}$. Since $\mathsf{cut}(\pi) \leq n$ for any graph, by Lemma 3.4.3 the result follows. □

We get another immediate Corollary.

**Corollary 3.4.7.** *Any non-commutative ABP computing the Cayley permanent of all matrices represented by outer planar graphs require size $2^{\Omega(n)}$ where $n$ is the number of vertices.*

*Proof.* Given an involution $\pi = (a_1, b_1), \dots, (a_{n/2}, b_{n/2})$ with associated graph $G$, the outerplanar graph is already given in a non outerplanar embedding. If we arrange the vertices in a way that $(a_1, b_1)$ lie next to each other we get an outerplanar embedding. Similarly we can get a graph with only one connected component by connecting these vertices with edges of weight zero. The result now follows from Corollary 3.4.6. □

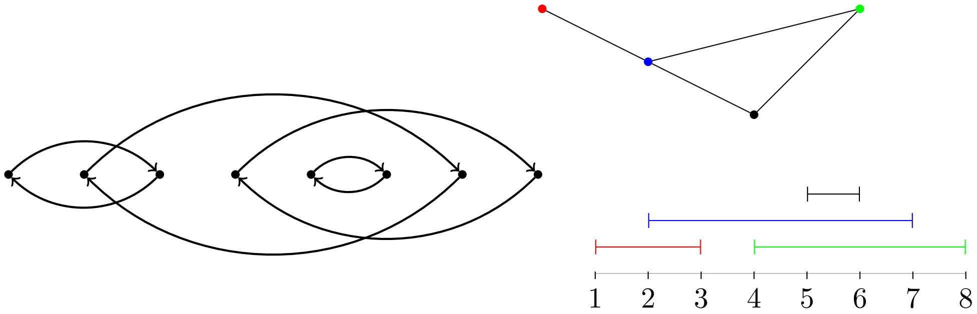


Figure 3.3: Graph and the corresponding intervals and the Interval Graph

Finally, we show that the hard instance exhibited in Corollary 3.4.6 is not an exception but in fact a rule, i.e., almost all involutions $\pi$ except an $1/\sqrt{n}$ fraction have $\mathsf{cut}(\pi) = \Omega(n)$. As before, let $n = 2m$. Then an involution $\pi$ on $\{1, \ldots, n\}$ with $\pi(i) \neq i$ represents a collection of $m$ intervals

$$I_\pi = \{[i, \pi(i)] \mid 1 \leq i \leq n,\ i < \pi(i)\}.$$

**Definition 3.2.** *The* Interval Graph *for a set of intervals $I = \{i_1, \ldots, i_n\}$ is given by $(V, E)$. Every interval $i_j \in I$ has exactly one vertex and two vertices $v_i, v_j$ have an edge between them if for the corresponding intervals $i_i \cap i_j \neq \emptyset$.*

Let $H_\pi$ be the interval graph formed by the intervals in $I_\pi$.

**Lemma 3.4.8.** *Let $\pi$ be an involution and $H_\pi$ be the interval graph as defined above. Then $\mathsf{cut}(\pi) \geq l/n$ where $l$ is the number of edges in $H_\pi$.*

*Proof.* For every edge $(a, b)$ in $H_\pi$, the corresponding intervals $I_a = [i, \pi(i)]$ and $I_b = [j, \pi(j)]$ have non empty intersection. Suppose $i < j$, then $j \in [i, \pi(i)]$. (In the case when $j > \pi(j)$, we have $\pi(j) \in [i, \pi(i)]$. Other cases can be handled analogously.) Thus every edge in $H_\pi$ contributes at least one distinct interval $[i, \pi(i)]$ with $i \leq k \leq \pi(i)$, i.e., it contributes a value to $C_k(\pi)$ for some $k$. Then

$$l \leq \sum_k |C_k(\pi)| \leq \sum_k \mathsf{cut}(\pi) \leq n \cdot \mathsf{cut}(\pi),$$

as $1 \leq k \leq n$. □

With this we have a correspondence between the graphs and interval graphs. As an example we show the graph on the left in Figure 3.3 and the corresponding intervals and the interval graph on the right.

Scheinerman [Sch88] showed that, random interval graphs have $\Omega(n^2)$ edges with high probability.

**Theorem 3.4.9** ([Sch88]). *Let $H_\pi$ be an interval graph where $\pi$ is an involution on $[n]$ chosen uniformly at random. Then $H_\pi$ has at least $n^2/3 - n^{7/4}$ edges with probability at least $1 - 1/\sqrt{n}$.*

**Corollary 3.4.10.** *For an involution $\pi$ on $[n]$ chosen uniformly at random, we have $\mathsf{cut}(\pi) = \Omega(n)$ with probability $1 - 1/\sqrt{n}$.*

The theorem now follows.

**Theorem 3.4.11.** *For all but a $1/\sqrt{n}$ fraction of graphs $G$ with connected component size 2, any ABP computing the* Cayley-perm *on $G$ requires size $2^{\Omega(n)}$.*

We can also prove a small corollary about the involutions.

**Corollary 3.4.12.** *Let $\pi$ be the involution in Corollary 3.4.6. Let $\pi'$ be an involution which can be constructed from $\pi$ with at most $O(\log n)$ transpositions. Then any non-commutative ABP computing the Cayley permanent on $\pi'$ has exponential size.*

We want to mention the relationship with the result by Hrubes, Wigderson, and Yehudayoff [HWY10] which shows that for any "ordered" non-commutative polynomial is it enough to prove lower bounds for multilinear circuits. However, the best known bound, to our knowledge, is by Raz and Yehudayoff [RY09] for constant depth multilinear circuits.

### 3.4.2 Weakly Skew Circuits

Recently Limaye, Malod, and Srinivasan [LMS15] showed lower bounds for non-commutative skew circuits and generalizations of these in independent work. In fact, they use the Palindrome Polynomial in variables $x_0, x_1$. Let $x_e = x_{e_1} x_{e_2} \dots x_{e_n}$ for $e \in \{0,1\}^n$ and $x_{e^R} = x_{e_n} x_{e_{n-1}} \dots x_{e_1}$, the reverse product of the binary string $e$. Then the Palindrome Polynomial is given by

$$\mathrm{Pal}(X) = \sum_{e \in \{0,1\}^n} x_e x_{e^R}.$$

They showed that the Palindrome Polynomial has exponential size ABPs in the non-commutative setting but can be easily computed with a weakly skew circuit based on the recursive structure $F(n) = x_0 F(n-1) x_0 + x_1 F(n-1) x_1$. We can view this polynomial with a different variable set. Let us order the variables occurring in the binary string from left to right and replace them by new variables $x_{0,i}$ or $x_{1,i}$ for $i \in [n]$. Now we can set $x_{0,i}$ corresponding to a self-loop and $x_{1,i}$ to the edges in the loop. In fact, this is a polynomial we shown to be hard for ABPs by Theorem 3.4.5 and is a generalization of the example in Figure 3.2.

Further, they showed some extension to this work, one of them being a lower bound for weakly skew circuits. They use the Palindrome Squared Polynomial given by

$$\mathrm{Pal}(X)^2 = \sum_{e_1, e_2 \in \{0,1\}^n} x_{e_1} x_{e_1^R} x_{e_2} x_{e_2^R}.$$

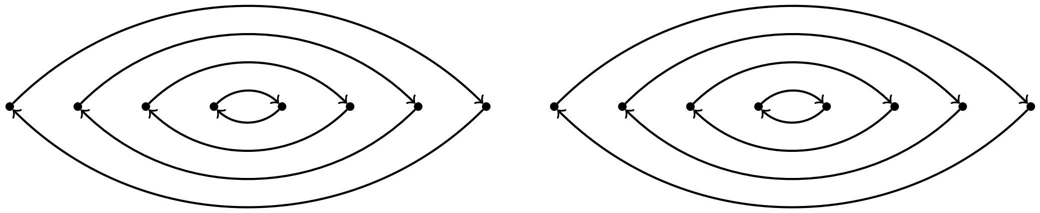


Figure 3.4: Palindrome Squared Graph

We can do a similar replacement as for the Palindrome Polynomial. We will take two different sets of variables, namely $x_{0,i}$, $x_{1,i}$, $y_{0,i}$, $y_{1,i}$ where we replace the first two sets of variables with $x_{0,i}$ and $x_{1,i}$, and the second set with the other variables. The polynomial can be represented similar to our previous example as a graph where we omitted the self-loops (Figure 3.4). Notice, that this graph also has exponential size ABP as the maximum for $\max_k |C_k|$ is obtained for $k = \frac{n}{4}$.

This immediately gives us the following corollary from their work.

**Corollary 3.4.13.** *There exists a graph with component size two such that every non-commutative skew circuit computing the permanent on this graph has exponential size.*

We will now spent a bit of time on explaining their lower bound for skew circuits as it is closely related to our previous proof. It is clear that simply using the same lower bound technique does not work as the Palindrome Polynomial is a counter example. It has exponential large rank for the Nisan matrix but a skew circuit of linear size. Hence, a new version needs to be introduced. A natural generalization of Nisan's matrix can be constructed. For this, we label the rows by all possible pairs of monomials $(m_1, m_2)$ and the columns by all possible monomials $m$ where $m$ has degree $k$ and $\deg(m_1) + \deg(m_2) = d - k$. The value in the cell then corresponds to the coefficient of the monomial $m_1 \cdot m \cdot m_2$. The matrix has $n^k$ rows and $(d-k+1)n^{d-k}$ columns. The intuition behind this measure can be given as follows. It is clear that a skew circuit can easily correlate the last and first few values together, in contrast to an ABP. However, it should have a hard time with correlations that are between the left part and parts left from the middle as the power of left and right multiplication can only give this with large skew circuit size.

We can imagine decomposing our polynomial $f$ as $\sum_{i \in [t]} h_i g_i h'_i$. And use this matrix to show a lower bound. However, this seems difficult as $h_i$ and $h'_i$ can have any degree in $[0, \frac{d}{4}]$. Hence, the matrix cannot even be defined.

Arvind and Raja [AR14] solved this problem by modifying the matrix. They defined that the coefficient of a monomial $m'$ is now given by $\sum_{m' = m_1 m m_2} M_k((m_1, m_2), m)$, the sum over all possible decompositon of $m'$ into $m_1 m m_2$. However, with this matrix they could only show a lower bound for monotone skew circuits. The many different possible "partitions" where to hard to analyze.

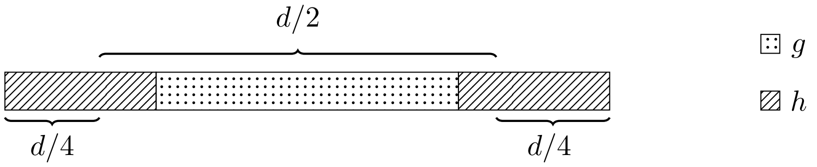


Figure 3.5: Partition

Limaye et al. could actually prove such a decomposition for weakly skew circuit, similar to the work by Hrubes, Wigderson, and Yehudayoff [HWY10].

**Lemma 3.4.14** ([LMS15])**.** *Let $f$ be a polynomial of degree $d$ computed by a non-commutative skew circuit of size $s$. Fix any $d' \in [d]$. Let $g_1, \ldots, g_t$ be the intermediate polynomials of degree $d'$ computed by $C$. Then there exist homogeneous polynomials $h_{i,j}$ of degree $j'$ and $h'_{i,j}$ of degree $d - d' - j$ for $i \in [t]$ and $j \in [d - d']$ such that*

$$f = \sum_{i \in [t]} \sum_{j \in [d-d']} h_{i,j} \cdot g_i \cdot h'_{i,j}.$$

*Furthermore, each $h_{i,j}$ and $h'_{i,j}$ can be computed by a skew circuit of size at most sd.*

One of the key insights was that $[t]$ is actually a small constant. They could now continue with the proof in the following way. Let us pick $d' = \frac{3d}{4}$ which gives us a small sum of homogeneous polynomials $g$ of degree $\frac{3d}{4}$ and $h_i$, $h'_i$ of combined degree $\frac{d}{4}$. Notice that $h_{i,j}$ and $h'_{i,j}$ now have a fixed degree. With this they can look at the rank for every $h_{i,j} g_i h'_{i,j}$ for $1 \leq i \leq t$ and $0 \leq j \leq \frac{d}{4}$.

They were able to show that for this partition $\Pi$, $\text{rank}(M(f, \Pi))$ is equal to the rank of

$$\sum_{i \in [t], j \in [\frac{d}{4}]} \text{rank}(M(g, \Pi_g)) \cdot \text{rank}(M((h_{i,j}, h'_{i,j}), \Pi_h))$$

for the restricted partitions $\Pi_h$ on $h$ and $\Pi_g$ on $g$. Here the partitions are actually sets of matrix coordinates. Let us clear up the different partitions. $\Pi$ is just the partition $([d/4] \cup [3d/4 + 1, d], [d/4 + 1, 3d/4])$. Then $\Pi_h$ is, in essence, the partition restricted to the set of indices that overlap with $h$ or $h'$ and $\Pi_g$ the one that overlap with $g$. As you see in Figure 3.5 our original partition does not necessarily correspond to the monomials $g_i$ and $h_{i,j}$, $h'_{i,j}$ and now it should be clear why we have to restrict our partitions. Their proof can be seen as showing that even using these restricted partitions is enough to show a lower bound.

Let us continue with the proof. Limaye et al. could prove that the $\text{rank}(M(h_{i,j}, h'_{i,j}), \Pi_h)$ is one for all $i$, $j$. This is intuitively clear as the there is only one index on the columns selected. They can further show that $\text{rank}(M(g, \Pi_g))$ is bounded by[1] $2^{\frac{d}{4}}$ by a similar tensor argument as in our Lemma 3.4.4. As the first $\frac{d}{4}$ many variables are fully

[1] In the paper, they give an upper bound of $n^{\frac{d}{4}}$ as they use $n$ variables instead of two.

correlated with the variables in $[\frac{d+1}{4}, \frac{d}{2}]$. Notice that this bound will only be met if $\deg(h_{i,j}) = 0$, or $\deg(h'_{i,j}) = 0$. With this they can combine their results to give an upper bound for $\operatorname{rank}(M(f))$ as $sd2^{\frac{d}{4}}$ where $s$ is the size of the skew circuit. This immediately implies that for a large size a large rank is necessary.

**Open Problem 3.1.** *Can we use the proof from [LMS15] to get a similar result for weakly skew circuits as Theorem 3.4.11 for graphs with connected components of size two?*

Similarly, they could show lower bounds for skew circuits where they allow at most a constant number of non-skew gates.

## 3.5 Completeness Results

In this section, we show multiple hardness results for simple polynomials over certain classes of non-commutative algebras. We give a #P completeness result for specific graphs of component size at most six. The completeness result is obtained by a careful analysis of the parameters in the reduction from $3-\#SAT$ to non-commutative determinant given recently by Gentry [Gen14] and the modification we will do to make this proof work for the Cayley permanent and the Cayley immanant.

### 3.5.1 A Recap of Gentry's Proof

For clarity we will repeat the proof of Gentry with some corrections by Goldreich. We will switch freely between denoting the inverse of an element $g$ in a group by $\frac{1}{g}$ or $g^{-1}$.

**Definition 3.3.** *Let $\mathcal{R}$ be some algebra. A product program $P$ over $\mathcal{R}$ with $n$ instructions for an input of length $m$ is given by $P = (a_0, (\iota_1, a_{1,0}, a_{1,1}), \ldots, (\iota_n, a_{n,0}, a_{n,1}))$. Where we call the sequence of length $n$ of the form $(\iota_i, a_{i,0}, a_{i,1})_{i\in[n]} \in ([m], \mathcal{R}, \mathcal{R})^n$ the instructions and $a_0$ the starting element.*

*It computes on an input $x_1, \ldots, x_m \in \{0,1\}$ the product*

$$a_0 \cdot \prod_{i=1}^{n} a_{i,x_{\iota_i}}.$$

In words, our product program decides for every instruction $(\iota_i, a_{i,0}, a_{i,1})$ if it should multiply $a_{i,0}$, if the bit of $x$ at the position $\iota_i$ is zero, or $a_{i,1}$, if the bit of $x$ at the position $\iota_i$ is one. These product programs where created in a line of research related to binary decision graphs and boolean branching program (cf. [Bar89]). They were famously used in Barrington's Theorem which states that constant width boolean branching programs are equal to the class $\mathsf{NC}_1$. We will generally not distinguish between an input as vector of length $m$ or a string of length $m$ and will index the string $x$ with $x_i$ to mean the $i$th bit.

**Lemma 3.5.1** ([Gen14, Lemma 2]). *For any division algebra $\mathcal{R}$, the group of units of $\mathcal{R}^{2\times 2}$ contains a subgroup isomoprhic to $S_3$. In particular, $\mathcal{R}^{2\times 2}$ contains the matrices*

$$r = \begin{pmatrix} 0 & -1 \\ 1 & -1 \end{pmatrix} \text{ and } s = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

*Proof.* It is clear that $\mathcal{R}^{2\times 2}$ contains these matrices. It is easy to see that $r$ has order three and $s$ order two and hence by Lagrange's Theorem they generate a group of order at least six. We know that the only elements in the group are $1, s, r, r^2, rs, sr$ as $rs = sr^2$ and $r^2 s = sr$ and $s$ has order two. As $S_3$ is the only non-abelian group of order six the lemma holds. □

With this we can now show a product program that outputs one if an assignment satisfies a $d$-CNF formula and zero else.

**Lemma 3.5.2** ([Gen14, Lemma 3]). *There exists a product program of length $2^d + 2^{d-1} - 2$ over the group $S_3$ that computes a disjunction of any $d$ literals. It outputs 1 if $x$ satisfies the disjunction and $r$ otherwise.*

*Proof.* Let 1 be the multiplicative neutral element of the $\mathcal{R}^{2\times 2}$.

We give a proof by induction and assume $a_0$ to be 1. Let $d = 1$. Either the bit $x_1$ is true, then $a_{1,1}$ is returned or the bit is zero and hence $a_{1,0}$ is returned. As $a_{1,0} = r$ and $a_{1,1} = 1$ the base case is clear.

Let us now assume that the lemma is true for $d-1$ and let $P_{d-1}$ be the constructed product program. Let $x_1, \ldots, x_n$ be the bits of our input and $b_0 = s$ and $b_1 = 1$. Then we construct the program such that the multiplication will be performed as follows:

$$b_{x_d} \cdot P_{d-1}(x_1, \ldots, x_{d-1}) \cdot b_{x_d} \cdot (P_{d-1}(x_1, \ldots, x_{d-1}))^{-1}.$$

Here $(P_{d-1}(x_1, \ldots, x_{d-1}))^{-1}$ is replacing all instructions $(\iota_i, (\alpha, \beta))$ by the corresponding instruction $(\iota_i, (\alpha^{-1}, \beta^{-1}))$.

Let us now prove the correctness. If $x_i = 1$ then it is clear that the layer $i$ evaluates to one as one commutes with all elements, especially $P_{d-1}$ and $P_{d-1}^{-1}$. Hence all layers above will also evaluate to one as $b_0^2 = b_1^2 = 1$.

If all bits of the input are zero then $P_{d-1}(x_1, \ldots, x_{d-1}) = r$ by induction, $b_{x_d} = s$ and hence $P_d(x_1, \ldots, x_d) = srsr^{-1}$. By the equalities above this is equal to

$$s(rs)r^{-1} = s(sr^2)r^{-1} = s^2 r = 1r.$$

□

Let

$$t = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.$$

**Theorem 3.5.3** ([Gen14, Theorem 6])**.** *For any division algebra $\mathcal{R}$ and any constant $d$ one can construct a product program of length $k(2^d + 2^{d-1} - 2)$ for a $d$-CNF formula with $k$ clauses. It outputs $t$ if the formula is satisfied by $x$ and $0$ otherwise.*

*Proof.* Let our clauses be given by $c_1, \dots, c_k$. By Lemma 3.5.2 we get a product program $P_{c_i}$ for every clause $c_i$. Then we construct our product program $P$ to compute the multiplication as follows:

$$\left( \prod_{i=1}^{k} t \cdot P_{c_i}(x_1, \dots, x_m) \right) t.$$

Here the multiplication with $t$ can easily be simulated with an instruction of the form $(1, (t, t))$.

For ease of notation we have written the complete variable set for our clause product programs but we can easily remove unneeded variables from a clause.

Let us give a correctness argument. Suppose one of the clauses is not satisfied. This then contributes a value of $t \cdot r$ to our product. It can be seen that this program has only two possible outcomes if one equation is not fulfilled. Namely,

$$t \cdot r = \begin{pmatrix} 0 & -1 \\ 0 & 0 \end{pmatrix}$$

and for any $g$

$$t \cdot r \cdot t \cdot g = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix},$$

as $t \cdot r \cdot t = 0$. Notice, that we will always multiply the value $t \cdot r$ from an unsatisfied clause with $t$ to the right. Hence, by associativity the resulting matrix will always be zero.

If all clauses are fulfilled the value computed is $t^{k+1}$. However, $t$ is idempotent in $\mathcal{R}^{2\times 2}$ and hence is equal to $t$. □

Let $P$ be the product program as in Theorem 3.5.3. Then it is obvious that

$$\#3\text{-SAT}(\phi) = \left( \sum_{e \in \{0,1\}^m} P_\phi(e) \right)_{(1,1)},$$

the first entry in the resulting matrix, as every satisfying assignment contributes exactly one and every unsatisfied assignment zero. To compute the sum with the Cayley determinant we will use the following special matrix form.

**Definition 3.4.** *We say a $n \times n$ matrix $M$ is a* barber pole matrix *if it is of the form*

$$M[i,j] = \begin{cases} \alpha_i & \text{if } i = j, \\ \beta_i & \text{if } i = j+1 \mod n, \\ 0 & \text{otherwise,} \end{cases}$$

*for $\alpha_i, \beta_i$ non zero.*

As an example, we can see that the following matrix is a barber pole matrix of size $5 \times 5$ over $\mathbb{N}$.

$$\begin{pmatrix} 1 & 2 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 4 & 5 & 0 \\ 0 & 0 & 0 & 8 & 3 \\ 4 & 0 & 0 & 0 & 7 \end{pmatrix}.$$

Notice, that for every barber pole matrix there exists only two cycle covers. Either the one where every vertex takes a self-loop or the single cycle.

**Theorem 3.5.4** ([Gen14, Theorem 5])**.** *The value $\sum_{e \in \{0,1\}^m} P_\phi(e)$ can be computed by the Cayley determinant of a matrix of size $n \times n$ over the algebra $\mathcal{R}^{2\times 2}$.*

*Proof.* Let $P = (1, (\iota_1, a_{1,0}, a_{1,1}), \ldots, (\iota_n, a_{n,0}, a_{n,1}))$. Let

$$\mathcal{I}_\ell = \{i \in [n] \mid \text{the } i\text{th instruction uses the } \ell\text{th bit of the input}\}.$$

Let $\mathcal{I}_\ell$ have the instructions $i_{\ell,1}, \ldots, i_{\ell,|\mathcal{I}_\ell|}$. Without loss of generality we can assume that these have odd size greater than one by adding dummy instructions. Let $\pi_0$ be the identity permutation and $\pi_1(i_{\ell,\kappa}) = i_{\ell,\kappa+1 \mod |\mathcal{I}_\ell|}$ the "shifted" permutation. Notice, that this corresponds to multiple cyclic permutation, consisting of cycles of length $|\mathcal{I}_\ell|$ for $2 \le \ell \le n$ where the elements of the cycle are the elements in $\mathcal{I}_\ell$. They are ordered in the natural order of the instructions.

Then we define the matrix

$$M[i,j] = \begin{cases} a_{i,b} & \text{if } j = \pi_b(i), \\ 0 & \text{otherwise.} \end{cases}$$

Left to show is that $\mathsf{Cayley\text{-}det}(M)$ is indeed computing the value of the product program. Let us look at this matrix a bit closer. We can see that this is a block barber pole matrix where the entry not on the diagonal are permuted.

Let us generate the matrix only for the set of instructions $\mathcal{I}_\ell = \{i_{\ell,1}, \ldots, i_{\ell,|\mathcal{I}_\ell|}\}$. At first we add all the entries for $\pi_0$ which are just on the diagonal. If we now look at $\pi_1$, we see that the first entry we add is at position $(1, i_{\ell,2})$ where $i_{\ell,2}$ is the index of the next instruction. We continue this until $\pi_1$ wraps around. This is clearly a cycle in the graph corresponding to the matrix.

It is now clear that $\pi_1$ produces a cycle for every $\mathcal{I}_\ell$. In essence it enforces that we either take all self-loops or all elements corresponding to $\pi_1$. Meaning we either multiply all values in the instructions asking $x_\ell$ for $x_\ell$ being zero or all values for the instructions where $x_\ell$ is one.

By this argument it is clear that one cycle cover is the value of the product program where we chosen every bit of the input and hence the value of all cycle cover is $\sum_{e\in\{0,1\}^m} P(e)$. As we enforced every cycle to have odd length and we have an even number of even length cycles, this can be computed with the determinant of the matrix. □

As the Cayley permanent is equal to the Cayley determinant for this construction we get the following corollary.

**Corollary 3.5.5.** *The value $\sum_{e\in\{0,1\}^m} P_\phi(e)$ can be computed by the Cayley permanent of a matrix of size $n \times n$ over the algebra $\mathcal{R}^{2\times 2}$.*

Notice, that the Cayley permanent does not need the requirement that $\mathcal{I}_\ell$ has odd size. This concludes the proof by Gentry.

Let us give a complete example how Gentry's Proof works. Let our formula be

$$(x_1 \vee x_2) \wedge (x_3 \vee x_1). \tag{3.2}$$

We then build the product programs for this. For the the disjunctions we get the following instructions for our product program

$$(2,(s,1)),(1,(r,1)),(2,(s,1)),(1,(r^{-1},1^{-1}))$$

and

$$(1,(s,1)),(3,(r,1)),(1,(s,1)),(3,(r^{-1},1^{-1})).$$

With this we can get the complete product program which is

$$\begin{aligned}(1,\Big(&(4,(t,t)),(2,(s,1)),(1,(r,1)),(2,(s,1)),(1,(r^{-1},1)),\\ &(4,(t,t)),(1,(s,1)),(3,(r,1)),(1,(s,1)),(3,(r^{-1},1)),(4,(t,t))\Big)).\end{aligned}$$

We make two conceits for clarity of presentation. Firstly, we did not pad the number of instructions to an odd number. Secondly, we added the new variable $x_4$ for the dummy instructions containing $t$.

Now with this we can build the matrix. Let us first state $\mathcal{I}_1,\dots,\mathcal{I}_4$.

$$\begin{aligned}\mathcal{I}_1 &= \{3,5,7,9\},\\ \mathcal{I}_2 &= \{2,4\},\\ \mathcal{I}_3 &= \{8,10\},\\ \mathcal{I}_4 &= \{1,6,11\}.\end{aligned}$$

$$\begin{pmatrix}
t & 0 & 0 & 0 & 0 & t & 0 & 0 & 0 & 0 & 0 \\
0 & s & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & r & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & s & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{r} & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & t & 0 & 0 & 0 & 0 & t \\
0 & 0 & 0 & 0 & 0 & 0 & s & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & r & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & s & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \frac{1}{r} & 0 \\
t & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t
\end{pmatrix}$$

Equation Figure 3.3: Matrix corresponding to Equation (3.2)

For ease of recognizing we color coded these cycles in this example. From this, we see that the permutation $\pi_1$, here stated in the standard notation, is given by

$$\pi_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 6 & 4 & 5 & 2 & 7 & 11 & 9 & 10 & 3 & 8 & 1 \end{pmatrix}.$$

Now the cycle covers are straightforward to see and we get the matrix as in Equation Figure 3.3 on Page 45.

### 3.5.2 Connected Components of Size 6 of Permanent and Immanant

With this recap we can now prove two new theorems.

**Theorem 3.5.6.** *Let $\mathcal{R}$ be a division algebra over a field $\mathbb{K}$ of characteristic zero containing the algebra of $2 \times 2$ matrices over $\mathbb{K}$. Computing the Cayley determinant or Cayley permanent on graphs with component size 6 with edges labeled from $\mathcal{R}$ is $\#\mathsf{P}$ complete.*

*Proof.* It is known that counting the number of satisfying assignments in a 2-CNF formula where every variable occurs at most three times is already $\#\mathsf{P}$ complete ([Rot96]). Let $\phi$ be a 2-CNF where every variable occurs at most three times with $k$ clauses.

With Lemma 3.5.2 we get a product program of length $2^2 + 2^{2-1} - 2 = 4$ for computing a disjunction of two literals. In fact the program for $x_1 \vee x_2$ is given by $(1, (s, I_2)), (2, (r, I_2)), (1, (s, I_2)), (2, (r^{-1}, I_2))$ where $I_2$ is the $2 \times 2$ identity matrix in $\mathcal{R}^{2\times 2}$.

If we now combine the clauses with the product program from Theorem 3.5.3 we see that every variable which occurs at most three times in $\phi$ gets read at most 6 times in the product program. This gives us a bound of 6 for $\mathcal{I}_\ell$ from Corollary 3.5.5.

Hence, the cycle in the barber pole matrix and the corresponding graph have a length of at most 6 for the case of the Cayley permanent.

If we look at the Cayley determinant, we get a bound of 7 for the component size by the previous argument as we padded the cycle length to be odd. However, we can push this down to 6.

We redefine $M$ to be

$$M[i,j] = \begin{cases} (-1)^{|\mathcal{I}_k|-1} a_{i,0} & \text{if } i = i_{k,1},\ j = i_{k,2} \text{ for some } k, \\ a_{i,b} & \text{otherwise, if } j = \pi_b(i), \\ 0 & \text{otherwise.} \end{cases}$$

With this we can remove the restriction that $\mathcal{I}_\ell$ has to have odd size as the Cayley determinant of $M$ is equal to the Cayley permanent of the matrix originally defined in Theorem 3.5.4. □

As a side note, the matrix defined here is the original matrix used in Gentry's proof.

With this framework we can also prove completeness for a related polynomial which has not been studied in the non-commutative setting, the immanant. Of course, we know that the immanant is already hard for some young diagrams. However, we can show that it is hard, irregardless of the young diagram, in the non-commutative setting. Seeing that the Cayley determinant is already #P complete this is not a surprising result but Gentry's result allows us to prove this easily.

**Definition 3.5.** *Let $\lambda \vdash n$. We define the* Cayley immanant *to be*

$$\mathsf{Cayley\text{-}Im}(X)_\lambda = \sum_{\sigma \in S_n} \chi_\lambda(\sigma) x_{1,\sigma(1)} \dots x_{n,\sigma(n)}.$$

**Theorem 3.5.7.** *The Cayley immanant is #P-hard for graphs with components of size 6.*

*Proof.* The proof works as the proof for Theorem 3.5.4 except that we modify our matrix. Let $\pi_0$, $\pi_1$ and $i_{\ell,\iota}$ be defined as in the proof of Theorem 3.5.4. Let $\pi_{0,\ell}$ be the identity permutation $\pi_0$ restricted to the elements in $\mathcal{I}_\ell$ and $\pi_{1,\ell}$ the permutation $\pi_1$ restricted to elements in $\mathcal{I}_\ell$, namely $(i_{\ell,2}, i_{\ell,3}, \dots, i_{\ell,|\mathcal{I}_\ell|}, i_{\ell,1})$.

We then set our matrix to be

$$M[i,j] = \begin{cases} \frac{1}{\chi_\lambda(\pi_{0,\ell})} a_{i,0} & \text{if } i = i_{k,1},\ j = i_{k,1} \text{ for some } k, \\ \frac{1}{\chi_\lambda(\pi_{1,\ell})} a_{i,1} & \text{otherwise, if } i = i_{k,1},\ j = i_{k,2} \text{ for some } k, \\ a_{i,b} & \text{otherwise, if } j = \pi_b(i), \\ 0 & \text{otherwise.} \end{cases}$$

In essence, we replace the first values in the cycle for every block with a scaled value.

In the following we will only reason about the character. We know that $\chi_\lambda$ is invariant under permutations from Proposition 2.1 Item 1. This means we can reorder our block barber pole matrix to have the blocks separately. Now we can use Lemma 2.6.3 and only look at one block.

We know that this corresponds to either the cycle cover with all self-loops which has character $\chi_\lambda(\pi_{0,\ell})$ or the cycle cover with one loop which has character $\chi_\lambda(\pi_{1,\ell})$. We chosen the first entry of the matrix cleverly to cancel these factors. As our cycles are of constant size, these constants are easy to compute by bruteforce.

With this the correctness proof works as in Theorem 3.5.4. □

Of course, this subsumes the previous result (Theorem 3.5.6) as the Cayley immanant is equal to the Cayley determinant or Cayley permanent for specific partitions $\lambda$.

### 3.5.3 Other Hard Polynomials

It is known that computing the commutative permanent of the weighted adjacency matrix of a planar graph is as hard as the general case as seen by Datta, Kulkarni, et al. [DKLM10]. We observe that their reduction extends to the non-commutative case.

**Theorem 3.5.8.** *The following reductions work over any non-commutative algebra.*

- $\mathsf{Cayley\text{-}perm} \leq_m^p \mathsf{planar{-}Cayley\text{-}perm}$,
- $\mathsf{Cayley\text{-}det} \leq_m^p \mathsf{planar{-}Cayley\text{-}det}$.

*Proof.* The proof is essentially the same as in [DKLM10]. We give a brief sketch here for the sake of completeness. Let $G$ be a weighted digraph. Consider an arbitrary embedding $\mathcal{E}$ of $G$. Obtain a new graph by changing the graph as follows:

- For each pair of edges $(u,v)$ and $(u',v')$ that cross each other in the embedding $\mathcal{E}$, do the following:
- introduce two new vertices $a$ and $b$; and
- new edges $\{(a,b),(b,a),(u',a),(a,v),(u,b),(b,v')\}$ replacing $(u,v)$ and $(u',v')$.

Note that any of the iterations above do not introduce any new crossings, and hence the process terminates after at most $O(n^2)$ many steps where $n$ is the number of vertices in $G$. The weight of $(u,v)$ is given to $(v,a)$ and the weight of $(u',v')$ to $(v',b)$. The remaining edges have the weight 1. By the construction, we can conclude that $\mathsf{Cayley\text{-}perm}(G) = \mathsf{Cayley\text{-}perm}(G')$ and $\mathsf{Cayley\text{-}det}(G) = \mathsf{Cayley\text{-}det}(G')$. □

We demonstrate some more families of polynomials whose commutative variants are easy but certain non-commutative variants are as hard as the permanent polynomial. We begin with a non-commutative variant of the elementary symmetric polynomial. The elementary symmetric polynomial $\mathsf{Sym}_{n,d}$ is given by

$$\mathsf{Sym}_{n,d}(x_1,\ldots,x_n) = \sum_{\substack{S\subseteq[n]\\|S|=d}} \prod_{i\in S} x_i.$$

There are several non-commutative variants of the above polynomial. The first one is analogous to the Cayley permanent, i.e.,

$$\mathsf{Cayley\text{-}Sym}_{n,d} = \sum_{S=\{i_1<i_2<\cdots<i_d\}} \prod_{j=1}^{d} x_{i_j}.$$

It is not hard to see that the above mentioned non-commutative version of $\mathsf{Cayley\text{-}Sym}_{n,d}$ can be computed by depth 3 non-commutative circuits for every value of $d \in [n]$. However, the above definition is not satisfactory, since it is not invariant under permutation of variables, which is the inherent property of elementary symmetric polynomials. We define a variant of non-commutative elementary symmetric polynomial which is invariant under the permutation of variables.

$$\mathsf{nc\text{-}Sym}_{n,d}(x_1, \ldots, x_n) \coloneqq \sum_{\{i_1,\ldots,i_d\}\subseteq[n]} \sum_{\sigma\in S_d} \prod_{j=1}^{d} x_{i_{\sigma(j)}}.$$

We show that with coefficients from the algebra of $n \times n$ matrices allowed, $\mathsf{nc\text{-}Sym}_{n,d}$ cannot be computed by polynomial size circuits unless $\mathsf{VP} = \mathsf{VNP}$. For this we need a small definition introduced in [AJS09, AS10].

**Definition 3.6.** *The* Hadamard product *between two polynomials $f = \sum_m \alpha_m m$ and $g = \sum_m \beta_m m$, written as $f \odot g$, is defined as $f \odot g = \sum_m \alpha_m \beta_m m$ where we sum over all possible monomials $m$.*

We can now show the following theorem. Keeping in mind the fact that the non-commutative Hadamard product of a polynomial size circuit $f$ with a polynomial size ABP $g$ can be computed efficiently as proven by Arvind, Joglekar, and Srinivasan [AJS09].

**Theorem 3.5.9.** *Over any $\mathbb{K}$ algebra $\mathcal{R}$ containing the $n\times n$ matrices as a sub-algebra, $\mathsf{nc\text{-}Sym}_{n,n}$ does not have polynomial size arithmetic circuits unless* $\mathrm{perm}_n \in \mathsf{VP}$.

Suppose that $\mathsf{nc\text{-}Sym}_{n,n}$ has a circuit $C$ of size polynomial in $n$. We need to show that perm $\in \mathsf{VP}$. Let $X = (x_{i,j})_{1\le i,j\le n}$ be matrix of variables, and $y_1, \ldots, y_n$ be distinct variables different from $x_{i,j}$. In the commutative setting, it was observed by Gathen [Gat87] that $\mathrm{perm}(X)$ equals the coefficient of $y_1 \cdots y_n$ in the polynomial

$$P(X,Y) \coloneqq \prod_{i=1}^{n} \left( \sum_{j=1}^{n} x_{i,j} y_j \right) \tag{3.4}$$

over the polynomial ring $\mathbb{K}[x_{1,1}, \ldots, x_{n,n}]$. However, the same cannot be said in the case of non-commuting variables. If $x_{i,j}y_k = y_k x_{i,j}$ for $i,j,k \in [n]$, then in the non-commutative development of Equation (3.4), the sum of coefficients of *all* permutations of the monomial $y_1 \cdots y_n$ equals $\mathrm{perm}(X)$. Hence the value $\mathrm{perm}(X)$ could be extracted using a Hadamard product with $\mathsf{nc\text{-}Sym}_{n,n}(y_1, \ldots, y_n)$ and then substituting $y_1 = 1, \ldots, y_n = 1$. However, we cannot assume $x_{i,j}y_k = y_k x_{i,j}$, since the Hadamard product may not be computable under this assumption.

*Proof of Theorem 3.5.9.* Let $\ell = \sum_{i,j} x_{i,j}$ and $P$ as above. Now we argue that

$$\mathrm{perm}(X) = (\textsf{nc-Sym}_{n,n}(\ell y_1, \ldots, \ell y_n) \odot P)|_{y_1=1,\ldots,y_n=1}.$$

Given a permutation $\sigma \in S_n$, there is a unique monomial $m_\sigma = x_{1,\sigma(1)} y_{\sigma(1)} \cdots x_{n,\sigma(n)} y_{\sigma(n)}$ in $P$ containing the variables $y_{\sigma(1)}, \ldots, y_{\sigma(n)}$ in that order. Thus taking Hadamard product with $P$ filters out all monomials but $m_\sigma$ from the term $\prod_{i=1}^{n} \ell y_{\sigma(i)}$. The monomials where a $y_j$ occurs more than once are eliminated by $\textsf{nc-Sym}_{n,n}(\ell y_1, \ldots, \ell y_n)$. Thus the only monomials that survive in the Hadamard product are of the form $m_\sigma$ where $\sigma \in S_n$. Now substituting $y_i = 1$ for $i \in [n]$ we get

$$\mathrm{perm}(X) = (\textsf{nc-Sym}_{n,n}(\ell y_1, \ldots, \ell y_n) \odot P)|_{y_1=1,\ldots,y_n=1}.$$

Note that the polynomial $P(X, Y)$ can be computed by an ABP of size $O(n^2)$. Then, by [AJS09, AS10], we obtain an arithmetic circuit $D$ of size $O(n^2 \textsf{size}(C))$ that computes the polynomial $\textsf{nc-Sym}_{n,n} \odot P$. Substituting $y_1 = 1, \ldots, y_n = 1$ in $D$ gives the required arithmetic circuit for $\mathrm{perm}(X)$. □

Note that by considering the following signed variant of $\textsf{nc-Sym}_{n,n}$, we can obtain a result analogous to Theorem 3.5.9 with $\textsf{Cayley-det}$. Let

$$\textsf{snc-Sym}_{n,n}(x_1, \ldots, x_n) := \sum_{\sigma \in S_n} \textsf{sgn}(\sigma) \prod_{i=1}^{n} x_{i,\sigma(i)}.$$

**Corollary 3.5.10.** *Over a $\mathbb{K}$ algebra containing the algebra of $n \times n$ matrices,* $\textsf{snc-Sym}_{n,n}$ *is not in* VP *unless* Cayley-det *has polynomial size arithmetic circuits.*

The reader should notice, that $\textsf{snc-Sym}_n, n$ is not an ordered polynomial in the sense of [HWY10].

Barvinok [Bar96] showed that computing the permanent of an integer matrix of constant rank can be done in strong polynomial time. In a similar spirit, we explore the complexity of computing the Cayley permanent of bounded rank matrices with entries from $\mathbb{K} \cup \{x_1, \ldots, x_n\}$. We consider the following notion of rank for matrices with variable entries. Let $A \in (\mathbb{K} \cup \{x_1, \ldots, x_n\})^{n \times n}$. Then

$$\textsf{row-rank}(A) = \max_{a_1,\ldots,a_n \in \mathbb{K}} \textsf{rank}(A|_{x_1=a_1,\ldots,x_n=a_n}).$$

The column rank of $A$ is defined analogously. As opposed to the case of the commutative permanent, for any algebra $\mathcal{R}$ containing the algebra of $n \times n$ matrices over $\mathbb{K}$, we have:

**Corollary 3.5.11.** Cayley-perm *and* Cayley-det *of rank one matrices with entries from $\mathbb{K} \cup \{x_1, \ldots, x_n\}$ over any $\mathbb{K}$ algebra do not have polynomial size arithmetic circuits unless* perm $\in$ VP.

*Proof.* We will argue the case of $\mathsf{Cayley\text{-}perm}$. Let $x_1, \ldots, x_n$ be non-commuting variables. Consider the matrix $A$ with $A[i,j] = x_j$, $1 \leq i,j \leq n$. $A$ has rank one over $\mathbb{K}$. We then have $\mathsf{nc\text{-}Sym}_{n,n}(x_1, \ldots, x_n) = \mathsf{Cayley\text{-}perm}(A)$. The result now follows by applying Theorem 3.5.9. For $\mathsf{Cayley\text{-}det}$, we can use Corollary 3.5.10 in place of Theorem 3.5.9 in the argument above. □

## 3.6 Computational Problems

In this section, we look at non-commutative computation. Are there computational problems that are easy in the non-commutative case but hard in the commutative case?

### Computing Coefficients

We start with the problem of computing the coefficient of a given monomial in the polynomial computed by an arithmetic circuit. In the commutative setting, the problem lies in the second level of the counting hierarchy, proven by Fournier, Malod, and Mengel [FMM15], and is known to be hard for $\#P$ (see [Mal07]). It was first seen by Arvind, Mukhopadhyay, and Srinivasan [AMS10] that $\mathsf{mcoeff}$ is easy to compute in the non-commutative case by simply using the Hadamard Product. We provide a different proof of the fact as it is useful in our later arguments.

**Problem 3.1** (Monomial Coefficient ($\mathsf{mcoeff}$)).

***Input:*** *A non-commutative arithmetic circuit $C$ and a non-commutative monomial $m$ of degree $d$.*

***Output:*** *The coefficient of monomial $m$ in the polynomial computed by $C$.*

**Theorem 3.6.1** ([FMM15]). $\mathsf{mcoeff}$ *is in* $\mathsf{P}$.

*Proof.* Suppose that the monomial $m = x_{j_1} \cdots x_{j_d}$ is given as an ordered listing of variables. Let $f$ be a non-commutative polynomial. Then we have the following recursive formulation for the coefficient function $\mathsf{mc} : \mathbb{K}\langle x_1, \ldots, x_n \rangle \times \mathcal{M} \to \mathbb{K}$, where $\mathcal{M}$ is the set of all non-commutative monomials in variables $\{x_1 \ldots, x_n\}$. Let $m_\ell = x_{i_1} \cdots x_{i_{\ell-1}}$ and $m'_\ell = x_{i_\ell} \cdots x_{i_d}$. Then

$$\mathsf{mc}(f,m) = \begin{cases} \alpha & \text{if } f = \alpha y_j \text{ and } m = y_j, \\ 0 & \text{if } f = \alpha y_j \text{ and } m = y_i, i \neq j \ , \\ \mathsf{mc}(g,m) + \mathsf{mc}(h,m) & \text{if } f = g + h, \\ \sum_{\ell=1}^{d+1} \mathsf{mc}(g, m_\ell) \times \mathsf{mc}(h, m'_\ell) & \text{if } f = g \times h. \end{cases} \tag{3.5}$$

However, if we apply the above recursive definition on the circuit $C$ in a straightforward fashion, the time required to compute $\mathsf{mc}(f,m)$ will be $d^{O(\mathsf{depth}(C))}$, since $\mathsf{depth}(C)$ could be as big as $\mathsf{size}(C)$, the running time would be exponential. However, we can have a more careful implementation of the above formulation by allowing a little more space.

We will use a similar construction as computing homogeneous component for a circuit. For $\ell < k \in [1, d]$, let $m_{\ell,k} = x_{i_{\ell+1}} \cdots x_{i_k}$ a "slice" of the monomial $m$ from $\ell$ to $k$. Let $M = \{m_{\ell,k} \mid 0 \leq \ell \leq d-1, 0 \leq k \leq d\}$ be the set of all such slices. Consider a gate $v$ in the circuit $C$. Note that in the process of computing $\mathsf{mc}(f, m)$, we require only the values from the set $M_v = \{\mathsf{mc}(p_v, m') \mid m' \in M\}$, where $p_v$ is the polynomial computed at $v$. Thus it is enough to compute and maintain the values $\mathsf{mc}(p_v, m')$, where $m' \in M$, in a bottom up fashion. For the base case, compute the values for polynomials computed at a leaf gate $v$ as follows, let $m' \in M$ and $\alpha \in \mathbb{K}$. Then

$$\mathsf{mc}(p_v, m') = \begin{cases} \alpha & \text{if } p_v = \alpha \text{ where } \alpha \in \mathbb{K} \text{ and } m' = \emptyset, \\ \alpha & \text{if } p_v = \alpha x_j \text{ where } \alpha \in \mathbb{K} \text{ and } m' = x_j, \\ 0 & \text{otherwise.} \end{cases}$$

For other nodes, we can apply the recursive formula given in Equation (3.5). Let us start with an addition gate. Let $p_v = p_{v_1} + p_{v_2}$. Then

$$\mathsf{mc}(p_{v_1} + p_{v_2}, m') = \mathsf{mc}(p_{v_1}, m') + \mathsf{mc}(p_{v_2}, m')$$

by induction. Let $v$ now be a multiplication gate such that $p_v = p_{v_1} \cdot p_{v_2}$. Then the values $\mathsf{mc}(p_{v_i}, m'')$ are available for prefix and suffix of the monomial $m'$, as every such monomial occurs as $m_{i,j} \in M$ for some $i < j$. Now, $\mathsf{mc}(p_v, m')$ can be computed by Equation (3.5).

The algorithm increases the size of the circuit by at most $d^2$ at every gate as we keep separate gates for every possible $m' \in M$. Every computation at a gate needs at most $d$ many operations. Hence the overall size is at most $O(d^3 \mathsf{size}(C))$. □

## Coefficient function as a polynomial

In the commutative setting, the coefficient function of a given polynomial can be represented as a polynomial (cf. [Mal07]). Thus it is desirable to study the arithmetic circuit complexity of coefficient functions. However, over non-commutative rings, we need a carefully chosen representation of monomials to obtain an arithmetic circuit that computes the coefficient function for a given polynomial with small circuits. In the proof of Theorem 3.6.1, we have used an ordered listing of variables as a representation of the monomial $m$. Here we use a vector representation for non-commutative monomials of a given degree $d$. Let $Y = \{y_{1,1}, \ldots, y_{1,n}, y_{2,1}, \ldots, y_{d,n}\}$ be a set of $nd$ distinct variables, and let $\tilde{Y}_i = (y_{i,1}, \ldots, y_{i,n})$. The vector of variables $\tilde{Y}_\ell$ can be seen as representing the characteristic vector of $x_j$, i.e., $y_{i,j} = 1$, and $y_{i,j'} = 0$, $\forall j' \neq j$. In essence, $y_{i,j}$ stands for the variable $x_j$ at the $i$th position in the monomial. Let $f(x_1, \ldots, x_n)$ be a polynomial of degree $d$, then we can define the coefficient polynomial $\mathsf{pc}_f(Y)$ as

$$\mathsf{pc}_f(Y) = \sum_{D=1}^{d} \sum_{(i_1,\ldots,i_D)\in[n]^D} \prod_{\ell=1}^{D} \left( \mathsf{mc}(f, x_{i_1} \cdots x_{i_D}) y_{\ell,i_\ell} \prod_{j \neq k} (1 - y_{\ell,j} y_{\ell,k}) \right).$$

Here the last product just enforces that we have a valid assignment to the $Y$ variables.

**Theorem 3.6.2.** *For any non-commutative polynomial $f$ that can be computed by a polynomial size arithmetic circuit, $\mathsf{pc}_f(Y)$ has a polynomial size arithmetic circuit.*

*Proof.* We will apply Equation (3.5) to obtain an arithmetic circuit computing the polynomial $\mathsf{pc}_f(Y)$. Let $C$ be an arithmetic circuit of size $s$, computing $f$. By induction on the structure of $C$, we construct a circuit $C'$ for $\mathsf{pc}_f(Y)$. Note that, it is enough to compute homogeneous degree $D$ components $\mathrm{HOMC}_D^X(\mathsf{pc}_f(Y))$ for the variable set $X$ of $\mathsf{pc}_f(Y)$, where

$$\mathrm{HOMC}_D^X(pc_f(Y)) = \sum_{(i_1,\ldots,i_D)\in[n]^D} \prod_{\ell=1}^{D} \left( \mathsf{mc}(f, x_{i_1}\cdots x_{i_D}) y_{\ell,i_\ell} \prod_{j\neq k}(1 - y_{\ell,j}y_{\ell,k}) \right).$$

Let $Y^{i,j}$ denote the set of variables in the vectors $\tilde{Y}_{i+1},\ldots,\tilde{Y}_j$. In the base case, we have $C = \gamma \in \{x_1,\ldots,x_n\} \cup R$. Then the all of the homogeneous components of $\mathsf{pc}_f(Y)$ can be described as follows.

$$\begin{aligned}
\mathrm{HOMC}_0^X(\mathsf{pc}_f(Y)) &= \begin{cases} \gamma & \text{if } Y = \emptyset \text{ and } \gamma \in R, \\ 0 & \text{otherwise.} \end{cases} \\
\mathrm{HOMC}_0^X\, \mathsf{pc}_f(Y)) &= \begin{cases} 1 & \text{if } Y = e_j \text{ and } \gamma = x_j, \\ 0 & \text{if } Y = \emptyset \text{ and } \gamma \in R, \\ 0 & \text{otherwise.} \end{cases} \\
\mathrm{HOMC}_{i>1}^X(\mathsf{pc}_f(Y)) &= 0.
\end{aligned}$$

Naturally, the induction step has two cases: $f = g + h$ and $f = g \cdot h$.

**Case 1:** $f = g + h$, then for any $D$

$$\mathrm{HOMC}_D^X(\mathsf{pc}_f(Y)) = \mathrm{HOMC}_D^X(\mathsf{pc}_g(Y)) + \mathrm{HOMC}_D^X(\mathsf{pc}_h(Y)).$$

**Case 2:** $f = g \times h$, then for any $D$

$$\mathrm{HOMC}_D^X(\mathsf{pc}_f(Y)) = \sum_{i=0}^{d}\sum_{j=0}^{D} \mathrm{HOMC}_j^X(\mathsf{pc}_g(Y^{1,i}))\, \mathrm{HOMC}_{D-j}^X(\mathsf{pc}_h(Y^{i+1,d}))$$

where $Y = \tilde{y}_1,\ldots,\tilde{y}_d$.

The size of the resulting circuit $C'$ is $O(d^3\mathsf{size}(C))$, and $C'$ can in fact be computed in time $O(d^3\mathsf{size}(C))$ given $C$ as the input. □

### Partial Coefficient functions

For a given commutative polynomial let $f(X) = \sum_m c_m m$, the partial coefficient of a given monomial $m$ (cf. [Mal07]) is a polynomial defined as

$$\mathsf{pcoeff}(f, m) = \sum_{\substack{m' \\ m | m'}} c_{m'} \frac{m'}{m}.$$

We extend the above definition to the case of non-commutative polynomials as follows. Let $f$ be non-commutative polynomial, and $m$ a non-commutative monomial. Then

$$\mathsf{pcoeffr}(f, m) = \sum_{m' = m \cdot m''} c_{m'} m''.$$

Similarly, we can define

$$\mathsf{pcoeffl}(f, m) = \sum_{m' = m'' \cdot m} c_{m'} m''.$$

The corresponding computational problem can be defined in the following way.

**Problem 3.2** (Coefficient Polynomial ($\mathsf{pcoeffl}$,$\mathsf{pcoeffr}$))**.**

***Input:*** *A non-commutative arithmetic circuit $C$ computing a polynomial $f$, and a monomial $m$.*

***Output:*** *A non-commutative arithmetic circuit that computes* $\mathsf{pcoeffl}(f, m)$*, (*$\mathsf{pcoeffr}(f, m)$*).*

**Theorem 3.6.3.** $\mathsf{pcoeffl}$ *and* $\mathsf{pcoeffr}$ *can be computed by a deterministic algorithm with a running time of polynomial in* $\mathsf{size}(C)$*, $n$ and* $\deg(m)$*).*

*Proof.* The algorithm is similar to the proof of Theorem 3.6.1, except that we need to construct an arithmetic circuit rather than a value. We will only proof this theorem for $\mathsf{pcoeffr}$ as $\mathsf{pcoeffl}$ behaves similar. We use the following recursive formulation similar to Equation (3.5).

If $f = \alpha \in R \cup \{x_1, \ldots, x_n\}$ and $m = \emptyset$ then $\mathsf{pcoeffr}(f, m) = \alpha$. For the summation $f = g + h$ we compute $\mathsf{pcoeffr}(f, m) = \mathsf{pcoeffr}(g, m) + \mathsf{pcoeffr}(h, m)$. The final case to handle is a multiplication gate. We define shorthand for sets of variables. Let $m = x_1 \cdots x_d$, $m_i = x_1 \cdots x_i$ and $m'_i = x_{i+1} \cdots x_d$ the rest of the monomial. We define $m_0 = \emptyset$. Then

$$\mathsf{pcoeffr}(f, m) = \mathsf{pcoeffr}(g, m) \cdot \mathsf{pcoeffr}(h, \emptyset) + \sum_{i=0}^{d-1} \mathsf{mc}(g, m_i) \mathsf{pcoeffr}(h, m'_i).$$

With this formula we simply check all possibilities to extend the monomial $m_i$ with the monomial $m'_i$ to get the monomial $m$. However, we only take the coefficient from the left part and the complete coefficient from the right part. Remember that $\mathsf{mc}(g, m_i)$ computes the exact coefficient in $\mathcal{R}$ and not the partial coefficient.

The rest of the proof is analogous to that of Theorem 3.6.1 except that, we need to compute and store the values $\mathsf{mc}(p_v, m_{i,j})$, and $\mathsf{pcoeffr}(p_v, m_{i,j})$ for every gate $v$ in the circuit in a bottom up fashion. □

# 4 A Fixed Parameter Theory of Arithmetic Circuits

## 4.1 Introduction

The field of Parameterized Complexity was developed to give us new insight into why finding fast algorithms for known and important problems is difficult. Many researchers before observed that a number of different problems exhibit some set of instances which are solvable in polynomial time. These instances can in general be characterized by having a specific parameter being small or constant. While we can prove theorems with these restrictions, the restrictions are rather ad-hoc and do not provide a consistent framework. Especially if we bound a running time of $n^k$ versus $2^k n^c$ which are similar if $k$ is constant but a running time of $2^k n^c$ is desired if $k$ can be asymptotically larger. To remedy this situation, Downey and Fellows [DF95a, DF95b, DF93] and Abrahamson, Downey, and Fellows [ADF95] as well as subsequent papers molded this distinction into a cohesive framework. They called the framework Fixed Parameter Tractable (FPT). With this they founded one of the exciting new research directions in Complexity Theory. And the field of FPT algorithms grows in importance steadily. With this popularity new parameters for old problems and new improved algorithms in this setting are found constantly.

As already hinted at, in our running time example, we can also define a notion of hardness. Intuitively, this will correspond to algorithms that do not have a running time of $2^k n^c$ but only $n^k$.[1] Notice that a running time of $n^k$ roughly corresponds to iterating over all sets of size $k$ and checking a condition with constant running time. We call such algorithms to not be fixed parameter tractable for a specific parameter. Beyond this intuition there exists a complete hierarchy of hardness, similar to the polynomial hierarchy. We will focus on this hardness aspect in this chapter.

While this field was initially only made for decision versions of problems, it was later extend to encompass counting algorithms independently by McCartin [McC03] and Flum and Grohe [FG04]. Later this theory was generalized to a framework based on logic by the second authors [FG06]. As with fixed parameter tractability, the counting version is an active research area and new and surprising results are found often.

We want to transfer the field of fixed parameter tractability to arithmetic circuit complexity. With the well known relationship between arithmetic circuit complexity

[1] As most of the time in Complexity Theory, we will conjecture that $\mathsf{FPT} \neq \mathsf{W}[1]$ but fail to prove it. Hence problems in $\mathsf{W}[1]$ will have algorithms with a running time of $n^k$ but it is unclear if they have algorithms with a running time of $f(k)n^c$ for some function $f : \mathbb{N} \to \mathbb{N}$.

of Generating Functions and counting complexity the theory should transfer. While just extending the FPT framework to arithmetic circuit is interesting in itself, there are many more natural questions we can try to answer. One of the core questions is the following. *What makes specific polynomials hard to compute?* We can see the study of arithmetic FPT algorithms and arithmetic FPT hardness to give a partial answer to this question. As we can see and prove directly how the complexity grows in specific parameters we can form an understanding how these parameters influence the complexity of the polynomial. We will first study specific generating functions, based on well known counting problems in the FPT world such as Independent Set or Dominating Set. This will base our framework in the familiar FPT framework while at the same time gives us an indication of the correctness of our newly introduced definitions. Additionally, we can ask the following question. *Does the same parameter as in the counting setting make these polynomials hard?*

The second major question we want to ask is about the immanant. As it “interpolates” between hard and easy polynomials, can we perhaps find an FPT algorithm in our framework?

We will give a recap of parameterized complexity in Section 4.2. Sections 4.3 and 4.4 define the basic arithmetic circuit FPT class as well as a notion of kernelization.

We then continue with the major definitions about fixed parameter hardness. We give a first definition for the $\mathsf{BVW}[t]$ hierarchy where most hardness results transfer (Section 4.5). This definition will be heavily based on the counting versions. However, this is rather unsatisfactory as they do not use the full power of arithmetic cricuits.. We will use this as a motivation to define $\mathsf{VW}[t]$. We give such definitions and show complete problems for our newly defined classes $\mathsf{VW}[1]$, $\mathsf{VW}[2]$, $\mathsf{VW}[3]$ (Section 4.6). Finally, we give in Section 4.7 two short proofs about the complexity of the immanant in this setting.

## 4.2 Parameterized Complexity

### 4.2.1 A Recap of Boolean Parameterized Complexity

In this section we will recap basic definitions from parameterized complexity theory. This will guide us to our new definition and show similarities and differences between the two settings. The major definitions and theorems can be found in the book by Flum and Grohe [FG06], the classic book by Downey and Fellows [DF99] or the new book by Downey and Fellows [DF13]. However, we will adjust some definitions for clarity.

**Definition 4.1.** *Let $\Sigma$ be a finite alphabet and $\kappa : \Sigma^* \to \mathbb{N}$ and $Q \subseteq \Sigma^*$. We call a tuple $(Q, \kappa)$ a* parameterized problem*.*

The central definition of fixed parameter tractability is as follows.

**Definition 4.2.** *Let $p(n)$ be a polynomially bounded function and $f(k)$ a computable function. A parameterized problem $(Q, \kappa)$ is* fixed parameter tractable *if there exists an algorithm $A$ that decides $Q$ and $A$ has a running time of $p(|x|)f(\kappa(x))$.*

With this we can present our first problem that is fixed parameter tractable. Let us remember that a vertex cover for a given graph $G$ is a set of vertices such that every edge is incident to at least one vertex in the set.

**Problem 4.1.** *Let $L$ be the set of all $(G, k)$ where $G$ is a graph, $k \in \mathbb{N}$ and $G$ has a vertex cover of size $k$. Then $(L, \kappa)$ where $\kappa((G, k)) = k$ is the parameterized problem we denote by p-Vertex Cover.*

**Theorem 4.2.1.** *p-Vertex Cover is fixed parameter tractable.*

We also define two notions of reductions.

**Definition 4.3.** *Let $(Q, \kappa)$, $(Q', \kappa')$ be two parameterized problems. We say $(Q, \kappa)$ reduces to $(Q', \kappa')$ with the reduction algorithm $R : \Sigma^* \to \Sigma^*$ and call this a* fpt many-one reduction*, written $(Q, \kappa) \leq_{fpt} (Q', \kappa')$, if the following holds:*

- *For all $x \in Q' \Leftrightarrow R(x) \in Q$.*
- *$R$ has an algorithm with running time bounded by $f(\kappa'(x))p(|x|)$ for every $x \in \Sigma^*$ for some polynomially bounded function $p(n)$ and some computable function $f(n)$.*
- *There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\kappa(R(x)) \leq g(\kappa'(x))$ for all $x \in \Sigma^*$.*

**Definition 4.4.** *Let $(Q, \kappa)$, $(Q', \kappa')$ be two parameterized problems. We say $(Q, \kappa)$ reduces to $(Q', \kappa')$. We call this a* fpt Turing reduction*, written $(Q, \kappa) \leq_{fpt,c} (Q', \kappa')$, if the following holds:*

- *$R$ is an algorithm with oracle access to $Q'$.*
- *$R$ decides $(Q, \kappa)$.*
- *$R$ is an algorithm with runtime $f(\kappa'(x))p(|x|)$ for a polynomial $p(n)$ and a computable function $f(k)$ on input $x \in \Sigma^*$.*
- *There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all oracle queries $y$ posed by $R$ on input $\kappa(y) \leq g(\kappa'(x))$.*

While we will not use Turing reductions in this chapter but we will later build our arithmetic reduction on this model.

Equivalently to fixed parameter tractability, there exists the notion of a fixed parameter kernel.

**Definition 4.5.** *Let $(Q, \kappa)$ be a parameterized problem. A polynomial time computable function $K : \Sigma^* \to \Sigma^*$ is a* kernelization *of $(Q, \kappa)$ if there is a computable function $h : \mathbb{N} \to \mathbb{N}$ such that for all $x \in Q$ that*

$$x \in Q \Leftrightarrow K(x) \in Q) \text{ and } |K(x)| \leq h(\kappa(x)).$$

*We call $K(x)$ the* kernel.

In essence, we can use kernels to decrease the size of the instances.

**Theorem 4.2.2** (see [Nie02])**.** *For every parameterized problem $(Q, \kappa)$, the following statements are equivalent:*

1. $(Q, \kappa) \in \mathsf{FPT}$.
2. *The language $Q$ is decidable and $(Q, \kappa)$ has a kernelization.*

We can now continue with defining the hard classes of parameterized complexity. We will follow closely the original definitions by Downey and Fellows as they give circuit constructions. While the new definition by Flum and Grohe are cleaner, the circuit model is easier to transfer to the arithmetic world.

**Definition 4.6.** *Let $C$ be a boolean circuit of constant depth $d$ with unbounded fan-in. We call this circuit having* weft *$t$ if there are at most $t$ layers where at least one gate in the layer has fan-in greater than two.*

*We call a family of boolean circuits to have weft $t$ if all circuits in the family have weft $t$.*

Let us remember that the hamming weight of an assignment $\phi$ is the number of variables set to true.

**Problem 4.2.** *Let $Q$ be the set of all $(C, k)$ such that $C$ is a boolean circuit which has a hamming weight $k$ satisfying assignment. We call the parameterized problem $(Q, \kappa)$ where $\kappa(((C_n), k)) = k$, $\mathsf{WSat}(C)$.*

**Theorem 4.2.3.** *The class $\mathsf{W}[t]$ is defined by all parameterized problems $(Q, \kappa)$ such that there exists a fpt reduction to $\mathsf{WSat}(C)$ for a family of boolean circuits $(C_n)$ which have depth $d$ and weft $t$ for some constants $d$ and $t$.*

We will introduce the notion of a $t$-normalized boolean circuit. This will be a mutal recursive definition as follows.

We call a boolean circuit 1-$\wedge$-normalized if it is of the form $\wedge_i L_i$ for literals $L_i$. We call a boolean circuit 1-$\vee$-normalized if it is of the form $\vee_i L_i$ for literals $L_i$.

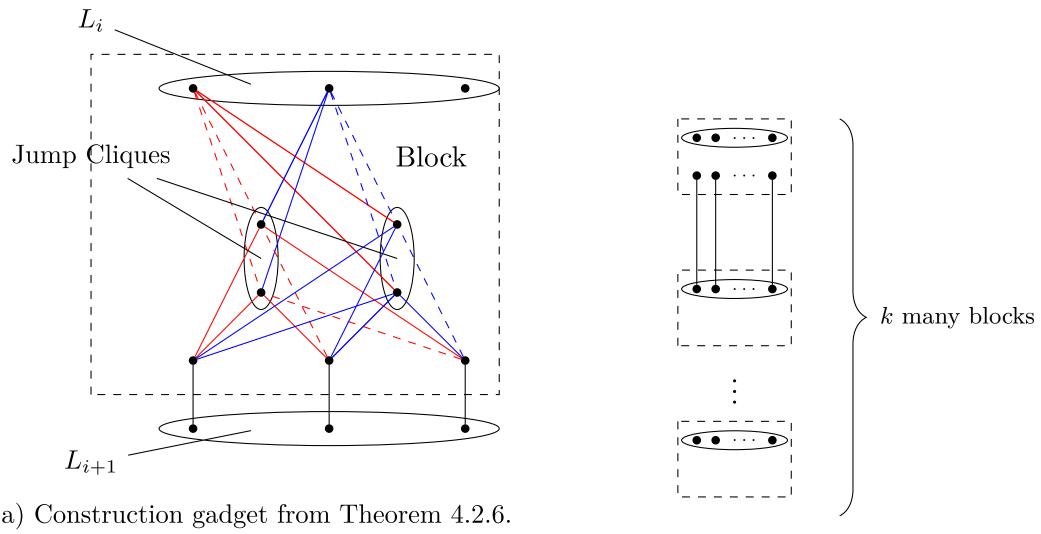
We call a circuit $t+1$-$\wedge$-normalized if it is of the form $\wedge_i C_i$ and $C_i$ are $t$-$\vee$-normalized circuits. Similarly, we call a circuit $t+1$-$\vee$-normalized if it is of the form $\vee_i C_i$ and $C_i$ are $t$-$\wedge$-normalized circuits.

Finally, we can define a $t$-normalized circuit for $t \in \mathbb{N}$ if it is either a $t$-$\wedge$-normalized circuit or a $t$-$\vee$-normalized circuit.

The following problem is the general complete problem.

**Problem 4.3.** *Let $Q$ be the set of all $(C, k)$ such that $C$ is a $t$-Normalized boolean formula and $C$ has a hamming weight $k$ satisfying assignment. Then we denote by $p$-*Weighted $t$-Normalized Satisfiability *the parameterized problem of $(Q, \kappa)$ where $\kappa((C, k)) = k$.*

This is essentially a specialized form of $\mathsf{WSat}(C)$.



(a) Construction gadget from Theorem 4.2.6.

(b) Combining the blocks.

Figure 4.1

**Theorem 4.2.4** (Normalization Theorem [DF95a]). *For all $t \geq 2$, p-Weighted t-Normalized Satisfiability is complete for* $\mathsf{W}[t]$.

With this the following natural complete problems are known. An Independent Set is a set of vertices such that no two vertices in the set are connected by an edge.

**Problem 4.4.** *Let $Q$ be the set of all $(G, k)$ such that $G$ is a graph and $G$ has an Independent Set of size exactly $k$. Then we call the parameterized problem $(Q, \kappa)$ where $\kappa((G, k)) = k$* $p$-Independent Set.

**Theorem 4.2.5** ([DF95b]). *p-Independent Set is complete for* $\mathsf{W}[1]$.

A Dominating Set of a graph is a set of vertices $S$ such that for every vertex either the vertex is in $S$ or $N(v)$, the neighbourhood of $v$, contains at least one vertex in $S$.

**Problem 4.5.** *Let $Q$ be the set of all $(G, k)$ where $G$ is a graph and $G$ has a Dominating Set of size exactly $k$. We then call the parameterized problem $(Q, \kappa)$ where $\kappa((G, k)) = k$ the* $p$-Dominating Set *problem.*

**Theorem 4.2.6** ([DF95a]). *p-Dominating Set is complete for* $\mathsf{W}[2]$.

We will describe the original reduction by Downey and Fellows [DF95a] as we later modify it for our purposes. Our construction will select $2k$ vertices such that they are in lexicographical order. We enforce this by having $k$ layers with $n$ vertices which select the vertices. These layers will be connected to other sets of vertices (which we later call jump cliques) which denote the gap between the current vertex and the next selected vertex in this order.

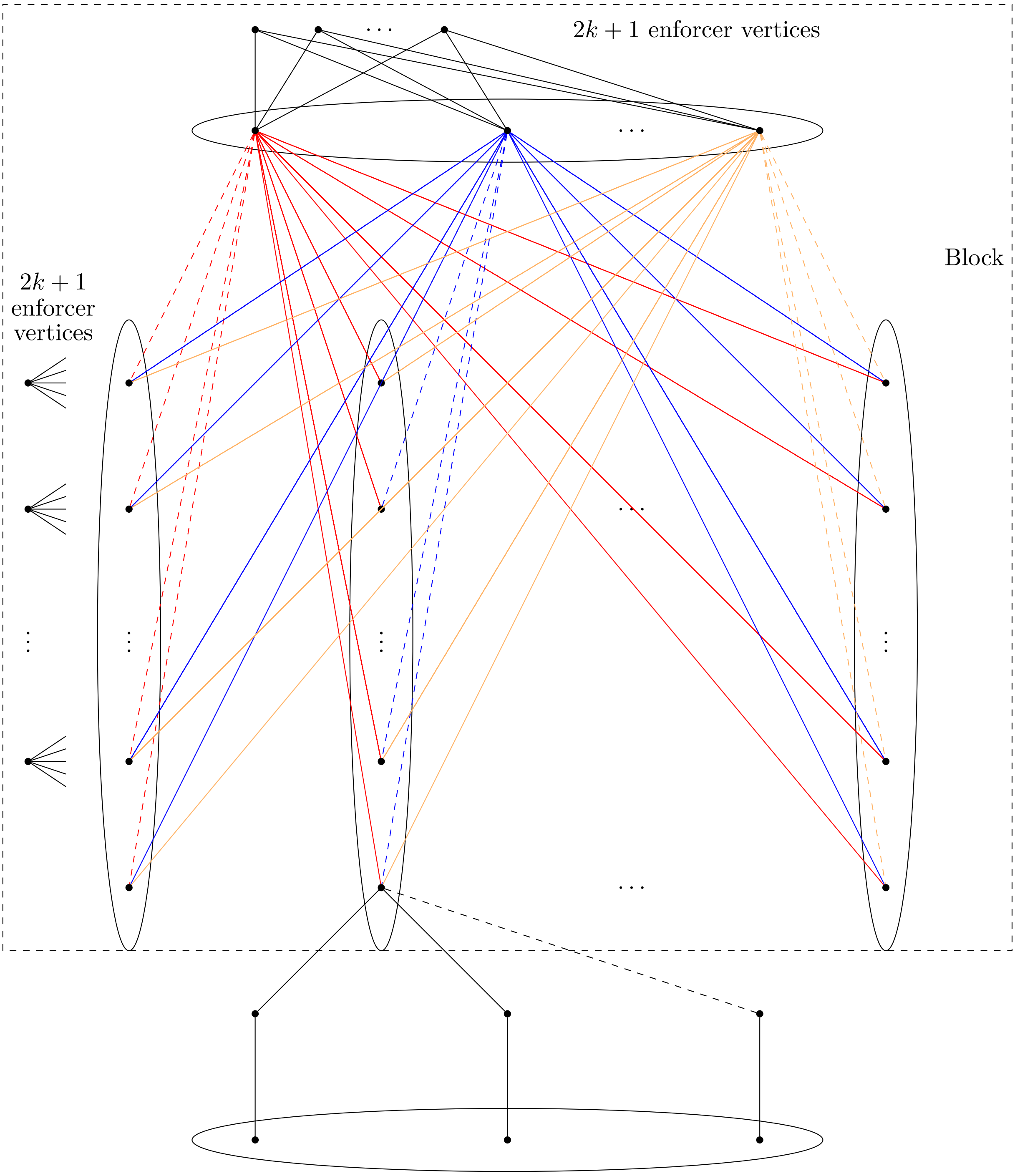


Figure 4.2: Partial construction from Theorem 4.2.6 in combined form.

*Proof.* We construct $nk$ vertices in layers and call them $L_i$ with vertices $v_{i,j}$ for $1 \leq i \leq k$ and $1 \leq j \leq n$. We then add for every vertex $v_{i,j}$ new vertices we denote by $u_{i,j,\nu}$ for $1 \leq \nu \leq n$. We will call these vertices the jump cliques and denote them by $C_{i,j} = \{u_{i,j,\nu} \mid 1 \leq \nu \leq n\}$. Additionally, we introduce new vertices we call $w_{i,j}$ for $2 \leq i \leq k$, $1 \leq j \leq n$. We denote by $L'_i = \{w_{i,j} \mid 1 \leq j \leq n\}$ and connect vertex $w_{i,j}$ in $L'_i$ with vertex $w_{i+1,j}$ in $L_{i+1}$ for all $1 \leq i \leq n-1$ and for all $j$.

For all $i$, we connect the vertex $v_{i,j}$ with all vertices in $C_{i,\mu}$ for $\mu \neq j$. We then connect a vertex $u_{i,j,\nu}$ with every vertex in $L'_{i+1}$ except the $\nu + j + 1$th one, namely we add edges $\{u_{i,j,\nu}, w_{i+1,\mu}\}$ where $\mu \neq j + \nu + 1$. The construction is almost complete. We are only missing some way to enforce that we have to take a vertex for every $i$ in $L_i$.

We connect vertices in $C_{i,j}$ for all $i, j$ such that $C_{i,j}$ is a clique and add in a same way edges to $L_i$ such that $L_i$ is a clique for every $i$. We now add for every $L_i$ $2k+1$ enforcer vertices, namely $2k+1$ new vertices connected to every vertex in $L_i$. We add another $2k+1$ enforcer vertices for every $C_{i,j}$ for all $i$ that is connected to every vertex in $C_{i,j}$ for all $j$.

The construction of one of these gadgets is given in Figure 4.1a with only three $y$ variables where we did not draw the edges and vertices for $y_3$ and removed the enforcer vertices. Additionally, we marked edges missing by dashed lines. Figure 4.1b gives an overview how the layers are connected and Figure 4.2 gives us a zoomed in view of one gadget.

$v_{i,j}$ will correspond to a variable $x_i$. We now have to use the formula $\wedge_i \left(\neg \wedge_j \neg x_{\varphi(i,j)}\right)$ which is given as our input to the reduction where $\varphi$ maps the indices $i, j$ to the index of $x$ in the formula. We rewrite it as $\wedge_i \left(x_{\varphi(i,1)} \vee \cdots \vee x_{\varphi(i,m)}\right)$. Now for every $i$ we construct a single new vertex that is connected to the following vertices.

$$\{v_{\varphi(i,1),\nu} \mid 1 \leq \nu \leq k\} \cup \cdots \cup \{v_{\varphi(i,m),\nu} \mid 1 \leq \nu \leq k\}.$$

If we see the set of $v_{i,j}$ as one vertex splitted by $j$, we connected it to all vertices in our $\wedge$ gate.

We skip the correctness proof as it can be found in the literature. Informally, we can argue the following way. The enforcer vertices enforce that every Layer $L_i$ has to have at least one vertex selected. The upper bound of $2k$ enforces that exactly one vertex in every layer has to be selected. A Dominating Set has to pick an ordered set of vertices from the different $L_i$. Picking a vertex in a jump cliques $C_{i,j}$ defines how many vertices are skipped from the current selected one to the next. Finally, the construction of the vertex $c_i$ enforces that we have to select a valid assignment for the formula. □

An *Independent Dominating Set* is an Dominating Set that is also an Independent Set. Unsurprisingly, Independent Dominating Set is complete for W[2] as well. We can even use the same reduction as in the Dominating Set case.

**Theorem 4.2.7.** *p-Independent Dominating Set is complete for* W[2].

We will need a graph problem complete for $\mathsf{W}[3]$. We chose the one defined by Chen and Zhang [CZ06]. We call a set of vertices a *$k$-3-SCM Single-Product Cover* on a graph $G = (S_1 \uplus \cdots \uplus S_k \cup M \cup R, E)$ if when selecting $k$ vertices out of $S$ every vertex in $R$ is covered. We call a vertex in $R$ covered if there exists an edge $(u, v) \in R \times M$ such that $v$ is covered. We call a vertex $v$ in $M$ covered if for all $1 \leq i \leq k$ there exists a $u \in S_i$ that is covered and the edge $(u, v)$ exists.

In words, a manufacturer needs $k$ different materials, where material $i$ is produced by all suppliers in $S_i$, to produce a product. A product can be at a retailer if it has a connection to a manufacturer producing it.

**Problem 4.6.** *Let $Q$ be the set of all $(G, k)$ that have a $k$-3-SCM Single-Product Cover. We call the parameterized problem $(Q, \kappa)$* $p$-3-SCM Single-Product Cover *where $\kappa((G, k)) = k$.*

**Theorem 4.2.8.** *p-3-SCM Single-Product Cover is complete for* $\mathsf{W}[3]$.

We will give a sketch of the proof for completeness.

*Proof.* By the Normalization theorem (Theorem 4.2.4) we can just reduce from a $\bigwedge\bigvee\bigwedge$ formula. Notice, that the formula for 3-SCM Single-Product Cover is given by

$$\bigwedge_{\nu\in R}\ \bigvee_{\mu\in M_\nu}\ \bigwedge_{i=1}^{k}\ \bigvee_{v\in S_i\cap N(\mu)} v.$$

We can transform this into a circuit of the form

$$\left(\bigwedge_{\nu\in R}\ \bigvee_{\mu\in M_\nu}\ \bigwedge_{i=1}^{k}\ \bigwedge_{v\in S_i\setminus N(\mu)} \neg v\right)\bigwedge_{i=1}^{k}\ \bigvee_{v\in S_i} v.$$

In essence, this formula enforces that every $S_i$ has at least one vertex selected by the second part. The first part now ensures that for every $k$ and for every $v$ which is not in our neighbourhood $v$ is unselected. As this is now a $\bigwedge\bigvee\bigwedge$ antimonotone formula the reduction is clear. □

### 4.2.2 A Recap of Parameterized Counting Complexity

With a basic understanding of the boolean parameterized complexity we can shift our gaze to the counting version. As we know that computing polynomials with arithmetic circuits is related to counting problems, we shall spent some time on understanding the basic parameterized definitions as well as the used reductions in this world.

**Definition 4.7.** *We call $(F, \kappa)$ where $F : \Sigma^* \to \mathbb{N}_0$ and $\kappa : \Sigma^* \to \mathbb{N}$ a* parameterized counting problem.

**Definition 4.8.** *We call a parameterized counting problem $(F, \kappa)$* fixed parameter tractable *if there exists an algorithm that computes $F(x)$ on input $x \in \Sigma^*$ with a running time bounded by $f(\kappa(x))\mathsf{poly}(|x|)$ for some computable function $f : \mathbb{N} \to \mathbb{N}$.*

As an example we can look at the vertex cover problem. Here we have to count the number of vertex cover of size $k$ and as expected this is fixed parameter tractable.

Let us look at the different definitions for the reductions.

**Definition 4.9.** *Let $(F, \kappa)$ and $(F', \kappa')$ be two parameterized counting problems.*

1. *A* fpt parsimonious *reduction from $(F, \kappa)$ to $(F', \kappa')$, written $(F, \kappa) \leq_{\#fpt,p} (F', \kappa')$, is a mapping $R : \Sigma^* \to \Sigma^*$ such that:*
   - *For all $x \in \Sigma^*$ we have $F(x) = F'(R(x))$.*
   - *$R$ is computable by an fpt algorithm with respect to $\kappa'(x)$.*
   - *There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for every $\kappa(R(x) \leq g(\kappa'(x))$.*
2. *A* fpt Turing reduction *from $(F, \kappa)$ to $(F', \kappa')$, written $(F, \kappa) \leq_{\#fpt,c} (F', \kappa')$, is an algorithm $A$ with oracle to $F'$ such that:*
   - *$A$ on input $x \in \Sigma^*$ is an fpt algorithm with respect to $\kappa'(x)$.*
   - *$A$ computes $F(x)$.*
   - *There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all oracle queries $y$ posed by $A$ on input $x$, $\kappa(y) \leq g(\kappa'(x))$.*

**Problem 4.7.** *Let $p$-$\#\mathsf{WSat}((C, k))$ be the problem of counting the number of $k$ hamming weight satisfying assignments of a boolean circuit $(C)$.*

**Definition 4.10.** *We can now define $\#\mathsf{W}[t]$ to be the class of all parameterized counting problems $(F, \kappa)$ which are in*

$$[\#\mathsf{WSat}((C, k))]^{\leq_{\#fpt,p}}$$

*where $C$ is a boolean circuits of weft $t$ and constant depth and $[\circ]^{\leq_{\#fpt,p}}$ is the closure under parsimonious fpt reductions.*

We now continue with two important results which are not surprising. They were first shown in the thesis by McCartin [McC03] but are heavily influenced by the previously stated result by Downey and Fellows [DF95a]. We will only give a brief sketch of the proofs here and make particularly note of the relations between the witnesses in the reduction. We especially skip the size bounds, the bounds on $k$ and the correctness arguments. We start with the proof for $\#p$-Independent Set which is just the analogue counting problem to $p$-Independent Set where $F((G, k))$ now counts the number of independent sets of size $k$ in $G$.

**Theorem 4.2.9.** *$\#p$-Independent Set is complete for $\#\mathsf{W}[1]$.*

The proof is divided into three lemmas. We define $\#\mathsf{W}[1, s]$ to be all counting problems that have a circuits of weft 1 and depth 2 with the $\vee$ gate on level 1 having fan-in bounded by $s$. Let us call the witness function $f : \{0, 1\}^* \to \mathcal{P}(\{0, 1\}^*)$ the function that maps instances to sets of witnesses.

**Lemma 4.2.10.** *If we have a circuit of weft 1 and constant depth then this is reducible to a family in* $\#\mathsf{W}[1, s]$*.*

*Proof (Sketch).*

**Step 1** We transform the circuit to a tree circuit. The witness function remains unchanged.

**Step 2** We move the not gates to the bottom of the circuit. The witness function remains unchanged.

**Step 3** Normalize the circuit to depth four. The step can be seen in [DF13, Theorem 21.2.1]. On page 651 they mention explicitly that the witness function remains unchanged after this step.

In the proof they transform a constant size circuit into a $\wedge\vee$ or $\vee\wedge$ circuit respectively where the gates have fan-in greater than 2. However, they can still be counted as small gates as the fan-in is bounded by a constant.

**Step 4** Employ additional variables to transform the circuit into a $\wedge\vee$ circuit. In [DF13, p. 656] it is explicitly mentioned that given a witness for the transformed circuit $C'$ of witness size $k'$ that then the restriction of $v$ of $v'$ is a witness for the original circuit of witness size $k$.

In the proof they employ additional variables which check that the constant size $\wedge\vee$, $\vee\wedge$ respectively, circuit is fulfilled.

□

We define a circuit to be antimonotone if all input variables are negated and the circuit has no other negations. We call the class Antimonotone $\#\mathsf{W}[1, s]$ to be the class of all antimonotone circuits that are in $\#\mathsf{W}[1, s]$.

**Lemma 4.2.11.** $\#\mathsf{W}[1, s] =$ *Antimonotone* $\#\mathsf{W}[1, s]$*,* $\forall s \geq 2$*.*

*Proof.* We will not discuss the whole proof here as it is a reduction to a new problem called Red-Blue Nonblocker. This problem will be hard for $\#\mathsf{W}[1, s]$ and belongs to Antimonotone $\#\mathsf{W}[1, s]$. The reduction shows the hardness of the problem. In essence, we have sets $A(0), \dots, A(k-1)$ of vertices that will give our assignment. Let $\tau$ be the selected vertices in the Red-Blue Nonblocker graph and $\tau'$ a satisfying assignment. In [McC03, p. 112f] we can see that any solution of Red-Blue Nonblocker uniquely corresponds to an assignment $\tau'$ where if a variable $i$ is set to true there exists a vertex in $A(i)$ selected in $\tau$.

Hence, there exists a projection to transform the new witness function into the old witness function. □

**Lemma 4.2.12.** $\#\mathsf{W}[1] = \#\mathsf{W}[1, 2]$.

*Proof.* Now we show that a problem in Antimonotone $\#\mathsf{W}[1, s]$ for $s \geq 2$ can be solved in $\#\mathsf{W}[1, 2]$. In this reduction the input variable $x[i]$ is replaced by a set of variables $x[i, j]$ for $0 \leq j \leq 2^k - 1$. Again by [McC03, p. 114] it follows that a if $x[i]$ was true in the previous circuit all $x[i, j]$ are true. As the implication $x[j, r] \rightarrow x[j, r + 1 \mod 2^k]$ is enforced it is again clear that the old witness function can be given by a projection of the new witness function. □

We can now prove the theorem.

*Proof of Theorem 4.2.9.* Taking an arbitrary circuit and using Lemmas 4.2.10 to 4.2.12 we get a circuit of the form $\bigwedge_{\{(i,j)\}\in S} (\neg v_i \vee \neg v_j)$ for some set $S$. We can easily transform this into an undirected graph where we add a vertex for every $v_i$ and add an edge between $v_i$ and $v_j$ if $\{(i, j)\} \in S$. □

A similar normalization theorem as in the decision setting also holds in the counting case.

**Theorem 4.2.13** ([McC03])**.** *For all $t \geq 2$, counting the number of Weighted t-Normalized Satisfiability is complete for $\#\mathsf{W}[t]$.*

We will use the shorthand notation of $\bigwedge^n$ and $\bigvee^n$ for unbounded fan-in gates.

*Proof (Sketch).* The reduction is based on transforming a circuit in a few basic steps.

**Step 1** Transform the circuit into a tree. The witness function is unchanged.

**Step 2** Transfer the negation gates to the bottom. The witness function is unchanged.

**Step 3a** We describe this reduction briefly. We want to transform the circuit into a circuit with a unbounded $\wedge$ gate on top. If this is not already the case, we transform the top layer of bounded $\vee$ and $\wedge$ gates into a $\wedge\vee$ gate. We then introduce new variables that for an input originating from an unbounded $\vee$ gate and from an unbounded $\wedge$ gate. We then enforce that for a $\vee$ gate the least index that is one is represented in our new variables. If it originates from a large $\wedge$ or input gate we take the smallest index such that the gate evaluates to zero. From these two inputs we can then reconstruct with a large $\wedge$ gate the value of the bounded $\vee$ gate. We then enforce the above described behaviour with a large $\wedge$ gate which collapse with the constant size $\wedge$ gate. For a detailed proof the reader should consult the original construction.

As we keep the old variables, the witness function of the circuit before the transformation is a projection of the witness function after the transformation.

**Step 3b** We iteratively do the following four procedures:

- Contract one layer of multiple small gates to a $\wedge^c\vee^{c'}$ or $\vee^c\wedge^{c'}$ construction for constants $c$, $c'$.
- Contract small gates into large gates of the same type.

- Transform $\vee^c\wedge^n$ into $\wedge^n\vee^c$ by distributive law.
- Transform $\vee^c$ with inputs from large $\wedge$ and $\vee$ gates with $\wedge^n\vee^n$ which now permits a contraction.

Through all this the witness function remains unchanged.

**Step 4** If it exists, remove the final large $\vee$ gate in the following way. Let $m$ be the fan-in of the large $\vee$ gate. We then copy the circuit $m$ times and add new variables $x_1, \dots, x_m$. We then enforce that exactly one $x_i$ is set to true and that when $x_i$ is true the subcircuit $C_i$ has to evaluate to true. The last thing we enforce is that $x_i$ is the smallest $i$ such that $C_i$ evaluates to true. For this we use the formula $\wedge_{i=1}^{m} \wedge_{j=1}^{i-1} (\neg x_i \vee C_j)$. There exists a projection from the new witness function to the old witness function.

**Step 5** Repeat step 1 and step 2 if necessary. Additionally, if the lowest large gate is a $\wedge$ replace the circuit of small gates with a $\wedge\vee$ gate. If the lowest large gate is an $\vee$ gate, replace the circuit of small gates with a $\vee\wedge$ gate. After the replacement we collapse gates of the same type.

This does not change the witness function.

**Step 6** Let $x_1, \dots, x_m$ be the inputs of the circuit. Take the graph from the Dominating Set reduction with inputs $z_1, \dots, z_{m'}$. By using the vertices and the Skip vertices we can unify if a variable has positive or negative connections. For each positive connection from $x_i$ add a large unbounded fan-in $\vee$ gate over all positive values. For each negative connection from $x_i$ take a large $\vee$ gate of all values that imply $x_i$ is false. Then we connect the to the top product gate a circuit corresponding to the Dominating Set formula.

Notice that while the witness function changes, there exists a projection to the old witness function as we still have to set the correct variables from $x_1, \dots, x_m$ to true.

**Step 7** Eliminating remaining small gates. We are now in two different cases. Either $t$ is even then the small gates at the bottom are $\wedge$ and $C$ is monotone, meaning all literals are positive. Or $t$ is odd and then the small gates are $\vee$ and $C$ has all inputs going to the level one small gates negated. With this we can gather the inputs into new variables which will represent the $\wedge$ or $\vee$ value of the set of variables and enforce the variables representing the sets to have the correct value.

The witness function after the transformation is again just a projection of the witness function before the transformation.

□

With this and the proof for the decision version, as well as the discussion about the witness function, we know that counting the number of Dominating Sets is complete. The definitions of the next two problems is obvious.

**Theorem 4.2.14.** *#p-Dominating Set is complete for* $\#\mathsf{W}[2]$.

Again with the same reduction as in the Dominating Set case, we can show the completeness for counting Independent Dominating Sets. The witness function behaves as in the $\#p$-Dominating Set case.

**Theorem 4.2.15.** *#p-Independent Dominating Set is complete for* $\#\mathsf{W}[2]$.

## 4.3 General Definitions for Parameterized Arithmetic Circuits

Before we study any classes, we need to define the problems we want to study.

**Definition 4.11.** *Let* $(p_n)$ *be a family of polynomials of degree* $k$ *in* $\mathcal{R}[x_1, \ldots, x_{q(n)}]$ *for a polynomially bounded function* $q(n)$. *We call the tuple* $((p_n), k)$ *a* parameterized family of polynomials.

**Definition 4.12.** *Let* $((p_n)_i, k_i)$ *be parameterized families of polynomials. We call the set* $\{((p_n)_i, k_i) \mid i \in S \subseteq \mathbb{N}\}$ *a* parameterized arithmetic problem *where* $(p_n)_i$ *is a family of polynomials.*

We will generally use the notation $\{((p_n)_i, k_i)\}$ as a shorthand notation. Our definition looks complicated but looking at this problem it is easy to explain. First notice, that for every polynomial family, we have a fixed $k$ as the parameter should not change on a polynomial family. But we also want to have different $k$ in the same problem as in the boolean case. Also the polynomial family obviously can change for different $k$.

We first introduce a reduction similar to projection in the arithmetic circuit model. This is the weakest form of reduction we know.

**Definition 4.13.** *Let* $L = \{((p_n)_i, k_i)\}$, $L' = \{((p'_n)_i, k'_i)\}$ *be two parameterized arithmetic problems. We call* $L$ *to be reducible to* $L'$ *with an* arithmetic fpt projection, *written* $L \leq_{afpt,p} L'$, *if for every* $((p_n), k) \in L$ *there exists a* $((p'_n), k') \in L'$ *such that the following conditions hold:*

- $p_n = p'_{q(q'(n)f(k))}(a_1, \ldots, a_{q(q'(n)f(k))})$ *where* $q'(n)$ *a polynomially bounded function and* $f(k)$ *some function and* $a_i \in \mathcal{R} \cup \{x_1, \ldots, x_{q(n)}\}$ *where* $p_n$ *has variables* $x_1, \ldots, x_{q(n)}$.
- *There is a function* $g : \mathbb{N} \to \mathbb{N}$ *such that* $k' \leq g(k)$.

Next we continue with a reduction more akin to Turing reductions.

**Definition 4.14.** *Let* $q(n)$ *and* $q'(n)$ *be polynomially bounded function and* $f(k)$ *a function.*

$L = \{((p_n)_i, k_i)\}$, $L' = \{((p'_n)_i, k'_i)\}$ *be two parameterized arithmetic problems. We call* $L$ *to be reducible to* $L'$ *with an* arithmetic fpt $c$-reduction, *written* $L \leq_{afpt,c} L'$, *if for all* $((p_n), k) \in L$ *there exists a* $((p'_n), k') \in L'$ *such that the following conditions hold:*

- *$p_n$ can be computed by a circuit $C$ with an oracle gate to $p'_{q(n)}$ and queries $(p_{q_1(n)}, k_1), \dots, (p_{q(q'(n)f(k))}, k_{f(k)q'(n)})$.*
- *$C$ has size $f(k)q'(n)$.*
- *There is a function $g : \mathbb{N} \to \mathbb{N}$ such that for all queries $(p_n, k_i)$ to the oracle, $k_i \leq g(k)$.*

Notice that the last item allows us to ask the oracle different questions for different $k_i$.

We now introduce a shorthand notation.

**Definition 4.15.** *We define $\langle{n \atop k}\rangle \in \{0,1\}^n$ to be the set of all vectors of length $n$ of hamming weight exactly $k$, i.e. the vectors with exactly $k$ ones and the rest of the entries being zero.*

Similar as in the boolean case, our circuits will have fan-in two but have large gates with larger fan-in.

**Definition 4.16.** *Let $C$ be a layered arithmetic circuit. We say a gate is* large *if the fan-in is greater than two but bounded by a polynomial in $n$ and* small *if it has fan-in two.*

## 4.4 VFPT

We first deal with the arithmetic FPT class. We introduce a definition which corresponds to a circuit of size $f(k)\mathsf{poly}(n)$.

**Definition 4.17.** *Let $q(n)$ be a polynomially bounded function and $f(k)$ a function.*

*We define the class $\mathsf{VFPT}$ to be the class of all parameterized arithmetic problems $\{((p_n)_i, k_i)\}$ such that for every $((p_n), k)$ there exists a family of arithmetic circuits $(C_n)$ that for all $n$ computes $p_n$ and the size of $C_n$ is bounded by $f(k)q(n)$.*

We can now show our first parameterized polynomial to be contained in $\mathsf{VFPT}$. As in the boolean case this will be related to vertex cover.

**Problem 4.8.** *The* parameterized $k$ Vertex Cover Generating Polynomial *is defined as follows.*

$$f_{\mathsf{VC},G,k} \coloneqq \mathrm{GF}(G, k\text{-}\mathcal{VC}_G)$$

*where $k$-$\mathcal{VC}_G$ is the graph property of all vertex covers in $G$ of size $k$.*

Notice that this polynomial is only given for a specific graph which is in contrast to the setting in Chapter 5.

As a reminder, let the symmetric polynomial of a set of $n$ vertices of degree $d$, written $\mathsf{Sym}^n_d(x_1, \dots, x_n)$ be the elementary symmetric polynomial of degree $d$ over the variables $x_1, \dots, x_n$. The elementary symmetric polynomial can be computed by taking the product $\prod_{i=1}^n (y + x_i)$ and extracting the homogeneous components of degree $n - d$ of variable $y$. This can be computed with a polynomial size circuit (Lemma 2.5.1).

**Theorem 4.4.1.** *$\{(f_{\mathsf{VC},G,k}, k) \mid k, G\}$ is in* VFPT.

*Proof.* Given a $G = (V, E)$ and $k$, we work in the same way as the FPT algorithm. We take an arbitrary order of edges. For every edge, we branch on which vertex we add to our cover and recurse. We define for an edge set $E_{\setminus\{v\}}$ to denote the set $\{(u, u') \in E \mid u \neq v \text{ and } u' \neq v\}$. Our formula can be written as

$$P(G,k) = \begin{cases} 1 & \text{if } k = 0 \text{ and } E = \emptyset, \\ 0 & \text{if } k = 0,\ E \neq \emptyset, \\ \mathsf{Sym}_k^{\ell}(x_{i_1}, \ldots, x_{i_\ell}) & \text{if } k \neq 0,\ E = \emptyset,\ V = x_{i_1}, \ldots, x_{i_\ell}, \\ x_u \cdot P((V \setminus \{u\}, E_{\setminus\{u\}}), k-1) & \text{otherwise.} \\ \quad + x_v P((V \setminus \{v\}, E_{\setminus\{v\}}), k-1) & \end{cases}$$

As every edge has to be covered by at least one vertex, the order does not matter. If the set of edges left after $k$ steps in a path is not the empty set, we remove this computation subtree by multiplying with zero. By construction it is easy to see that these are precisely all vertex covers.

The number of operations and hence the size of the circuit is bounded by $T(n, k) = 2T(n, k-1) + 1$ plus $O(nk)$ for computing the symmetric polynomial. □

### 4.4.1 Kernelization

Another important topic in fixed parameter tractable algorithm theory is the notion of a kernel and kernelization. We can construct a similar notion here by using a modification of the definition in the paper by Creignou, Meier, et al. [CMMSV13].

**Definition 4.18.** *Let $L = \{((p_n)_i, k_i)\}$ be a parameterized arithmetic problem and $x_1, \ldots, x_{f(k)}$ a set of variables. We call $(K_n) : L \to L$ a kernel if the following conditions hold:*

1. *$K_n$ can be computed by a polynomial size family of circuits for all $n$.*
2. *$K_n((p_n), k) = ((p'_n), k')$ and $k' \leq f(k)$ for some function $f$ and $((p'_n), k') \in L$ if and only if $((p_n), k) \in L$.*
3. *$p'_n$ is a polynomial on at most $f'(k')$ variables.*

**Theorem 4.4.2.** *There exists a kernel for $\{(f_{\mathsf{VC},G,k}, k) \mid G, k\}$.*

*Proof.* We can follow the famous Buss' kernelization [FG06, pp. 208ff]. This kernelization has the following rules:

1. If $v$ is a vertex of degree greater than $k$, remove $v$ from the graph and decrease $k$ by one.
2. If $v$ is an isolated vertex, remove $v$.
3. Terminate if the remaining graph has less than $k^2$ edges.

If no rule can be applied we return the complete graph on $k+1$ vertices.

Now we can use the proof from Creignou, Meier, et al. [CMMSV13] to finish. One can verify that whenever we can apply one of the rules for a set of vertices, we can apply the same rule to the same set of vertices at a later time. Let $V_1$ be the vertices that are removed by rule 1 and $V_2$ the vertices removed by rule 2. No vertex in $V_2$ is in the vertex cover and hence can be removed. On the other hand every vertex cover of size less than $k$ has to contain every vertex in $V_1$. This shows we can look at the reduced graph.

Assume $W$ is a vertex cover of the reduced graph and let $S$ be our final vertex cover, meaning $V \cup V_1 \cup V_2'$ where $|V_2'| \leq k - |V| - |V_1|$. It is clear that different vertex cover in our reduced graph $W_1, W_2$ give different vertex cover in our final graph and that we get all vertex cover by the previous discussion.

Our fpt reduction $R$ is given by applying the reduction backwards for every valid set of variables $y_1, \ldots, y_{f(k)}$. □

We can also prove the general theorem as in the boolean setting.

**Theorem 4.4.3.** *Let $L$ be a parameterized arithmetic problem. Then $L$ has a kernel if and only if $L$ is in* VFPT.

*Proof.* If a parameterized arithmetic problem has a kernel then it has a VFPT circuit by the following argument. We can compute the polynomial $p_{f(k')}$ by brute force enumeration of all monomials. The circuit is then given by the reduction from the kernelization.

If a parameterized family is in VFPT, then it has a kernel. Take a polynomial $p_c$ for some constant $c$. Now our reduction is simple. Because $L$ was in VFPT the reduction just uses this algorithm and ignores the input. □

## 4.5 Boolean-Arithmetic and BVW$[t]$

Now we can introduce our first definition of larger arithmetic circuit classes for the fixed parameter world. These will generally be the "hard" problems and corresponds to the #W$[t]$ hierarchy in the counting world. In essence, we will simulate boolean circuits with arithmetic circuits and hence can use definitions and reductions close to the boolean counting setting which we described in Section 4.2.2.

Let $X = \{x_1, \ldots, x_{q(n)}\}$ and $Y = \{y_1, \ldots, y_{q(n)}\}$ be a set of variables where $q(n)$ is a polynomially bounded function. Let us identify the value true of a boolean formula with 1 and the value 0 with false.

**Definition 4.19.** *We call a family of arithmetic formulas* boolean-arithmetic *if every circuit in the family is of the following form. It has at the top a product gate with two children $C_l$ and $C_r$. $C_r = \prod_{i=1}^{q(n)} (1 - y_i + x_i y_i)$. For $C_l$ there exists a boolean formula $\phi$ on $\wedge$ and $\vee$ gates over inputs $Y$ such that for all assignments $e \in \{0,1\}^{q(n)}$*

$$C_l(e) = \alpha\phi(e)$$

*for some $\alpha \in \mathbb{N}$. Additionally, $C_l$ has polynomial size and constant depth.*

Now we can define our central definition of hard classes. We will first define a set called BA[$t$] and then use the closure under our reduction for the class.

**Definition 4.20.** *We call a parameterized polynomial family $\{((p_n), k)\}$ to be in BA[$t$] if for every $((p_n), k)$ there exists a family of boolean-arithmetic formula where $\phi$ which is equivalent to the left part of the formula has weft t,*

$$p_n = \sum_{e \in \binom{q(n)}{f(k)}} C_n(X, e)$$

*and $p_n$ has $q(n)$ variables.*

*We call* $\mathsf{BVW}[t] = [BA[t]]^{\leq_{afpt,c}}$.

Notice that the $\alpha$ is chosen beforehand and hence the same for all possible assignments. We can generally ignore $\alpha$ as we can add a $k$ fraction of the value to every element in our reduction or if we use oracle reductions we can multiply it in the reduction. Notice that we did not restrict the weft of the arithmetic realization of the boolean formula. In our case the distinction is not necessary as a $\wedge$ gate can be directly simulated with a product gate and a $\vee$ gate by using DeMorgan's rule with two negations and a product gate. However, we formulated our definition this way to distinguish explicitly between the boolean equivalent of the boolean-arithmetic formula and the boolean-arithmetic formula itself.

As BA[$t$] is just a syntactic defined class, it is unclear if two different circuits computing the same polynomial will be in the class. This is the reason why we chose the closure under a reduction. We chose oracle reductions because they are more powerful than projections. Additionally, they give us various tools such as extracting homogeneous components which make the proofs technically easier. However, we will see that our main result of this section transfers also in the case for fpt projections. From now on we will only work with $\mathsf{BVW}[t]$ and essentially ignore BA[$t$].

We can now introduce some sanity checks for the definition. First we obviously have a hierarchy for $\mathsf{BVW}[t]$.

**Proposition 4.1.** $\mathsf{BVW}[t] \subseteq \mathsf{BVW}[t+1]$.

The next proposition holds trivially by the closure under arithmetic fpt $c$-reduction.

**Proposition 4.2.** $\mathsf{VFPT} \subseteq \mathsf{BVW}[1]$.

Second our now defined classes are trivially included in $\mathsf{VNP}$.

**Lemma 4.5.1.** *If a parameterized arithmetic problem $L = \{((p_n), k)\}$ is in* $\mathsf{BVW}[t]$ *then all elements $((p_n), k) \in L$, $(p_n)$ are in* $\mathsf{VNP}$.

*Proof.* Given a polynomial family $((p_n), k) \in \mathsf{BVW}[t]$ we construct a new polynomial $p'$ where we multiply $p_n$ with an arithmetic circuit computing

$$\frac{\prod_{i \neq k}(i - \sum_{j=1}^{q(n)} y_j)}{\prod_{i \neq k}(i - k)}$$

where $q(n)$ gives us the number of variables $p_n$ has.

This ensures exactly $k$ of our $y$ variables are set to one. As we can transform the large gates to binary gates we see that this circuit is in VP. The summation over all $\{0,1\}^n$ vectors gives us the required polynomial. □

### 4.5.1 Independent Set and BVW[1]

Now we can show the first problem that is complete for BVW[1]. As in the boolean case a variant of Independent Set will be our complete problem.

**Problem 4.9.** *The* parameterized $k$ Independent Set Generating Polynomial *is defined as follows:*

$$f_{\mathsf{IS},G,k} := \mathrm{GF}(G, k\text{-}\mathcal{IS}_G)$$

*where $k$-$\mathcal{IS}_G$ is the graph property of all Independent Sets in $G$ of size $k$.*

Let us look at how we want to construct the reduction. We want to take the parsimonious reduction from the completeness of W[1] to Independent Set on the formula corresponding to $\phi$ and set the weights correctly for the fulfilling the right side. For this we need the following property the parsimonious reduction has to have.

**Definition 4.21.** *We call a fpt parsimonious reduction from a boolean formula $F$ on variables $y_1, \dots, y_n$ to a boolean formula $F'$ on variables $y'_1, \dots, y'_m$* witness projectable *if the following holds. There exists a projection $\psi : \{0,1\}^m \to \{0,1\}^n$ such that for the sets*

$$\begin{aligned} S_F &:= \{a \quad\ \ \mid a \text{ is a valid } k\text{-weight assignment for } F\}, \\ S_{F'} &:= \{\psi(b) \mid b \text{ is a valid } k\text{-weight assignment for } F'\} \end{aligned}$$

*it holds that*

$$S_F = S_{F'}.$$

This definition is needed to know that we can transfer a witness between these two problems. We will hence transfer the $\prod(1 - y_i + x_i y_i)$ part with this projection $\psi$ to construct the correct witness.

**Theorem 4.5.2.** $\{(f_{\mathsf{IS},G,k}, k) \mid G, k\}$ *is complete for* BVW[1].

*Proof.* Because the formula has as left child a formula corresponding to $\phi$ we can transform this part to a boolean formula $\phi'$ of weft one. We can now use the reduction from $k$-Independent Set to #W[1] for this part (see Theorem 4.2.9). Notice that this reduction is witness projectable. This will then give us a graph which has an Independent Set for every $k$ weight assignments to the $y$s.

We can then put the weights $x_i$ in the variable $y_i$. Selecting $k$ vertices from the set $Y$ and using the projection stemming from the witness projectable of the reduction finishes the reduction. The correctness is obvious from the previous lemma and the correctness of the original reduction.

The membership is easy to see with the following formula.

$$\prod_{\{u,v\}\in E} (1 - y_u y_v) \cdot \prod_{v\in V} (1 - y_v + x_v y_v) .$$

□

Notice that directly transforming the arithmetic formula corresponding to $\phi$ could result in a vastly different formula which might not fulfill the requirements in the reduction. It is best to see this as transforming the boolean-arithmetic formula into a boolean formula, transform this with the reduction and then transfer this back to a boolean-arithmetic form

### 4.5.2 Dominating Set and BVW[2]

**Problem 4.10.** *The* parameterized $k$ Dominating Set Generating Polynomial *is defined as follows:*

$$f_{\mathsf{DS},G,k} \coloneqq \mathrm{GF}(G, k\text{-}\mathcal{DOM}_G)$$

*where $k$-$\mathcal{DOM}_G$ is the graph property of all Dominating Sets in $G$ of size $k$.*

**Theorem 4.5.3.** $\{(f_{\mathsf{DS},G,k}, k) \mid G, k\}$ *is* BVW[2] *complete.*

*Proof.* We now have a reduction for the arithmeterized boolean formula with the boolean reduction to $k$-Dominating Set. Notice that this reduction is witness projectable which can be seen by the proof given in Theorems 4.2.6 and 4.2.13.

For the right part we can take the reduction and set the weight of the vertices corresponding to $y_i$ to $x_i$. Notice that the reduction already prevents us from taking multiple vertices assigned to the same $y_i$. As the previous reduction was witness projectable this one is too.

The membership is again easy to see with the following formula where $N(v)$ is the neighbourhood of $v$.

$$\prod_{v\in V} \left( 1 - \prod_{u\in N(v)\cup\{v\}} (1 - y_u) \right) \cdot \prod_{v\in V} (1 - y_v + y_v x_v) .$$

□

Now we can already see a pattern in our reductions. The majority of the work is done in the boolean formula and the “arithmetic complexity” does not increase.

### 4.5.3 3-SCM Single-Product Cover and BVW[3]

**Problem 4.11.** *The* parameterized $k$-3-SCM Single-Product Cover Polynomial *is defined as follows:*

$$f_{\mathsf{SCM},G,k} \coloneqq \mathrm{GF}(G, k - \mathcal{SCM}_G)$$

*where $k$-$\mathcal{SCM}_G$ is the graph property of all 3-SCM Single-Product Covers in $G$ of size $k$.*

**Theorem 4.5.4.** $\{(f_{\mathsf{SCM},G,k}, k) \mid G, k\}$ *is complete for* $\mathsf{BVW}[3]$.

*Proof.* We take the original reduction from Theorems 4.2.8 and 4.2.13 to reduce the boolean part. Notice that this again is a witness projectable reduction.

As in the Dominating Set case, we can set the weight of the vertices corresponding to $y_i$ to $x_i$. With this we can then use the projection stemming from the reduction being witness projectable to get the correct weights.

The membership is easy to see with the following formula.

$$\left(\prod_{\nu \in R}\left(1 - \prod_{\mu \in M_\nu}\prod_{i=1}^{k}\left(1 - \prod_{v \in S_i \setminus N(\mu)}(1-v)\right)\right)\right) \cdot \prod_{i=1}^{k}\left(1 - \prod_{v \in S_i} v\right)$$
$$\cdot \prod_{i=1}^{k}\prod_{v \in S_i}(1 - y_v + y_v x_v)$$

□

### 4.5.4 Discussion

We now want to discuss our various decisions in defining this model. The first one is the top summation of $e \in \left\langle {n \atop k} \right\rangle$. This seems necessary as it is unclear how to check over all sets of size $k$ without such a summation. Additionally, the definition seems reasonable because it is similar to the definition of $\mathsf{VNP}$.

Let us now look at how we could define the circuits we sum over. The obvious first idea would be without any restrictions, meaning to just have unbounded fan-in gates for products and for summations of some weft $t$ with some constant depth $t$. However, finding complete problems for this seems to be hard. We can easily assume a circuit being $\mathrm{HOMC}^k \prod_i^n (p_i + i \cdot y)$ where $p_i$ is the $i$th prime. Now this polynomial has only degree $k$ but we still need to consider all $2^n$ possible monomials in some sense. This seems impossible with some complete problems which can only have order of $\binom{n}{k}$ many monomials. It is also unclear how to "collapse" multiple of these monomials with the same variables but different coefficients into one circuit for every computed monomial. This implies we need to have some restriction on the gates we allow. We postpone some additional argument about our model to the next section.

## 4.6 $\mathsf{VW}[t]$

We have seen in the previous section that taking a definition close to the boolean setting gives us consistent classes. However, this classes are far away from having the full power of arithmetic circuits. Can we, for example, construct a model that can give different weights to different assignments? We will reexam the defining problems of the $\mathsf{W}[t]$ hierarchy to construct different problems for $\mathsf{VW}[t]$.

First, we will give our model which is based on the simple requirement that we want to incorporate constants.

**Definition 4.22.** *Let $\alpha$, $\alpha_i$, $\beta_i \in \mathbb{K}[x_1, \dots, x_{q(n)}]$. We call an arithmetic formula an extended boolean-arithmetic formula if the formula is iteratively constructed from gates of one of the following forms*

- $C = \prod_{i=1}^{q(n)} (\alpha_i + \beta_i C_i)$ *for some polynomially bounded function $q(n)$ where $C_i$ is an extended boolean-arithmetic formula.*
- $C = \alpha \prod_{i=1}^{c} y_{\varphi(i)} \prod^{c'} x_{\varphi'(i)}$ *for some mapping $\varphi$, $\varphi'$ and constants $c$, $c'$.*

*We call an extended boolean-arithmetic formula of weft t if it has at most t layers of gates of the first type. We call the closure under arithmetic fpt c-reduction the class* VW[$t$].

The reader should notice that we again build our model only on product gates. We do not know how to handle summation gates in our model. The problem arises from the following simple observation. As soon as multiple branches of a summation gate are selected, a simple generating function which multiplies the weight of all selected vertices will make mistakes by multiplying weights that should be summed up. It might multiply two different trees connected by a summation gate, unless the graph is carefully crafted to remove these mistakes in a later step. Hence, we would need to treat every summand separately, similarly to the notion of a proof tree in an arithmetic circuit. But here we again meet our limit of choices. As we have unbounded product gates, a $\prod \sum$ formula can already have $n^n$ proof trees while we only have $\binom{n}{k}$ choices.

It would be interesting if we can find a theory that encompasses summation gates despite these difficulties. However, we will not use unbounded summation gates in this thesis.

As it turns out, we need to change the defining problems of our hierarchy. We will now discuss why. In the boolean setting, where we look at weighted satisfiability with exactly $k$ many values set to true, we can imagine selecting $k$ points and then checking some condition. In essence, checking a boolean condition, like checking if the selected vertices are a Dominating Set, is easy for boolean circuits. For arithmetic circuits, this problem gets more complicated. While we can easily deny certain products to take place which we shown in Section 4.5 it is unclear if this captures the essence of arithmetic computation. We argue that to be closer to the power of arithmetic circuits we need to be able to change the value by arbitrary constants if a certain condition (here represented by some $y$ variables) is not fulfilled.

This can be seen in an example for Dominating Set where we just change the formula slightly from the case in BVW[2]. Let a circuit be given by $\prod^n (1 - \beta_i \prod^n (1 - y_j))$. Here it seems that we also need to look at variables that are not 1 to calculate the correct value of the polynomial. It is unclear how we could avoid having to change constants depending on specific vertices not selected without introducing more restrictions.

To simplify notation we will use `UDEF` $(T)$ to denote the following function. Let $T$ be a constant arithmetic term and $\beta$ the evaluation of this term if it is defined. Then we define

$$\texttt{UDEF}\,(T) = \begin{cases} \beta & \text{if } T \text{ is defined,} \\ 0 & \text{otherwise.} \end{cases}$$

We will need a short lemma to simplify the proofs for constant size products at the bottom.

Let a set of vertices $y_1, \dots, y_c$ be given. Let for every set $\emptyset \neq S \subseteq \{y_1, \dots, y_c\}$ a new vertex with the weight of $\alpha$ if $|S|$ is odd and $\frac{1}{\alpha}$ if $|S|$ is even be connected to all vertices in the set $S$.

We will call a vertex that is included in an instance to a problem to be *selected*.

We say a set $S$ is *counted* if it is connected to a $y_i$ and $y_i$ is selected and otherwise not counted. The weight of an assignment where some $y_i$ are selected is the product over all sets that are counted where every set is the product of its elements.

**Lemma 4.6.1.** *Let the setting be as above. Then the value of the assignment where all $y_i$ are selected is $\alpha$ and 1 otherwise.*

*Proof.* We can see the construction as counting positive and negative numbers by identifying $\alpha$ with a positive one and $\frac{1}{\alpha}$ with a negative one.[2]

We know that

$$(-1) \cdot \sum_{i=0}^{c} 1^{c-i} \binom{c}{i} (-1)^i = (-1) \cdot (1-1)^c = 0.$$

Notice that in our construction the set has to be of odd size to have a weight of $\alpha$ which corresponds to a weight of one here.

Then the sum over all weights in our construction is given by

$$(-1) \cdot \sum_{i=1}^{c} \binom{c}{i} (-1)^i = 1 + (-1) \cdot \sum_{i=0}^{c} \binom{c}{i} (-1)^i = 1 + 0 = 1. \tag{4.1}$$

Hence we know that our value is 1 if all $y_i$ are selected.

Now we pick $\ell \neq 0$ where $\ell$ is the number of $y_i$ selected.

We know that the uncounted values are

$$(-1) \sum_{i=1}^{\ell} \binom{\ell}{i} (-1)^i = 1$$

as these are all vertices that have no selected vertex adjacent.

Hence the sum over the uncounted values is one. Notice that the summation of the counted and uncounted values is exactly the value from Equation (4.1). Hence, we follow that the sum over all counted vertices is zero.

As the counted vertices correspond to the set we actually sum over, the proof is finished. □

[2] Essentially we take the logarithm of our values and compute with these.

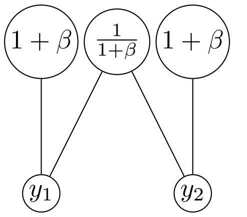


Figure 4.3: Example Independent Set construction for $1 + \beta y_1 y_2$

As an example we can look at Figure 4.3 which results in the following values.

| $y_i$ | $y_j$ | |
|---|---|---|
| 0 | 0 | $1$ |
| 0 | 1 | $\texttt{UDEF}\left((1+\beta_i)\cdot\frac{1}{1+\beta_i}=1\right)$ |
| 1 | 0 | $\texttt{UDEF}\left((1+\beta_i)\cdot\frac{1}{1+\beta_i}=1\right)$ |
| 1 | 1 | $\texttt{UDEF}\left((1+\beta_i)\cdot\frac{1}{1+\beta_i}\cdot(1+\beta_i)=1+\beta_i\right)$ |

In general, we will again ignore a constant $\alpha$ in front of the product as we can just multiply every variable by $\frac{\alpha}{k}$ to get the correct value.

In general, we will assume our circuits to have top fan-in bounded by $n$ instead of $q(n)$ to simplify notation.

### 4.6.1 VW[1]

**Problem 4.12.** *Let a graph $G$ be given with every vertex having two values out of $\mathbb{K}[x_1,\dots,x_n]$. Let us denote the values by $(x_v,\alpha_v)$. Let now a* full cover $k$-Independent Set *be a set of $k$ vertices $\{v_1,\dots,v_k\}$. Let $R$ be the set of all vertices that are not selected in the chosen Independent Set and when would be selected would have an edge with a vertex in $\{v_1,\dots,v_k\}$. The weight of a full cover $k$-Independent Set $v_{e_1},\dots,v_{e_k}$ is given by*

$$\prod_{i=1}^{k} v_{e_i}\cdot\prod_{v\in R}\alpha_v.$$

We define the polynomial $f_{\mathsf{fc-IS},G,k}$ to be the sum over all the weights of all full cover $k$-Independent Set in a graph $G$.

**Theorem 4.6.2.** *$\{(f_{\mathsf{fc-IS},G,k},k)\mid G,k\}$ is complete for* VW[1]*.*

*Proof.* The membership is obvious with the formula

$$\prod_{(i,j)\in E}^{\binom{n}{2}}(1-y_iy_j)\prod_{v\in V}(\alpha_v+(x_v-\alpha_v)\,y_v)\,.$$

For the completeness, we have a circuit of the form $\alpha\prod_i^n\left(1+\beta_i\prod_j^c y_{\iota_{i,j}}\right)$. Notice that here $\beta_i$ can be a constant size product of variables $x_i$. We build now the following construction. Connect a new vertex to every $y_j$ and set the weight as $(1, 1+\beta_i)$. Now, if $c \geq 1$ similar to the inclusion exclusion formula like in Lemma 4.6.1, connect more vertices with the weight given by either $(1, 1+\beta_i)$ if the number of connected vertices is odd or $(1, \frac{1}{1+\beta_i})$ when the number of selected vertices is even. We give a short example in Figure 4.3.

By Lemma 4.6.1 this is always calculating the correct weight.

□

### 4.6.2 VW[2]

**Problem 4.13.** *Let a graph $G$ be given with every vertex having two values out of $\mathbb{K}[x_1, \ldots, x_n]$. Let us denote the values by $(x_v, \alpha_v)$. Let now a* full cover $k$-Dominating Set *be a set of $k$ vertices such that there exists set $\{v_1, \ldots, v_k\} \cup A \cup R = V$ and the following conditions hold:*

- *For every $v \in A$ there exists $i$ such that the edge $(v, v_i) \in E$.*
- *For every $v \in R$ and all $i$ no edge $(v, v_i) \in E$ exist.*

*The weight of a full cover $k$-Dominating Set with vertices $v_{i_1}, \ldots, v_{i_k}$ is*

$$\prod_{\ell=1}^{k} v_{i_\ell} \cdot \prod_{v \in R} \alpha_v .$$

We define the polynomial $f_{\mathsf{fc-DS},G,k}$ to be the sum over the weights of all full cover $k$-Dominating Set.

In essence we take all possible Dominating Sets of a graph and weight them by the vertices which violate the Dominating Set condition.

**Lemma 4.6.3.** *Let $C = \prod_{i=1}^{n}\left(1 + \alpha_i \prod_{j=1}^{n_i}\left(1 + \beta_{i,j} y_{\varphi(i,j)}\right)\right)$ be a circuit. Then $C$ has a graph such that $f_{\mathsf{fc-DS},G,k}$ is equal to $\sum_{e \in \left\langle {q(n) \atop k} \right\rangle} C(x, e)$ for some polynomially bounded function $q(n)$.*

*Proof.* We take the general Dominating Set construction and modify it in the following way. For every branch in the formula corresponding to $y_{\phi(i,j)}$[3] we add a vertex that is connected with all representatives of $y_{\phi(i,j)}$ in our construction. We set the weight of this vertex to

$$\left(0, \mathtt{UDEF}\left(\frac{1+\alpha_i}{1+\alpha_i\left(1+\beta_{i,j}\right)}\right)\right).$$

Notice that if the fraction is undefined the overall value of our product is zero.

Additionally, for every variable $y_i$ we add the weight

$$\left(\frac{\prod_{i=1}^{n}\left(1+\alpha_i \prod_{j=1}^{n_i}\left(1+\beta_{i,j}\right)\right)}{k}, 1\right).$$

[3] Notice that this can have repetition corresponding to the same variable $y_\ell$

This corresponds to the overall value of the circuit if all $y_{i_{i,j}}$ would be one.

Now by construction, whenever we have a specific variable set to zero, we cancel the weight of this variable in our overall value of the circuit. □

**Lemma 4.6.4.** *Every arithmetic formula of the form*

$$f = \prod_{i=1}^{n} \left( 1 + \beta_i \prod_{j=1}^{n_i} \left( 1 + \gamma_{i,j} \prod_{m=1}^{c_{i,j}} y_{\varphi(i,j,m)} \right) \right)$$

*admits a graph $G$ such that $f_{\mathsf{fc-DS},G,k}$ on this graph is $\sum_{e \in \left\langle {q(n) \atop k} \right\rangle} f(X, e)$ for some polynomially bounded function $q(n)$.*

*Proof.* We transform the circuit into the following form for all $\gamma_{i,j} \neq -1$

$$\prod_{i=1}^{n} \left( 1 + \beta_i \prod_{j=1}^{n_i} (1 + \gamma_{i,j}) \cdot \prod_{j=1}^{n_i} \left( 1 + \left( \frac{1}{1 + \gamma_{i,j}} - 1 \right) \prod_{m=1}^{c_{i,j}} y_{\varphi(i,j,m)} \right) \right). \tag{4.2}$$

If we now look at Lemma 4.6.1 we see that we can use the same construction. But instead of a vertex $v$ being denied when it has a neighbour it is now allowed (and hence not counted) on the other hand if a vertex does not have a neighbour it is allowed in the original construction and denied now. With this it is clear that we just need to use this construction to compute the "inverse". For this, as you can see in Equation (4.2) we did exactly that. Whenever our construction produces a one, we remove the value $1 + \gamma_{i,j}$ from $\prod_{j}^{n_i} (1 + \gamma_{i,j})$. Whenever we have a zero, we do not remove it. For $\gamma_{i,j} = -1$ we treat it normally but with a vertex of weight $(0, 0)$.

Let us call the constants from the formula above by $\beta'_i = \beta_i \prod_{j=1}^{n_i} \left( 1 + y_{i_{i,j}} \right)$ and $\gamma'_{i,j} = \frac{1}{1+\gamma_{i,j}} - 1$. As described earlier, we put $\gamma'_{i,j}$ onto the vertices as in the construction in Lemma 4.6.1. □

**Theorem 4.6.5.** $\{(f_{\mathsf{fc-DS},G,k}, k) \mid G, k\}$ *is complete for* VW[2].

*Proof.* From Lemma 4.6.4 we can prove the completeness. The membership is easy to see with the formula

$$\prod_{v \in V} \left( 1 - (1 - \alpha_v) \cdot \prod_{u \in N(v) \cup \{v\}} (1 - y_u) \right) \prod_{v \in V} (1 + (x_v - 1)\, y_v).$$

□

### 4.6.3 VW[3]

We again look at a variant of $k$-3-SCM Single-Product Cover.

**Problem 4.14.** *Given a graph $G = S \cup M \cup R$ a tripartite graph with every vertex having two linear forms out of $\mathbb{K}[x_1, \ldots, x_n]$ as weights. Let us denote by by a full cover 3-SCM Single-Product Cover a set of $k$ vertices $\{v_1, \ldots, v_k\}$ from the set $S$. Then the weight is given by*

$$\prod_{v \in \{v_1,\ldots,v_k\}} x_v \cdot \prod_{f \in F} \gamma(v_f) \cdot \prod_{s \in S} \gamma'(v_s).$$

*Here $\gamma(v_f)$ is $w_1$ if all values $v \in P$ with $(v, v_f) \in E$ are selected and $w_2$ otherwise. Similarly, $\gamma'(v_f) = w_1$ if there exists a vertex $v$ such that for all vertices $u$ with $(u, v) \in E, u \in P$ $u$ is selected and $w_2$ otherwise.*

We define $f_{\mathsf{fc-SCM},G,k}$ to be the sum over the weights of all full cover 3-SCM Single-Product Cover.

In essence, we set weights onto the vertices and select the weights if this is a valid assignment for the subgraph or not.

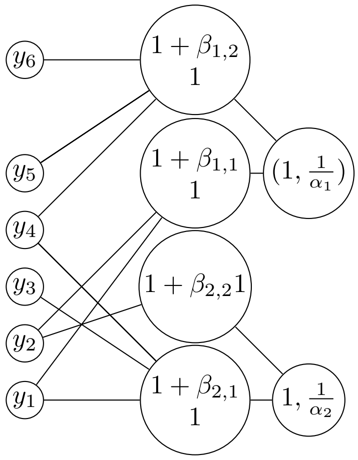


Figure 4.4: Gadget for polynomial $(1 + \alpha_1 (1 + \beta_{1,2} y_4 y_5 y_6)(1 + \beta_{1,2} y_1 y_2))$
$\cdot (1 + \alpha_2 (1 + \beta_{2,1} y_1 y_3 y_4)(1 + \beta_{2,2} y_2))$

**Lemma 4.6.6.** *Let*

$$C = \prod_{i=1}^{n} \left( 1 + \alpha_i \prod_{j=1}^{n_i} \left( 1 + \beta_{i,j} \prod_{m=1}^{n_{i,j}} \left( 1 + \gamma_{i,j,m} y_{\phi(i,j,m)} \right) \right) \right).$$

*Then $C$ has a graph such that $f_{\mathsf{fc-SCM},G,k}$ is equal to $\sum_{e \in \left\langle {q(n) \atop k} \right\rangle} C(X, e)$.*

*Proof.* For $\alpha_i$, $\beta_{i,j}$ or $\gamma_{i,j,m}$ being zero, we can add the resulting coefficient to a separate vertex that is enforced to be taken. We can build a graph for this type of monomial in a way as in Figure 4.4. For every $\beta_{i,j}$ we introduce a new vertex into the set $F$ that is connected to all $y_{\varphi(i,j,m)}$ with the weight $(1 + \beta_{i,j}, 1)$. Let us call this vertex $f_{i,j}$. For the second layer, we add the following vertices into $S$. We connect a vertex $v$ with the weight $(1, \frac{1}{\alpha_i})$ to every $f_{i,j}$ for every $j$. We produce $n$ such vertices. For the third layer we just add the weight product of all $\gamma_{i,j,m}$ that $y_{\phi(i,j,m)}$ has. □

**Theorem 4.6.7.** *$\{(f_{\mathsf{fc-SCM},G,k}, k) \mid G, k\}$ is complete for* $\mathsf{VW}[3]$.

*Proof.* After Lemma 4.6.6 we only need to handle the case of constant size products at the bottom. We can handle this again with Lemma 4.6.1. The correctness now follows from the construction.

The membership is easy to see with the formula

$$\prod_{i=1}^{n}\left(\alpha_i + (\alpha_i - 1)\prod_{j=1}^{n_i}\left(\beta_{i,j} + (\beta_{i,j} - 1)\prod_{m=1}^{n_{i,j}}\left(1 + \gamma_{i,j,m} y_{\phi(i,j,m)}\right)\right)\right).$$

□

## 4.7 The Immanant

The immanant is an interesting problem in arithmetic circuit complexity. As described in Section 2.6 it is a polynomial that can encompass the determinant or the permanent, two polynomials that could not be any different in their complexity. This is why it makes sense to look at the parameterized complexity of the permanent.

Perhaps for hook diagrams of horizontal size at most $k$ we can find a fixed parameter algorithm? Instead want to show that a $k$-hook diagram is already $\mathsf{VW}[1]$ hard.

However, we are faced with one problem. The immanant is a multilinear polynomial of degree $n$ but our parameterized families of polynomials are only allowed to have degree $f(k)$ for some function $f : \mathbb{N} \to \mathbb{N}$. We need to define a polynomial related to the immanant that has degree depending only on $k$.

**Problem 4.15.** *Let $\lambda = (k, 1, \ldots, 1)$ be a $k$-hook diagram. Let*

$$S'_{n,k} := \{\pi : [k] \to [n] \mid \ \pi(1), \ldots, \pi(k) \text{ describes a cycle cover}\}$$

*and $\psi : ([k] \to [n]) \to ([n] \to [n])$ which maps the partial permutations to complete permutations by setting $\psi(\pi(i)) = i$ for every $i \notin \mathrm{Dom}(\pi)$.*

*Then we define the $k$-*cycle immanant *to be*

$$\mathrm{Im}_{k,hook} := \sum_{\pi \in S'_{n,k}} \chi_\lambda(\psi(\pi)) \prod_{i=1}^{k} A_{i,\pi(i)}.$$

Notice that we can easily have a $c$-reduction from the immanant polynomial to the $\mathrm{Im}_k$ polynomial. We do this by just enforcing $n - k$ self-loops via taking homogeneous components.

We will first reduce immanant to a modified cycle format polynomial. Here we denote by $\mathcal{F}_{cf} := \mathcal{CF}_{(k,1,\ldots,1)|\forall i, x_i \to 1}$, the cycle format polynomial where evaluate all variables corresponding self-loops with the constant one. This polynomial has a degree of $k$.

**Lemma 4.7.1.** *Let $\rho$ be a frequency notation. Then for any cycle $\sigma$ that has the same frequency as $\rho$ the character is the same.*

*Proof.* Let us remember the Murnaghan-Nakayama Rule (Theorem 2.6.1). By the decomposition given by the rule it is clear that it only matters what cycles the frequency notation has and that for every cycle cover with the same cycle format the character is the same. □

**Theorem 4.7.2.** $\{(\mathcal{F}_{cf}, k) \mid k\} \leq_c \{(\mathrm{Im}_{k,hook}, k) \mid k\}$.

*Proof.* These two polynomials are almost equal. However, $\mathrm{Im}_{k,hook}$ has monomials corresponding to self-loops while the cycle format polynomial has not. We can extract these by taking the homogeneous components of all self-loops of degree zero. Left to do is the corresponding coefficient. By Lemma 4.7.1 this is the same for all cycle formats (which we only have one). Hence, we can hardcode this into the reduction by adding a $\frac{1}{\chi(\rho)}$ multiplication at the top. □

Now the idea is clear. We just need to prove that $\mathcal{F}_{cf}$ is hard for $\mathsf{VW}[1]$ to show that hook immanants are hard. In the counting world, there is a proof based on fpt Turing reduction ([FG06, pp. 372ff]). However, it is unclear how to transfer this proof to our setting. In their proof, they look at two different homomorphisms. Let us call the first homomorphism of type $f$ which is given by the following description. The sum of specific homomorphisms from cycles of length $k \cdot l$ to graphs $\mathcal{H}_0, \ldots, \mathcal{H}_m$ where these are all graphs on $k$ vertices not identical under isomorphism. The second type they look at, let us call it of type $g$ is the following. They look at homomorphisms from cycles of length $k \cdot l$ to a given graph $G$[4], let us call them of type $g$. While they could use relations of the number of homomorphism between type $f$ and type $g$ for different parameters $l$, it is unclear how to transfer these in our settings. Our homomorphisms of type $f$ would have different variables from the homomorphisms of type $g$ and we would need to rename these somehow.

**Conjecture 4.1.** $\{(\mathcal{CF}_{(k,1,\ldots,1)|\forall i, x_i \to 1}, k)\}$ *is hard for* $\mathsf{VW}[1]$.

## 4.8 Open Problems

Our theory of fixed parameter tractable arithmetic problems is far from complete. We will discuss some further directions for this topic.

The class $\mathsf{VFPT}$ is very sparsely populated at the moment. While it seems easy to transfer Generating Functions based on well known fixed parameter tractable counting problems we can study this class with respect to the setting. A good start would be to study various homomorphism polynomials as in Chapter 5. As we will discuss in

[4] To be precise they look at directed versions of the graphs.

that chapter there exists a relationship between homomorphism polynomials and VP. Can we get more insight into this by finding more homomorphism polynomials in VFPT?

Secondly, an important question regarding the complexity of determinant and permanent would be to find various parameters for these problems where they are actually fixed parameter tractable. But even proving hardness for these problems for natural parameters would give us some insight into the complexity of these problems.

As for the hardness, there are some obvious hanging threads. Solving Conjecture 4.1 would be an interesting result. Recently, Curticapean [Cur15] gave a different proof for hardness of $k$-cycles in the counting setting. Can we perhaps transfer this proof?

Finally, research into different fixed parameter tractable models can be very interesting. With our omission of summation gates, finding a different model which either includes bounded summation gates or even unbounded one would be interesting. Not only would the model be more flexible in representing polynomials but it also would be closer to encompassing the power of arithmetic circuits.

# 5 Homomorphism Polynomials

The following chapter is an extension of the work submitted to WALCOM [Eng15].

## 5.1 Introduction to Homomorphism Polynomials

Graph homomorphisms of undirected graphs are studied because they give important generalizations of many natural questions ($k$-coloring, acyclicity, binary CSP and many more cf. [HN04]). One major theorem, proved by Lovász [Lov67], showing the importance of homomorphisms is the following. Two graphs $H$ and $H'$ are isomorphic if and only if for all $G$ the number of homomorphisms from $G$ to $H$ and from $G$ to $H'$ are identical. Similarly, the relation to coloring is obvious as every graph homomorphic to a complete graph $K_k$ (if we disregard self-loops) is $k$ colorable but the other relations mentioned earlier are more involved. We can also look at vertex cover. Let $H$ be a single edge and a loop at one vertex. Then the homomorphisms from $G$ to $H$ give us the vertex covers in $G$. These are just some examples of the usefulness of graph homomorphisms.

One of the first results on the decision problem of graph homomorphisms was given by Hell and Nešetřil [HN90]. They showed the following dichotomy: Deciding if there exists a homomorphism from some graph $G$ to a fixed graph $H$ is polynomial time computable if $H$ is bipartite and NP complete otherwise. Some different but related problems were studied by Chekuri and Rajaraman [CR00], Dalmau, Kolaitis, and Vardi [DKV02], and Freuder [Fre90]. They looked at the constraint satisfaction side of the graph homomorphism problem which was finally generalized by Grohe [Gro07]. He studied the problem even for arbitrary relational structures. In summary, they looked at the following graph homomorphism problem: For a restricted graph class $\mathcal{G}$, decide if a given graph $G \in \mathcal{G}$ is homomorphic to a given graph $H$. Grohe indeed showed that the problem is fixed parameter tractable if every graph in the graph class has bounded treewidth[1] and is otherwise $\mathsf{W}[1]$-hard. In contrast to the decision problem they could not show a dichotomy for the problem in relation to $\mathsf{P}$ and $\mathsf{NP}$. Such a dichotomy seems unlikely as they give a reduction to the Log-Clique problem. The Log-Clique problem is defined by deciding if a given graph $G$ has a clique of size at least $\log n$ and was studied by Papadimitriou and Yannakakis [PY96]. It is contained in $\mathsf{NP}$ but unlikely to be complete for $\mathsf{NP}$ and similarly unlikely to be in $\mathsf{P}$. $\mathsf{NP}$ completeness would imply that $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log n)})$. Recently, a new conditional lower bound was found for the first problem. Fomin, Golovnev, et al. [FGKM15] showed that any

[1] Actually, he showed that if every graph in the graph class is homomorphic to a bounded treewidth graph then the problem is fixed parameter tractable.

algorithm needs time at least $2^{\Omega(\frac{n \log h}{\log \log h})}$ if the Exponential Time Hypothesis holds.

Later, focus shifted onto the counting versions of these two sides where we have to count the number of homomorphisms. Dyer and Greenhill [DG00, DG04] solved the first side in the counting case and Dalmau and Jonsson [DJ04] the second. Let us state the results precisely. Dyer and Greenhill showed that counting graph homomorphism from any graph $G$ to a fixed graph $H$ is #P complete unless every connected component of $H$ is either an isolated vertex without a self-loop, a complete graph with all self-loops or a complete bipartite graph without self-loops. Dalmau and Jonsson, who looked at the second case, gave a similar answer as in the decision case. Counting the number of homomorphisms from a given graph $G \in \mathcal{G}$ to a given graph $H$ is fixed parameter tractable if all graphs in $\mathcal{G}$ have bounded treewidth otherwise it is complete for #W[1]. In fact, the algorithm for the bounded treewidth case was already shown by Flum and Grohe [FG04] in their seminal paper about parameterized counting complexity.

Later this problem was extended by Bulatov and Grohe [BG05] to graphs with multiple edges. They also noticed some interesting connections to statistical physics and constraint satisfaction problems. Statistical physics defines the partition function as follows:

$$Z_A(G) = \sum_{\zeta : V(G) \to [m]} \prod_{(u,v) \in E(G)} A_{\zeta(u),\zeta(v)}$$

where $A \in \mathbb{K}^{m \times m}$. In essence it counts the number of homomorphisms from some graph $G$ to a fixed graph given by the matrix $A$. Research continued to have dichotomy theorems for different type of matrices $A$, with two noticeable works being by Goldberg, Grohe, et al. [GGJT10] who proved a dichotomy for real valued symmetric matrices $A$ and by Cai, Chen, and Lu [CCL13] who showed a dichotomy for all symmetric matrices with complex entries. Even today the research field for the counting case is very active as the recent results on counting specific restrictions of homomorphisms such as in [GJ14, GGR14, GGR15] show. A good, short introduction to graph homomorphism can be found in the excellent introduction of Cai, Chen, and Lu [CCL13] and a richer historic background in [GT11].

However, the arithmetic circuit complexity of graph homomorphisms was still open. The previous results could only show that the hard cases have no polynomial size circuits for counting the number of homomorphisms but it was unclear if these problems are VNP complete. Recently, a dichotomy for graph homomorphisms was shown by Rugy-Altherre [Rug12]. While not explicit, this paper also uses generating functions as the base structure which we explained in Section 2.4. However, their result was for the first side of the graph homomorphism problem.

In this chapter we look at a problem related to the first side of the problem to complete the picture for the arithmetic circuit world. In words, we will look at all graphs from some graph class $\mathcal{G}$ that are homomorphic to a given graph $H$ and encode these graphs into polynomials. In contrast to [DJ04] we will look at the arithmetic circuit complexity in terms of VP and VNP instead of a parameterized counting version. Hence, we could not get a general dichotomy theorem. We will look at cycles, cliques, trees, outerplanar graphs, planar graphs and graphs of bounded genus (for various different genera).

Recently, a new use was found for homomorphism polynomials. They can be used to give natural characterizations of VP independent of the circuit definition as seen by Durand, Mahajan, et al. [DMMRS14]. They showed that all homomorphisms from a balanced binary tree with $n$ leaves to a complete graph on $n^6$ vertices on specific weights is VP complete. However, their polynomially differs from the one presented here. They looked at a polynomial encoding all homomorphism from a given graph $G$ to a given graph $H$ where they have weights on edges and vertices. Our polynomials encode all graphs from some graph class homomorphic to a given graph $H$. While these two problems are very different, we can see our result as showing that that some straightforward candidates originating from the counting world do not give a characterization of VP.

Section 5.2 gives a formal introduction to our model. We prove our dichotomies in Sections 5.3.1 to 5.3.6 where the constructions in Sections 5.3.4 to 5.3.6 build upon each other. The construction in Section 5.3.3 will use a slightly different model as the other sections. We will give a brief introduction into concepts from graph genus in Section 5.2.2 and the completeness proof for these homomorphism polynomials in Section 5.3.6.

## 5.2 Model and Notation of Homomorphism Polynomials

We refer the reader to Chapter 2 and especially Section 2.4 for graph properties and for some complete problems we need. However, we first introduce some simple facts about planar and outerplanar graphs. After this in Section 5.2.2 the concept of graph genus will be introduced to later define some interesting graph classes. A rigorous description of the problem will be given in Section 5.2.3.

### 5.2.1 Facts about Planar and Outerplanar Graphs

This subsection gives some background knowledge on the few graph classes we use. If the reader is familiar with planar and outerplanar graphs and minors he is free to skip to the next subsection.

We first state what a minor of a graph is.

**Definition 5.1.** *$G'$ is a* minor *of a graph $G$ if $G'$ can be obtained by deleting edges or contracting edges. Here, contracting the edge $(u, v)$ deletes the edge $(u, v)$ adds $N(u)$ to $N(v)$ and deletes the vertex $u$.*

We call the operation *edge deletion* and *edge contraction*. The inverse operation for edge contraction is denoted by *edge subdivision*.[2]

**Definition 5.2.** *A graph $G$ is called* planar *if it can be drawn in the plane without any edges crossing.*

[2] As the reader might notice there are multiple inverses of one contracting operation depending on how we split the edges. It will be clear from the proof which one we took.

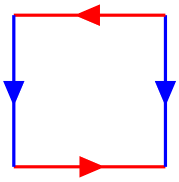


Figure 5.1: Crosscap

This is a well known graph class. There are two well known facts about planar graphs which we state in the next fact.

**Proposition 5.1.** *Let $G = (V, E)$ be a finite planar graph. Then*

- *If $|V| \geq 3$ then $|E| \leq 3|V| - 6$.*
- *It does not have $K_{3,3}$ or $K_5$ as a minor.*

The second fact is the famous Kuratowski's theorem. We will only use the second fact in this chapter but state the first for completeness.

Next we will briefly state the definition for outerplanar graphs which is a well known restriction of planar graphs.

**Definition 5.3.** *A graph $G$ is called* outerplanar *if it is planar and it can be drawn such that every vertex belongs to the unbounded face of the drawing.*

We can have a similar minor theorem for outerplanar graphs.

**Proposition 5.2.** *A graph $G$ is outerplanar if and only if $G$ does not have $K_{3,2}$ as a minor.*

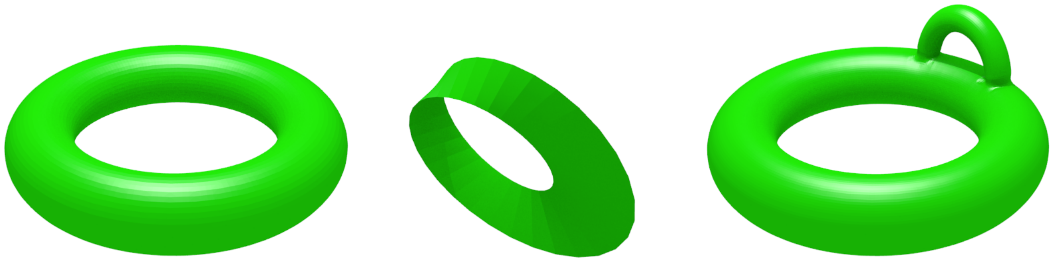
### 5.2.2 A Short Introduction to Graph Genus

As we later need graph genus in Section 5.3.6, we will give a short introduction to this topic here. However, this introduction is not mathematically rigorous and interested readers should look into the textbooks [Arc96] or [Die00] and accompanying literature for a precise handling. Instead we will focus on building an intuition while keeping precise definitions for later reference. To keep this introduction short and useful, we will not define every concept used.

**Definition 5.4.** *We call $\Sigma$ a* surface *if it is a compact Hausdorf topological space.*

Intuitively, the reader can imagine it as being a space where every point has an $\epsilon$ environment around it homeomorphic to some open subset of the euclidean plane in two dimensions. Surfaces give us an extension of a 2 dimensional space.

**Definition 5.5.** *We call a surface* orientable *if it contains no subset that is homeomorphic to a mobius strip.*



(a) Orientable genus one surface (b) Non-orientable genus one surface (c) Orientable genus two surface

Figure 5.2: Surfaces with different genera

Intuitively we can assume a surface be constructed from a disc by either attaching *handles* or *crosscaps* to it. Handles, similar to their real life counterpart, give a loop protruding from the surface. Crosscaps are constructed by gluing the same colored edges in Figure 5.1 together such that the directions match.

Now we can define the three genus definitions we will use. Let $\Sigma$ be a surface. We call a sphere or disc to have orientable, non-orientable and euler genus zero. We can iteratively construct a surface of genus $g$ for the different types of genera with the following definition.

**Definition 5.6.**

- *$\Sigma$ has orientable genus, $\gamma(\Sigma)$, $g$ if it can be constructed from adding $g$ handles to the surface.*
- *$\Sigma$ has non-orientable genus, $\gamma'(\Sigma)$, $g$ if it can be constructed from adding $g$ crosscaps to the surface.*
- *$\Sigma$ has euler genus* $\bar{\gamma}(\Sigma) = \begin{cases} 2-2g & \textit{if the surface can be constructed from } g \textit{ handles,} \\ 2-g & \textit{if the surface can be constructed from } g \textit{ crosscaps.} \end{cases}$

Sometimes the euler genus is named the general genus.

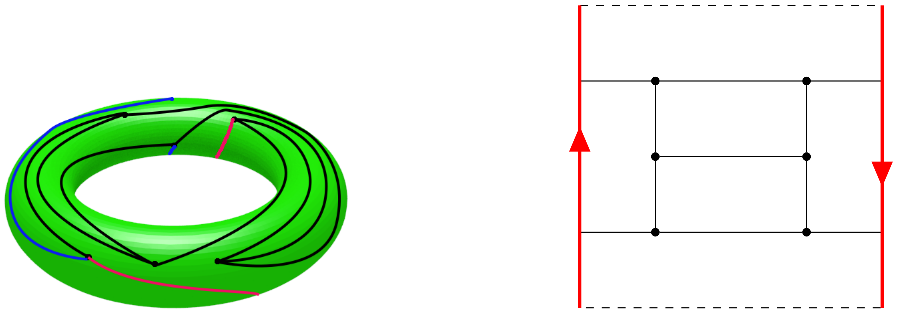
For illustration, we can look at the following figures (Figures 5.2a to 5.2c) for examples of surfaces.

Now we can talk about embedding graphs into these surfaces.

**Definition 5.7.** *We call a graph $G = (V, E)$ embeddable into a surface $\Sigma$ if we can map every vertex $v$ to a point on the surface $\Sigma$ and edges to paths on $\Sigma$ such that no two edges cross each other. We call the mapping from vertices to points the* embedding *of $G$ onto $\Sigma$.*

Intuitively, the handles and crosscaps allow us “hop over” some edges without crossing an actual edge. The next example should make this clear.

In the following we will talk about genus (without any further specification) if it doesn’t matter which genus we talk about. It is easy to see that a planar graph has



(a) Orientable genus one graph (b) Non-orientable genus one graph

Figure 5.3: Graph Genus

genus zero. And that the graph in Figure 5.3a has genus one. Notice, that the graph has a higher number of crossings than genus. Indeed, the genus is only a lower bound for the number of crossings a planar drawing of a graph has to have. The graph in Figure 5.3a is complete bipartite graph on 6 vertices, $K_{3,3}$ with edges colored for clarification.

We also show a graph with non-orientable genus one in Figure 5.3b. Here we have taken the mobius strip but for clarity did not yet glue the red edges together. The graph is also $K_{3,3}$.

**Definition 5.8.** *We call a graph $G$ having orientable genus $\gamma(G)$, non-orientable genus $\hat{\gamma}(G)$ or euler genus $\bar{\gamma}(G)$ if the specific condition holds:*

$$\begin{aligned}\gamma(G) &= \min\{\gamma(\Sigma) \mid \; G \textit{ can be embedded into } \Sigma\},\\ \hat{\gamma}(G) &= \min\{\hat{\gamma}(\Sigma) \mid \; G \textit{ can be embedded into } \Sigma\},\\ \bar{\gamma}(G) &= \min\{\bar{\gamma}(\Sigma) \mid \; G \textit{ can be embedded into } \Sigma\}.\end{aligned}$$

The genus of a graph gives us now a good restriction on the type of graphs we can have. However, dealing with genus and proving by hand that a graph has a specific genus is rather hard. As it turns out it is even hard to test if a graph has a fixed genus as the problem is NP complete ([Tho89]). There exists a fixed parameter tractable algorithm by Filotti, Miller, and Reif [FMR79] or the more recent algorithm with double exponential time in the genus but linear time in the size of the graph by Mohar [Moh99]. As we only need NP completeness and not the fixed paramter tractable algorithms, we state the theorem by Thomassen.

**Theorem 5.2.1** ([Tho89])**.** *Let $k$ be fixed. Testing if a given graph $G$ has $\gamma(G) \leq k$ or $\gamma'(G) \leq k$ or $\bar{\gamma}(G) \leq k$ is* NP *complete respectively.*

While the proof for non-orientable genus was not given by Thomassen it follows from his proof, as was mentioned in [Arc96].

However, we can even use an easier approach. We only need to be able to construct specific graphs with a fixed genus. Intuitively, we should be able to take two graphs with genus one, connect them in a specific way (or create edges such that both graphs are fully enclosed by them) to construct a graph with genus two, the sum of the genera of both graphs. While this looks simple to prove, it is actually more complicated which a simple try on the part of the reader will show. Let us start looking at the orientable genus first.

An additivity theorem was shown by Battle, Harary, and Kodama [BHK62]. We first need a way to glue two graphs together we can either glue them together at vertices, as in the intuition, or on edges.

**Definition 5.9** ([BHK62])**.** *G is a* vertex (edge) amalgam *of $H_1, H_2$ if $G$ is obtained from disjoint graphs $H_1$ and $H_2$ where we identify one vertex (edge) from $H_1$ with one vertex (edge) from $H_2$.*

With this we can state a theorem from [BHK62] to compute the orientable genus of a given graph.

**Theorem 5.2.2** ([BHK62])**.** *Let $\gamma(G)$ be the orientable genus of a graph $G$. Let $G$ be constructed from vertex amalgams of graphs $G_1, \dots, G_n$. Then $\gamma(G) = \sum_{i=1}^{n} \gamma(G_i)$.*

Next we will look at the euler genus which is additive over edge amalgams proven by Miller [Mil87]. The operation of edge amalgams is associative.

**Theorem 5.2.3** ([Mil87])**.** *Let $\bar{\gamma}(G)$ be the euler genus of a graph $G$. Let $G$ be constructed from edge amalgams of graphs $G_1, \dots, G_n$. Then $\bar{\gamma}(G) = \sum_{i=1}^{n} \bar{\gamma}(G_i)$.*

The non-orientable genus is unclear in its additivity property. It was shown by Stahl and Beineke [SB77] that the non-orientable genus of the blocks differ from the non-orientable genus of the vertex amalgam by only one. However, this is not useful for us as we will need to construct graphs with a precise genus. Thankfully, we do not need an additivity theorem, just the existence of a non-orientable genus $k$ graph is necessary for us. Let $K_{n,m}$ be a complete bipartite graph with $n$ vertices on one side and $m$ vertices on the other side. We then have the following theorem proven by Mohar [Moh88].

**Theorem 5.2.4** ([Moh88])**.** *Let $K_{n,m}^{(k)}$ be the graph obtained from $K_{n,m}$ by deleting an arbitrary set of $k$ edges. Then the non-orientable genus of $K_{n,m}^{(k)}$ is given by*

$$\hat{\gamma}(K_{n,m}^{(k)}) = \max\{\lceil \frac{(m-2)(n-2)-k}{2} \rceil, 1\}$$

*unless $n = k, m = k+1$ or $n = k+1, m = k$ and $k$ even or $n = m = k$ for every $k$.*

This immediately shows that there exists a graph for every non-orientable genus $g$ which is even constructible.

As a final ingredient we need this short fact.

**Proposition 5.3.** *The orientable genus, non-orientable genus and the euler genus is closed under edge subdivision.*

This is rather easy to see. If we just subdivide our edge we cannot decrease the genus as the original edge could take the same path as the two new subdivided edges.

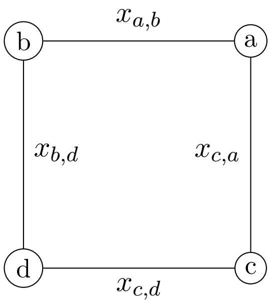


Figure 5.4: Graph $G$

### 5.2.3 The Problem and Related Definitions

We now formulate our main problems we want to study in this chapter. Let $G = (V, E)$, $H = (V', E')$ be undirected graphs. A homomorphism from $G$ to $H$ is a mapping $f : V \to V'$ such that for all edges $\{u, v\} \in E$ there exist an edge $\{f(u), f(v)\} \in E'$. We can define the corresponding generating function as follows.

**Definition 5.10.** *Let $\mathcal{H}_H$ be the property of all connected graphs homomorphic to a fixed $H$. We denote by $\mathcal{F}^{H,n}$ the generating function $\mathcal{F}^{H,n} := \mathrm{GF}(K_n, \mathcal{H}_H)$.*

Let us give a short example for this polynomial. Let $H$ be a triangle and let $\mathcal{H}_H$ as above and $n = 4$. Then all graphs homomorphic to a triangle on four vertices give us the following polynomial.

$$\begin{aligned}\mathrm{GF}(K_4, \mathcal{H}_H) &= x_{1,2}x_{2,4}x_{4,1}\left(1 + x_{3,1} + x_{3,2} + x_{3,4} + x_{1,3}x_{3,2} + x_{1,3}x_{3,4} + x_{4,3}x_{3,2}\right)\\ &\quad + x_{1,2}x_{2,3}x_{3,1}\left(1 + x_{4,1} + x_{4,2} + x_{4,3} + x_{1,4}x_{4,2} + x_{1,4}x_{4,3} + x_{2,4}x_{4,3}\right)\\ &\quad + x_{1,3}x_{3,4}x_{4,1}\left(1 + x_{2,1} + x_{2,3} + x_{2,4} + x_{1,2}x_{2,3} + x_{1,2}x_{2,4} + x_{3,2}x_{4,2}\right)\\ &\quad + x_{2,3}x_{3,4}x_{4,2}\left(1 + x_{1,2} + x_{1,3} + x_{1,4} + x_{2,1}x_{1,3} + x_{3,1}x_{1,4} + x_{2,1}x_{1,4}\right).\end{aligned}$$

Either it is a triangle or a triangle with one edge connected to the fourth vertex or it is a triangle with the remaining vertex connected by two edges. The polynomial gives us exactly these graphs.

We can now continue and state the first dichotomy theorem.

**Theorem 5.2.5** (Rugy-Altherre [Rug12])**.** *If $H$ has a loop or no edges, $\mathcal{F}^{H,n}$ is in* $\mathsf{VAC}_0$ *and otherwise it is* $\mathsf{VNP}$ *complete under c-reductions.*

Instead of looking at all graphs, we want to look at a restricted version. What happens if we do not want to find every graph homomorphic to a given $H$ but every *cycle* homomorphic to a given $H$? We state our problem in the next definitions.

**Definition 5.11.** *Let $\mathcal{E}_n$ be a graph property. Then $\mathcal{F}^{H,n}_{\mathcal{E}_n}$ is the generating function for all graphs in $\mathcal{E}_n$ on $n$ vertices homomorphic to a fixed graph $H$.*

Let us give a simple example. Let $G$ be the graph from Figure 5.4 and $\mathcal{H}_{H,\mathcal{E}_n}$ all *trees* homomorphic to a path of length two. Notice, that this includes connected components which are only single vertices. Hence the trees do not have to span the graph. As all trees are homomorphic to a path of length two, we get the following polynomial.

$$\begin{aligned}\mathrm{GF}(G, \mathcal{H}_{H,\mathcal{E}_n}) = {} & x_{a,b}x_{b,c}x_{c,d} + x_{b,c}x_{c,d}x_{d,a}\\ & + x_{c,d}x_{d,a}x_{b,c} + x_{d,a}x_{a,b}x_{b,c}\\ & + x_{a,b}x_{b,c} + x_{b,c}x_{c,d} + x_{c,d}x_{d,a}\\ & + x_{d,a}x_{a,b}\\ & + x_{a,b} + x_{b,c} + x_{c,d} + x_{d,a}.\end{aligned}$$

Let us now change the graph class to be all triangles, we notice that no triangle is homomorphic to a path of length two and hence our polynomial $\mathrm{GF}(G, \mathcal{H}_{H,\mathcal{E'}_n})$ is zero. This shows that similar settings can give very different sets of graphs and hence polynomials. In general it is unclear how easy or hard the resulting polynomials are. We will look at the following polynomials in this chapter.

**Definition 5.12.** *We define the following graph polynomials.*

- $\mathcal{F}^{H,n}_{cycle_n}$ *where* $cycle_n$ *is the property where one connected component is a cycle and the others are single vertices in a graph of size* $n$*.*
- $\mathcal{F}^{H,n}_{clique_n}$ *where* $clique_n$ *is the property where one connected component is a clique and the others are single vertices in a graph of size* $n$*.*
- $\mathcal{F}^{H,n}_{tree_n}$ *where* $tree_n$ *is the property where one connected component is a tree and the others are single vertices in a graph of size* $n$*.*
- $\mathcal{F}^{H,n}_{outerplanar_n}$ *where* $outerplanar_n$ *is the property where one connected component is a outerplanar graph and the others are single vertices in a graph of size* $n$*.*
- $\mathcal{F}^{H,n}_{planar_n}$ *where* $planar_n$ *is the property where one connected component is a planar graph and the others are single vertices in a graph of size* $n$*.*
- $\mathcal{F}^{H,n}_{\gamma,k,n}$ *where* $\gamma, k, n$ *is the property where one connected component has orientable genus* $k$ *and the others are single vertices in a graph of size* $n$*.*
- $\mathcal{F}^{H,n}_{\hat{\gamma},k,n}$ *where* $\hat{\gamma}, k, n$ *is the property where one connected component has non-orientable genus* $k$ *and the others are single vertices in a graph of size* $n$*.*
- $\mathcal{F}^{H,n}_{\bar{\gamma},k,n}$ *where* $\bar{\gamma}, k, n$ *is the property where one connected component has euler genus* $k$ *and the others are single vertices in a graph of size* $n$*.*

We will use the notation $\mathcal{F}_{\text{cycle}}$, $\mathcal{F}_{\text{clique}}$, $\mathcal{F}_{\text{tree}}$, $\mathcal{F}_{\text{outerplanar}}$, $\mathcal{F}_{\text{planar}}$, $\mathcal{F}_{\gamma,k}$, $\mathcal{F}_{\hat{\gamma},k}$ and $\mathcal{F}_{\bar{\gamma},k}$ as a shorthand.

## 5.3 Dichotomies

We now assume all our circuits to be over the field of the real numbers $\mathbb{R}$. The reductions also work over $\mathbb{C}$.

Before we show the dichotomies we need one hardness proof which will be very useful to us throughout this section.

**Lemma 5.3.1.** *Let $\mathcal{HP}$ is the property where the single connected component is a Hamiltonian path from vertex denoted by one to vertex denoted by $n$ that are not cycles. Then* $\mathrm{GF}(K_n, \mathcal{HP})$ *is* $\mathsf{VNP}$*-hard.*

*Proof.* We will reduce this to Hamiltonian cycle on $K_n$. Let $F$ be the polynomial $\mathrm{GF}(K_n, \mathcal{HP})$. We will compute $\sum_{1<i<j<n} \mathrm{HOMC}_1^{(1,i)}(\mathrm{HOMC}_1^{(n,j)}(F))$. Now we can replace every $x_{\{u,n\}}$ with $x_{\{u,1\}}$. This gives us now all Hamiltonian cycles on $n-1$ vertices. □

In general we call the Hamiltonian paths without cycles Hamiltonian paths.

Almost all of our dichotomy proofs work the following way. Given the polynomial of all graphs in some specific graph class homomorphic to $H$ we will extract only a sum of some specific monomials with the help of homogeneous components. In general, we restrict all the sets of graphs to have a specific type. We can then evaluate some variables in this sum with constants to get the sum over all monomials for Hamiltonian cycles, Matchings or Hamiltonian paths. This will yield the reduction. Most of our polynomials will be hard even if $H$ has just one edge.

### 5.3.1 Cycles

As a first graph class we look at cycles. The proof for the dichotomy will be relatively easy and gives us a nice example to get familiar with homomorphism polynomials and hardness proofs.

Our main dichotomy for cycles is the following theorem.

**Theorem 5.3.2.** *If $H$ has at least one edge or has a self-loop, then $\mathcal{F}_{cycle}$ is* $\mathsf{VNP}$ *complete under c-reductions. Else it is in* $\mathsf{VAC}_0$*.*

The next simple fact shows us which cycles are homomorphic to a given graph $H$. Let $\mathrm{ev}(n)$ be defined as $n$ if $n$ is even and $n-1$ if $n$ is odd.

**Proposition 5.4.** *Given $H$ a graph with at least one edge, all cycles of length* $\mathrm{ev}(n)$ *are homomorphic to $H$.*

It is easy to see that by folding the graph in half we get one path which is trivially homomorphic to an edge. Our hardness proof will only be able to handle cycles of even length. Luckily this is enough to prove hardness.

**Lemma 5.3.3.** *Let $\mathcal{UHC}_{\mathrm{ev}(n),even}$ be the graph property of all cycles of length* $\mathrm{ev}(n)$*. Then* $\mathrm{GF}(K_{\mathrm{ev}(n)}, \mathcal{UHC}_{\mathrm{ev}(n),even})$ *is* $\mathsf{VNP}$*-hard under c-reductions.*

*Proof.* If $n$ is even, we can immediately use the hardness of $\mathrm{GF}(K_n,\mathcal{UHC}_n)$ (cf. Theorem 2.4.1).

If $n$ is odd, we need to fix one edge to get all cycles of some even length. We do this by evaluating $\mathrm{GF}(K_{n+1},\mathcal{UHC}_{\mathrm{ev}(n),\mathrm{even}})$. Notice how $K_{n+1}$ now has $\mathrm{ev}(n+1) = n+1$ vertices. We now need to contract one edge to get the polynomial $\mathrm{GF}(K_n,\mathcal{UHC}_n)$. We do this with the following argument. We enforce, via taking the homogeneous component of degree one of $x_{n+1,1}$, all cycles to use $x_{n+1,1}$. Additionally to this restriction, we will sum over all our given cycles where $x_{1,i}$ and $x_{n,j}$ are enforced for $i < j$ (cf. the proof of Lemma 5.3.1). We then replace $x_{i,n+1}$ by $x_{i,1}$ for all $i$ and set $x_{n+1,1}$ to one. This gives us all cycles of length $n$.

To prove the contraction let us look at the following argument. Let the edge $(n+1,1)$ be the edge we contract and let $i,j$ be the points picked in the sum. If we connect $i,j$ with a path through every point we can complete this into a cycle only one way. Notice, that every different choice of $i,j$ will construct a different cycle if we contract 1 and $n+1$ and that this construction even works for the characteristic of the field being equal to two.

This concludes our reduction to $\mathrm{GF}(K_n,\mathcal{UHC}_n)$. As our circuit can easily divide by two if the polynomial is over any field. □

Later proofs will also use the contracting idea from the previous lemma. A simple case distinction will give us the proof of the theorem.

*Proof of Theorem 5.3.2.* If $H$ has at least one edge, we know from Proposition 5.4 that all even cycles are homomorphic to $H$ and by this represented in our polynomial. If we take the homogeneous components of degree $\mathrm{ev}(n)$, we extract all even cycles of length $\mathrm{ev}(n)$. This is VNP-hard via the previous Lemma (5.3.3).

If $H$ has a self-loop, we can map all cycles to the one vertex in $H$. We can then extract the Hamiltonian cycles of length $n$ by using the homogeneous degree of $n$ as all cycles are homogeneous to a self-loop.

If $H$ has no edge, our polynomial is the zero polynomial as we cannot map any graph $G$ containing an edge to $H$.

Using Valiant's Criterion, we can prove membership of $\mathcal{F}_{\mathrm{cycle}}$ in VNP (Theorem 2.1.1). □

### 5.3.2 Cliques

Here, we will not use cycles in the hardness proof but use the clique polynomial which was defined by Bürgisser, directly. The completeness proof is an easy exercise. In contrast to the other results, we show that computing $\mathcal{F}_{\mathrm{clique}}$ is easy for most choices of $H$.

**Theorem 5.3.4.** *If $H$ has a self-loop then $\mathcal{F}_{clique}$ is* VNP *complete under c-reductions. Otherwise $\mathcal{F}_{clique}$ is in* $\mathsf{VAC}_0$.

*Proof.* Let $H$ have no self-loop. We can use that $H$ has constant size which implies that $H$ has a maximal subgraph which forms a clique or $H$ has no clique. We include cliques of size two (edges) and of size one (single vertices without a self-loop) in this.

Let us now look at the case for cliques of size $c$. We can compute $\mathcal{F}_{\text{clique}}$ explicitly by a brute-force algorithm. The number of monomials can be bounded by the following argument. There are $\sum_{i=1}^{c} \binom{n}{i}$ many different cliques. As we can bound $\binom{n}{i}$ by $n^i$ we get an upper bound of $cn^c$ monomials. Further inspection yields, that constant depth, unbounded fan-in circuits of polynomial size are enough to compute all cliques up to size $c$.

Let $H$ now have one self-loop. The fact that all cliques are homomorphic to a given graph with a self-loop tells us that $\mathcal{F}_{\text{clique}}$ contains different monomials for all cliques of size $i$ for $i = 1, \ldots, n$. The $\mathsf{VNP}$ hardness follows via Theorem 2.4.2.

The empty graph has the zero polynomial.

As a polynomial time deterministic machine can easily check if a given instance is a clique, we can use Valiant's Criterion to show membership in $\mathsf{VNP}$. □

### 5.3.3 Trees

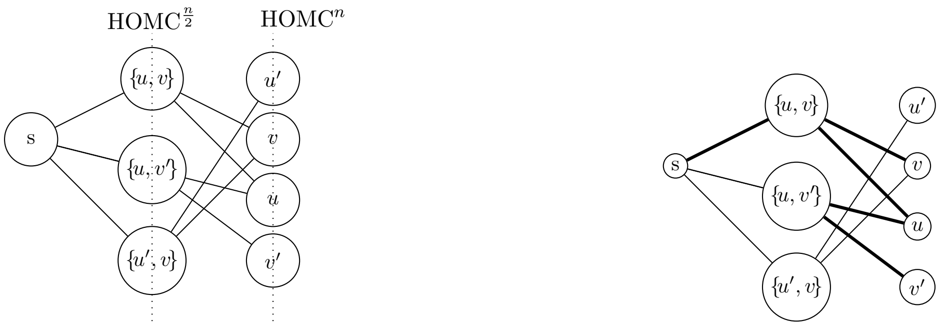
As the new characterization of VP had a specific tree structure we want to look at the general problem. In previous sections our polynomial just contained the edges of the graph but for this section we need a slightly different model. But first, let us motivate why this is necessary. In this introduction, we will keep the definitions of the homomorphisms and graph classes informal. If our graph class is the set of all spanning trees then we can use a slight modification of Kirchhoff's algorithm. We set our matrix to have the variables $x_{i,j}$ and the Laplacian to be $a_{i,j} = \sum_{j' \in N(j)} x_{i,j'} - x_{i,i}$ where $N(j)$ is the neighborhood of $j$ excluding $j$. The determinant of $(a_{i,j})$ gives us all spanning trees of the graph. As trees are homomorphic to an edge we get immediatly the fact that computing all homomorphisms from all spanning trees is easy.

However, we generally defined our homomorphisms polynomials to have one connected component which can have smaller size than $n$. Another try might be to get all forests instead of all trees. It is unclear how we can modify Kirchhoff's algorithm to achieve this. Just inserting $1 + x_{i,j}$ for variable $x_{i,j}$ gives us a weighted sum of all trees as we will count different forests a different number of times. In fact, the problem of counting forests is #$\mathsf{P}$-hard as it is equal to evaluating the Tutte polynomial at the point $(2, 1)$ which was proven to be hard by Jaeger, Vertigan, and Welsh [JVW90].

Instead we will use our original definition for trees[3] and the following modification to our generating function. If a monomial in our polynomial would select the edges $E'$ we also select the vertices $\{u, v \mid \{u, v\} \in E'\}$ in our monomial. In essence, we will also select the vertices forming the edges, giving us polynomials with variables $X = \{x_e \mid e \in E\} \cup \{x_v \mid v \in V\}$. We will later give details why this special form has an advantage.

While this might look rather arbitrary, a similar generating function was used in

[3] Remember, here one connected component is a tree and the others are single vertices.



(a) Reduction from Trees to matching

(b) Invalid Construction

[DMMRS14] but augmented with an additional function $\alpha : V \to \mathbb{N}$ as the power of the variable $x_v$ and some restrictions. As they can show equality of their model with $\mathsf{VP}$ there is hope that this addition to our model does not increase the complexity. While we give hardness results, it is unclear if the original problem is $\mathsf{VNP}$-hard or if our extension made the model to powerful.

**Theorem 5.3.5.** *If $H$ contains an edge, then $\mathcal{F}_{tree}$ is $\mathsf{VNP}$ complete under c-reductions. Otherwise $\mathcal{F}_{tree}$ is in $\mathsf{VAC}_0$.*

*Proof.* We use a reduction from connected partial trees to perfect matchings. It is obvious that a tree is always homomorphic to one edge.

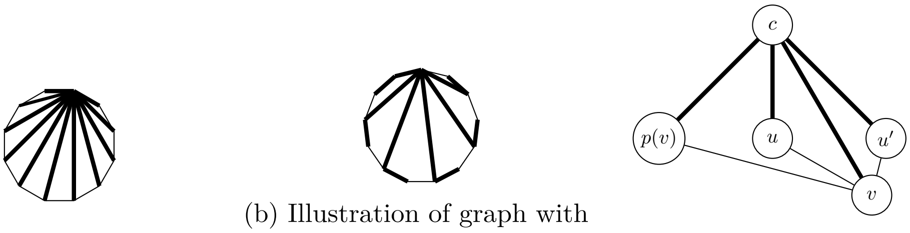
We want to compute a matching on a graph given by $(V, E)$. We can build a graph as in Figure 5.5a from a $K_n$ by setting the weight of every edge not given to zero. In detail, our graph has vertices $\{v \in V\} \cup \{v_e \mid e \in E\} \cup \{s\}$. We add the edges $\{(u, v), u\}, \{(u, v), v\}$ and $\{s, v_e\}$ for every $e \in E$. Vertices of the form $\{v_e \mid e \in E\}$ will be called *edge-vertices* in this proof. Now as the vertices are given by our polynomials we can take the homogeneous components over vertices. We take the homogeneous components of degree $n/2$ over vertices $\{v_e \mid e \in E\}$ and of degree $n$ of vertices $v \in V$. Our matching in the original graph is given by the edges $(s, v_e)$. Every matching in the original graph has obviously a tree in our graph.

The other direction is left. Given a tree in our graph, we know that only $n/2$ edge-vertices are selected. As every vertex $v \in V$ has to be connected by an edge, edge-vertices have to go to pairwise different sets of $v \in V$. Hence we can compute a perfect matching which is as hard as computing the permanent.

Valiant's Criterion will again show the membership. □

We crucially need the fact that we get the adjacent vertices for free in our homomorphism polynomials. The reader might think restricting the edges out of $s$ might suffice but this is not the case. Let us look at Figure 5.5b where we removed the restriction that a specific number of vertices have to be selected. The thick path then gives us a valid tree but an invalid matching. In this case our monomial would be

$$x_{s,\{u,v\}}x_{\{u,v\},v}x_{\{u,v\},u}x_{\{u,v'\},u}x_{\{u',v\},v'}$$



(a) Triangle graph

(b) Illustration of graph with buddy vertices

(c)

Figure 5.6

if we ignore the variables $x_v$ for all $v$. We can see that replacing every edge $x_{a,b}$ with the variable $y_a y_b x_{a,b}$ for vertices $a, b$ in our constructed graph and new variables $y_a$, $y_b$ can be used for extracting homogeneous components. However, the monomial above has already degree 4 in the new $y$ variables, namely $y_v y_u y_u y_{v'}$ is a factor of our monomial. Hence, this cannot be used to forbid the wrong instances.

It is unclear how to forbid this general behaviour without using our modified generating function. Splitting the vertices into two parts does not help but splitting a vertex into $n$ many different vertices might. However, taking homogeneous components then would give us exponentially sized circuits. If we restrict the graph such that we would always select all edges outgoing from $s$ we would prevent this case but the reconstruction of a matching is non trivial.

### 5.3.4 Outerplanar Graphs

Next we will show a dichotomy for outerplanar graphs. We start with the case of a triangle homomorphic to $H$. We will use the case of triangles homomorphic to $H$ as a simple stepping stone.

**Lemma 5.3.6.** *If a triangle is homomorphic to $H$ then $\mathcal{F}_{outerplanar}$ is* VNP*-hard under c-reductions.*

*Proof.* We have given an outerplanar graph that is homomorphic to a triangle and want to compute the Hamiltonian path polynomial.

We will enforce a construction as in Figure 5.6a to occur exclusively in a polynomial $p$ we compute with the help of homogeneous components from $\mathcal{F}_{\text{outerplanar}}$. For this, we pick an arbitrary vertex $c$ and enforce all $n$ outgoing edges from this vertex via homogeneous components. We further enforce the whole graph to have $n + n - 3$ edges. The graph given in Figure 5.6a is outerplanar as the outerplanar embedding (where all vertices belong to the unbounded face) is given in the figure. Let us call the set of graphs represented by the polynomial $p$ now $S$. We still need to proof that all graphs in $S$ are isomorphic to Figure 5.6a.

We call the implied order of the graph, the order of the outer circle of vertices starting from the star and ending at it again without any edges crossing. As there are two such orderings let us fix an arbitrary one for every graph. Let us now look at a graph which has not an implied order of the outer vertices. This implies that there exists a vertex $v$ which has a degree of at least 4. With our ordering every vertex (except $c$) up to and including the later defined vertex $v$ has a single predecessor. Furthermore, let $v$ be the first vertex with a degree of at least 4 in this order and let $p(v)$ be the predecessor of $v$ in this order. Notice that by enforcing all $n$ instead of just $n-2$ edges starting at the center, a predecessor $p(v) \neq c$ has to exist.

Let $u, u'$ denote the other vertices adjacent to $v$ different than $p(v)$ and $c$. As we enforced edges from $c$ to every vertex, we can easily see the $K_{2,3}$ with $v, c$ on the one side and $u, u', p(v)$ on the other side. Hence the graph cannot be outerplanar. This implies that every vertex except $c$ and the two neighbouring vertices have degree at most 3. Enforcing the overall number of edges gives us at least degree 3 and hence implies equality.

Now we the sum of all monomials corresponding to the graph in Figure 5.6a. The outer path gives us almost all Hamiltonian paths. In fact, it gives us all permutation of $n-2$ vertices. We need to remove the center of the star by evaluating the edges with one as well as evaluate the other enforced edges with one. This polynomial is now VNP-hard by Lemma 5.3.1. Taking the homogeneous components as described only increases the circuit by a factor of $n$.

It is easy to see that our graph is homomorphic to a triangle. □

With this in mind we can now show a completeness result.

**Theorem 5.3.7.** *If $H$ has an edge then $\mathcal{F}_{outerplanar}$ is* VNP *complete under c-reductions and otherwise trivial.*

*Proof.* For every vertex $v$, except $c$, we choose a *buddy vertex $v'$*. We enforce the edge between every vertex and his buddy vertex and set the edge between a buddy vertex and $c$ to zero. Additionally, we set all edges from $v$ to any other non buddy vertex to zero and all edges from a buddy vertex to a different buddy vertex to be zero. In essence this splits every vertex into a left and right part (see Figure 5.6b). Similar to the proof of Lemma 5.3.6 we enforce edges from $c$ to $v$ for every non buddy vertex and to take exactly $n + 2n - 3$ edges. Let us call the sets of all graphs here $S'$ and the set of all graphs from Lemma 5.3.6 $S$.

Let us now prove that the set of all graphs constructed here and in Lemma 5.3.6 are of the same size. Take an outerplanar graph $G'$ from $S'$. We can then contract the edges between a vertex and its buddy vertex. This gives us a graph $G$ in $S$. If the combined degree of a vertex and its buddy vertex (disregarding the connecting edge) would be greater than 3 $G'$ would be not outerplanar. The reason for this is again the contraction. As we can contract it to a graph with a vertex of degree 4 of the form in $S$ the proof of Lemma 5.3.6 would tell us that this graph would be not outerplanar. As this graph is a minor of the graph in $S'$, $G'$ would not be outerplanar. In essence, our graph with combined degree greater than 3 can be constructed from a $G$ by adding and subdividing edges which can be constructed from $K_{2,3}$ if $G$ was not outerplanar.

For the other direction a similar proof holds. If we have given a graph as in Figure 5.6a we can subdivide the edges as stated earlier and have a graph of the form

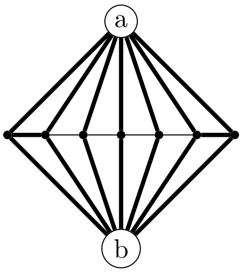


Figure 5.7: Planar Gadget

Figure 5.6b.

Now that we know that the sets of graphs $S$ and $S'$ are of the same size, we need to transfer the hardness. This is easy as we can do the following variable replacement. Let us look at the variable $x_{u',v}$ corresponding to the edge $(u',v)$, where $u'$ is the buddy vertex of $u$ and $v$ is a non buddy vertex. We can evaluate this with $x_{u,v}$. Similarly, for the edge $(v',u)$ we can evaluate $x_{v',u}$ with $x_{v,u}$. Finally, we evaluate $x_{v,v'}$ with 1. As there exists a bijection from $S$ to $S'$ and our replacement is also bijective, we constructed the polynomial $\mathcal{F}_{\text{outerplanar}}$ as in Lemma 5.3.6 even if $H$ is homomorphic to an edge.

Taking the homogeneous components increases the circuit size by a factor of $n$.

We know by [Mit79] that checking if a graph is outerplanar is possible in linear time. With this we can use Valiant's Criterion to show the membership. □

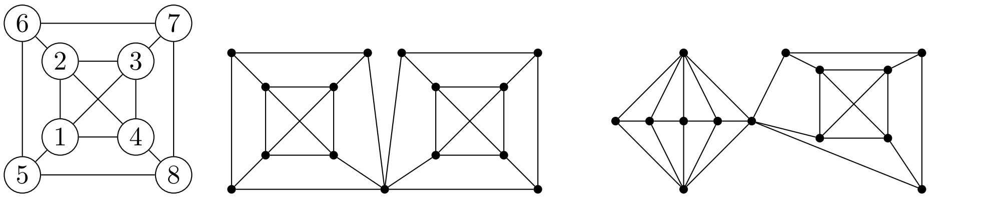
### 5.3.5 Planar Graphs

The next lemma will show that we can use homomorphisms restricted to planar graphs to construct all permutations of a set of vertices.

**Lemma 5.3.8.** *All graphs isomorphic to Figure 5.7 with the thick edges fixed and $n+2+2(n+2)$ edges required contain all possible paths on a set of vertices we can denote by $v'_1, \dots, v'_n$.*

*Proof.* Take an embedding in the plane of the graph without any crossings. If we show that every vertex has at most one edge going to the right, it follows that the set of vertices from left to right ordered is a permutation of the vertices. By slightly adjusting this to exclude the first and last two vertices and the vertices denoted by $a$ and $b$ we get a path for these vertices. We call these vertices $v'_1, \dots, v'_n$.

Let us look at the following subgraph with the left to right ordering. Let $v$ be a vertex with two right successors $u, u'$ and a predecessor $p(v)$. By construction the predecessor always exists. We denote the top and bottom vertex by $a$ and $b$ in our graph. We can now build a $K_{3,3}$ minor in the following way. $S_1 = \{v, a, b\}$ and $S_2 = \{u, u', p'\}$. As $a$ and $b$ are connected to every vertex we only need to check that $u$ is connected to $u, u'$ and $p$ which is by assumption. This proves that via edge deletion our graph would have a $K_{3,3}$ minor if the vertices would not give us a permutation. □



(a) Gadget (b) Two Gadgets (c) Gadget with planar gadget

Figure 5.8

**Theorem 5.3.9.** *If $H$ has an edge then $\mathcal{F}_{planar}$ is* VNP *complete under c-reductions. Otherwise $\mathcal{F}_{planar}$ is in* $\mathsf{VAC}_0$.

*Proof.* We again get all paths by enforcing our polynomial to have the graph from Figure 5.7. By Lemma 5.3.8 this gives us all path which is VNP-hard by Lemma 5.3.1.

However, this graph is not yet homomorphic to a single edge. To accomplish this, we will use a graph of size $2n$. We, as in the outerplanar case, enforce every vertex, except $a$ and $b$, to have a buddy vertex $u_v$. Then we subdivide the edge $(a, v)$ and $(b, v)$ for every original, meaning none buddy, vertex $v$ with a new vertex $v'_a$, $v'_b$ respectively. This will give us for every part a square consisting of the vertices $a, v, v'_a, u_v$ and the square $b, v, v'_b, u_v$.

Now it is easy to see that we can fold $a$ to $b$ which leaves us with a grid of height one. A grid can be easily folded to one edge. The size of the circuit is increased by a factor of at most $2n$.

As testing planarity is easy, we can use Valiant's Criterion to show membership. □

### 5.3.6 Genus $k$ graphs

With the planar result in place we can use the simple proof strategy. Construct a genus $k$ graph where we append the planar construction. If we now enforce all our graphs in the homomorphism polynomial to have these graphs our genus bound will ensure that our planar gadget gives us all Hamiltonian paths of vertices. Of course this holds only if the combination of the genus $k$ graph and the planar construction does not reduce the genus.

**Lemma 5.3.10.** *The graph in Figure 5.8a has orientable genus one.*

*Proof.* We can use the given embedding with one handle for the crossing in the middle to show an upper bound of one.

We again construct a $K_{3,3}$ with the sets $S_1 = \{2, 1, 6'\}, S_2 = \{3, 4, 7'\}$ where $6'$ is the vertex constructed from contracting the edge $(5, 6)$ and $7'$ from the edge $(7, 8)$. And hence the graph is not planar and has a lower bound for the orientable genus of one. □

Now we can use Theorem 5.2.2 to glue these graphs together. This now gives us immediately the result that a graph constructed as in Figure 5.8b with $k$ gadgets has orientable genus $k$.

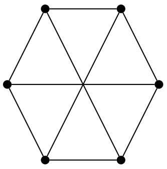


Figure 5.9: Non-orientable Genus one graph

**Theorem 5.3.11.** *If $H$ has an edge then $\mathcal{F}_{\gamma,k}$ is* VNP *complete under c-reductions for any $k$. Otherwise $\mathcal{F}_{\gamma,k}$ is in* $\mathsf{VAC}_0$*.*

*Proof.* With Theorem 5.2.2, Lemma 5.3.8 and the construction in Figure 5.8 we are almost done. Because we enforced a orientable genus $k$ graph to occur. All graphs that homomorphic to the planar gadget have orientable genus zero and hence are planar. Therefore, as in Lemma 5.3.8 we can extract all Hamiltonian paths. This is VNP-hard according to Lemma 5.3.1.

The only thing left to do is to modify our graphs such that they are homomorphic to an edge without violating the properties. It is clear that we can fold our orientable genus one gadgets together. If we then subdivide the edge $(1, 3)$ and $(2, 4)$ (which keeps our block property) we can first fold 7 to 5 and 3 to 1. Folding then again 6 to 8 and 2 to 4 we get a square with two dangling edges. The dangling edges can be folded onto the square and the square is homomorphic to one edge. This construction increases the size of the circuit at most by a factor of $14k + 2n$. As testing for a fixed orientable genus is in NP (see Theorem 5.2.1), we can use Valiant's Criterion to show membership. □

We only need a similar gadget with euler genus one. By the definition of euler genus it is enough to give a graph with non-orientable genus one. We could use the later Theorem 5.2.4 to construct but in the interest of the understanding of the reader we will show a simple graph with euler genus one.

**Lemma 5.3.12.** *The graph in Figure 5.9 has non-orientable genus one and hence euler genus one.*

*Proof.* It is clear that the graph is not planar. Let us enumerate the vertices counterclockwise with the numbers $1, \dots, 6$. The sets $\{1, 3, 5\}$ and $\{2, 4, 6\}$ give us a $K_{3,3}$ minor. It is also clear that we can use one crosscap in the middle to embed this graph into a surface of non-orientable genus one. □

We can also now prove a similar theorem for the euler genus and the non-orientable genus.

**Theorem 5.3.13.** *If $H$ has an edge then $\mathcal{F}_{\bar{\gamma},k}$ is* VNP *complete under c-reductions for any $k$. Otherwise $\mathcal{F}_{\bar{\gamma},k}$ is in* $\mathsf{VAC}_0$*.*

*Proof.* The proof works as in the previous theorem. As every genus is closed under edge subdivision (Proposition 5.3) and contraction we can similarly augment our graph to construct a graph homomorphic to one edge. We get again membership with Valiant's criterion as testing euler genus is in NP (see Theorem 5.2.1). □

**Theorem 5.3.14.** *If $H$ has an edge then $\mathcal{F}_{\hat{\gamma},k}$ is* VNP *complete under c-reductions for any $k$. Otherwise $\mathcal{F}_{\gamma,k}$ is in* $\mathsf{VAC}_0$.

*Proof.* By Theorem 5.2.4 we now can construct a graph that has non-orientable genus $k$. With this we can continue as in the proof of Theorems 5.3.11 and 5.3.13. Notice that a bipartite graph can always be folded to an edge. Finally, with the fact that testing for a fixed non-orientable genus is in NP (see Theorem 5.2.1) we get again membership. □

## 5.4 Open Problems

We have shown many dichotomy results for different graph classes but some classes are still open. We want to especially mention the case of our graph class being the class of trees. We mentioned earlier that we can use Kirchoff's Theorem to find all spanning trees of a given graph. This, however, does not include monomials of total degree less than $n-1$ which our polynomials include. It is unclear how to decrease the size of the trees without disconnecting them into forests[4]. And even constructing all forests is non trivial. From the algebraic view, the knowledge ends here. In the counting view, where we solve the task of counting all trees in a graph, a bit more is known. Goldberg, Grohe, et al. [GGJT10] showed that counting the number of subtrees that are distinct up to isomorphism is #P complete. This, combined with our dichotomy for trees including the vertices, gives us a strong indication that the similar problem is VNP-hard in the algebraic world.

A different expansion of these results would be the case of bounded treewidth. As mentioned earlier, in the counting version the case of bounded treewidth is indeed the most general form and completely characterizes the easy and hard instances of counting graph homomorphisms. Additionally, recent advancements showed that graph homomorphisms of a specific type characterize VP. Can homomorphism from graph classes parameterized by treewidth, similar to the counting case, be used for a complete characterization of VP and VNP?

An interesting research direction would be the case of disconnected graph properties. Rugy-Altherre looked at the property that any graph is homomorphic to a given graph $H$. This includes disconnected graphs with connected components larger than one vertex. We instead only looked at restricted homomorphisms where one major connected component exists. It is unclear to the author if our proofs could be adapted to this case.

Finally, the presented hardness proofs rely solely on the fact that we almost always get all permutation of a graph for free. Instead we could look at the following model. Instead of graph properties, we take a graph class without isomorphic copies and then

[4] Here forests are the set of all edges that do not have any cycle.

weight the isomorphic copies of the graphs by the sign of their mapping. In essence, our polynomial would be

$$\sum_{G \in \mathcal{G}} \sum_{\sigma \in S_n} \psi(G, H) \operatorname{sgn}(\sigma) \prod_{e \in G} x_{e,\sigma(e)}$$

where $\psi(G, H)$ is one if there exists a homomorphism from $G$ to $H$ and zero otherwise.

Even the case where $G$ is a fixed graph might be interesting. Here, while the permutation is still for free, it is also weighted similar to the determinant which could drastically decrease the complexity of the polynomials. However, while every graph with their permutations could now easily be constructed with the help of the determinant, it is unclear how different graphs homomorphic to the same $H$ could be incorporated.

# 6 Average Case Analysis of Graph Algorithms on Metric Graphs

The following chapter is based on joint work with Bringmann, Manthey and Rao and was published in MFCS under [BEMR13].

## 6.1 Introduction

In this chapter we step away from arithmetic circuits and study general optimization problems. For large-scale optimization problems, finding optimal solutions within reasonable time is often impossible, because many such problems, like the traveling salesman problem (TSP), are NP-hard. Nevertheless, we often observe that simple heuristics succeed surprisingly quickly in finding close-to-optimal solutions. Many such heuristics perform well in practice but have a poor worst-case performance. In order to explain the performance of such heuristics, probabilistic analysis has proved to be a useful alternative to worst-case analysis. Probabilistic analysis of optimization problems has been conducted with respect to arbitrary instances (without the triangle inequality) [Fri04, Kar77] or instances embedded in Euclidean space. In particular, the asymptotic behavior of various heuristics for many of the Euclidean optimization problems is known precisely [Yuk98].

However, the average-case performance of heuristics for general metric instances is not well understood. This lack of understanding can be explained by two reasons: First, independent random edge lengths (without the triangle inequality) and random geometric instances are relatively easy to handle from a technical point of view – the former because of the independence of the lengths, the latter because Euclidean space provides a structure that can be exploited. Second, analyzing heuristics on random metric spaces requires an understanding of random metric spaces in the first place. While Vershik [Ver04] gave an analysis of a process for obtaining random metric spaces, using this directly to analyze algorithms seems difficult. Let us give a quick overview how the construction works on the example of the exponential distribution. First we pick a distance between vertices $v_1, v_2$ by the exponential distribution. Depending on $d(v_1, v_2)$ we have to pick two new distances such that the triangle inequalities between $v_3$, $v_1$ and $v_2$ is fulfilled. We do this by restricting and scaling the exponential distribution to these intervals. With this we can iteratively construct a "most random metric space" as Vershik calls it.

In order to initiate systematic research of heuristics on general metric spaces, we use the following model, proposed by Karp and Steele [KS85, Section 3.4]. Given an undirected complete graph, we draw edge weights independently at random according to exponential distributions with parameter one. The distance between any two vertices is the total weight of the shortest path between them, measured with respect to the random weights. We call such instances *random shortest path metrics*.

This model is also known as *first-passage percolation*, and has been introduced by Broadbent and Hammersley as a model for passage of fluid in a porous medium [Bla10, BH57]. More recently, it has also been used to model shortest paths in networks such as the Internet [EGHN13]. The appealing feature of random shortest path metrics is their simplicity, which enables us to use them for the analysis of heuristics.

### Known and Related Results

There has been significant study of random shortest path metrics or first-passage percolation. The expected length of an edge is known to be $\ln n/n$ [DP93, Jan99]. Asymptotically the same bound holds also for the longest edge almost surely [HZ85, Jan99]. These results hold not only for the exponential distribution, but for every distribution with the cumulative distribution function $F$ satisfying $F(x) = x + o(x)$ for small values of $x$ [Jan99]. (See also Section 6.6.) This model has been used to analyze algorithms for computing shortest paths [HZ85, PSSZ13, FG85]. Kulkarni and Adlakha have developed algorithmic methods to compute distribution and moments of several optimization problems [Kul88, KA85, Kul86]. Beyond shortest path algorithms, random shortest path metrics have been applied only rarely to analyze algorithms. Dyer and Frieze [DF90], answering a question raised by Karp and Steele [KS85, Section 3.4], analyzed the patching heuristic for the asymmetric TSP (ATSP) in this model. They showed that it comes within a factor of $1 + o(1)$ of the optimal solution with high probability. Hassin and Zemel [HZ85] applied their findings to the 1-center problem.

From a more structural point of view, first-passage percolation has been analyzed in the area of complex networks, where the hop-count (the number of edges on a shortest path) and the length of shortest path trees have been analyzed [HHM06]. These properties have also been studied on random graphs with random edge weights in various settings [BHH10, HHM01, BHH11, KK12, BHH12]. Addario-Berry, Broutin, and Lugosi [ABL10] have shown that the number of edges in the longest of the shortest paths is $O(\log n)$ with high probability, and hence the shortest path trees have depth $O(\log n)$.

### Our Results

As far as we are aware, simple heuristics such as greedy heuristics have not been studied in this model yet. Understanding the performance of such algorithms is particularly important as they are easy to implement and used in many applications.

We provide a probabilistic analysis of simple heuristics for optimization under random shortest path metrics. First, we provide structural properties of random shortest path metrics (Section 6.3). Our most important structural contribution is proving the existence of a good clustering (Lemma 6.3.9). Then we use these structural insights to analyze simple algorithms for minimum weight matching and the TSP to obtain better expected approximation ratios compared to the worst-case bounds. In particular, we show that the greedy algorithm for minimum-weight perfect matching (Theorem 6.4.2), the nearest-neighbor heuristic for the TSP (Theorem 6.4.4), and every insertion heuristic for the TSP (Theorem 6.4.5) achieve constant expected approximation ratios. We also analyze the 2-opt heuristic for the TSP and show that the expected number of 2-exchanges required before the termination of the algorithm is bounded by $O(n^8 \log^3 n)$ (Theorem 6.4.6). Investigating further the structural properties of random shortest path metrics, we then consider the $k$-median problem (Section 6.5), and show that the most trivial procedure of choosing $k$ arbitrary vertices as $k$-median yields a $1 + o(1)$ approximation in expectation, provided $k = O(n^{1-\varepsilon})$ for some $\varepsilon > 0$ (Theorem 6.5.5).

## 6.2 Preliminaries

### 6.2.1 Model and Notation

In this chapter we consider undirected complete graphs $G = (V, E)$ without self-loops. First, we draw *edge weights* $w(e)$ independently at random according to the exponential distribution[1] with parameter 1.

Second, let the *distances* $d : V \times V \to [0, \infty)$ be given as follows: the distance $d(u, v)$ between $u$ and $v$ is the minimum total weight of a path connecting $u$ and $v$. In particular, we have $d(v, v) = 0$ for all $v \in V$, $d(u, v) = d(v, u)$ because $G$ is undirected, and the triangle inequality: $d(u, v) \leq d(u, x) + d(x, v)$ for all $u, x, v \in V$. We call the complete graph with distances $d$ obtained from random weights $w$ a *random shortest path metric.*

We use the following notation: Let $\Delta_{\max} = \max_{u,v} d(u, v)$ denote the *diameter* of the random shortest path metric. Let $B_\Delta(v) = \{u \in V \mid d(u, v) \leq \Delta\}$ be the ball of radius $\Delta$ around $v$, i.e., the set of all nodes whose distance to $v$ is at most $\Delta$.

We denote the minimal $\Delta$ such that there are at least $k$ nodes within a distance of $\Delta$ of $v$ by $\tau_k(v)$. Formally, we define $\tau_k(v) = \min\{\Delta \mid |B_\Delta(v)| \geq k\}$.

By $\mathrm{Exp}(\lambda)$, we denote the exponential distribution with parameter $\lambda$. If a random variable $X$ is distributed according to a probability distribution $P$, we write $X \sim P$. In particular, $X \sim \sum_{i=1}^{m} \mathrm{Exp}(\lambda_i)$ means that $X$ is the sum of $m$ independent exponentially distributed random variables with parameters $\lambda_1, \ldots, \lambda_m$. Let $F, G$ be two cumulative distribution functions. We say $F$ *stochastically dominates* $G$ if for all $x$, $F(x) \geq G(x)$. In general the definition of stochastically dominance assumes that at least for one $x$ the value is strictly greater. As we will only use inequalities with equality, we can ignore this.

For $n \in \mathbb{N}$ let $H_n = \sum_{i=1}^{n} 1/i$ be the $n$th harmonic number.

[1] Exponential distributions are technically the easiest to handle because they are memoryless. We will discuss other distributions in Section 6.6.

### 6.2.2 Facts about Exponential Distributions

We will extensively need some facts about the exponential distribution which we gather in this section.

**Proposition 6.1.** *The random variable $Y \sim \min\{X_1, \dots, X_n\}$ where $X_i$ are independent exponential distributed random variables with parameter $\lambda_i$ has the same distribution as* $\operatorname{Exp}(\lambda_1 + \cdots + \lambda_n)$.

*Proof.* Follows from a simple argument and is widely known. Let $X_i$ be an exponential variable distributed with parameter $\lambda_i$.

$$\begin{aligned}\mathbb{P}[Y \geq \alpha] &= \mathbb{P}[X_1 \geq \alpha, \dots, X_n \geq \alpha]\\ &= \prod_{i=1}^{n} \mathbb{P}[X_i \geq \alpha]\\ &= \prod_{i=1}^{n} \exp(-\alpha\lambda_i)\\ &= \exp(-\alpha \sum_{i=1}^{n} \lambda_i).\end{aligned}$$

□

**Proposition 6.2** ([Ros10] p. 308)**.** *Let $X \sim \sum_{i=1}^{n} \operatorname{Exp}(\lambda)$ then the probability density function of $X$ is given by*

$$f(x) = \lambda \exp(-\lambda x) \frac{(\lambda x)^{n-1}}{(n-1)!}.$$

**Proposition 6.3** ([Ros10] p. 308ff)**.** *Let $X \sim \sum_{i=1}^{n} \operatorname{Exp}(\lambda_i)$. Then the probability density function of $X$ is given by*

$$f(x) = \sum_{i=1}^{n} \frac{\lambda_i \exp(-\lambda_i x)}{\prod_{\substack{j=1\\ j\neq i}}^{n} \left(1 - \frac{\lambda_i}{\lambda_j}\right)}.$$

Hence the density for $X \sim \sum_{i=1}^{n} \operatorname{Exp}(i)$ is given by

$$\begin{aligned}&\sum_{i=1}^{n} \frac{i \exp(-ix)}{\prod_{\substack{j=1\\ j\neq i}}^{n} \left(1 - \frac{i}{j}\right)}\\ &= \sum_{i=1}^{n} \left( \prod_{\substack{j=1\\ i\neq j}}^{n} \frac{j}{j-i} \right) i \exp(-ix).\end{aligned}$$

**Lemma 6.2.1.** *Let $X \sim \sum_{i=1}^{n} \operatorname{Exp}(ci)$. The random variable $X$ has the same distribution as $\max\{Y_1, \dots, Y_n\}$ where $Y_i \sim \operatorname{Exp}(c)$. Then $\mathbb{P}(X \leq \alpha) = (1 - e^{-c\alpha})^n$.*

*Proof.* By Proposition 6.3 we can write the cumulative distribution function of $X$ as

$$\begin{aligned}\mathbb{P}[X \leq x] &= \int_0^x \sum_{i=1}^{n} \left( \prod_{j\in[n]\setminus\{i\}} \frac{j}{j-i} \right) \cdot ci \cdot \exp(-ciy)\,\mathrm{d}y \\ &= \int_0^x \sum_{i=1}^{n} \frac{n!}{i} \cdot \left( \prod_{j\in[n]\setminus\{i\}} \frac{1}{j-i} \right) \cdot ci \cdot \exp(-ciy)\,\mathrm{d}y.\end{aligned}$$

Notice that we multiply $\frac{1}{j-i} \cdots 1 \cdot (-1) \cdots \frac{1}{1-i}$ and hence we can simplify this to

$$\begin{aligned}&\int_0^x \sum_{i=1}^{n} \frac{n!}{i} \cdot \frac{(-1)^{i-1}}{(i-1)!(n-i)!} \cdot ci \cdot \exp(-ciy)\,\mathrm{d}y \\ &= \int_0^x \sum_{i=1}^{n} \binom{n}{i} (-1)^{i-1} \cdot ci \cdot \exp(-ciy)\,\mathrm{d}y.\end{aligned}$$

Integrating over $y$ yields

$$\sum_{i=1}^{n} \left( \binom{n}{i} \frac{-\left(\exp(-ciy)\right)}{ci} \cdot (-1)^{i-1} \cdot ci \right)\Bigg|_0^x$$

which we rewrite to

$$\sum_{i=1}^{n} \left( \binom{n}{i} \left(-\exp(-cy)\right)^{i} \right)\Bigg|_0^x$$

and finally with the binomial theorem we get

$$\left(1-\exp(-cy)\right)^{n}\big|_0^x .$$

Evaluating this at 0 and $x$ we get $(1-\exp(-cx))^n$.

By the following argument, we can get the cumulative distribution function of the maximum of $n$ random variables distributed according to $\mathrm{Exp}(c)$.

$$\begin{aligned}\mathbb{P}(\max\{Y_1,\ldots,Y_n\} \leq x) \leq &= \mathbb{P}(Y_1 \leq x) \cdots \mathbb{P}(Y_n \leq x) \\ &= (1-e^{-cx}) \cdots (1-e^{-cx}) \\ &= (1-e^{-cx})^n.\end{aligned}$$

This shows that these two distributions are equal. □

**Lemma 6.2.2.** *The density $f$ of $\sum_{i=k}^{n} \mathrm{Exp}(i)$ is given by*

$$f(x) = k \cdot \binom{n}{k} \cdot \exp(-kx) \cdot \left(1-\exp(-x)\right)^{n-k}.$$

*Proof.* By the previous argument the density is the same as the $k$th smallest element of a set of $n$ independent, exponentially distributed random variables with parameter 1. We know from [Ros10, Example 2.37] that the density function of the $i$th minimum for exponentially distributed random variables with parameter 1 is given by

$$i\binom{n}{i}\exp(-x)(1-e^{-x})^{i-1}(1-(1-e^{-x}))^{n-i}.$$

Now as $i = n-k+1$ we get

$$\begin{aligned}&(n-k+1)\binom{n}{n-k+1}\exp(-x)(1-e^{-x})^{n-k}\exp(-(k-1)x)\\&=(n-k+1)\frac{n!}{(n-k+1)!(k-1)!}\exp(-kx)(1-\exp(-x))^{n-k}\\&=k\binom{n}{k}\exp(-kx)(1-\exp(-x))^{n-k}.\end{aligned}$$

□

## 6.3 Structural Properties of Shortest Path Metrics

### 6.3.1 Random Process

To understand random shortest path metrics, it is convenient to fix a starting vertex $v$ and see how the lengths from $v$ to the other vertices develop. In this way, we analyze the distribution of $\tau_k(v)$. Remember that $\tau_k(v)$ is the $\Delta$ we need to have at least $k$ vertices in a ball around $v$ with the radius $\Delta$.

The values $\tau_k(v)$ are generated by a simple birth process as we will describe in the next proof. The same process has been analyzed by Davis and Prieditis [DP93], Janson [Jan99], and also in subsequent papers.

**Lemma 6.3.1.** *$\tau_{k+1}(v) - \tau_k(v) \sim \text{Exp}(k \cdot (n-k))$.*

*Proof.* For $k = 1$, we have $\tau_k(v) = 0$ because $d(v, v) = 0$.

For $k \geq 1$, let us look at the set $B_{\tau_k(v)}(v)$. If we look at the closest vertex to any vertex in this set, we can obtain $\tau_{k+1}(v)$. This conditions all edges $(u, x)$ with $u \in B_{\tau_k(v)}(v)$ and $x \notin B_{\tau_k(v)}(v)$ to be of length at least $\tau_k(v) - d(v, u)$. This holds because we know that $d(u, x) + d(u, v) \geq d(v, x) \geq \tau_k(v)$.

The set $B_{\tau_k(v)}(v)$ contains exactly $k$ vertices as the probability that two vertices have the same distance from $v$ is zero. Thus, there are $k \cdot (n-k)$ edges from $B_{\tau_k(v)}$ to the rest of the graph. Consequently, the difference $\tau_{k+1}(v) - \tau_k(v)$ is distributed as the minimum of $k(n-k)$ exponential random variables (with parameter 1). Equivalently, we can state this as distributed the same as $\text{Exp}(k \cdot (n-k))$ by Proposition 6.1. □

**Lemma 6.3.2.**

$$\tau_k(v) \sim \sum_{i=1}^{k-1} \text{Exp}\left(i \cdot (n-i)\right).$$

*Proof.* We obtain that

$$\tau_{k+1} \sim \sum_{i=1}^{k} \tau_{i+1} - \tau_i \sim \sum_{i=1}^{k} \operatorname{Exp}(i \cdot (n-i)).$$

Note that these exponential distributions as well as the random variables $\tau_2 - \tau_1, \ldots, \tau_n - \tau_{n-1}$ are independent. □

Exploiting linearity of expectation and that the expected value of $\operatorname{Exp}(\lambda)$ is $1/\lambda$ we obtain the following lemma.

**Lemma 6.3.3.** *For any $k \in [n]$ and any $v \in V$, we have*

$$\mathbb{E}\left(\tau_k(v)\right) = \frac{1}{n} \cdot \left(H_{k-1} + H_{n-1} - H_{n-k}\right).$$

*Proof.* The proof is by induction on $k$. For $k = 1$, we have $\tau_k(v) = 0$ and $H_{k-1} + H_{n-1} - H_{n-k} = H_0 + H_{n-1} - H_{n-1} = 0$. Now assume that the lemma holds for some $k \geq 1$. By Lemma 6.3.1 we have seen that $\tau_{k+1}(v) - \tau_k(v) \sim \operatorname{Exp}(k(n-k))$. Thus, $\mathbb{E}(\tau_{k+1}(v) - \tau_k(v)) = \frac{1}{k(n-k)}$. Plugging in the induction hypothesis yields

$$\begin{aligned}
\mathbb{E}(\tau_{k+1}(v)) =& \mathbb{E}(\tau_k(v)) + \frac{1}{k \cdot (n-k)} \\
=& \frac{1}{n} \cdot \left( H_{k-1} + H_{n-1} - H_{n-k} + \frac{n-k+k}{k(n-k)} \right) \\
=& \frac{1}{n} \cdot \left( H_{k-1} + H_{n-1} - H_{n-k} + \frac{1}{k} + \frac{1}{n-k} \right) \\
=& \frac{1}{n} \cdot \left( H_k + H_{n-1} - H_{n-(k+1)} \right).
\end{aligned}$$

□

From this result, we can easily deduce two known results: averaging over $k$ yields the following equations for the expected distance of an edge.

$$\begin{aligned}
\sum_{k=0}^{n} \mathbb{E}\left(\tau_k(v)\right) &= \frac{1}{n^2} \sum_{k=0}^{n} \left(H_{k-1} + H_{n-1} - H_{n-k}\right) \\
&= \frac{H_{n-1}}{n} + \frac{1}{n^2} \sum_{k=0}^{n} (H_{k-1} - H_{n-k}).
\end{aligned}$$

As the last summand is equal to zero we get that the expected distance of an edge is roughly $\frac{H_{n-1}}{n}$ as in [DP93, Jan99]. The longest distance from $v$ to any other node is $\tau_n(v)$, which is $2H_{n-1}/n \approx 2 \ln n/n$ in expectation [Jan99]. For completeness, let us mention that the diameter $\Delta_{\max}$ is approximately $3 \ln n/n$ [Jan99]. However, this does not follow immediately from Lemma 6.3.3.

### 6.3.2 Distribution of $\tau_k(v)$

Let us now have a closer look at cumulative distribution function of $\tau_k(v)$ for fixed $v \in V$ and $k \in [n]$. To do this, the following lemma is very useful.

In the following, let $T_k$ denote the cumulative distribution function of $\tau_k(v)$ for some fixed vertex $v \in V$, i.e., $T_k(x) = \mathbb{P}(\tau_k(v) \leq x)$.

**Lemma 6.3.4.** *For every $\Delta \geq 0$, $v \in V$, and $k \in [n]$, we have*

$$(1 - \exp(-(n-k)\Delta))^{k-1} \leq T_k(\Delta) \leq (1 - \exp(-n\Delta))^{k-1}.$$

*Proof.* Lemma 6.3.2 states that $\tau_k(v) \sim \sum_{i=1}^{k-1} \mathrm{Exp}(i(n-i))$. Let us look at the distribution for $ci$ for $c \in \{n-k, n\}$. We can use the following inequalities to see which distribution dominates.

$$\sum_{i=1}^{k} \mathrm{Exp}\left(i(n-k)\right) \leq \sum_{i=1}^{k} \mathrm{Exp}\left(i(n-i)\right) \leq \sum_{i=1}^{k} \mathrm{Exp}\left(in\right).$$

Hence, the distribution with $c = n$ is stochastically dominated by the true distribution which in turn is dominated by the distribution obtained for $c = n - k$. We apply Lemma 6.2.1 with $c = n$ and $c = n - k$ to finish the proof. □

We can use this to bound the probability of having at least $k$ vertices in a ball of diameter $\Delta$ around $v$.

**Lemma 6.3.5.** *Fix $\Delta \geq 0$ and a vertex $v \in V$. Then*

$$(1 - \exp(-(n-k)\Delta))^{k-1} \leq \mathbb{P}(|B_\Delta(v)| \geq k) \leq (1 - \exp(-n\Delta))^{k-1}.$$

*Proof.* We have $|B_\Delta(v)| \geq k$ if and only if $\tau_k(v) \leq \Delta$. In words, if we need $\Delta' \leq \Delta$ distance from $v$ to already have $k$ vertices in $B_{\Delta'}(v)$ then $|B_\Delta(v)|$ is greater or equal to $k$. The other direction follows with a similar argument.

Then the lemma follows from Lemma 6.3.4. □

We can improve Lemma 6.3.4 slightly in order to obtain even closer upper and lower bounds. For $n, k \geq 2$, combining Lemmas 6.3.4 and 6.3.6 yields tight upper and lower bounds if we disregard the constants in the exponent, namely $T_k(\Delta) = (1 - \exp(-\Theta(n\Delta)))^{\Theta(k)}$.

**Lemma 6.3.6.** *For all $v \in V$, $k \in [n]$, and $\Delta \geq 0$, we have*

$$T_k(\Delta) \geq (1 - \exp(-(n-1)\Delta/4))^{n-1}$$

*and*

$$T_k(\Delta) \geq (1 - \exp(-(n-1)\Delta/4))^{\frac{4}{3}(k-1)}.$$

*Proof.* As $\tau_k(v)$ is monotonically increasing in $k$, we have $T_k(\Delta) \geq T_{k+1}(\Delta)$ for all $k$. Thus, we have to prove the first claim only for $k = n$. In this case, $\tau_n(v) \sim \sum_{i=1}^{n-1} \mathrm{Exp}(\lambda_i)$, with $\lambda_i = i(n-i) = \lambda_{n-i}$. Setting $m = \lceil n/2 \rceil$ and exploiting the symmetry around $m$ yields

$$\tau_n(v) \leq \sum_{i=1}^{m} \mathrm{Exp}(\lambda_i) + \sum_{i=1}^{m} \mathrm{Exp}(\lambda_i) \sim \tau_m(v) + \tau_m(v).$$

Here, "$\leq$" means stochastic dominance and "$+$" means adding up two independent random variables. Hence,

$$T_n(\Delta) = \mathbb{P}(\tau_n(v) \leq \Delta) \geq \mathbb{P}(\tau_m(v) + \tau_m(v) \leq \Delta) \geq \mathbb{P}(\tau_m(v) \leq \Delta/2)^2.$$

By Lemma 6.3.4, and using $m \leq (n+1)/2$, this is bounded by

$$T_n(\Delta) \geq (1 - \exp(-(n-m)\Delta/2))^{2(m-1)} \geq (1 - \exp(-(n-1)\Delta/4))^{n-1}.$$

For the second inequality, we use the first inequality of Lemma 6.3.6 for $k - 1 \geq \frac{3}{4}(n-1)$ and 6.3.4 for $k - 1 < \frac{3}{4}(n-1)$ as then $n - k \geq (n-1)/4$. □

### 6.3.3 Tail Bounds for $|B_\Delta(v)|$ and $\Delta_{\max}$

Our first tail bound for $|B_\Delta(v)|$, which is the number of vertices within distance $\Delta$ of a given vertex $v$, follows directly from Lemma 6.3.4. From this lemma we derive the following corollary, which is a crucial ingredient for the existence of good clusterings and, thus, for the analysis of heuristics in the remainder of this paper.

**Corollary 6.3.7.** *Let $n \geq 5$ and fix $\Delta \geq 0$ and a vertex $v \in V$. Then we have*

$$\mathbb{P}\left(|B_\Delta(v)| < \min\left\{\exp\left(\Delta n/5\right), \frac{n+1}{2}\right\}\right) \leq \exp\left(-\Delta n/5\right).$$

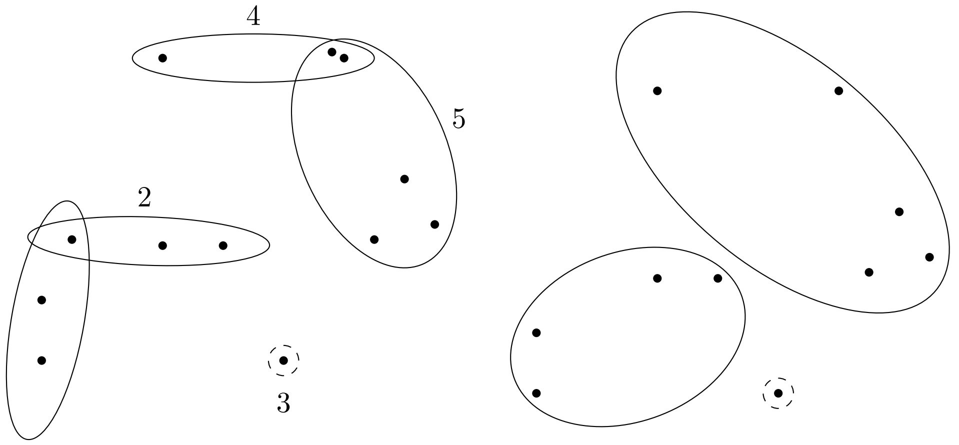
*Proof.* Lemma 6.3.5 yields

$$\begin{aligned}&\mathbb{P}\left(|B_\Delta(v)| < \min\left\{\exp\left(\Delta\frac{n-1}{4}\right), \frac{n+1}{2}\right\}\right)\\ &\leq 1 - \left(1 - \exp\left(-\Delta\left(n - \frac{n+1}{2}\right)\right)\right)^{\exp(\Delta\frac{n-1}{4})}.\end{aligned}$$

In the last inequality, we used the fact that the minimum of two values is always smaller or equal to one of the values. This can then be upper bounded by

$$\begin{aligned}&1 - \left(1 - \exp\left(-\Delta\frac{n-1}{2}\right)\right)^{\exp(\Delta\frac{n-1}{4})}\\ &\leq \exp\left(-\Delta\frac{n-1}{2}\right)\exp\left(\Delta\frac{n-1}{4}\right)\\ &\leq \exp\left(-\Delta\frac{n-1}{4}\right),\end{aligned}$$

where the last two inequalities follows from $(1-x)^y \geq 1 - xy$ for $y \geq 1$, $x \geq 0$. Using $(n-1)/4 \geq n/5$ for $n \geq 5$ completes the proof. □



(a) Clustering before the matching

(b) Complete Clustering

Figure 6.1: Clustering Example

Janson [Jan99] derived the following tail bound for the diameter $\Delta_{\max}$. A qualitatively similar bound can be derived from Hassin and Zemel [HZ85]'s analysis. However, Janson's bound is stronger with respect to the constants in the exponent.

**Lemma 6.3.8** ([Jan99], p. 352)**.** *For any fixed* $c > 3$*, we have* $\mathbb{P}(\Delta_{\max} > c\ln(n)/n) \leq O(n^{3-c}\log^2 n)$*.*

### 6.3.4 Balls and Clusters

In this section, we show our main structural contribution, which is a global property of random shortest path metrics. We show that such instances can be divided into a small number of clusters of any given diameter.

From now on, let $s_\Delta = \min\{\exp(\Delta n/5), (n+1)/2\}$, as in Corollary 6.3.7. If $|B_\Delta(v)|$, the number of vertices within distance $\Delta$ of $v$, is at least $s_\Delta$, then we call the vertex $v$ a *dense* $\Delta$*-center*, and we call the set $B_\Delta(v)$ of vertices within distance $\Delta$ of $v$ (including $v$ itself) the $\Delta$*-ball of* $v$. Otherwise, if $|B_\Delta(v)| < s_\Delta$, and $v$ is not part of any $\Delta$-ball, we call the vertex $v$ a *sparse* $\Delta$*-center*. Any two vertices in the same $\Delta$-ball have a distance of at most $2\Delta$ because of the triangle inequality.

If $\Delta$ is clear from the context, then we also speak about centers and balls without parameter. We can bound the expected number of sparse $\Delta$-centers to be at most $O(n/s_\Delta)$ as we can bound the expected value by $n \cdot \exp(-\Delta n/5)$ using Corollary 6.3.7.

We want to partition the graph into a small number of clusters, each of diameter at most $6\Delta$. For this purpose, we put each sparse $\Delta$-center in its own cluster (of size 1). Then the diameter of each such cluster is 0, which is trivially upper-bounded by $6\Delta$, and the number of these clusters is expected to be at most $O(n/s_\Delta)$.

We are left with the dense $\Delta$-centers, which we cluster using the following algorithm: Consider an auxiliary graph whose vertices are all dense $\Delta$-centers. We draw an edge between two dense $\Delta$-centers $u$ and $v$ if $B_\Delta(u) \cap B_\Delta(v) \neq \emptyset$. Now consider any maximal Independent Set of this auxiliary graph (for instance, a greedy independent set), and let $t$ be the number of its vertices. Then we form initial clusters $C'_1, \ldots, C'_t$, each containing one of the $\Delta$-balls corresponding to the vertices in the Independent Set. By the independence, all these $t$ $\Delta$-balls are disjoint, which implies $t \leq n/s_\Delta$. The ball of every remaining center $v$ has at least one vertex in one of the $C'_i$. We add all remaining vertices of $B_\Delta(v)$ to such a $C'_i$ to form the final clusters $C_1, \ldots, C_t$. By construction, the diameter of each $C_i$ is at most $6\Delta$: Consider any two vertices $u, v \in C_i$. The distance of $u$ towards its closest neighbor in the initial ball $C'_i$ is at most $2\Delta$. The same holds for $v$. Finally, the diameter of the initial ball $C'_i$ is also at most $2\Delta$.

We shown an example in Figure 6.1a. Here the ellipses show the dense $\Delta$-centers where the dashed ellipse is a sparse $\Delta$-center. We named the $\Delta$ centers from 1 to 5. It is easy to see that the graph has an edge between the clusters denoted by 1 and 2 and an edge between 4 and 5. We chose the Independent Set corresponding to 1, 3 and 5 and get the resulting clustering as in Figure 6.1b.

With this partitioning, we have obtained the following structure: We have an expected number of $O(n/s_\Delta)$ clusters of size 1 and diameter 0, and a number of $O(n/s_\Delta)$ clusters of size at least $s_\Delta$ and diameter at most $6\Delta$ as as shown earlier $t < n/s_\Delta$. Thus, we have $O(n/s_\Delta) = O(1 + n/\exp(\Delta n/5))$ clusters in total. We summarize these findings in the following lemma. This lemma is the crucial ingredient for bounding the expected approximation ratios of the greedy, nearest-neighbor, and insertion heuristics.

**Lemma 6.3.9.** *Consider a random shortest path metric and let $\Delta \geq 0$. If we partition the instance into clusters, each of diameter at most $6\Delta$, then the expected number of clusters is $O(1 + n/\exp(\Delta n/5))$.*

## 6.4 Analysis of Heuristics

### 6.4.1 Greedy Heuristic for Minimum-Length Perfect Matching

Finding minimum-length perfect matchings in metric instances is the first problem that we consider. This problem has been widely considered in the past and had applications in, e.g., optimizing the speed of mechanical plotters [RT81, SPR80]. The worst-case running-time of $O(n^3)$ for finding an optimal matching is prohibitive if the number $n$ of points is large. Thus, simple heuristics are often used, with the greedy heuristic being probably the simplest one: at every step, choose an edge of minimum length incident to the unmatched vertices and add it to the partial matching. Let GREEDY denote the cost of the matching output by this greedy matching heuristic, and let MM denote the optimum value of the minimum-length perfect matching. The worst-case approximation ratio for greedy matching on metric instances is $\Theta(n^{\log_2(3/2)})$ [RT81], where $\log_2(3/2) \approx 0.58$. In the case of Euclidean instances, the greedy algorithm has an approximation ratio of $O(1)$ with high probability on random instances [ADS88].

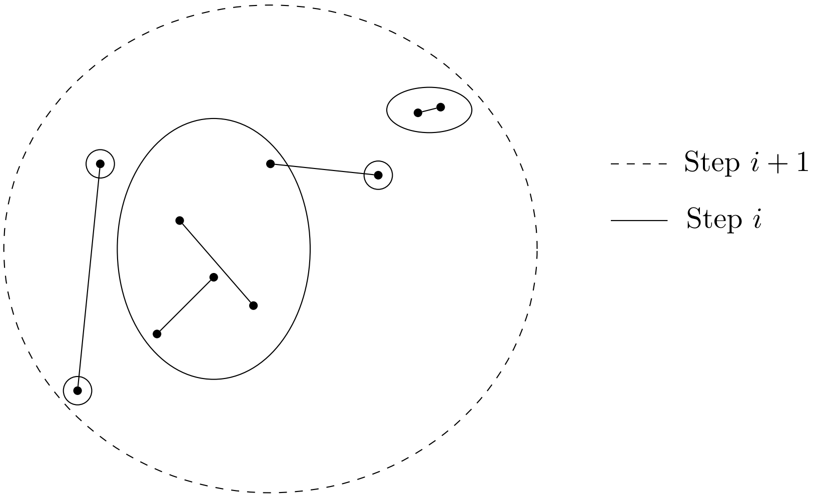


Figure 6.2: Greedy Matching Algorithm Example

For independent random edge weights (without the triangle inequality), the expected weight of the matching computed by the greedy algorithm is $\Theta(\log n)$ [DFP93] whereas the optimal matching has a weight of $\Theta(1)$ with high probability, which gives an $O(\log n)$ approximation ratio.

We show that greedy matching finds a matching of constant expected length on random shortest path metrics.

**Theorem 6.4.1.** $\mathbb{E}(\mathsf{GREEDY}) = O(1)$ *for random shortest path metric with the edge weights drawn from the exponential distribution.*

*Proof.* Let $\Delta_i = \frac{i}{n}$. We divide the run of GREEDY in phases as follows: we say that GREEDY is in phase $i$ if edges $\{u, v\}$ are inserted such that $d(u, v) \in (6\Delta_{i-1}, 6\Delta_i]$. Lemma 6.3.8 allows to show that the expected sum of all edges longer than $\Delta_{\Omega(\log n)}$ is $o(1)$, so we can ignore them.

GREEDY goes through phases $i$ with increasing $i$ (phases can be empty). We now estimate the contribution of phase $i$ to the matching computed by GREEDY. Using Lemma 6.3.9, after phase $i-1$ we can find a clustering into clusters of diameter at most $6\Delta_{i-1}$ using an expected number of $O(1 + n/e^{(i-1)/5})$ clusters. Each such cluster can have at most one unmatched vertex. Thus, we have to add at most $O(1 + n/e^{(i-1)/5})$ edges in phase $i$. Each such edge connects vertices at a distance of at most $6\Delta_i$. Hence, the contribution of phase $i$ is $O(\frac{i}{n} \cdot (1 + n/e^{(i-1)/5}))$ in expectation. Summing over all phases yields the desired bound. Notice that we have at most $O(\log n)$ phases as the longest edge is almost surely of length at most $O(\log n)$.

$$\mathbb{E}(\mathsf{GREEDY}) = o(1) + \sum_{i=1}^{O(\log n)} O\left(\frac{i}{e^{(i-1)/5}} + \frac{i}{n}\right) = O(1).$$

□

We gave an illustration of the phases in Figure 6.2.

Careful analysis allows us to bound the expected approximation ratio.

**Theorem 6.4.2.** *The greedy algorithm for minimum-length perfect matching has constant approximation ratio on random shortest path metrics, i.e.,* $\mathbb{E}\left(\frac{\mathsf{GREEDY}}{\mathsf{MM}}\right) = O(1)$.

For the proof we will use the following tail bound to estimate the approximation ratio of the greedy heuristic for matching as well as the nearest-neighbor and insertion heuristics for the TSP.

**Lemma 6.4.3.** *Let* $\alpha \in [0,1]$. *Let* $S_m$ *be the sum of the lightest* $m$ *edge weights, where* $m \geq \alpha n$. *Then, for all* $c \in [0,1)$, *we have*

$$\mathbb{P}(S_m \leq c) \leq \left(\frac{e^2 c}{2\alpha^2}\right)^{\alpha n}.$$

*Furthermore,* $S_{n/2} \leq \mathsf{MM} \leq \mathsf{TSP}$, *where* $\mathsf{TSP}$ *and* $\mathsf{MM}$ *denote the length of the shortest TSP tour and the minimum-weight perfect matching, respectively, in the corresponding shortest path metric.*

Notice that this lemma proves a fact about the weight of the edges and not the shortest path distances. This will suffice for bounding the expected ratio.

*Proof.* Let $X \sim \sum_{i=1}^{m} \operatorname{Exp}(1)$, and let $Y$ be the sum of $m$ independent random variables drawn uniformly from $[0,1]$. The random variable $Y$ stochastically dominates $X$. Let us study $Y$ further. This distribution is also known as the Irwin-Hall Distribution or Uniform Sum Distribution and we know that the cumulative distribution function is

$$\mathbb{P}(Y \leq c) = \frac{1}{m!}\sum_{k=0}^{\lfloor c \rfloor}(-1)^k \binom{m}{k}(c-k)^m$$

as, for example, given in [Usp37, p. 277f].

As $c \in [0,1)$ we get that $\mathbb{P}(Y \leq c) = c^m/m!$.

The probability that $S_m \leq c$ is at most the probability that there exists a subset of the edges of cardinality $m$ whose total weight is at most $c$. By a union bound and using $\binom{a}{b} \leq (ae/b)^b$, $\binom{n}{2} \leq n^2/2$, and $a! > (a/e)^a$, we obtain

$$\begin{aligned}\mathbb{P}(S_m \leq c) &\leq \binom{\binom{n}{2}}{m} \cdot \frac{c^m}{m!} \leq \left(\frac{en^2}{2m}\right)^m c^m \left(\frac{e}{m}\right)^m \\ &= \left(\frac{n^2 e^2 c}{2m^2}\right)^m \leq \left(\frac{e^2 c}{2\alpha^2}\right)^m.\end{aligned}$$

Here this bound only makes sense if $c < \frac{2\alpha^2}{e^2}$. As we will later fix $c$ to be a small enough constant, this does not matter.

We can replace $m$ by its lower bound $\alpha n$ in the exponent [AMR11, Fact 2.1] to obtain the first claim.

It remains to prove $\mathsf{TSP} \geq \mathsf{MM} \geq S_{n/2}$. The first inequality is trivial. For the second inequality, consider a minimum-weight perfect matching in a random shortest path metric. We replace every edge by the corresponding paths. If we disregard multiple edges, then we are still left with at least $n/2$ edges whose length is not shortened by taking shortest paths. The sum of the weights of these $n/2$ edges is at most $\mathsf{MM}$ and at least $S_{n/2}$. □

*Proof of Theorem 6.4.2.* The worst-case approximation ratio of $\mathsf{GREEDY}$ for minimum-weight perfect matching is $n^{\log_2(3/2)}$ [RT81]. Let $c > 0$ be a sufficiently small constant. Then the approximation ratio of $\mathsf{GREEDY}$ on random shortest path instances is

$$\mathbb{E}\left(\frac{\mathsf{GREEDY}}{\mathsf{MM}}\right) \leq \mathbb{E}\left(\frac{\mathsf{GREEDY}}{c}\right) + \mathbb{P}(\mathsf{MM} < c) \cdot n^{\log_2(3/2)}.$$

By Theorem 6.4.1, the first term is $O(1)$. Since $c$ is sufficiently small, we know by Lemma 6.4.3 that

$$\mathbb{P}(\mathsf{MM} < c) \leq \left(2e^2 c\right)^{\frac{n}{2}}.$$

This shows that the second term is $o(1)$. □

### 6.4.2 Nearest-Neighbor Algorithm for the TSP

A greedy analogue for the traveling salesman problem (TSP) is the *nearest neighbor* heuristic:

1. Start with some starting vertex $v_0$ as the current vertex $v$.
2. At every iteration, choose the nearest yet unvisited neighbor $u$ of the current vertex $v$ (called the successor of $v$) as the next vertex in the tour, and move to the next iteration with the new vertex $u$ as the current vertex $v$.
3. Go back to the first vertex $v_0$ if all vertices are visited.

Let $\mathsf{NN}$ denote both the nearest-neighbor heuristic itself and the cost of the tour computed by it. Let $\mathsf{TSP}$ denote the cost of an optimal tour. The nearest-neighbor heuristic $\mathsf{NN}$ achieves a worst-case ratio of $O(\log n)$ for metric instances and also an average-case ratio (for independent, non-metric edge lengths) of $O(\log n)$ [ACG+99]. We show that $\mathsf{NN}$ achieves a constant approximation ratio on random shortest path instances.

**Theorem 6.4.4.** *For random shortest path instances we have* $\mathbb{E}(\mathsf{NN}) = O(1)$ *and* $\mathbb{E}\left(\frac{\mathsf{NN}}{\mathsf{TSP}}\right) = O(1)$.

*Proof.* The proof is similar to the proof of Theorem 6.4.2. Let $\Delta_i = i/n$ for $i \in \mathbb{N}$. Let $Q = O(\log n/n)$ be sufficiently large.

Consider the clusters obtained with parameter $\Delta_i$ as in the discussion preceding Lemma 6.3.9. These clusters have diameters of at most $6\Delta_i$. We refer to these clusters as the *$i$-clusters*. Let $v$ be any vertex. We call $v$ *bad at $i$*, if $v$ is in some $i$-cluster and $\mathsf{NN}$ chooses a vertex at a distance of more than $6\Delta_i$ from $v$ for leaving $v$. Hence, if $v$ is bad at $i$, then the next vertex lies outside of the cluster to which $v$ belongs. Note that $v$ is not bad at $i$ if the outgoing edge at $v$ leads to a neighbor outside of the cluster of $v$ but at a distance of at most $6\Delta_i$ from $v$.

In the following, let the cost of a vertex $v$ be the distance from $v$ to its successor $u$. The length of the tour produced by $\mathsf{NN}$ is equal to the sum of costs over all vertices.

**Claim 6.1.** *The expected number of vertices with costs in the range* $(6\Delta_i, 6\Delta_{i+1}]$ *is at most* $O(1 + n/\exp(i/5))$.

*Proof of Claim 6.1.* Suppose that the cost of the neighbor chosen by $\mathsf{NN}$ for a vertex $v$ is in the interval $(6\Delta_i, 6\Delta_{i+1}]$ and hence $v$ is bad at $i$. This happens only if all other vertices of the $i$-cluster containing $v$ have already been visited. Otherwise, there would be another vertex $u$ in the same $i$-cluster with a distance of at most $6\Delta_i$ to $v$. By Lemma 6.3.9, the number of $i$-clusters is at most $O(1 + n/\exp(i/5))$. □

If $\Delta_{\max} \le Q$, then it suffices to consider $i$ for $i \le O(\log n)$. If $\Delta_{\max} > Q$, then we bound the value of the tour produced by $\mathsf{NN}$ by $n\Delta_{\max}$. This failure event, however, contributes only $o(1)$ to the expected value by Lemma 6.3.8. For the case $\Delta_{\max} \le Q$, the contribution to the expected length of the $\mathsf{NN}$ tour is bounded from above by

$$\sum_{i=0}^{O(\log n)} 6\Delta_{i+1} \cdot O\left(1 + \frac{n}{\exp(i/5)}\right) = \sum_{i=0}^{O(\log n)} O\left(\frac{i+1}{n} + \frac{i+1}{\exp(i/5)}\right) = O(1).$$

Using the fact that the worst-case approximation ratio of $\mathsf{NN}$ is $O(\log n)$, the proof of the constant expected approximation ratio is similar to the proof of Theorem 6.4.2.

$$\mathbb{E}\left(\frac{\mathsf{NN}}{\mathsf{TSP}}\right) \le \mathbb{E}\left(\frac{\mathsf{NN}}{c}\right) + \mathbb{P}(\mathsf{TSP} \le c) \cdot O(\log n).$$

Using again Lemma 6.4.3 shows that $\mathbb{P}(\mathsf{TSP} \le c)$ is in $o(1)$. □

### 6.4.3 Insertion Heuristics for the TSP

An insertion heuristic for the TSP is an algorithm that starts with an initial tour on a few vertices and extends this tour iteratively by adding the remaining vertices. In every iteration, a vertex is chosen according to some rule, and this vertex is inserted at the place in the current tour where it increases the total tour length the least. The approximation ratio achieved depends on the rule used for selecting the next node to insert. Certain insertion heuristics such as nearest neighbor insertion (which is different from the nearest neighbor algorithm from the previous section) achieve constant approximation ratio [RSI77]. The random insertion algorithm, where the next vertex is chosen uniformly at random from the remaining vertices, has a worst-case approximation ratio of $\Omega(\log\log n/\log\log\log n)$, and there are insertion heuristics with a worst-case approximation ratio of $\Omega(\log n/\log\log n)$ [Aza94].

A rule $R$ that specifies an insertion heuristic can be viewed as follows. Depending on the distances $d$, it:

1. Choose a set $R_V$ of vertices for computing an initial tour.
2. Given any tour of vertices $V' \supseteq R_V$, describes how to choose the next vertex.

Let $\mathsf{INSERT}_R$ denote the length of the tour produced with rule $R$.

For random shortest path metrics, we show that any insertion heuristic produces a tour whose length is expected to be within a constant factor of the optimal tour. This result holds irrespective of which insertion strategy we actually use.

**Theorem 6.4.5.** *For every rule $R$, we have $\mathbb{E}(\mathsf{INSERT}_R) = O(1)$ and $\mathbb{E}(\frac{\mathsf{INSERT}_R}{\mathsf{TSP}}) = O(1)$.*

*Proof.* Let $\Delta_i = i/n$ for $i \in \mathbb{N}$ and $Q = O(\log n/n)$ be sufficiently large. Assume that $\Delta_{\max} \leq Q$. If $\Delta_{\max} > Q$, then we bound the length of the tour produced by $n \cdot \Delta_{\max}$. This contributes only $o(1)$ to the expected value of length of the tour produced by Lemma 6.3.8.

Suppose we have a partial tour $T$ and $v$ is the vertex that we have to insert next. If $T$ has a vertex $u$ such that $v$ and $u$ are in a common $i$-cluster, then the triangle inequality implies that the costs of inserting $v$ into $T$ is at most $12\Delta_i$ because the diameters of $i$-clusters are at most $6\Delta_i$ [RSI77, Lemma 2]. For each $i$, only the insertion of the first vertex of each $i$-cluster can possibly cost more than $12\Delta_i$. Thus, the number of vertices whose insertion would incur costs in the range $(12\Delta_i, 12\Delta_{i+1}]$ is at most $O\big(1 + \frac{n}{\exp(i/5)}\big)$ in expectation. Note that we only have to consider $i$ with $i \leq O(\log n)$ since $\Delta_{\max} \leq Q$. The expected cost of the initial tour is at most $\mathsf{TSP} = O(1)$ [Fri04]. Summing up the expected costs for all $i$ plus the costs of the initial tour, we obtain that the expected costs of the tour obtained by an insertion heuristic is bounded from above by

$$\mathbb{E}(\mathsf{INSERT}_R) = O(1) + \sum_{i=0}^{O(\log n)} \Delta_i \cdot O\left(1 + \frac{n}{\exp(i/5)}\right) = O(1).$$

Note that the above argument is independent of the rule $R$ used.

The proof for the approximation ratio is similar to the proof of Theorem 6.4.2 and uses the worst-case ratio of $O(\log n)$ for insertion heuristics for any rule $R$ [RSI77, Theorem 3]. □

### 6.4.4 Running-Time of 2-Opt for the TSP

The 2-opt heuristic for the TSP starts with an initial tour and successively improves the tour by so-called 2-exchanges until no further refinement is possible. In a 2-exchange, a pair of edges $e_{12} = \{v_1, v_2\}$ and $e_{34} = \{v_3, v_4\}$, where $v_1, v_2, v_3, v_4$ appear in this order in the Hamiltonian tour, are replaced by a pair of edges $e_{13} = \{v_1, v_3\}$ and $e_{24} = \{v_2, v_4\}$ to get a shorter tour. The 2-opt heuristic is easy to implement and widely used but can have exponential running time. In practice, it usually converges quite quickly to close-to-optimal solutions [JM02]. To explain its performance in practice, probabilistic analyses of its running-time on geometric instances [ERV14, MV13, Ker89] and its approximation performance on geometric instances [ERV14] and with independent, non-metric edge lengths [EM09] have been conducted. We prove that for random shortest path metrics, the expected number of iterations that 2-opt needs is bounded by a polynomial.

**Theorem 6.4.6.** *The expected number of iterations that 2-opt needs to find a local optimum is bounded by $O(n^8 \log^3 n)$.*

*Proof.* The proof is similar to the analysis of 2-opt by Englert, Röglin, and Vöcking [ERV14]. Consider a 2-exchange where edges $e_1 = \{v_1, v_2\}$ and $e_2 = \{v_3, v_{4\}}$ are replaced by edges $f_1 = \{v_1, v_3\}$ and $f_2 = \{v_2, v_4\}$ as described above. The improvement obtained from this exchange is given by $\delta = \delta(v_1, v_2, v_3, v_4) = d(v_1, v_2) + d(v_3, v_4) - d(v_1, v_3) - d(v_2, v_4)$.

We estimate the probability $\mathbb{P}(\delta \in (0, \varepsilon])$ of the event that the improvement is at most $\varepsilon$ for some $\varepsilon > 0$. The distances $d(v_i, v_j)$ correspond to shortest paths with respect to the exponentially distributed edge weights $w$. Assume for the moment that we know these paths. We will now rewrite the equation for $\delta$. Let the shortest path between $v_1, v_2$ be given by the edges $E_{v_1,v_2}$. Then

$$\delta = \sum_{e \in E_{v_1,v_2}} w(e) + \sum_{e \in E_{v_3,v_4}} w(e) - \sum_{e \in E_{v_1,v_3}} w(e) - \sum_{e \in E_{v_2,v_4}} w(e).$$

We can simplify this to

$$\delta = \sum_{e \in E} \alpha_e w(e) \tag{6.1}$$

for some coefficients $\alpha_e \in \{-2, -1, 0, 1, 2\}$. If the exchange considered is indeed a 2-exchange, then $\delta > 0$. Thus, in this case, there exists at least on edge $e = \{u, u'\}$ with $\alpha_e > 0$. Let $I \subseteq \{e_{12}, e_{34}, e_{13}, e_{24}\}$ be the set of edges of the 2-exchange such that the corresponding paths use $e$.

For all combinations of $I$ and $e = \{u, u'\}$, let $\delta_{ij}^{I,e}$ be the following quantity:

- If $e_{ij} \notin I$, then $\delta_{ij}^{I,e}$ is the length of a shortest path from $v_i$ to $v_j$ without using $e$.
- If $e_{ij} \in I$, then $\delta_{ij}^{I,e}$ is the minimum of
  - the length of a shortest path from $v_i$ to $u$ without $e$ plus the length of a shortest path from $u'$ to $v_j$ without $e$ and
  - the length of a shortest path from $v_i$ to $u'$ without $e$ plus the length of a shortest path from $u$ to $v_j$ without $e$.

Let $\delta^{e,I} = \delta^{e,I}_{12} + \delta^{e,I}_{34} - \delta^{e,I}_{13} - \delta^{e,I}_{24}$.

**Claim 6.2.** *For every outcome of the random edge weights, there exists an edge $e$ and a set $I$ such that $\delta = \delta^{e,I} + \alpha w(e)$, where $\alpha \in \{-2,-1,1,2\}$ is determined by $e$ and $I$.*

*Proof of Claim 6.2.* Fix the edge weights arbitrarily and consider any four shortest paths all different. Then there exists some edge $e$ with non-zero $\alpha_e$ in Equation (6.1). We choose this $e$, an appropriate set $I$, such that all edges except $e$ are in $I$, and we choose $\alpha = \alpha_e$. Then the claim follows from the definition of $\delta^{e,I}$. □

Claim 6.2 yields that $\delta \in (0,\varepsilon]$ implies that there are an $e$ and an $I$ with $\delta^{e,I}+\alpha w(e) \in (0,\varepsilon]$.

**Claim 6.3.** *Let $e$ and $I$ be arbitrary with $\alpha = \alpha_e > 0$. Then $\mathbb{P}(\delta^{e,I}+\alpha w(e) \in (0,\varepsilon]) \leq \varepsilon$.*

*Proof of Claim 6.3.* We fix the edge weights of all edges except for $e$. This determines $\delta^{e,I}$. Thus, $\delta^{e,I} + \alpha w(e) \in (0,\varepsilon]$ if and only of $w(e)$ assumes a value in a now fixed interval of size $\varepsilon/\alpha \leq \varepsilon$. Since the density of the exponential distribution is bounded from above by 1, the claim follows. □

The number of possible choices for $e$ and $I$ is $O(n^2)$. Thus, $\mathbb{P}(\delta \in (0,\varepsilon]) = O(n^2\varepsilon)$.

Let $\delta_{\min} > 0$ be the minimum improvement made by any 2-exchange. Since there are at most $n^4$ different 2-exchanges, we have $\mathbb{P}(\delta_{\min} \leq \varepsilon) = O(n^6\varepsilon)$.

The initial tour has a length of at most $n\Delta_{\max}$. Let $T$ be the number of iterations that 2-opt takes. Then $T \leq n\Delta_{\max}/\delta_{\min}$. Now, $T > x$ implies $\Delta_{\max}/\delta_{\min} > x/n$. The event $\Delta_{\max}/\delta_{\min} > x/n$ is contained in the union of the events $\Delta_{\max} > \log x \ln n/n$, and $\delta_{\min} < \ln n \cdot \log x/x$. To further explain this, when one of these conditions is fulfilled we know that

$$\frac{\Delta_{\max}}{\delta_{\min}} > \frac{\log x \ln n}{n} \cdot \frac{x}{\ln n \log x} = \frac{x}{n}.$$

The first happens with a probability of at most $n^{-\Omega(\log(x))}$ by Lemma 6.3.8. The second happens with a probability of at most $O(n^6 \log(x)/x)$. Thus, we obtain

$$\mathbb{P}(T > x) \leq n^{-\Omega(\log(x))} + O(n^6 \ln n \cdot \log(x)/x).$$

Since the number of iterations is at most $n!$, we obtain an upper bound of

$$\mathbb{E}(T) \leq \sum_{x=1}^{n!} \left(n^{-\Omega(\log(x))} + O(n^6 \ln n \log(x)/x)\right).$$

The sum over $n^{-\Omega(\log(x))}$ is negligible. The sum over $O(n^6 \ln n \log(x)/x)$ contributes $O(n^6 \ln n \log(n!)^2) \subseteq O(n^8 \log^3 n)$. □

## 6.5 $k$-Median

In the (metric) $k$-median problem, we are given a finite metric space $(V, d)$ and should pick $k$ points $U \subseteq V$ such that $\sum_{v \in V} \min_{u \in U} d(v, u)$ is minimized. We call the set $U$ a $k$-median. Regarding worst-case analysis, the best known approximation algorithm for this problem achieves an approximation ratio of $3 + \varepsilon$ [AGK+04].

In this section, we consider the $k$-median problem in the setting of random shortest path metrics. In particular we examine the approximation ratio of the algorithm TRIVIAL, which picks $k$ points independently of the metric space, e.g., $U = \{1, \ldots, k\}$ or $k$ random points in $V$. We show that TRIVIAL yields a $(1 + o(1))$-approximation for $k = O(n^{1-\varepsilon})$. This can be seen as an algorithmic result since it improves upon the worst-case approximation ratio, but it is essentially a structural result on random shortest path metrics. It means that any set of $k$ points is, with high probability, a very good $k$-median, which gives some knowledge about the topology of random shortest path metrics. For larger, but not too large $k$, i.e., $k \leq (1 - \varepsilon)n$, TRIVIAL still yields an $O(1)$-approximation.

The main insight comes from generalizing the growth process described in Section 6.3.2. Fixing $U = \{v_1, \ldots, v_k\} \subseteq V$ we sort the vertices $V \setminus U$ by their distance to $U$ in ascending order, calling the resulting order $v_{k+1}, \ldots, v_n$. Now we consider $\delta_i = d(v_{i+1}, U) - d(v_i, U)$ for $k \leq i < n$. These random variables are generated by a simple growth process analogous to the one described in Section 6.3.2. This shows that the $\delta_i$ are independent and $\delta_i \sim \mathrm{Exp}(i \cdot (n - i))$. We know that $\mathrm{Exp}(\lambda)/a \sim \mathrm{Exp}(\lambda a)$ by the simple inequality: $\mathbb{P}[\frac{1}{a} X \leq x] = \mathbb{P}[X \leq ax] \leq 1 - \mathrm{Exp}(\lambda a)$. Hence

$$\mathsf{cost}(U) = \sum_{i=k}^{n-1} (n - i) \cdot \delta_i \sim \sum_{i=k}^{n-1} (n - i) \cdot \mathrm{Exp}(i \cdot (n - i)) \sim \sum_{i=k}^{n-1} \mathrm{Exp}(i).$$

From this, we can read off the expected cost of $U$ immediately, and thus the expected cost of TRIVIAL.

**Lemma 6.5.1.** *Fix $U \subseteq V$ of size $k$. We have*

$$\mathbb{E}(\mathsf{TRIVIAL}) = \mathbb{E}(\mathsf{cost}(U)) = H_{n-1} - H_{k-1} = \ln(n/k) + \Theta(1).$$

*Proof.* We have $\mathbb{E}(\mathsf{cost}(U)) = \sum_{i=k}^{n-1} \frac{1}{i} = H_{n-1} - H_{k-1}$. Using $H_n = \ln(n) + \Theta(1)$ yields the last equality. □

By closely examining the random variable $\sum_{i=k}^{n-1} \mathrm{Exp}(i)$, we can show good tail bounds for the probability that the cost of $U$ is lower than expected. Together with the union bound this yields tail bounds for the optimal $k$-median MEDIAN, which implies the following theorem. In this theorem, the approximation ratio becomes $1 + O(\frac{\ln \ln(n)}{\ln(n)})$ for $k = O(n^{1-\varepsilon})$.

We need the following lemmas to prove Theorem 6.5.5.

**Lemma 6.5.2.** *Let $c > 0$ be sufficiently large, and let $k \leq c'n$ for $c' = c'(c) > 0$ be sufficiently small. Then*

$$\mathbb{P}\left(\mathsf{MEDIAN} < \ln\left(\frac{n}{k}\right) - \ln\ln\left(\frac{n}{k}\right) - \ln c\right) = n^{-\Omega(c)}.$$

*Proof.* Fix $U \subseteq V$ of size $k$ and consider $\mathsf{cost}(U) \sim \sum_{i=k}^{n-1} \mathrm{Exp}(i)$. In the following we set $m := n-1$ to shorten notation. Let $f(x)$ be the probability density function of $\mathsf{MEDIAN}$ as in Lemma 6.2.2. We now want to bound $f(x)$ from above at $x = \ln(\frac{m}{ak})$ for a sufficiently large $a$ with $1 \leq a \leq m/k$ (such an $a$ exists since $k$ is small enough). Plugging in this particular $x$ and using $\binom{m}{k} \leq m^k e^k / k^k$ yields the following inequations.

$$\begin{aligned}
f\left(\ln\left(\frac{m}{ak}\right)\right) =& k \cdot \binom{m}{k} \cdot \left(\frac{ak}{m}\right)^k \cdot \left(1 - \frac{ak}{m}\right)^{m-k} \\
=& k \cdot \binom{m}{k} \cdot \frac{a^k k^k}{m^k} - \frac{a^k k^k}{m^k} \cdot \left(\frac{ak}{m}\right)^{m-k} \\
=& k \cdot \binom{m}{k} \cdot \frac{a^k k^k m^{m-k}}{m^m} - \frac{a^k k^k}{m^m} \cdot a^{m-k} k^{m-k} \\
=& k \cdot \binom{m}{k} \cdot \frac{a^k k^k (m - ak)^{m-k}}{m^m} \\
\leq& k e^k \frac{m^k}{k^k} \cdot \frac{a^k k^k (m - ak)^{m-k}}{m^m} \\
\leq& k(ea)^k \left(\frac{m - ak}{m}\right)^{m-k} \\
\leq& k(ea)^k \left(1 - \frac{ak}{m}\right)^{m-k}.
\end{aligned}$$

Using $1 + x \leq e^x$ and $m - k = \Omega(m)$, so that $(m-k)/m = \Omega(1)$, yields

$$f(x) \leq k(ea)^k \exp(-\Omega(ak)).$$

Since $a$ is sufficiently large, the first two factors are lower order terms that we can hide by the $\Omega$. Thus, we can simplify this further to

$$f(x) \leq \exp(-\Omega(ak)).$$

Rearranging this using $a = \frac{m}{k} e^{-x}$ yields

$$f(x) \leq \exp(-\Omega(m \exp(-x)), \tag{6.2}$$

which holds for any $x \in [0, \ln(\frac{m}{\alpha k})]$ for any sufficiently large $\alpha \geq 1$. Now we can bound the probability that $\mathsf{cost}(U) < \ln(\frac{m}{\alpha k})$. We can rewrite our probability

$$\int_0^{\ln(\frac{m}{\alpha k})} f(x)\,\mathrm{d}x$$

to

$$= \int_0^{\ln(\frac{m}{\alpha k})} f\left(\ln\left(\frac{m}{\alpha k}\right) - x\right) \mathrm{d}x$$

as this is just the mirrored function. We can then use Equation (6.2) and get the upper bound

$$\begin{aligned}
&\int_0^{\ln(\frac{m}{\alpha k})} \exp(-\Omega(\alpha k \exp(x))) \,\mathrm{d}x \\
&\leq \int_0^{\infty} \exp(-\Omega(\alpha k(1+x))) \,\mathrm{d}x \\
&\leq \exp(-\Omega(\alpha k))
\end{aligned}$$

since $e^x \leq 1 + x$ and $\int_0^\infty \exp(-\Omega(\alpha k x)) \,\mathrm{d}x = O(1/(\alpha k)) \leq 1$ as $\alpha$ is sufficiently large.

In order for $\mathsf{MEDIAN}$ to be less than $\ln(\frac{m}{\alpha k})$, one of the subsets $U \subseteq V$ of size $k$ has to have cost less than $\ln(\frac{m}{\alpha k})$. We bound the probability of the latter using the union bound and get

$$\begin{aligned}
\mathbb{P}\left(\mathsf{MEDIAN} < \ln\left(\frac{m}{\alpha k}\right)\right) &= \mathbb{P}\left(\exists U \subseteq V, |U| = k\colon\ \mathsf{cost}(U) < \ln\left(\frac{m}{\alpha k}\right)\right) \\
&\leq \binom{n}{k} \cdot \mathbb{P}\left(\mathsf{cost}(U) < \ln\left(\frac{m}{\alpha k}\right)\right) \\
&\leq \binom{n}{k} \cdot \exp(-\Omega(\alpha k)).
\end{aligned}$$

By setting $\alpha = c\ln(\frac{n}{k})$ for sufficiently large $c \geq 1$, we fulfill all conditions on $\alpha$. If we plug this into $\ln\left(\frac{m}{\alpha k}\right)$ we get

$$\ln\left(\frac{m}{c \ln \frac{n}{k} k}\right) = \ln\left(\frac{m}{k}\right) - \ln\ln\left(\frac{n}{k}\right) - \ln c.$$

Plugging this into our probability estimation we get

$$\begin{aligned}
&\mathbb{P}\left(\mathsf{MEDIAN} < \ln\left(\frac{n}{k}\right) - \ln\ln\left(\frac{n}{k}\right) - \ln c\right) \\
&\leq \binom{n}{k} \cdot e^{-\left(\Omega(kc \ln \frac{n}{k})\right)} \\
&\leq \left(\frac{en}{k}\right)^k \cdot \left(\frac{n}{k}\right)^{-\Omega(ck)}.
\end{aligned}$$

Since $k$ is sufficiently smaller than $n$, we have $\frac{en}{k} \leq (\frac{n}{k})^2$. Thus, for sufficiently large $c$, the right hand side simplifies to $(\frac{n}{k})^{-\Omega(ck)}$. Since $k$ is at least 1 and sufficiently smaller than $n$, we have $(\frac{n}{k})^k \geq n$. Thus, the probability is bounded by $n^{-\Omega(c)}$, which finishes the proof. □

To bound the expected value of the quotient $\mathsf{TRIVIAL} / \mathsf{MEDIAN}$, we further need to bound the probabilities that $\mathsf{TRIVIAL}$ is much too large or $\mathsf{MEDIAN}$ is much too small. This is achieved by the following two lemmas.

**Lemma 6.5.3.** *Let $k \leq (1-\varepsilon)n$ for some constant $\varepsilon > 0$. Then, for any $c > 0$, we have*

$$\mathbb{P}(\mathsf{MEDIAN} < c) = O(c)^{\Omega(n)}.$$

*Proof.* Since $n-k$ vertices have to be connected to the $k$-median, the cost of the $k$-median is the sum of $n-k$ shortest path lengths. Thus, the cost of the minimal $k$-median is at least the sum of the smallest $n-k$ edge weights $w(e)$. We use Lemma 6.4.3 with $\alpha = \varepsilon$. □

**Lemma 6.5.4.** *For any $c \geq 3$, we have $\mathbb{P}(\mathsf{TRIVIAL} > n^c) \leq \exp(-n^{c/3})$.*

*Proof.* We can bound very roughly $\mathsf{TRIVIAL} \leq n \max_e\{w(e)\}$. As $\max_e\{w(e)\}$ is the maximum of $\binom{n}{2}$ independent exponentially distributed random variables, we have

$$\begin{aligned}\mathbb{P}(\mathsf{TRIVIAL} \leq n^c) &\geq (1-\exp(-n^{c-1}))^{\binom{n}{2}} \geq 1 - \binom{n}{2} \cdot \exp(-n^{c-1}) \\ &\geq 1-\exp(-n^{c-2}) \geq 1-\exp(-n^{c/3}).\end{aligned}$$

□

These two lemmas give us the final theorem.

**Theorem 6.5.5.** *Let $k \leq (1-\varepsilon)n$ for some constant $\varepsilon > 0$. Then*

$$\mathbb{E}\left(\frac{\mathsf{TRIVIAL}}{\mathsf{MEDIAN}}\right) = O(1).$$

*If we have $k \leq \kappa n$ for some sufficiently small constant $\kappa \in (0,1)$, then*

$$\mathbb{E}\left(\frac{\mathsf{TRIVIAL}}{\mathsf{MEDIAN}}\right) = 1 + O\left(\frac{\ln\ln(n/k)}{\ln(n/k)}\right). \tag{6.3}$$

We know that $\mathsf{TRIVIAL}$ and $\mathsf{MEDIAN}$ are not independent and hence we do not know if $\frac{\mathbb{E}[\mathsf{TRIVIAL}]}{\mathbb{E}[\mathsf{MEDIAN}]} = \mathbb{E}[\frac{\mathsf{TRIVIAL}}{\mathsf{MEDIAN}}]$. Hence, to get a good bound on the expected difference we need a slightly more complicated proof.

*Proof.* Let $T = \mathsf{TRIVIAL}$ and $M = \mathsf{MEDIAN}$ for short. We have for any $m \geq 0$

$$\mathbb{E}\left(\frac{T}{M}\right) \leq \mathbb{E}\left(\frac{T}{m}\right) + \mathbb{P}(M < m) \cdot \mathbb{E}\left(\frac{T}{M} \mid M < m\right). \tag{6.4}$$

**Case 1, $k \leq c'n$, $c'$ sufficiently small:** Using Lemma 6.5.2, we can pick $c > 0$ such that

$$\mathbb{P}\left[M < \ln\left(\frac{n}{k}\right) - \ln\ln\left(\frac{n}{k}\right) - \ln c\right] \leq n^{-7}. \tag{6.5}$$

Set $m = \ln\left(\frac{n}{k}\right) - \ln\ln\left(\frac{n}{k}\right) - \ln c$. Then, by Lemma 6.5.1

$$\mathbb{E}\left(\frac{T}{m}\right) \leq \frac{\ln(n/k) + O(1)}{m} \leq O(1)$$

where the last part holds for large enough $n$ as the limit of $\ln(n/k)/m$ is 1.

We show that the second summand of inequality (6.4) is $O(1/n)$ in the current situation, which shows the claim. As $\frac{T}{M}$ is non-negative, we can bound the expectation in the following way.

$$\begin{aligned}\mathbb{P}(C < m) \cdot \mathbb{E}\left(\frac{T}{M} \mid M < m\right) &= \mathbb{P}(M < m) \cdot \int_0^\infty \mathbb{P}\left(\frac{T}{M} \geq x \mid M < m\right) \mathrm{d}x \\ &\leq \mathbb{P}(M < m) \cdot \left(n^6 + \int_{n^6}^\infty \mathbb{P}\left(\frac{T}{M} \geq x \mid M < m\right) \mathrm{d}x\right).\end{aligned}$$

As we chosen $c$ such that Equation (6.5) holds, we can bound the equation by

$$\begin{aligned}&n^{-1} + \int_{n^6}^\infty \mathbb{P}\left(\frac{T}{M} \geq x \text{ and } M < m\right) \mathrm{d}x \\ &\leq n^{-1} + \int_{n^6}^\infty \mathbb{P}\left(\frac{T}{M} \geq x\right) \mathrm{d}x \\ &\leq n^{-1} + \int_{n^6}^\infty 2\max\left\{\mathbb{P}\left(T \geq \sqrt{x}\right), \mathbb{P}\left(M \leq \frac{1}{\sqrt{x}}\right)\right\} \mathrm{d}x\end{aligned}$$

since $T/M \geq x$ implies $T \geq \sqrt{x}$ or $M \leq 1/\sqrt{x}$. Using Lemmas 6.5.3 and 6.5.4, this yields

$$\begin{aligned}&\mathbb{P}(M < m) \cdot \mathbb{E}\left(\frac{T}{C} \mid M < m\right) \\ &\leq n^{-1} + \int_{n^6}^\infty 2\max\left\{\exp(-x^{1/6}), O\left(\frac{1}{\sqrt{x}}\right)^{\Omega(n)}\right\} \mathrm{d}x.\end{aligned}$$

This is now equal to $O(1/n)$.

**Case 2, $c'n < k \leq (1-\varepsilon)n$:** We repeat the proof above, now choosing $m$ to be a sufficiently small constant. Then $\mathbb{P}(M < m) = O(m)^{\Omega(n)} \leq O(n^{-7})$ by Lemma 6.5.3, and we have

$$\mathbb{E}\left(\frac{T}{m}\right) = \frac{\ln(n/k) + O(1)}{m} = O(1),$$

since $k > c'n$. Together with the first case, this shows the first claim.

□

## 6.6 Concluding Remarks

### 6.6.1 General Probability Distributions

Using a coupling argument, Janson [Jan99, Section 3] proved that the results about the length of a fixed edge and the longest edge carry over if the exponential distribution is replaced by a probability distribution with the following property: The probability that an edge weight is smaller than $x$ is $x + o(x)$. This property is satisfied, e.g., by the exponential distribution with parameter 1 and by the uniform distribution on the interval $[0, 1]$. The intuition is that, because the longest edge has a length of $O(\log n/n) = o(1)$, only the behavior of the distribution in a small, shrinking interval $[0, o(1)]$ is relevant and the $o(x)$ term becomes irrelevant.

For completeness we will sketch a standard coupling argument. We can see the exponential distribution as resulting from the uniform distribution on $[0, 1]$. For this we take the cummulative distribution function $F(u) = 1 - \exp(-\lambda u)$. Then there exists an inverse $F^{-1} : [0, 1] \to [0, \infty)$ such that $F^{-1}(U)$ gives the exponential distribution with parameter $\lambda$.

We believe that also all of our results carry over to such probability distributions by a similar argument. In fact, we started our research using the uniform distribution and only switched to exponential distributions because they are technically easier to handle. However, we decided not to carry out the corresponding proofs because, we feel that they do not add much to our understanding of algorithms on general metric spaces.

### 6.6.2 Open Problems on Metric Graphs

To conclude this chapter, let us list the open problems that we consider most interesting:

1. While the distribution of distances in asymmetric instances does not differ much from the symmetric case, an obstacle in the application of asymmetric random shortest path metrics seems to be the lack of clusters of small diameter (see Section 6.3). Is there an asymmetric counterpart for this?

2. Is it possible to prove an $1 + o(1)$ approximation ratio (like Dyer and Frieze [DF90] for the patching algorithm) for any of the simple heuristics that we analyzed?

3. What is the approximation ratio of 2-opt in random shortest path metrics? In the worst case on metric instances, it is $O(\sqrt{n})$ [CKT99]. For independent, non-metric edge lengths drawn uniformly from the interval $[0, 1]$, the expected approximation ratio is $O(\sqrt{n} \cdot \log^{3/2} n)$ [EM09]. For $d$-dimensional geometric instances, the smoothed approximation ratio is $O(\phi^{1/d})$ [ERV14], where $\phi$ is the perturbation parameter.

   We easily get an approximation ratio of $O(\log n)$ based on the two facts that the length of the optimal tour is $\Theta(1)$ with high probability and that $\Delta_{\max} =$

$O(\log n/n)$ with high probability. Can we prove that the expected ratio of 2-opt is $o(\log n)$?

# Bibliography

[AW09] Scott Aaronson and Avi Wigderson. “Algebrization: A New Barrier in Complexity Theory”. In: *TOCT* 1.1 (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”. In: *Ann. Pure Appl. Logic* 73.3 (1995), pp. 235–276.

[ABL10] Louigi Addario-Berry, Nicolas Broutin, and Gábor Lugosi. “The Longest Minimum-Weight Path in a Complete Graph”. In: *Combinatorics, Probability & Computing* 19.1 (2010), pp. 1–19.

[AB03] Manindra Agrawal and Somenath Biswas. “Primality and identity testing via Chinese remaindering”. In: *J. ACM* 50.4 (2003), pp. 429–443.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. “PRIMES is in P”. In: *Ann. of Math. (2)* 160.2 (2004), pp. 781–793.

[AV08] Manindra Agrawal and V. Vinay. “Arithmetic Circuits: A Chasm at Depth Four”. In: *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*. IEEE Computer Society, 2008, pp. 67–75.

[Arc96] Dan Archdeacon. “Topological Graph Theory – A Survey”. In: *CONG. NUM* 115 (1996), pp. 115–5.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[AMR11] David Arthur, Bodo Manthey, and Heiko Röglin. “Smoothed Analysis of the k-Means Method”. In: *J. ACM* 58.5 (2011), p. 19.

[AJS09] Vikraman Arvind, Pushkar S. Joglekar, and Srikanth Srinivasan. “Arithmetic Circuits and the Hadamard Product of Polynomials”. In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India.* Ed. by Ravi Kannan and K. Narayan Kumar. Vol. 4. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009, pp. 25–36.

[AMS10] Vikraman Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan. “New Results on Noncommutative and Commutative Polynomial Identity Testing”. In: *Computational Complexity* 19.4 (2010), pp. 521–558.

[AR14] Vikraman Arvind and S. Raja. “The Complexity of Two Register and Skew Arithmetic Computation”. In: *Electronic Colloquium on Computational Complexity (ECCC)* 21 (2014).

[AS10] Vikraman Arvind and Srikanth Srinivasan. “On the hardness of the noncommutative determinant”. In: *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010.* Ed. by Leonard J. Schulman. ACM, 2010, pp. 677–686.

[AGK+04] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. “Local Search Heuristics for k-Median and Facility Location Problems”. In: *SIAM J. Comput.* 33.3 (2004), pp. 544–562.

[Asl96] Helmer Aslaksen. “Quaternionic determinants”. In: *Math. Intelligencer* 18.3 (1996), pp. 57–65.

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.* Springer, 1999.

[ADS88] David Avis, Burgess Davis, and J. Michael Steele. “Probabilistic Analysis of a Greedy Heuristic for Euclidean Matching”. In: *Probability in the Engineering and Informational Sciences* 2 (02 Apr. 1988), pp. 143–156.

[Aza94] Yossi Azar. “Lower Bounds for Insertion Methods for TSP”. In: *Combinatorics, Probability & Computing* 3 (1994), pp. 285–292.

[BGS75] Theodore P. Baker, John Gill, and Robert Solovay. “Relativizations of the P =? NP Question”. In: *SIAM J. Comput.* 4.4 (1975), pp. 431–442.

[Bar89] David A. Barrington. “Bounded-width polynomial-size branching programs recognize exactly those languages in {NC1}”. In: *Journal of Computer and System Sciences* 38.1 (1989), pp. 150–164.

[Bar90] Alexander I. Barvinok. “Computational complexity of immanents and representations of the full linear group”. In: *Functional Analysis and Its Applications* 24.2 (1990), pp. 144–145.

[Bar96] Alexander I. Barvinok. “Two Algorithmic Results for the Traveling Salesman Problem”. In: *Mathematics of Operations Research* 21.1 (1996), pp. 65–84.

[BHK62] Joseph Battle, Frank Harary, and Yukihiro Kodama. “Additivity of the genus of a graph”. In: *Bull. Amer. Math. Soc.* 68.6 (Nov. 1962), pp. 565–568.

[BC92] Michael Ben-Or and Richard Cleve. “Computing Algebraic Formulas Using a Constant Number of Registers”. In: *SIAM J. Comput.* 21.1 (1992), pp. 54–58.

[BHH10] Shankar Bhamidi, Remco van der Hofstad, and Gerard Hooghiemstra. “First Passage Percolation on Random Graphs with Finite Mean Degrees”. In: *Annals of Applied Probability* 20.5 (2010), pp. 1907–1965.

[BHH11] Shankar Bhamidi, Remco van der Hofstad, and Gerard Hooghiemstra. “First Passage Percolation on the Erdős-Rényi Random Graph”. In: *Combinatorics, Probability & Computing* 20.5 (2011), pp. 683–707.

[BHH12] Shankar Bhamidi, Remco van der Hofstad, and Gerard Hooghiemstra. *Universality for first passage percolation on sparse random graphs.* Tech. rep. 1210.6839 [math.PR]. arXiv, 2012.

[Bla10] Nathaniel D. Blair-Stahn. *First passage percolation and competition models.* Tech. rep. 1005.0649v1 [math.PR]. arXiv, 2010.

[Blä13] Markus Bläser. “Noncommutativity Makes Determinants Hard”. In: *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I.* Ed. by Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg. Vol. 7965. Lecture Notes in Computer Science. Springer, 2013, pp. 172–183.

[BEMR13] Karl Bringmann, Christian Engels, Bodo Manthey, and B. V. Raghavendra Rao. “Random Shortest Paths: Non-euclidean Instances for Metric Optimization Problems”. In: *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings.* Ed. by Krishnendu Chatterjee and Jiri Sgall. Vol. 8087. Lecture Notes in Computer Science. Springer, 2013, pp. 219–230.

[BH57] S. R. Broadbent and J.M. Hammersley. “Percolation Processes. I. Crystals and Mazes”. In: *Proceedings of the Cambridge Philosophical Society* 53.3 (1957), pp. 629–641.

[BB03] Jean-Luc Brylinski and Ranee Brylinski. “Complexity and Completeness of Immanants”. In: *CoRR* cs.CC/0301024 (2003).

[BG05] Andrei A. Bulatov and Martin Grohe. “The complexity of partition functions”. In: *Theor. Comput. Sci.* 348.2-3 (2005), pp. 148–186.

[Bür00a] Peter Bürgisser. *Completeness and reduction in algebraic complexity theory.* Vol. 7. Springer, 2000.

[Bür00b] Peter Bürgisser. “The Computational Complexity of Immanants”. In: *SIAM J. Comput.* 30.3 (2000), pp. 1023–1040.

[Bür00c] Peter Bürgisser. “The Computational Complexity to Evaluate Representations of General Linear Groups”. In: *SIAM J. Comput.* 30.3 (2000), pp. 1010–1022.

[CCL13] Jin-Yi Cai, Xi Chen, and Pinyan Lu. “Graph Homomorphisms with Complex Values: A Dichotomy Theorem”. In: *SIAM J. Comput.* 42.3 (2013), pp. 924–1029.

[CKT99] Barun Chandra, Howard J. Karloff, and Craig A. Tovey. “New Results on the Old k-opt Algorithm for the Traveling Salesman Problem”. In: *SIAM J. Comput.* 28.6 (1999), pp. 1998–2029.

[CR00] Chandra Chekuri and Anand Rajaraman. “Conjunctive query containment revisited”. In: *Theor. Comput. Sci.* 239.2 (2000), pp. 211–229.

[CZ06] Jianer Chen and Fenghui Zhang. “On product covering in 3-tier supply chain models: Natural complete problems for $W[3]$ and $W[4]$”. In: *Theor. Comput. Sci.* 363.3 (2006), pp. 278–288.

[CKW11] Xi Chen, Neeraj Kayal, and Avi Wigderson. “Partial Derivatives in Arithmetic Complexity and Beyond”. In: *Foundations and Trends in Theoretical Computer Science* 6.1-2 (2011), pp. 1–138.

[CHSS11] Steve Chien, Prahladh Harsha, Alistair Sinclair, and Srikanth Srinivasan. “Almost settling the hardness of noncommutative determinant”. In: *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011.* Ed. by Lance Fortnow and Salil P. Vadhan. ACM, 2011, pp. 499–508.

[CS07] Steve Chien and Alistair Sinclair. “Algebras with Polynomial Identities and Computing the Determinant”. In: *SIAM J. Comput.* 37.1 (2007), pp. 252–266.

[CSS11] Sam Clearman, Brittany Shelton, and Mark Skandera. “Path tableaux and combinatorial interpretations of immanants for class functions on $S_n$”. In: *The 23rd International Conference on Formal Power Series and Algebraic Combinatorics, FPSAC 2011.* 2011, pp. 233–244.

[CMMSV13] Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. “Paradigms for Parameterized Enumeration”. In: *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings.* Ed. by Krishnendu Chatterjee and Jiri Sgall. Vol. 8087. Lecture Notes in Computer Science. Springer, 2013, pp. 290–301.

[Cur15] Radu Curticapean. “The simple, little and slow things count”. PhD thesis. Saarland University, 2015.

[DJ04] Víctor Dalmau and Peter Jonsson. “The complexity of counting homomorphisms seen from the other side”. In: *Theor. Comput. Sci.* 329.1-3 (2004), pp. 315–323.

[DKV02] Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. “Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics”. In: *The 8th International Conference on Principles and Practice of Constraint Programming - CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings.* Ed. by Pascal Van Hentenryck. Vol. 2470. Lecture Notes in Computer Science. Springer, 2002, pp. 310–326.

[DKLM10] Samir Datta, Raghav Kulkarni, Nutan Limaye, and Meena Mahajan. “Planarity, Determinants, Permanents, and (Unique) Matchings”. In: *TOCT* 1.3 (2010).

[DP93] Robert Davis and Armand Prieditis. “The Expected Length of a Shortest Path”. In: *Inf. Process. Lett.* 46.3 (1993), pp. 135–141.

[Die00] Reinhard Diestel. *Graph Theory.* Springer, 2000.

[DF93] Rod G. Downey and Michael. R. Fellows. “Fixed parameter tractability and completeness III: some structural aspects of the W hierarchy”. In: *Complexity Theory* (1993), pp. 166–191.

[DF95a] Rodney G. Downey and Michael R. Fellows. “Fixed-Parameter Tractability and Completeness I: Basic Results”. In: *SIAM J. Comput.* 24 (4 1995), pp. 873–921.

[DF95b] Rodney G. Downey and Michael R. Fellows. “Fixed-Parameter Tractability and Completeness II: On Completeness for W[1]”. In: *Theor. Comput. Sci.* 141.1&2 (1995), pp. 109–131.

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Vol. 3. Springer Heidelberg, 1999.

[DF13] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.

[DMMRS14] Arnaud Durand, Meena Mahajan, Guillaume Malod, Nicolas de Rugy-Altherre, and Nitin Saurabh. “Homomorphism Polynomials Complete for VP”. In: *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India.* Ed. by Venkatesh Raman and S. P. Suresh. Vol. 29. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014, pp. 493–504.

[DF90] Martin E. Dyer and Alan M. Frieze. “On Patching Algorithms for Random Asymmetric Travelling Salesman Problems”. In: *Math. Program.* 46 (1990), pp. 361–378.

[DG00] Martin E. Dyer and Catherine S. Greenhill. “The complexity of counting graph homomorphisms”. In: *Random Struct. Algorithms* 17.3-4 (2000), pp. 260–289.

[DG04] Martin E. Dyer and Catherine S. Greenhill. “Corrigendum: The complexity of counting graph homomorphisms”. In: *Random Struct. Algorithms* 25.3 (2004), pp. 346–352.

[DFP93] Martin Dyer, Alan Frieze, and Boris Pittel. “The Average Performance of the Greedy Matching Algorithm”. In: *Annals of Applied Probability* 3.2 (1993), pp. 526–552.

[EGHN13] Maren Eckhoff, Jesse Goodman, Remco van der Hofstad, and Francesca R. Nardi. “Short Paths for First Passage Percolation on the Complete Graph”. In: *Journal of Statistical Physics* 151.6 (2013), pp. 1056–1088.

[Eng14] Christian Engels. “Dichotomy Theorems for Homomorphism Polynomials of Graph Classes”. In: *CoRR* abs/1412.0423 (2014).

[Eng15] Christian Engels. “Dichotomy Theorems for Homomorphism Polynomials of Graph Classes”. In: *WALCOM: Algorithms and Computation - 9th International Workshop, WALCOM 2015, Dhaka, Bangladesh, February 26-28, 2015. Proceedings.* Ed. by M. Sohel Rahman and Etsuji Tomita. Vol. 8973. Lecture Notes in Computer Science. Springer, 2015, pp. 282–293.

[EM09] Christian Engels and Bodo Manthey. “Average-case approximation ratio of the 2-opt algorithm for the TSP”. In: *Oper. Res. Lett.* 37.2 (2009), pp. 83–84.

[ER14] Christian Engels and B. V. Raghavendra Rao. “New Algorithms and Hard Instances for Non-Commutative Computation”. In: *CoRR* abs/1409.0742 (2014).

[ERV14] Matthias Englert, Heiko Röglin, and Berthold Vöcking. “Worst Case and Probabilistic Analysis of the 2-Opt Algorithm for the TSP”. In: *Algorithmica* 68.1 (2014), pp. 190–264.

[FMR79] I. S. Filotti, Gary L. Miller, and John H. Reif. “On Determining the Genus of a Graph in O(vˆO(g)) Steps”. In: *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA.* Ed. by Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho. ACM, 1979, pp. 27–37.

[FKL07] Uffe Flarup, Pascal Koiran, and Laurent Lyaudet. “On the Expressive Power of Planar Perfect Matching and Permanents of Bounded Treewidth Matrices”. In: *Algorithms and Computation, 18th International Symposium, ISAAC 2007, Sendai, Japan, December 17-19, 2007, Proceedings.* Ed. by Takeshi Tokuyama. Vol. 4835. Lecture Notes in Computer Science. Springer, 2007, pp. 124–136.

[FL10] Uffe Flarup and Laurent Lyaudet. "On the Expressive Power of Permanents and Perfect Matchings of Matrices of Bounded Pathwidth/Cliquewidth". In: *Theory Comput. Syst.* 46.4 (2010), pp. 761–791.

[FG04] Jörg Flum and Martin Grohe. "The Parameterized Complexity of Counting Problems". In: *SIAM J. Comput.* 33.4 (2004), pp. 892–922.

[FG06] Jörg Flum and Martin Grohe. *Parameterized complexity theory.* Vol. 3. Springer, 2006.

[FGKM15] Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, and Ivan Mihajlin. "Lower Bounds for the Graph Homomorphism Problem". In: *CoRR* abs/1502.05447 (2015).

[FLMS14] Hervé Fournier, Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. "Lower bounds for depth 4 formulas computing iterated matrix multiplication". In: *Proceedings of the 46th ACM Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014.* Ed. by David B. Shmoys. ACM, 2014, pp. 128–135.

[FMM15] Hervé Fournier, Guillaume Malod, and Stefan Mengel. "Monomials in Arithmetic Circuits: Complete Problems in the Counting Hierarchy". In: *Computational Complexity* 24.1 (2015), pp. 1–30.

[Fre90] Eugene C. Freuder. "Complexity of K-Tree Structured Constraint Satisfaction Problems". In: *Proceedings of the 8th National Conference on Artificial Intelligence. Boston, Massachusetts, July 29 - August 3, 1990, 2 Volumes.* Ed. by Howard E. Shrobe, Thomas G. Dietterich, and William R. Swartout. AAAI Press / The MIT Press, 1990, pp. 4–9.

[Fri04] Alan M. Frieze. "On Random Symmetric Travelling Salesman Problems". In: *Math. Oper. Res.* 29.4 (2004), pp. 878–890.

[FG85] Alan M. Frieze and G. R. Grimmett. "The Shortest-Path Problem for Graphs with Random Arc-Lengths". In: *Discrete Applied Mathematics* 10 (1985), pp. 57–77.

[FH91] William Fulton and Joe Harris. *Representation theory.* Vol. 129. Springer, 1991.

[Gat87] Joachim von zur Gathen. "Feasible Arithmetic Computations: Valiant's Hypothesis". In: *J. Symb. Comput.* 4.2 (1987), pp. 137–172.

[Gen14] Craig Gentry. "Noncommutative Determinant is Hard: A Simple Proof Using an Extension of Barrington's Theorem". In: *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014.* IEEE, 2014, pp. 181–187.

[GGR14] Andreas Göbel, Leslie Ann Goldberg, and David Richerby. "The complexity of counting homomorphisms to cactus graphs modulo 2". In: *TOCT* 6.4 (2014), p. 17.

[GGR15] Andreas Göbel, Leslie Ann Goldberg, and David Richerby. "Counting Homomorphisms to Square-Free Graphs, Modulo 2". In: *CoRR* abs/1501.07539 (2015).

[GGJT10] Leslie Ann Goldberg, Martin Grohe, Mark Jerrum, and Marc Thurley. "A Complexity Dichotomy for Partition Functions with Mixed Signs". In: *SIAM J. Comput.* 39.7 (2010), pp. 3336–3402.

[GJ14] Leslie Ann Goldberg and Mark Jerrum. "The Complexity of Approximately Counting Tree Homomorphisms". In: *TOCT* 6.2 (2014), p. 8.

[Gro07] Martin Grohe. "The complexity of homomorphism and constraint satisfaction problems seen from the other side". In: *J. ACM* 54.1 (2007).

[GT11] Martin Grohe and Marc Thurley. "Counting Homomorphisms and Partition Functions". In: *CoRR* abs/1104.0185 (2011).

[GM84] Robert Grone and Russell Merris. "An algorithm for the second immanant". In: *Math. Comp.* 43.168 (1984), pp. 589–591.

[GKKS14] Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. "Approaching the Chasm at Depth Four". In: *J. ACM* 61.6 (2014), p. 33.

[Har85] Wolfgang Hartmann. "On the complexity of immanants". In: *Linear and Multilinear Algebra* 18.2 (1985), pp. 127–140.

[HZ85] Refael Hassin and Eitan Zemel. "On Shortest Paths in Graphs with Random Weights". In: *Mathematics of Operations Research* 10.4 (1985), pp. 557–564.

[Hei12] Uffe Heide-Jørgensen. "On the determinantal complexity of the 2-Hook-Immanant". PhD thesis. Aarhus University, 2012.

[HN90] Pavol Hell and Jaroslav Nešetřil. "On the complexity of $H$-coloring". In: *J. Comb. Theory, Ser. B* 48.1 (1990), pp. 92–110.

[HN04] Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms.* Vol. 28. Oxford University Press Oxford, 2004.

[Hep94] Charles Thomas Hepler. "On the complexity of computing characters of finite groups". Master Thesis. Dept. of Computer Science, University of Calgary, Canada, 1994.

[HHM01] Remco van der Hofstad, Gerard Hooghiemstra, and Piet van Mieghem. "First Passage Percolation on the Random Graph". In: *Probability in the Engineering and Informational Sciences* 15.2 (2001), pp. 225–237.

[HHM06] Remco van der Hofstad, Gerard Hooghiemstra, and Piet Van Mieghem. "Size and Weight of Shortest Path Trees with Exponential Link Weights". In: *Combinatorics, Probability & Computing* 15.6 (2006), pp. 903–926.

[HWY10] Pavel Hrubes, Avi Wigderson, and Amir Yehudayoff. “Non-commutative circuits and the sum-of-squares problem”. In: *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010.* Ed. by Leonard J. Schulman. ACM, 2010, pp. 667–676.

[JVW90] F. Jaeger, D. L. Vertigan, and Dominic J. A. Welsh. “On the computational complexity of the Jones and Tutte polynomials”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 108 (01 July 1990), pp. 35–53.

[Jan99] Svante Janson. “One, Two And Three Times Log N/N For Paths In A Complete Graph With Random Weights”. In: *Combinatorics, Probability & Computing* 8.4 (1999), pp. 347–361.

[JS82] Mark Jerrum and Marc Snir. “Some Exact Complexity Results for Straight-Line Computations over Semirings”. In: *J. ACM* 29.3 (1982), pp. 874–897.

[JM02] David S. Johnson and Lyle A. McGeoch. “Experimental Analysis of Heuristics for the STSP”. In: *The Traveling Salesman Problem and its Variations.* Ed. by Gregory Gutin and Abraham P. Punnen. Kluwer, 2002. Chap. 9.

[KI04] Valentine Kabanets and Russell Impagliazzo. “Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds”. In: *Computational Complexity* 13.1-2 (2004), pp. 1–46.

[Kar77] Richard M. Karp. “Probabilistic Analysis of Partitioning Algorithms for the Traveling-Salesman Problem in the Plane”. In: *Mathematics of Operations Research* 2.3 (1977), pp. 209–224.

[KS85] Richard M. Karp and J. Michael Steele. “Probabilistic Analysis of Heuristics”. In: *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Ed. by Eugene L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys. Wiley, 1985, pp. 181–205.

[Kay12] Neeraj Kayal. “An exponential lower bound for the sum of powers of bounded degree polynomials”. In: *Electronic Colloquium on Computational Complexity (ECCC)* 19 (2012).

[KSS14] Neeraj Kayal, Chandan Saha, and Ramprasad Saptharishi. “A superpolynomial lower bound for regular arithmetic formulas”. In: *Proceedings of the 46th ACM Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014.* Ed. by David B. Shmoys. ACM, 2014, pp. 146–153.

[KS14] Neeraj Kayal and Ramprasad Saptharishi. “A selection of lower bounds for arithmetic circuits”. In: *Perspectives in Computational Complexity.* Springer, 2014, pp. 77–115.

[Ker89] Walter Kern. “A probabilistic analysis of the switching algorithm for the euclidean TSP”. In: *Math. Program.* 44.1-3 (1989), pp. 213–219.

[Koi05] Pascal Koiran. “Valiant’s model and the cost of computing integers”. In: *Computational Complexity* 13.3-4 (2005), pp. 131–146.

[Koi12] Pascal Koiran. “Arithmetic circuits: The chasm at depth four gets wider”. In: *Theor. Comput. Sci.* 448 (2012), pp. 56–65.

[KK12] István Kolossváry and Júlia Komjáthy. *First Passage Percolation on Inhomogeneous Random Graphs.* Tech. rep. 1201.3137v1 [math.PR]. arXiv, 2012.

[KA85] V. G. Kulkarni and V. G. Adlakha. “Maximum Flow in Planar Networks in Exponentially Distributed Arc Capacities”. In: *Communications in Statistics. Stochastic Models* 1.3 (1985), pp. 263–289.

[Kul86] Vidyadhar G. Kulkarni. “Shortest paths in networks with exponentially distributed arc lengths”. In: *Networks* 16.3 (1986), pp. 255–274.

[Kul88] Vidyadhar G. Kulkarni. “Minimal spanning trees in undirected networks with exponentially distributed arc weights”. In: *Networks* 18.2 (1988), pp. 111–124.

[LMS15] Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. “Lower bounds for non-commutative skew circuits”. In: *Electronic Colloquium on Computational Complexity (ECCC)* 22 (2015).

[LR34] Dudley E. Littlewood and Archibald R. Richardson. “Group Characters and Algebra”. In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 233.721-730 (1934), pp. 99–141.

[Lov67] László Lovász. “Operations with structures”. English. In: *Acta Mathematica Academiae Scientiarum Hungarica* 18.3-4 (1967), pp. 321–328.

[Mah13] Meena Mahajan. “Algebraic Complexity Classes”. In: *CoRR* abs/1307.3863 (2013).

[MR13] Meena Mahajan and B. V. Raghavendra Rao. “Small Space Analogues of Valiant’s Classes and the Limitations of Skew Formulas”. In: *Computational Complexity* 22.1 (2013), pp. 1–38.

[MV99] Meena Mahajan and V. Vinay. “Determinant: Old Algorithms, New Insights”. In: *SIAM J. Discrete Math.* 12.4 (1999), pp. 474–490.

[Mal07] Guillaume Malod. “The Complexity of Polynomials and Their Coefficient Functions”. In: *22nd Annual IEEE Conference on Computational Complexity (CCC 2007), 13-16 June 2007, San Diego, California, USA*. IEEE Computer Society, 2007, pp. 193–204.

[MP08] Guillaume Malod and Natacha Portier. “Characterizing Valiant’s algebraic complexity classes”. In: *J. Complexity* 24.1 (2008), pp. 16–38.

[MV13] Bodo Manthey and Rianne Veenstra. “Smoothed Analysis of the 2-Opt Heuristic for the TSP: Polynomial Bounds for Gaussian Noise”. In: *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings.* Ed. by Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam. Vol. 8283. Lecture Notes in Computer Science. Springer, 2013, pp. 579–589.

[McC03] Catherine McCartin. “Contributions to Parameterized Complexity”. PhD thesis. Victoria University of Wellington, 2003.

[Men13] Stefan Mengel. “Arithmetic Branching Programs with Memory”. In: *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings.* Ed. by Krishnendu Chatterjee and Jiri Sgall. Vol. 8087. Lecture Notes in Computer Science. Springer, 2013, pp. 667–678.

[Mer83] Russell Merris. “Single-hook characters and hamiltonian circuits”. In: *Linear and Multilinear Algebra* 14.1 (1983), pp. 21–35.

[MM13] Stephan Mertens and Cristopher Moore. “The Complexity of the Fermionant and Immanants of Constant Width [Note]”. In: *Theory of Computing* 9 (2013), pp. 273–282.

[Mil87] Gary L. Miller. “An additivity theorem for the genus of a graph”. In: *J. Comb. Theory, Ser. B* 43.1 (1987), pp. 25–47.

[Mit79] Sandra L. Mitchell. “Linear Algorithms to Recognize Outerplanar and Maximal Outerplanar Graphs”. In: *Inf. Process. Lett.* 9.5 (1979), pp. 229–232.

[Moh88] Bojan Mohar. “Nonorientable Genus of Nearly Complete Bipartite Graphs”. In: *Discrete & Computational Geometry* 3 (1988), pp. 137–146.

[Moh99] Bojan Mohar. “A Linear Time Algorithm for Embedding Graphs in an Arbitrary Surface”. In: *SIAM J. Discrete Math.* 12.1 (1999), pp. 6–26.

[MS01] Ketan Mulmuley and Milind A. Sohoni. “Geometric Complexity Theory I: An Approach to the P vs. NP and Related Problems”. In: *SIAM J. Comput.* 31.2 (2001), pp. 496–526.

[Nie02] Rolf Niedermeier. “Invitation to fixed-parameter algorithms”. Habilitation Thesis. 2002.

[Nis91] Noam Nisan. “Lower Bounds for Non-Commutative Computation (Extended Abstract)”. In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, STOC 1991, May 5-8, 1991, New Orleans, Louisiana, USA*. Ed. by Cris Koutsougeras and Jeffrey Scott Vitter. ACM, 1991, pp. 410–418.

[NW97] Noam Nisan and Avi Wigderson. “Lower Bounds on Arithmetic Circuits Via Partial Derivatives”. In: *Computational Complexity* 6.3 (1997), pp. 217–234.

[PY96] Christos H. Papadimitriou and Mihalis Yannakakis. “On Limited Nondeterminism and the Complexity of the V-C Dimension”. In: *J. Comput. Syst. Sci.* 53.2 (1996), pp. 161–170.

[PSSZ13] Yuval Peres, Dmitry Sotnikov, Benny Sudakov, and Uri Zwick. “All-pairs shortest paths in $O(n^2)$ time with high probability”. In: *J. ACM* 60.4 (2013), p. 26.

[Raz09] Ran Raz. “Multi-linear formulas for permanent and determinant are of super-polynomial size”. In: *J. ACM* 56.2 (2009).

[RY09] Ran Raz and Amir Yehudayoff. “Lower Bounds and Separations for Constant Depth Multilinear Circuits”. In: *Computational Complexity* 18.2 (2009), pp. 171–207.

[RY11] Ran Raz and Amir Yehudayoff. “Multilinear formulas, maximal-partition discrepancy and mixed-sources extractors”. In: *J. Comput. Syst. Sci.* 77.1 (2011), pp. 167–190.

[RR97] Alexander A. Razborov and Steven Rudich. “Natural Proofs”. In: *J. Comput. Syst. Sci.* 55.1 (1997), pp. 24–35.

[RT81] Edward M. Reingold and Robert Endre Tarjan. “On a Greedy Heuristic for Complete Matching”. In: *SIAM J. Comput.* 10.4 (1981), pp. 676–681.

[RSI77] Daniel J. Rosenkrantz, Richard Edwin Stearns, and Philip M. Lewis II. “An Analysis of Several Heuristics for the Traveling Salesman Problem”. In: *SIAM J. Comput.* 6.3 (1977), pp. 563–581.

[Ros10] Sheldon M. Ross. *Introduction to Probability Models*. 10th. Academic Press, 2010.

[Rot96] Dan Roth. “On the Hardness of Approximate Reasoning”. In: *Artif. Intell.* 82.1-2 (1996), pp. 273–302.

[Rug12] Nicolas de Rugy-Altherre. “A Dichotomy Theorem for Homomorphism Polynomials”. In: *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*. Ed. by Branislav Rovan, Vladimiro Sassone, and Peter Widmayer. Vol. 7464. Lecture Notes in Computer Science. Springer, 2012, pp. 308–322.

[Sch88] Edward R. Scheinerman. "Random interval graphs". In: *Combinatorica* 8.4 (1988), pp. 357–371.

[SY10] Amir Shpilka and Amir Yehudayoff. "Arithmetic Circuits: A survey of recent results and open questions". In: *Foundations and Trends in Theoretical Computer Science* 5.3-4 (2010), pp. 207–388.

[ST04] Daniel A. Spielman and Shang-Hua Teng. "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time". In: *J. ACM* 51.3 (2004), pp. 385–463.

[SB77] Saul Stahl and Lowell W. Beineke. "Blocks and the nonorientable genus of graphs". In: *Journal of Graph Theory* 1.1 (1977), pp. 75–78.

[Str90] Volker Strassen. "Algebraic complexity theory". In: *Handbook of theoretical computer science, Vol. A*. Elsevier, Amsterdam, 1990, pp. 633–672.

[SPR80] Kenneth J. Supowit, David A. Plaisted, and Edward M. Reingold. "Heuristics for Weighted Perfect Matching". In: *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*. Ed. by Raymond E. Miller, Seymour Ginsburg, Walter A. Burkhard, and Richard J. Lipton. ACM, 1980, pp. 398–419.

[Tav13] Sébastien Tavenas. "Improved Bounds for Reduction to Depth 4 and Depth 3". In: *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*. Ed. by Krishnendu Chatterjee and Jiri Sgall. Vol. 8087. Lecture Notes in Computer Science. Springer, 2013, pp. 813–824.

[Tes13] Rebecca Tessier. "Path Tableaux and the Combinatorics of the Immanant Function". Master Thesis. University of Waterloo, 2013.

[Tho89] Carsten Thomassen. "The Graph Genus Problem is NP-Complete". In: *J. Algorithms* 10.4 (1989), pp. 568–576.

[Tod92] Seinosuke Toda. "Classes of arithmetic circuits capturing the complexity of computing the determinant". In: *IEICE Transactions on Information and Systems* 75.1 (1992), pp. 116–124.

[Usp37] James V. Uspensky. *Introduction to mathematical probability.* McGraw-Hill, 1937.

[Val79a] Leslie G. Valiant. "Completeness Classes in Algebra". In: *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*. Ed. by Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho. ACM, 1979, pp. 249–261.

[Val79b] Leslie G. Valiant. “The Complexity of Computing the Permanent”. In: *Theor. Comput. Sci.* 8 (1979), pp. 189–201.

[Ver04] Anatoly M. Vershik. “Random Metric Spaces and Universality”. In: *Russian Mathematical Surveys* 59.2 (2004), pp. 259–295.

[Wil14] Ryan Williams. “Nonuniform ACC Circuit Lower Bounds”. In: *J. ACM* 61.1 (2014), 2:1–2:32.

[Yuk98] Joseph E. Yukich. *Probability Theory of Classical Euclidean Optimization Problems.* Vol. 1675. Lecture Notes in Mathematics. Springer, 1998.