Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik
Lehrstuhl Logik in der Informatik

Diplomarbeit

Tractability and Intractability of Parameterized Counting Problems

Marc Thurley

Gutachter:
Prof. Dr. Martin Grohe
Prof. Dr. Johannes Köbler

Berlin, im Mai 2006

I want to thank my supervisor, Martin Grohe, for his patience, many illuminative discussions and for encouraging me to develop questions of my own. I would like to thank Dzifa Ametowobla for supporting me throughout this work especially by reading several parts of it and helping me greatly with the design. Furthermore, I am indebted to Bettina Hepp and Kay Thurley for helpful comments on earlier versions of this thesis.

Contents

The interest in solving counting problems has many practical motivations. Notably, many issues that are important in statistical physics and artificial intelligence can be modelled as counting problems. For example, in the part of artificial intelligence that is devoted to approximate reasoning, several problems arise that are equivalent to counting the number of solutions of a propositional formula (cf. [Rot96]).

Most naturally, however, counting problems arise in a purely mathematical context as generalizations of decision problems. The latter ask for the existence of certain combinatorial objects, whereas the former pose the question for the number of these objects. One important example of a counting problem is the problem of computing the permanent, which, in its combinatorial interpretation, asks for the number of perfect matchings in a given bipartite graph.

Already in the 19th century it has been observed that, apparently, there is no way to solve the permanent problem without much computational effort. But it was not until the development of computational complexity theory that the measurement of computational effort was grasped formally. Hence only about 30 years ago the necessary complexity theoretic notions of counting problems could be formed by Valiant, who even found formal evidence for the apparent hardness of the permanent problem.

More formally, Valiant defined the class #P and showed that the permanent problem is complete for this class (cf. [Val79a]). In short, a counting problem is contained in #P if its results can be represented by the number of accepting runs of a polynomial time nondeterministic Turing machine. A class of computationally easy problems within this class is called FP, which denotes all counting problems that can be computed by a polynomial time deterministic Turing machine.

With the proof that the permanent problem is #P complete, Valiant showed that it is very unlikely for this problem to be contained in FP. Unfortunately, analogous results have been shown for many interesting problems, implying that one cannot hope to solve them efficiently.

As analogous hardness results have been shown for many decision problems as well, complexity theory soon became devoted to the question how to circumvent this hardness. Much effort has been spent in trying to approximate the solutions of these problems or to find restrictions of the problems under consideration that could be solved efficiently.

If exact solutions are desired, one has to rely on the restricted case analysis. For counting problems, especially in the work of Vadhan (cf. [Vad01]) and Roth ([Rot96]), it has been shown that the approach of considering only very restricted subsets of certain problems does not help much in most of the cases, as even many restrictions remain #P complete. Thus one might want a different way of finding exact solutions and circumventing the hardness to some extend.

In the field of decision problems there is such an approach called *parameterized complexity*, which has become quite popular in recent years. Its underlying idea is readily understood.

Complexity theory measures, for example, the time needed to solve a certain problem by the size of the input. Especially with respect to classification results

as mentioned above, this way of measuring complexity might seem unfortunate, since almost all structural information of the instances is lost. Parameterized complexity aims at describing some (important) part of the structure of a problem by a *parameter*. This idea together with some of its implications has led to new paradigms in designing efficient algorithms and encouraged the development of a whole new theory of intractability.

As opposed to decision problems, a theory of parameterized counting problems is still in its early stages. Structural issues of this theory, for example, have been considered only by McCartin (see [McC02]) and Flum and Grohe (cf. [FG06]).

In this thesis we will consider the two branches of parameterized complexity in the context of counting problems. The first part will be devoted mainly to parameterized algorithm design. For decision problems there is an important technique called *kernelization* that is applied in many parameterized algorithms. As this technique has yet no analog for counting problems, we will examine ways of applying it to counting problems. This will be the work of chapter 1 and 2. In chapter 3, we will know enough about the application of this technique such that we will be able to develop a formal characterization of it. Furthermore we will see that this characterization is equivalent to the notion of the tractability of parameterized counting problems.

As certain results from chapter 2 have implications on the structural theory of parameterized counting problems, we will reckon these by proposing a redefinition of some of its complexity classes.

The second part of this work deals with the intractability of parameterized counting problems. Most importantly, we will see that the parameterized problem of counting bipartite cliques in bipartite graphs is complete for the class #A[1]. To a certain extend, this class can be considered as a parameterized analog of #P.

Furthermore, in chapter 6, we will prove that the parameterized problems of counting induced cycles and paths are #A[1]-complete as well. Eventually, we will summarize some of the results demonstrated and give an impression of the difference between (classical) complexity theory and parameterized complexity theory in classifying certain problems.

Preliminary Definitions

We assume that the reader is familiar with some basic notions from discrete mathematics, and in particular with the "big Oh" notation and its relatives.

For a set S let $\binom{S}{k}$ denote the family of all k-element subsets of S. Furthermore, 2^S denotes the family of all subsets of S.

A *graph* is a pair $\mathcal{G} = (V, E)$ with V being a set of *vertices* and $E \subseteq \binom{V}{2}$ a set of *edges*. For $v \in V$ we call $N_{\mathcal{G}}(v) := \{w \in V \mid \{v, w\} \in E\}$ the *neighborhood* of v in $\mathcal{G}$ and $d_{\mathcal{G}}(v) := |N_{\mathcal{G}}(v)|$ the *degree* of v in $\mathcal{G}$. If the graph $\mathcal{G}$ is clear from context, we simply omit it from the subscript of the given concepts.

We define $\Delta(\mathcal{G}) := max\{d_{\mathcal{G}}(v) \mid v \in V\}$ as the *maximum degree* of $\mathcal{G}$ and analogously $\delta(\mathcal{G}) := min\{d_{\mathcal{G}}(v) \mid v \in V\}$ denotes the *minimum degree* of $\mathcal{G}$.

A *path* in a graph $\mathcal{G} = (V, E)$ is a sequence of vertices $v_1, \dots v_n$ such that $v_i \in V$ for all $i \in [n]$ and $\{v_i, v_{i+1}\} \in E$ for all $i \in [n-1]$. Two vertices $u, v \in V$ are *connected* in $\mathcal{G}$ if there is a path from u to v in $\mathcal{G}$. A *connected component* of $\mathcal{G}$ is a maximal subset $C \subseteq V$ such that every pair of vertices $u, v \in C$ is connected in $\mathcal{G}$. $\mathcal{G}$ is connected if $C = V$ holds for a connected component C.

A *directed* graph or *digraph* is a pair $\mathcal{G} = (V, E)$ where $E \subseteq V \times V$.

A *bipartite* graph is a triple $\mathcal{B} = (U, W, F)$ such that F is a set of edges $e \in F$ that satisfy $e \cap U \neq \emptyset$ and $e \cap W \neq \emptyset$. We say that F is a set of edges *between* U and W. Bipartite *digraphs* are defined analogously.

A *hypergraph* is a pair $\mathcal{H} = (V, E)$ with V being a set of *vertices* and $F \subseteq 2^V$ a family of sets of vertices, called *hyperedges*. $\mathcal{H}$ is called d-uniform if all of its hyperedges have cardinality d. The *size* of a hypergraph $\mathcal{H} = (V, E)$ is defined as

$$\|\mathcal{H}\| := |V| + \sum_{e \in E} |e|$$

Note that this defines the size $\|\mathcal{G}\|$ of a graph $\mathcal{G}$ as well, since every graph is a 2-uniform hypergraph.

Now, we will explain the basic notions needed to understand parameterized complexity. We will introduce these notions for decision problems, which facilitates explaining their analogs for counting problems.

Parameterized complexity mainly relies on two notions, that of *parameterized problems* and *fixed parameter tractability*. Recall that a *language* Q over a finite alphabet Σ is a set of strings, that is $Q \subseteq \Sigma^*$.

Definition 0.1. *Let Σ be a finite alphabet*

- *A* parameterization *of Σ^* is a mapping $\kappa : \Sigma^* \to \mathbb{N}$ that is polynomial time computable.*
- *A* parameterized problem *over Σ is a pair (Q, κ) that consists of a language Q over Σ and a parameterization κ.*

For natural problems $Q \subseteq \Sigma^*$ we mainly consider parameterizations κ such that for $x \in \Sigma^*$ the value $\kappa(x)$ (i.e. the *parameter* of x) is small in comparison to the length $|x|$ of x. An example will make this more clear.

Consider a graph $\mathcal{G} = (V, E)$. A *vertex cover* in $\mathcal{G}$ is a set $S \subseteq V$ of vertices such that $S \cap e \neq \emptyset$ holds for all edges $e \in E$. A vertex cover S has *size* k if $|S| = k$. The classical vertex cover problem is defined as follows.

VertexCover
Instance: A graph $\mathcal{G} = (V, E)$ and $k \in \mathbb{N}$ *Problem:* Decide if there is a vertex cover of size k in $\mathcal{G}$

The corresponding parameterized problem, defined by a "natural" parameterization, includes the size k of the vertex cover sought into the parameter.

p-VertexCover
Instance: A graph $\mathcal{G} = (V, E)$ and $k \in \mathbb{N}$ *Parameter:* k *Problem:* Decide if there is a vertex cover of size k in $\mathcal{G}$

Considerations about the complexity of this problem regard k as fixed whereas the size of the input graph may grow arbitrarily. From a practical point of view this is motivated by the fact that often one is interested in small vertex covers no matter how large the graph $\mathcal{G}$ may be. This is the notion of the small parameter, which in turn motivates a refinement of the definition of a tractable problem. In parameterized complexity a problem is regarded as tractable if, except for an arbitrary dependence on the parameter, the problem can be solved in polynomial time. The following definition formalizes this notion.

Definition 0.2. *A parameterized problem (Q, κ) is said to be* fixed parameter tractable *if there is a (deterministic) algorithm $\mathbb{A}$ and a computable function $f : \mathbb{N} \to \mathbb{N}$ that for any $x \in \Sigma^*$ decides $x \in Q$ in time*

$$f(\kappa(x)) \cdot |x|^c$$

for some constant c. In this context $\mathbb{A}$ is an fpt-algorithm *with respect to κ.*

By FPT we denote the class of all fixed parameter tractable problems.

In the case of p-VertexCover, for example, there is an algorithm, solving this problem in time $2^{\mathcal{O}(k)}n$ with $n := |V| + |E|$. Hence for small values of k this problem can be solved efficiently.

These notions can be transferred to counting problems. In general, a *counting problem* is a function $F : \Sigma^* \to \mathbb{N}$ for a finite alphabet Σ. Similarly to the case of decision problems, one can define *parameterized counting problems.*

Definition 0.3. *Given a finite alphabet Σ.*

- *A* parameterized counting problem *is a pair (F, κ), such that $F : \Sigma^* \to \mathbb{N}$ and κ is a parameterization.*
- *Accordingly, a parameterized counting problem (F, κ) is* fixed parameter tractable *if there is an* fpt-algorithm *with respect to κ that computes F.*
- *The class of all fixed parameter tractable counting problems is denoted by $\#FPT$.*

With our example from above the natural counting problem corresponding to p-VertexCover is defined as follows.

p-#VertexCover
Instance: A graph $\mathcal{G} = (V, E)$ and $k \in \mathbb{N}$ *Parameter:* k *Problem:* Compute the number of size k vertex covers in $\mathcal{G}$

We will fix some further notation. Given an instance $x \in \Sigma^*$ of a parameterized counting problem p-#Problem we define #problem(x) as the image of x under the function realized by p-#Problem. For example #vertexcover($\mathcal{G}, k$) denotes the number of size k vertex covers of the graph $\mathcal{G}$.

From now on, we will define only parameterized counting problems. Given such a problem p-#Problem we always refer by p-Problem to the decision problem connected with p-#Problem, without explicitly defining it.

Part I.

Kernelization and The Tractability of Parameterized Counting Problems

In the field of parameterized decision problems, the concept of *kernelization* is one of the most powerful tools for designing efficient fpt-algorithms. Kernelization itself, however, is only a general notion that comprises a variety of techniques for reducing the size of the input of a problem.

Definition 0.4. *Given a parameterized decision problem (Q, κ) over a finite alphabet Σ.*

A kernelization *of (Q, κ) is a polynomial time computable function $K : \Sigma^* \to \Sigma^*$ satisfying the following conditions:*

- *$x \in Q \Leftrightarrow K(x) \in Q$, for all $x \in \Sigma^*$*
- *There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \Sigma^*$ we have*

$$|K(x)| \leq g(\kappa(x))$$

$K(x)$ is called the kernel *of x.*

Informally, given an instance x of any problem, a kernelization plays the role of a precomputation, which reduces the size of the instance until it depends only on the parameter. Then an algorithm, which might even be a brute force algorithm, can solve the problem on the kernel efficiently, as the kernel $K(x)$ may be assumed to be small in comparison to x.

This technique has indeed created some of the most efficient fpt-algorithms and it is one of the most frequently applied approaches in parameterized algorithm design. Considering this popularity of kernelization in the field of decision problems one would naturally ask, if this notion can be applied to counting problems as well.

Unfortunately, the notion given above is generally not sufficient to satisfy a definition of a kernelization for counting problems. This is due to the fact that, by definition 0.4 given a counting problem (F, κ) and an instance $x \in \Sigma^*$, it is not necessarily the case that $F(x) = F(K(x))$. In a reasonable notion of kernelizations of counting problems, however, this would be a necessary condition. Nonetheless, an adaptation of the notion above for counting is not as obvious as is seems. For example, if we stipulate that $F(x) = F(K(x))$ for all instances $x \in \Sigma^*$, then this would be way too restricted. Even the p-#VertexCover problem could not be described by this refined notion. The following example illustrates why this is the case.

Example 0.5. *Let $\mathcal{G} = (V, E)$ be a graph and $k \in \mathbb{N}$. Let $K : \Sigma^* \to \Sigma^*$ be a function mapping instances of* p-#VertexCover *to instances of the same problem and let $g : \mathbb{N} \to \mathbb{N}$ be a computable function. Furthermore, let K be such that for all graphs $\mathcal{H}$ we have $|K(\mathcal{H}, k)| \leq g(k)$.*

Suppose that $\#\text{vertexcover}(K(\mathcal{G}, k)) = \#\text{vertexcover}(\mathcal{G}, k)$. As $K(\mathcal{G}, k)$ is an instance of p-#VertexCover*, we have $K(\mathcal{G}, k) = (\mathcal{G}', k')$ for a graph $\mathcal{G}' = (V', E')$ and $k' \in \mathbb{N}$. As $\|\mathcal{G}'\| \leq g(k)$, we know that there can be no more than $\binom{g(k)}{k'}$ vertex covers of cardinality k' in $\mathcal{G}'$.*

Let $\mathcal{H} = (W, \emptyset)$ be a graph with $|W| = g(k) + k$. It is easy to see that

$$\#\text{vertexcover}(\mathcal{H}, k) = \binom{g(k)+k}{k} > \binom{g(k)}{k} \geq \#\text{vertexcover}(K(\mathcal{H}, k)).$$

This example shows that a mapping K cannot be applied to p-#VertexCover such that it satisfies both definition 0.4 and $F(x) = F(K(x))$ for all $x \in \Sigma^*$.

For the time being, we will abandon any attempt to give a formal notion of "kernelization of counting problems". Hence our aim in this part of the thesis will be to examine certain techniques that are applied in kernelizations of decision problems. Learning about the way these techniques can be applied in counting algorithms will help to understand what might be a reasonable notion for these counting kernels.

Even though we drop the formal definition of a kernel, we want to fix some important requirements that we think are essential for a counting analog of kernelizations.

Requirement 0.6. *A preprocessing, if performed, can be done in polynomial time. Furthermore, after preprocessing, the part of the instance that will be solved by a search algorithm has a size that is bounded by a function of the parameter.*

For convenience we will denote these preprocessings or data reductions by *kernelizations*. This will cause no ambiguities as it will be clear from context if we refer to counting or decision problems. In cases where ambiguities might arise, we refer to the precomputations as *counting kernelizations*.

Time measures for arithmetic computations

We will use a standard model of *random access machines* (RAM) for analyzing the time complexity of the algorithms discussed in this work. In this model we allow addition, subtraction, multiplication and divison of natural numbers as arithmetic operations. Note that models of this kind are applied only in the analysis of practical algorithms. In a purely theoretical context allowing multiplication and division as operations is not common.

In algorithms for counting problems the arithmetic computations play an important part. Therefore, in determining the time complexity of an algorithm it is advisable to explicitly measure the time spent in these computations. Naturally, there are two different measures that come to mind (see, for example [AHU74]):

1. The *uniform cost measure* (UCM), considers every basic operation to take unit time.
2. The *logarithmic cost measure* (LCM) treats the operations, depending on the length of the operands.

Generally, the uniform cost measure is used, when it is known that the numbers the algorithm has to deal with do not exceed certain bounds. Such a bound might, for example, be the register size of a computer. Accordingly, the logarithmic cost measure applies to all cases in which this cannot be guaranteed.

For simplicity, we will use the uniform cost measure if not stated otherwise. This implies that all basic arithmetic computations can be carried out in constant time. However, as the numbers involved in the computations can become quite large we will pick some cases were, additionally, we apply the logarithmic cost measure.

We are mainly interested in the complexity of arithmetic operations. Therefore, when we apply the logarithmic cost measure this will only be the case for arithmetic operations. All other operations will still be treated by the UCM. This distinction is motivated by the fact that apart from an extensive use of arithmetic operations, counting algorithms do not differ very much from decision algorithms.

To be able to apply the LCM, we need to know the time arithmetic computations take. All numbers will be considered as binary. Given two numbers a and b, assume w.l.o.g. $a = \max\{a, b\}$ and let $n := \lceil \log a \rceil$. For simplicity, we refer to both a and b as n-bit numbers.

Note that adding two n bit numbers can always be done in time $\mathcal{O}(n)$ by a straightforward algorithm. For multiplication and division this is more complicated. As a detailed discussion would be beyond the scope of this work, we only state the existence of an algorithm whose running time is less than the trivial time bound of $\mathcal{O}(n^2)$.

Fact 0.7 (Schönhauser, Strassen). *On a random access machine two n-bit numbers can be multiplied in time $\mathcal{O}(n \cdot \log n)$. Integer division can be carried out within a constant factor of multiplication.*

A discussion of this can be found, for example, in [Knu81].

We will refer to the time needed to multiply two n-bit numbers as $t_{mul}(n)$. Implicitly, this time will be always assumed to equal the term from fact 0.7.

Chapter 1.

Vertex Cover

The parameterized decision problem p-VertexCover is known to admit a wide range of kernelizations. In this chapter we will study three important techniques, to wit, the *Buss Kernelization* the *Crown Rule Reduction* and the *Linear Programming* method, and the way of applying them to counting vertex covers.

In most cases kernelization techniques are combined with fast fpt algorithms for solving a given problem. This helps in improving the time bounds further. Here, we will introduce such an algorithm for p-#VertexCover which we will use later on to combine it with several kernelization techniques.

Algorithm 1 is an adaptation of the CountGHS algorithm given in [FG06]. Since it will play an important part in this chapter, we provide a bound on the running time of this algorithm.

Let $\#vc(\mathcal{G}, k)$ denote the number of size k vertex covers in a graph $\mathcal{G} = (V, E)$.

```
   CountVC(G, k, F) // G a graph, k ∈ ℕ, F ⊆ V
 1 if k > |V \ F| then return 0;
 2 else if E = ∅ then return (|V\F| choose k);
 3 else
 4     choose e ∈ E;
 5     h ← 0 ;
 6     forall S_0 ⊆ e \ F with 0 < |S_0| ≤ k do
 7         V' ← V \ S_0 ;
 8         E' ← {e ∈ E | e ⊆ V'};
 9         G' ← (V', E');
10         F' ← F ∪ (e \ S_0);
11         k' ← k − |S_0| ;
12         h ← h + CountVC(G', k', F');
13     end
14     return h;
15 end
```

Algorithm 1: Counting vertex covers

Theorem 1.1. *Given a graph $\mathcal{G} = (V, E)$ and $k \in \mathbb{N}$. The algorithm* CountVC *solves* p-#VertexCover *in time $\mathcal{O}((1+\sqrt{3})^k \cdot \|\mathcal{G}\|)$*

Proof. Note that, CountVC is a trivial p-#VertexCover adaptation of the algorithm CountGHS given in [FG06]. This implies the correctness of the algorithm.

We tighten the running time analysis. Let $\|\mathcal{G}\| = n$ and let $T(k, n)$ denote the maximum running time of CountVC$(\mathcal{G}, k, \emptyset)$. Then, we have:

$$\begin{aligned} T(0,n) &= \mathcal{O}(1) \\ T(1,n) &= 2 \cdot T(0,n) + \mathcal{O}(n) \\ T(k,n) &= T(k-2,n) + 2 \cdot T(k-1,n) + \mathcal{O}(n) \end{aligned}$$

for $k > 1, n \in \mathbb{N}$.

Let c be a constant large enough such that the terms $\mathcal{O}(1)$ and $\mathcal{O}(n)$ above are bounded by c and $c \cdot n$, respectively.

By induction on k, we show that $T(k, n) \leq c \cdot (1 + \sqrt{3})^k \cdot n$. For $k = 0$ this is trivial. For $k = 1$ we have $T(1, n) = 2 \cdot T(0, n) + \mathcal{O}(n) \leq 2cn$.
Now, let $k > 1$ and define $b := (1 + \sqrt{3})$ then

$$\begin{aligned} T(k,n) &\leq T(k-2,n) + 2 \cdot T(k-1,n) + \mathcal{O}(n) \\ &\leq (b^{k-2} + 2b^{k-1} + 1) \cdot cn \\ &\leq b^{k-2} \cdot (4 + 2\sqrt{3}) \cdot cn \\ &= b^{k-2+\log_b(4+2\sqrt{3})} \cdot cn = b^k \cdot cn \end{aligned}$$

□

1.1. Applying Buss' Kernelization to p-#VertexCover

A simple but powerful kernelization of p-VertexCover is known as *Buss' Kernelization*. Among the kernelizations for this problem this one is the simplest but least efficient. However, as far as p-#VertexCover is concerned, we will see that its efficiency is much better than that of the other approaches.

The main idea behind Buss' kernelization is expressed in the following lemma.

Lemma 1.2. *Let $\mathcal{G} = (V, E)$ be a graph and $k \in \mathbb{N}$ then:*

1. *Any $v \in V$ with $d(v) > k$ is contained in every k-element vertex cover of $\mathcal{G}$.*
2. *If $\Delta(\mathcal{G}) \leq k$ and $\mathcal{G}$ has a k-element vertex cover, then $|E| \leq k^2$.*

The proof is immediate, therefore we omit it. Algorithm 2 shows how to apply this lemma to counting vertex covers. Here, $\mathcal{G}_0 = (\{a, b\}, \{\{a, b\}\})$ is the graph that consist of exactly one edge and $\mathcal{G}_1 = (\{a\}, \emptyset)$. Hence $\#vc(\mathcal{G}_0, 0) = 0$ and $\#vc(\mathcal{G}_1, 0) = 1$.

Theorem 1.3. *Given a graph $\mathcal{G} = (V, E)$ and $k \in \mathbb{N}$.*
The algorithm CountBussVC *correctly solves* p-#VertexCover *on $\mathcal{G}$. With*

```
   CountBussVC(G, k) // G = (V, E) a graph, k ∈ ℕ
 1 V' ← V; E' ← E;
 2 k' ← k; G' ← (V', E');
 3 while there is a v ∈ V' with d_{G'}(v) > k' do
 4     E' ← E' \ {e ∈ E | ∃ w ∈ V' : e = {v, w}};
 5     V' ← V' \ {v}; G' ← (V', E');
 6     k' ← k' − 1;
 7 end
 8 if k' = 0 and E' = ∅ then G' = (V', E') ← G_1;
 9 if k' ≤ 0 or |E'| > (k')^2 or |V'| < k' then
       // i.e.  #vc(G', k') = 0
10     G' = (V', E') ← G_0;
11     k' ← 0 ;
12 end
13 I ← {v ∈ V' | d_{G'}(v) = 0};
14 V' ← V' \ I;
15 return Σ_{i=0}^{k'} CountVC(G', i, ∅) · C(|I|, k'−i);
```

Algorithm 2: Counting Vertex Covers via Buss' Kernelization

a uniform cost measure this takes time

$$\mathcal{O}((1+\sqrt{3})^k k^2 + k \cdot \|\mathcal{G}\|)$$

Proof. Call lines 1-14 the *kernelization phase* of the algorithm. To see the correctness of the algorithm, consider $k \in \mathbb{N}$ and $\mathcal{G} = (V, E)$. Furthermore, let $v \in V$ be a vertex with $d(v) > k$ and define the graph $\mathcal{G}^* = (V \setminus \{v\}, E^*)$ with $E^* := E \setminus \{\, \{v, u\} \in E \mid u \in V\}$. Lemma 1.2 implies that $\#vc(\mathcal{G}, k) = \#vc(\mathcal{G}^*, k-1)$. Thus, the kernelization phase is correct.

For the remainder of the algorithm, consider the reduced graph $\mathcal{G}' = (V', E')$ and $k' \in \mathbb{N}$. Note that for $\mathcal{G}' \in \{\mathcal{G}_0, \mathcal{G}_1\}$ with $k' = 0$ the last line of the algorithm is correct. Hence assume that $V' \neq \emptyset$ and $E' \neq \emptyset$. Note that $\mathcal{G}'$ may contain isolated vertices, which may be contained in a vertex cover of size k'. Let I be the set of these isolated vertices and consider them as deleted from V'. Thus any size k' vertex cover C in $\mathcal{G}'$ satisfies $|V' \cap C| = i \in [k']$ and $|I \cap C| = k' - i$. As I and V' are disjoint, this implies, that the union of any size i vertex cover of $\mathcal{G}'$ and any subset $A \subseteq I$ with $|A| = k' - i$ yields a size k' vertex cover. This shows that the computation carried out in the last line of the algorithm is correct.

Time Complexity. The kernelization phase of the algorithm can be carried out in time $\mathcal{O}(k \cdot \|\mathcal{G}\|)$. To see this, note that the **while**-loop is repeated at most k times and each iteration can be carried out in linear time. Furthermore, the other operations in the kernelization phase do not increase this bound.

The last line of the algorithm induces at most $k+1$ additions in the sum. For

the reduced graph $\mathcal{G}' = (V', E')$ lemma 1.2 implies that $|E'| \leq k^2$ and $|V'| \leq 2k^2$. Hence, the call CountVC$(\mathcal{G}', i, \emptyset)$ completes in time $c \cdot b^i 2k^2$ with $b = (1 + \sqrt{3})$ and an appropriate constant c.

Note that, by the definition of the binomial coefficient, computing $\binom{|I|}{k'-i}$ takes at most $\mathcal{O}(k)$ many steps which involve $k' - i$ multiplications, $k' - i$ divisions and $2(k - i + 1)$ subtractions. We simply assume that this takes time $c \cdot k$.

Thus, evaluating $\sum_{i=0}^{k'}$CountVC$(\mathcal{G}', i, \emptyset) \cdot \binom{|I|}{k'-i}$ takes time

$$\begin{aligned} k + \sum_{i=0}^{k}(c \cdot b^i 2k^2 + c \cdot k) &\leq k + c \cdot k^2 + c \cdot 2k^2 \cdot \sum_{i=0}^{k} b^i \\ &\leq k + c \cdot k^2 + c \cdot 2k^2 \cdot \frac{b^{k+1} - 1}{b - 1} \\ &= k + c \cdot k^2 + c \cdot 2k^2 \cdot \frac{b^{k+1} - 1}{\sqrt{3}} \\ &\in \mathcal{O}(k^2 b^k) \end{aligned}$$

Accordingly, the whole algorithm completes in time $\mathcal{O}((1 + \sqrt{3})^k k^2 + k \cdot \|\mathcal{G}\|)$, as claimed. □

1.1.1. Logarithmic Cost Measure

The time bound given in theorem 1.3 suggests that the time spent in the computation on the reduced instance does not depend on the size of the original graph $\mathcal{G}$ anymore. In kernelizations of decision problems this is always the case. But we will see now that the situation changes if we consider the arithmetic computations involved in the CountBussVC algorithm under the logarithmic cost measure. Note first that the running time of the CountVC algorithm does not change under the LCM.

Observation. Under the logarithmic cost measure p-#VertexCover can be solved by CountVC in time $\mathcal{O}((1 + \sqrt{3})^k \|\mathcal{G}\|)$.

This is due to the fact that in a graph $\mathcal{G} = (V, E)$ with $n = |V|$ there can be at most n^k vertex covers of size k. Therefore, the numbers summed in the algorithm are of size at most $k \log n$. And as addition takes linear time, the LCM does not change the estimate.

Lemma 1.4. *With a logarithmic cost measure the algorithm* CountBussVC *solves* p-#VertexCover *in time*

$$\mathcal{O}(k \cdot \|\mathcal{G}\| + k^2(1 + \sqrt{3})^k + k^3 \log n \cdot (\log k + \log\log n)).$$

Proof. Let $|V| = n$. Define $t_{add}(x)$ as the time needed to add two x bit numbers. Let c be a constant larger than the constants hidden in t_{add} and the time bound

of CountVC. That is, we assume that $t_{add}(x) \leq c \cdot x$ and CountVC completes in time $c \cdot b^i \|\mathcal{G}\|$ with $b = (1 + \sqrt{3})$.

We reconsider algorithm 2 with a logarithmic cost measure. Note that the time bounds for the computations do not change, except for the last line of the algorithm.

First, we determine the time to compute the binomial coefficient:

$$\binom{|I|}{k' - i} = \frac{|I| \cdot \ldots \cdot (|I| - k' - i + 1)}{(k' - i) \cdot (k' - i - 1) \ldots \cdot 2}.$$

We have to perform $k' - i$ multiplications, $k' - i$ divisions and $2(k' - i + 1)$ subtractions. Hence, by the fact that division is within a constant factor of multiplication (see fact 0.7), we may assume that at most $2k$ multiplications and $2k$ additions are performed.

As $|I|$ can be represented by a $\log n$ bit number, the largest numbers multiplied have a size of at most $k \log n$ bits and the numbers involved in subtractions have size at most $\log n$ bits. Thus, the binomial coefficient can be computed in time at most

$$2k \cdot t_{add}(\log n) + 2k \cdot t_{mul}(k \log n).$$

Note that for the reduced graph $\mathcal{G}' = (V', E')$ we have $|V'| \leq 2k^2$ and $|E'| \leq k^2$. Thus, by observation 1 the time spent in CountVC$(\mathcal{G}', i, \emptyset)$ is at most $\mathcal{O}(b^i \cdot k^2)$ with $b = (1 + \sqrt{3})$. And as $\mathcal{G}'$ may contain at most $(2k^2)^i$ vertex covers of size i, the number returned is of size at most $2i \cdot \log 2k \leq k \log n$. Thus the time needed to multiply CountVC$(G', i, \emptyset)$ and $\binom{|I|}{k'-i}$ is at most $t_{mul}(k \log n)$.

Altogether, the ith term of the sum in the last line of the algorithm can be computed in time at most

$$t_{mul}(k \log n) + c \cdot b^i \cdot k^2 + 2k \cdot t_{add}(\log n) + 2k \cdot t_{mul}(k \log n)$$

For the whole last line we have to reckon the addition of the $k+1$ different terms of the sum, of which each has size at most $k \cdot \log n$. Let c' be a constant large enough to bound all of the constants given above, then we have:

$$c' \cdot k \cdot t_{add}(k \cdot \log n) + \sum_{i=0}^{k} c' \cdot b^i \cdot k^2 + c' \cdot k \cdot t_{add}(\log n) + c' \cdot k \cdot t_{mul}(k \log n)$$

Therefore, we may simply omit c' and we obtain:

$$\begin{aligned} & k \cdot (t_{add}(k \cdot \log n) + k \cdot t_{add}(\log n) + k \cdot t_{mul}(k \log n)) + k^2 \sum_{i=0}^{k} b^i \\ = \;& k \cdot (k \cdot \log n + k \cdot \log n + k \cdot t_{mul}(k \log n)) + k^2 \sum_{i=0}^{k} b^i \\ \in \;& \mathcal{O}(k^2 \cdot \log n + k^2 \cdot t_{mul}(k \log n) + k^2 \cdot b^k) \end{aligned}$$

As the kernelization phase takes still $\mathcal{O}(k \cdot \|\mathcal{G}\|)$, the time bound for the whole algorithm is:

$$\mathcal{O}(k \cdot \|\mathcal{G}\| + k^2 b^k + k^2 \log n + k^2 \cdot t_{mul}(k \log n)) \qquad \square$$

The time bound just derived shows that even in the computations on the reduced graph $\mathcal{G}'$ the time still depends on the original input graph $\mathcal{G}$. Fortunately, this dependence does not show in the part $k^2 b^k$ of the time bound that is exponential in k. Thus for the algorithm at hand, the application of the logarithmic cost measure does not change much in comparison to the UCM. However, in the next chapter we will see that this is not always the case.

1.2. Crown Rule Reduction

One of the most efficient kernelization techniques known in parameterized algorithm design is the application of the so-called *Crown Rule Reduction*.

Definition 1.5. *Let $\mathcal{G} = (V, E)$ be a graph. A* crown *in $\mathcal{G}$ is a bipartite subgraph $\mathcal{C} = (I, N(I), F)$ of $\mathcal{G}$ satisfying three conditions:*

(1) I is an independent set in $\mathcal{G}$ and $N(I)$ is the set of all neighbors of vertices from I in $\mathcal{G}$.

(2) F contains all edges from E that connect vertices in $N(I)$ to vertices in I.

(3) $\mathcal{C}$ has a matching of cardinality $|N(I)|$.

Let $vc(\mathcal{G})$ denote the minimum size of a vertex cover in $\mathcal{G}$.

Let $\mathcal{G} = (V, E)$ be a graph. A *matching* in $\mathcal{G}$ is a set $M \subseteq E$ of pairwise disjoint edges. A vertex $v \in V$ is *free* with respect to M if it is not incident with an edge in M. We say that M is *maximal* if there is no matching M' in $\mathcal{G}$ such that $M \subset M'$. The matching M is *maximum* if there is no matching M' in $\mathcal{G}$ with $|M| < |M'|$. An *M-alternating* path in $\mathcal{G}$ is a path whose edges alternately belong to M and $E \setminus M$.

The possibility of computing a crown efficiently is tighly connected to the following two facts. Their proof can be found, for example, in [FG06].

Lemma 1.6. *(1) A maximal matching in a given graph can be computed in linear time.*

(2) Given a bipartite graph $\mathcal{B}$ and $k \in \mathbb{N}$. There is an algorithm that decides the existence of a size k matching in $\mathcal{B}$ in time $\mathcal{O}(k \cdot \|\mathcal{B}\|)$. If there exists such a matching then the algorithm computes it within the same time bounds.

Let $\mathcal{G} = (V, E)$ be a graph and $k \in \mathbb{N}$. We will follow an algorithm given in [FG06] that constructs a crown in $\mathcal{G}$. For the time being, let us assume, that $\mathcal{G}$ contains no isolated vertices. The case of isolated vertices will be considered later. After we have seen the construction, we will show how to use the special structure of a crown in a counting algorithm. In order to do this it will be convenient

to explicitly refer to the different exit conditions of the construction algorithm. Therefore, we denote them by (E1), (E2), ...

First, compute a maximal matching L of $\mathcal{G}$. If $|L| > k$ then $vc(\mathcal{G}) > k$ and thus the algorithm returns zero. (E1)

Assume that $|L| \leq k$ and let I be the set of all vertices that are free with respect to L. By the maximality of L, I is an independent set. If $|I| \leq k$ then $|V| = 2 \cdot |L| + |I| \leq 3k$. That is, $\mathcal{G}$ itself is a "kernel" and we call CountVC$(\mathcal{G}, k)$ (see algorithm 1) to solve the problem. (E2)

In the following assume $|I| > k$. We will see now how to construct a crown $\mathcal{C}$ in $\mathcal{G}$ on at least $|V|-3k$ vertices. Let $\mathcal{B} = (I, N(I), F)$ be the bipartite graph such that F contains all edges from $\mathcal{G}$ that connect vertices in $N(I)$ to vertices in I. Then a *maximum* matching M is constructed in $\mathcal{B}$. If $|M| > k$ then $vc(\mathcal{G}) \geq |M| > k$ and the algorithm returns zero. (E3)

If $|M| \leq k$ and $|M| = |N(I)|$ then $\mathcal{B}$ is the crown sought, as $\mathcal{B}$ contains at least $|V| - 2|L| \geq |V| - 2k$ vertices. (E4)

Otherwise, if $|M| \leq k$ and $|M| < |N(I)|$ then denote by J the set of all vertices in I that are free with respect to M. As $|I| > k$ the set J is nonempty. Let C be the set of all vertices of $\mathcal{B}$ that can be reached from a vertex in J by an M-alternating path and define $\mathcal{C}$ as the induced subgraph of $\mathcal{B}$ with vertex sets $I' := I \cap C$ and $N(I')$.

Let M' be the restriction of M to the edges in $\mathcal{C}$. The following claim is proven in [FG06]. So we omit the proof here.

Claim 1. $\mathcal{C}$ is a crown with $|I' \cup N(I')| \geq |V| - 3k + 2|M'|$

By this fact, we know that the number of vertices in $\mathcal{G}$ that do not belong to $\mathcal{C}$ is at most $3k$ and therefore $\mathcal{C}$ is a crown, as desired. (E5)

In the cases where a crown is constructed ((E4) and (E5)) let this crown be $\mathcal{C} = (I, N(I), F)$. We will show now how to obtain a graph whose size is bounded by a function of k and that is appropriate for counting.

Applying a crown in a counting algorithm. Recall that $|N(I)| < k$. Note furthermore that with respect to the edges in $\mathcal{G}$ (and hence in $\mathcal{C}$ as well), all vertices in I have neighbors only in $N(I)$. Therefore, we can define an equivalency relation with at most 2^k equivalency classes by defining for all $v, w \in I$:

$$v \sim w \;:\Leftrightarrow\; N(v) = N(w) \tag{1.1}$$

Furthermore, for every $v \in I$ define $[v]$ as the equivalency class of v, that is $[v] := \{w \in I \mid v \sim w\}$.

Claim 2. Given a vertex cover S of $\mathcal{C}$ such that there is a vertex $y \in N(I) \setminus S$. Let $v \in I$ be a vertex with $y \in N(v)$, then $[v] \subseteq S$.

Proof. Assume that there is a $w \in [v]$ with $w \notin S$. Then, by the definition of $[v]$ there is an uncovered edge $\{y, w\}$ in $\mathcal{C}$. Contradiction. ✓

This claim opens up a possibility to count the vertex covers in a graph by exploiting the existence of a large crown.

Let $\mathcal{G} \setminus \mathcal{C}$ be the graph obtained from $\mathcal{G}$ by deleting all vertices in $\mathcal{C}$ and all edges incident to vertices in $\mathcal{C}$.

Let S be a vertex cover of $\mathcal{C}$ and let $\mathcal{G}'$ be the graph obtained from $\mathcal{G} \setminus \mathcal{C}$ by deleting all edges covered by vertices in S. One can easily see that the number $\#vc(\mathcal{G}', k - |S|)$ equals the number of size k vertex covers in $\mathcal{G}$ that are supersets of S.

As the size of $\mathcal{G}'$ depends only on k, $\#vc(\mathcal{G}', k - |S|)$ can be computed by CountVC in time depending only on k. Therefore, to show how to compute the number of size k vertex covers in $\mathcal{G}$ it remains to show, how to enumerate the vertex covers of $\mathcal{C}$.

Fix a set $S \subseteq N(I)$. Claim 2 entails a way of constructing a set $L \subseteq I$ such that $S \cup L$ is a minimal (with respect to inclusion) vertex cover of $\mathcal{C}$. Let

$$\mathbf{A}(S) := \{[v] \mid v \in I,\ \exists\, y \in N(I) \setminus S \text{ such that } y \in N(v)\}.$$

and define

$$L(S) := \bigcup_{[v] \in \mathbf{A}(S)} [v] \tag{1.2}$$

Intuitively, $\mathbf{A}$ is the (unique) minimal family of equivalency classes that is necessary such that $S \cup L(S)$ is a vertex cover in $\mathcal{C}$. Thus we only need to see how to generate all vertex covers of size k including those that are not necessarily minimal as in the above sense.

Let $R \subseteq I \setminus L(S)$ (R1) be a set of vertices with $|R| \leq k - |S \cup L(S)|$ (R2). It is easy to see that for every set R satisfying these conditions the set $R \cup S \cup L(S)$ is a vertex cover of $\mathcal{C}$ of size at most k. Hence, we can find all vertex covers of $\mathcal{C}$ of cardinality at most k by enumerating all sets $S \subseteq N(I)$, and forming the corresponding set $L(S)$ according to equation (1.2). Then, if $|S \cup L(S)| \leq k$ it suffices to compute the number of sets R satisfying conditions (R1) and (R2).

These considerations form the main building block of algorithm 3 which computes the number of size k vertex covers of $\mathcal{G}$ with a crown $\mathcal{C}$ present. Noteworthily, for each set $S \subseteq N(I)$ the set $L(S)$ is computed. Then the number of sets R satisfying (R1) and (R2) is accommodated in the **for**-loop below line 10. This entails the correctness of the algorithm.

Theorem 1.7. *Given a graph $\mathcal{G} = (V, E)$, $k \in \mathbb{N}$ and a crown $\mathcal{C} = (I, N(I), F)$ in $\mathcal{G}$ as defined above, such that $\mathcal{C}$ contains at least $|V| - 3k$ vertices. Then the algorithm* CountCrownVC *(algorithm 3) correctly computes the number of size k vertex covers in $\mathcal{G}$ in time*

$$\mathcal{O}(k \cdot \|\mathcal{G}\| + (2 + 2\sqrt{3})^k \cdot k^3)$$

Furthermore, if $\mathcal{G}$ contains isolated vertices, this case can be handled without time overhead.

```
   CountCrownVC(G, k, C)
   // G = (V, E) a graph, k ∈ ℕ and
   // a crown C = (I, N(I), F) on at least |V| − 3k vertices
 1 c ← 0;
 2 Let A be the set of all equivalency classes defined by equation (1.1) ;
 3 forall y ∈ N(I) do compute L_y ← { [v] | y ∈ N(v)};
 4 forall [v] ∈ A do compute |[v]| ;
 5 forall X ⊆ N(I) do
 6     compute L = ⋃_{y∈N(I)\X} L_y;
 7     G' ← the graph G \ C with all edges covered by X deleted;
 8     r ← Σ_{[v]∈L} |[v]|;
 9     k' ← k − |X| − r;
10     if k' ≥ 0 then
11         for i ← 0 to k' do
12             c ← c + C(|I|−r, i) · CountVC(G', k' − i) ;
13         end
14     end
15 end
16 return c;
```

Algorithm 3: Counting Vertex Covers using a Crown

Proof. The correctness of the algorithm follows directly from the considerations above.

For the time complexity of the algorithm, note that each equivalency class $[v]$ can be described by a string $a_v \in \{0,1\}^k$. Computing these strings takes at most $|I| \cdot k$ steps. The sets L_y can be generated in at most $|N(I)| \cdot |I| \leq k \cdot |I|$ steps.

Furthermore, computing the values $|[v]|$ can be done in time $\mathcal{O}(k \cdot |I|)$as follows:

In a single pass through the vertices in I all equivalency classes that contain at least one vertex from I can be determined by analyzing the strings a_v for all $v \in I$. Thus there are at most $|I|$ nonempty equivalency classes. For each of these classes a counter is initialized to zero. Then, in another pass through I, for each $v \in I$ the string a_v is taken to increase the counter for the corresponding equivalency class. After this the counter values contain the sizes of the equivalency classes. Thus, the operations up to line 4 can be carried out in time $\mathcal{O}(k \cdot |I|)$.

Now, consider one of the at most 2^k iterations of the **for**-loop beginning in line 5. Note that L can be computed in time $2^k \cdot k$ and if we keep a fixed copy of $\mathcal{G} \setminus \mathcal{C}$ we can compute $\mathcal{G}'$ in k^2 steps. Observe that if $|X| + |L| > k$ we can simply skip to he next iteration as no solution can be found in this case. Thus, computing r and k' together takes at most k steps. Furthermore, recall that computing the binomial coefficient can be done in time $\mathcal{O}(k)$. Hence, the inner **for**-loop takes time at most $\mathcal{O}(k + k^2 + k^3 \cdot (1+\sqrt{3})^k) \leq \mathcal{O}(k^3 \cdot (1+\sqrt{3})^k)$.

For the big **for**-loop, this implies time $\mathcal{O}(2^k \cdot (2^k k + k^3 \cdot (1+\sqrt{3})^k))$. Thus, the

whole algorithm completes in time

$$\mathcal{O}(\|\mathcal{G}\| + k \cdot |I| + 2^k(2^k \cdot k + (1+\sqrt{3})^k \cdot k^3)) \leq \mathcal{O}(k \cdot \|\mathcal{G}\| + (2+2\sqrt{3})^k \cdot k^3).$$

For the case of isolated vertices, note that, if U is the set of all isolated vertices then the value $|U|$ can be reckoned in the **for**-loop below line 10. Clearly, the increase in running time is swallowed by the "big Oh" notation. □

Corollary 1.8. *Let $\mathcal{G} = (V, E)$ be a graph and $k \in \mathbb{N}$. In applying the Crown Rule Reduction as described above,* p-#VertexCover *can be solved in time*

$$\mathcal{O}(k \cdot \|\mathcal{G}\| + (2+2\sqrt{3})^k \cdot k^3)$$

Proof. Note that the construction of a crown can be carried out in time $\mathcal{O}(k \cdot \|\mathcal{G}\|)$. Thus we only have to check the exit conditions (E1) - (E5).

Note that in conditions (E4) and (E5) the time bound from theorem 1.7 above applies, as in these cases a crown is constructed. Furthermore, in cases (E1) and (E3) no additional time is spent as the algorithm returns zero. In the case that the algorithm returns in (E2) the graph returned contains at most $3k$ vertices. Thus solving this instance with CountVC does not exceed the running time from theorem 1.7. Hence, the time bound follows. □

1.3. Linear Programming

For the decision problem p-VertexCover a kernel is known that is even smaller that the one obtained by the crown rule reduction. The main idea underlying the construction of this kernel is the application of linear programming techniques.

We will show now how to apply this technique to p-#VertexCover which in turn will reveal that its efficiency in counting is comparable to that of the crown rule.

Note that a *linear program* is a system of linear inequalities together with some linear objective function. As an example we show how to define a linear program for the vertex cover problem.

For a graph $\mathcal{G} = (V, E)$ we define the linear program $L(\mathcal{G})$ as follows:

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{v \in V} \quad \text{subject to} \\ x_v + x_w &\geq 1 && \forall \{v, w\} \in E \\ x_v &\geq 0 && \forall v \in V \\ x_v &\leq 1 && \forall v \in V \end{aligned}$$

It is not hard to see that integral solutions of $L(\mathcal{G})$ correspond to vertex covers of $\mathcal{G}$. Call a solution $(x_v)_{v \in V} \in \mathbb{R}^{|V|}$ of $L(\mathcal{G})$ *half-integral* if $x_v \in \{0, \frac{1}{2}, 1\}$ holds for all $v \in V$.

The possibility to apply linear programming techniques to kernelization arises from the following fact which we state without proof.

Fact 1.9. *There is a polynomial time algorithm that computes an optimal solution for a given linear program with rational coefficients.*

The information necessary for computing a kernel for the p-VertexCover problem is provided by the following two lemmas. Proofs of these can be found, for example, in [FG06]. The latter of which is known as the Nemhauser-Trotter theorem.

Lemma 1.10. *Given a graph $\mathcal{G} = (V, E)$. Then $L(\mathcal{G})$ has an optimal half-integral solution that can be computed in polynomial time.*

Lemma 1.11. *Let $\mathcal{G} = (V, E)$ be a graph and $(x_v)_{v \in V}$ an optimal half-integral solution of $L(\mathcal{G})$. Define $V_r := \{v \in V | x_v = r\}$ for $r \in \{0, \frac{1}{2}, 1\}$ and let $\mathcal{G}_r$ be the induced subgraph of $\mathcal{G}$ with vertex set V_r. Then*

(1) $vc(\mathcal{G}_{\frac{1}{2}}) \geq |V_{\frac{1}{2}}|/2$

(2) $vc(\mathcal{G}_{\frac{1}{2}}) = vc(\mathcal{G}) - |V_1|$

By these two facts, we can show how to apply the linear programming technique to counting.

Theorem 1.12. *Let $\mathcal{G} = (V, E)$ be a graph and $k \in \mathbb{N}$. Then applying the above mentioned linear programming method,* p-#VertexCover *can be solved in time*

$$\mathcal{O}(\|\mathcal{G}\|^c + (2 + 2\sqrt{3})^k \cdot k^3)$$

where c is a constant.

Proof. First we compute $L(\mathcal{G})$ and an optimal half-integral solution $(x_v)_{v \in V}$ of $L(\mathcal{G})$. Let I be the set of isolated vertices of $\mathcal{G}$. By the optimality of $(x_v)_{v \in V}$, we know that $I \subseteq V_0$. Thus define $V_0' := V_0 \setminus I$ and let $\mathcal{D} = (V_0', V_1, F)$ be the bipartite graph that contains all the edges between V_0' and V_1 in $\mathcal{G}$. These computations account for the term $\|\mathcal{G}\|^c$ in the statement of the theorem, as they can be carried out in polynomial time.

Note that $\mathcal{D}$ is not necessarily a crown, but all vertices in V_0' have neighbors only in V_1 and $|V_1| \leq k$.

Furthermore, observe that in for the algorithm CountCrownVC it is not essential that the second graph $\mathcal{C} = (I, N(I), F)$ in the input is a crown. That is, it is not necessary that the graph $\mathcal{C}$ has a perfect matching of cardinality $|I|$. As the graph $\mathcal{D}$ obviously satisfies all conditions for being a crown but not necessarily the perfect matching condition, the instance $(\mathcal{G}, \mathcal{D}, k)$ is a valid input of CountCrownVC. Hence, the time bound derives from this algorithm. □

1.4. A Note on the Crown Rule Reduction

The time bounds we have given for applying the Crown Rule and Linear Programming to p-#VertexCover are surprisingly high. When comparing these

results with the decision problem these techniques were originally designed for this seems even more unfortunate. However, a closer look on the structure of these techniques reveals some details that might help interpreting these results.

In contrast to the Buss Kernelization these techniques consider *minimum* vertex covers, that is, vertex covers of the least possible cardinality. For p-VertexCover this entails no problem, as every minimum vertex cover can easily be extend to a size k vertex cover as long as the graph given contains at least k vertices. However, in solving p-#VertexCover we cannot restrict our attention to *minimum* vertex covers and hence this optimality criterion cannot be exploited.

Conversely, it is easy to see that a refinement of the definition of p-#VertexCover to counting optimal solutions of size at most k would prove the Crown Rule and Linear Programming more efficient than the Buss kernelization again. We might define this problem in the following way:

p-#MinimumVertexCover

Instance: A graph $\mathcal{G} = (V, E)$ and $k \in \mathbb{N}$
Parameter: k
Problem: Compute the number of minimum vertex covers of cardinality at most k in $\mathcal{G}$

It is easy to see that, for example, the Crown Rule is more efficient when applied to this problem than in application to p-#VertexCover. This is due to the fact that for an instance $(\mathcal{G}, k)$, once the crown $\mathcal{C} = (I, N(I), F)$ is computed, it suffices to consider the graph $\mathcal{G}' := \mathcal{G} \setminus \mathcal{C}$, as every minium vertex cover S of $\mathcal{G}$ satisfies $S \cap I = \emptyset$ and $N(I) \subset S$. Then, by the construction of the crown, the graph $\mathcal{G}' = (V', E')$ satisfies $|V'| \leq 3k$ and hence $\|\mathcal{G}\|$ is bounded by some function of k.

Moreover, for p-#MinimumVertexCover these considerations reveal the surprising fact that it is parametrically restricted in the sense that the number of solutions it admits is bounded by some function of k.

Apart from p-VertexCover, the crown rule reduction has been applied to a variety of parameterized decision problems. For example in [MPS04] an application to the p-DisjointTriangle problem has been shown. For the so-called p-SetSplitting problem, kernelizations using Crown Rule reductions can be found in [LS05] and [DFRS04]. For definitions of these problems we refer the reader to the literature.

Unfortunately, a closer look on the algorithms solving these problems reveals that they are not applicable in counting problems. A detailed discussion of this fact is beyond our focus, therefore we will give only a short clue about the reason for this. In the papers mentioned, all the algorithms applying the Crown Rule stop whenever it is clear that there is at least one solution. In counting problems these cases have to be treated differently. This implies that new techniques have to be developed to address these cases. But it might even turn out that - apart from p-#VertexCover - the Crown Rule is not applicable to any counting problem.

Chapter 2.

Exploring #FPT

In this chapter we will examine further parameterized counting problems. It will be shown how kernelization techniques can be applied to find fpt algorithms for these problems. Many of these will exhibit a strong connection to vertex cover which in turn can be used to develop simple fpt algorithms. Other problems that are less correlated to vertex cover will require more effort.

To illustrate the strong connections between vertex cover and certain counting problems it will be convenient to have a notion of "parameterized counting reduction".

Definition 2.1. *Let (F, κ) and (G, λ) be parameterized counting problems over the alphabets Σ and Γ, respectively.*

An fpt parsimonious reduction *from (F, κ) to (G, λ) is a mapping $R : \Sigma^* \to \Gamma^*$ satisfying:*

a) For all $x \in \Sigma^*$ we have $F(x) = G(R(x))$.

b) R is computable by an fpt algorithm (with respect to κ).

c) There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\lambda(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

For parameterized counting problems (F, κ) and (G, λ) we write $(F, \kappa) \leq^{fpt} (G, \lambda)$ if there is an fpt parsimonious reduction from (F, κ) to (G, λ).

We will begin with generalizations of the vertex cover problem to hypergraphs. As hypergraph analogs of vertex cover there are two notions that come to mind. First, one might look for sets C of vertices such that each hyperedge has a most one vertex not in C. This is known as the *cover* problem. Secondly, one might look for sets of vertices that contain at least one vertex in common with every hyperedge. This is the *hitting set* problem.

2.1. p-#Cover

Formally, a *cover* in a hypergraph $\mathcal{H} = (V, E)$ is a set $X \subseteq V$ such that $\forall\, e \in E : |e \setminus X| \leq 1$. Hence, we can define the following parameterized problem of counting covers:

p-#Cover
Instance: A hypergraph $\mathcal{H} = (V, E)$ and $k \in \mathbb{N}$ *Parameter:* k *Problem:* Compute the number of covers of cardinality k in $\mathcal{H}$.

Via the concept of the *Gaifman graph*, solving this problem is easily seen to be tightly connected to solving p-#VertexCover. Given a hypergraph $\mathcal{H} = (V, E)$ the *Gaifman graph* of $\mathcal{H}$ is defined as $\mathcal{G}_\mathcal{H} := (V, E^*)$ with

$$E^* = \{\, \{u, v\} \mid ex.\ e \in E :\ \{u, v\} \subseteq e\}$$

Lemma 2.2. *There is a polynomial time algorithm that witnesses*

$$\text{p-\#Cover} \ \leq^{fpt}\ \text{p-\#VertexCover}$$

Proof. Let $(\mathcal{H}, k)$ with $\mathcal{H} = (V, E)$ be an instance of p-#Cover. Let $\mathcal{G} = (V, E^*)$ be the Gaifman graph of $\mathcal{H}$.

Claim 1. Let $X \subseteq V$. X is a cover in $\mathcal{H}$ iff X is a vertex cover in $\mathcal{G}$.

Proof. Let X be a cover in $\mathcal{H}$ and $f \in E^*$ an edge in $\mathcal{G}$. By definition of $\mathcal{G}$ there is a hyperedge $e \in E$ s.t. $f \subseteq e$ and as X is a cover we have $|e \setminus X| \leq 1$ thus $|f \setminus X| \leq 1$. That is, f is covered by X and X is a vertex cover.
For the backward direction, let X be a vertex cover of $\mathcal{G}$. Note that any hyperedge e with $|e| > 1$ in $\mathcal{H}$ is represented by a size $|e|$ clique $\mathcal{K}$ in $\mathcal{G}$. As $\mathcal{K} = (K, K \times K)$ is covered by X, we have $|K \setminus X| \leq 1$, because otherwise there would be $u, v \in K$ such that $\{u, v\} \subseteq K \setminus X$ in contradiction to X being a vertex cover. Therefore, X is a cover in $\mathcal{H}$. ✓

Claim 1 shows that the number of size k covers in $\mathcal{H}$ equals the number of size k vertex covers in $\mathcal{G}$. Note that, unless there is no cover in $\mathcal{H}$, the cardinality of the hyperedges is bounded by $k + 1$. Thus, we can transform $\mathcal{H}$ into $\mathcal{G}$ in time $\mathcal{O}(|V| + k^2|E|)$ □

Corollary 2.3. *Given a hypergraph $\mathcal{H} = (V, E)$ and $k \in \mathbb{N}$.*
With a uniform cost measure p-#Cover *can be solved in time*

$$\mathcal{O}((1 + \sqrt{3})^k k^2 + |V| + k^2 \cdot |E|)$$

Proof. Given $(\mathcal{H}, k)$ an instance of p-#Cover with $\mathcal{H} = (V, E)$. Let $|V| = n$ and $|E| = m$. If there is an $e \in E$ with $|e| \geq k + 2$ return 0. Otherwise, we compute the Gaifman graph $\mathcal{G} = (V, E^*)$ of $\mathcal{H}$ according to lemma 2.2. By the construction of $\mathcal{G}$ we have $|E| \leq k^2 \cdot m$ and by lemma 1.3 p-#VertexCover on $\mathcal{G}$ can be solved in time $\mathcal{O}((1 + \sqrt{3})^k k^2 + k + n + k^2 \cdot m)$. □

2.2. p-card-#HittingSet

Let us consider the second of the above mentioned generalizations of vertex cover.

Definition 2.4. *A* hitting set *in a hypergraph $\mathcal{H} = (V, E)$ is a set $S \subseteq V$ such that $S \cap e \neq \emptyset$ for all $e \in E$.*

Contrarily to the problems discussed so far, no fpt algorithm is known for the hitting set problem when parameterized only by the cardinality of the hitting sets sought. Indeed, for both the decision and counting problem, there is strong evidence that there is no such algorithm. Therefore, different parameterizations are considered. One straightforward parameterization assumes the hyperedges to be of a small cardinality and hence includes this value into the parameter. This version of the counting problem looks as follows:

p-card-#HITTINGSET
Instance: A hypergraph $\mathcal{H} = (V, E)$ and $k \in \mathbb{N}$ *Parameter:* $k + d$ with $d := max_{e \in E}\|e\|$ *Problem:* Compute the number of hitting sets of cardinality k in $\mathcal{H}$

Note that the convention to include several parameter values into one parameter by summation (as above for $k+d$) was introduced only for simplicity (cf. [FG06]). Intuitively, however, we may assume that this problem has two parameters. We define $\#hs(\mathcal{H}, k)$ to denote the number of hitting sets of size k in the hypergraph $\mathcal{H}$.

One element of the algorithm that solves this problem with kernelization techniques is an fpt algorithm which will be combined with the kernelization. We cite its existence without proof.

Theorem 2.5 (Theorem 14.3 from [FG06]).
p-*card-#*HITTINGSET *is fixed parameter tractable. More precisely, there is an algorithm solving* p-*card-#*HITTINGSET *in time*

$$\mathcal{O}(d^{2k} \cdot \|\mathcal{H}\|).$$

The pseudocode of the algorithm is presented in algorithm 4.

The kernelization of the decision problem p-card-HITTINGSET given in [FG06] utilizes the well known *sunflower lemma*. It is a little bit more complicated than the Buss kernelization of vertex cover but intuitively, one can view it as a generalization of Buss' idea. We will see that it can be applied to the counting problem without alteration.

Given a hypergraph $\mathcal{H} = (V, F)$. A *sunflower* in $\mathcal{H}$ is a family $\mathbf{S} = \{S_1, \ldots, S_k\} \subseteq F$ of hyperedges such that there is a set $C \subseteq V$ satisfying

$$S_i \cap S_j = C \qquad \text{for all } 1 \leq i < j \leq k \tag{2.1}$$

```
   CountGHS(H, k, F) // H a hypergraph, k ∈ ℕ, F ⊆ V
 1 if k > |V \ F| then return 0;
 2 else if E = ∅ then return (|V\F| choose k);
 3 else
 4     choose e ∈ E;
 5     h ← 0 ;
 6     forall S_0 ⊆ e \ F with 0 < |S_0| ≤ k do
 7         V' ← V \ S_0 ;
 8         E' ← {e ∈ E | e ⊆ V'};
 9         H' ← (V', E');
10         F' ← F ∪ (e \ S_0);
11         k' ← k − |S_0| ;
12         h ← h + CountGHS(H', k', F');
13     end
14     return h;
15 end
```

Algorithm 4: Counting generalized hitting sets

C is the *core* of the sunflower $\mathbf{S}$ and for all $i \in [k]$ the set $S_i \setminus C$ is called a *petal* if it is nonempty.

Lemma 2.6 (Sunflower Lemma). *Let $k, d \in \mathbb{N}$ and be $\mathcal{H} = (V, F)$ a d-uniform hypergraph with more than $(k-1)^d \cdot d!$ hyperedges. Then there is a sunflower S of cardinality k in $\mathcal{H}$.*

Proof. We prove this by induction on d. For $d = 1$ each collection of k distinct 1-element hyperedges clearly forms a sunflower with empty core. For $d > 1$ let $D = \{f_1, \ldots, f_m\} \subseteq F$ be a maximal family of pairwise disjoint hyperedges. If $|D| \geq k$, then D is a sunflower, as desired. Otherwise, let $W := f_1 \cup \ldots \cup f_m$. Then $|W| \leq (k-1) \cdot d$ and by the maximality of D, every hyperedge $f \in F$ satisfies $W \cap f \neq \emptyset$. Thus there is an element $x \in W$ contained in at least

$$\frac{|F|}{|W|} \geq \frac{(k-1)^d \cdot d!}{(k-1)d} = (k-1)^{d-1} \cdot (d-1)!$$

hyperedges in F. For such an $x \in W$ let $F_x := \{f \setminus \{x\} \mid f \in F \wedge x \in f\}$. Then, by the induction hypothesis, F_x contains a sunflower $\{f'_1, \ldots, f'_k\}$ of cardinality k and thus $\{f'_1 \cup \{x\}, \ldots, f'_k \cup \{x\}\}$ is a sunflower in $\mathcal{H}$. □

Corollary 2.7. *There is an algorithm that computes a sunflower S according to the sunflower lemma in time $\mathcal{O}(d\|\mathcal{H}\|)$.*

Proof. The proof of the sunflower lemma can be turned into an algorithm (see algorithm 5) that computes sunflowers. Its correctness follows directly from the sunflower lemma.

We prove the time bound. Note that the family D and the set W can be computed by greedily searching all hyperedges in a fixed order. For each $v \in V$ a flag is maintained indicating whether v is contained in a hyperedge in D. Then for every edge $e \in F$ we can decide in time $\mathcal{O}(d)$ whether e can be added to D. As initializing the array of flags takes $|V|$ steps and the set W can be computed by a single pass through the array, the first three lines of the algorithm take time $\mathcal{O}(|V| + d|F|)$.

Obviously, checking the cardinality of D does not increase this time bound. Finding $w \in W$ as in line 5 can be done within the same time bound by keeping a counter for each vertex in V. In a single pass through all hyperedges the counters for a $v \in V$ are increased everytime v occurs. Then all counters of vertices not in W are set to zero and in a second pass a vertex with a highest counter value is chosen. This can be done in time at most $\mathcal{O}(|V| + d|F|)$.

Similarly, F' and V' can be computed within the same time bound and as the cardinality of S is at most $|F|$ the family S' can be computed in at most $d \cdot |F|$ steps.

This yields a time bound of $\mathcal{O}(|V|+d|F|)$ for one recursive call, and via a straightforward inductive argument one can see that the overall algorithm takes time $\mathcal{O}(d\cdot(|V|+d|F|))$ which, by the definition of $\|H\|$ is identical to the time claimed.□

FindSunflower($\mathcal{H}, k$)
// $\mathcal{H} = (V, F)$ a d-uniform hypergraph, $k \in \mathbb{N}$
Output: a pair (S, C) where S is a sunflower and C its core
1 **if** $|F| \leq (k-1)^d \cdot d!$ **then return** $(\emptyset, \emptyset)$;
2 Greedily find a maximal family $D \subseteq F$ of pairwise disjoint hyperedges;
3 $W \leftarrow \bigcup_{A \in D} A$;
4 **if** $|D| > k$ **then return** D;
5 Pick a $w \in W$ that occurs in a maximum number of hyperedges;
6 $F' \leftarrow \{e \setminus \{w\} \,|e \in F, w \in e\}$;
7 $V' \leftarrow \bigcup_{e \in F} e$;
8 $(S, C) \leftarrow$ FindSunflower$((V', F'), k)$;
9 $S' \leftarrow \{e \cup \{w\} \mid e \in S\}$;
10 **return** $(S', C \cup \{w\})$;

Algorithm 5: Finding sunflowers in a hypergraph

The algorithm FINDSUNFLOWER for finding sunflowers and the algorithm COUNTGHS are now combined to form the kernelization algorithm COUNTSUNFLOWERHS.

Theorem 2.8. *Given a hypergraph $\mathcal{H} = (V, E)$, $d := max_{e \in E}|e|$ and $k \in \mathbb{N}$. The algorithm* COUNTSUNFLOWERHS *solves* p*-card-#*HITTINGSET *on $\mathcal{H}$ in time*

$$\mathcal{O}(d^2 \cdot |E| \cdot \|\mathcal{H}\| + k^d \cdot d! \cdot d^{2k+2})$$

```
   CountSunflowerHS(H, k) // H = (V, E) a hypergraph, k ∈ ℕ
 1 d ← max_{e∈E}|e|;
   // Begin kernelization
 2 for i ← 1 to d do
 3     E_i ← {e ∈ E : |e| = i };
 4     V_i ← ⋃_{e∈E_i} e;
 5     while FindSunflower((V_i, E_i), k + 1) ≠ (∅, ∅) do
 6         Let (S, C) be the sunflower returned by FindSunflower ;
 7         E_i ← (E_i \ S) ∪ {C};
 8         V_i ← (V_i \ ⋃_{X∈S} X) ∪ C;
 9     end
10 end
   // End kernelization
11 V' ← V_1 ∪ V_2 ∪ ... ∪ V_d;
12 E' ← E_1 ∪ E_2 ∪ ... ∪ E_d;
13 I ← V \ V';
14 H' ← (V', E') ;
15 return Σ_{i=0}^{k} CountGHS(H', i, ∅) · (|I| choose k−i);
```

Algorithm 6: Counting hitting sets by the sunflower kernelization

Proof. Let $\mathcal{H}$, d and k be defined as above.

To show the correctness of the algorithm we prove the following claim.

Claim 1. Let S with $|S| = k+1$ be a sunflower in $\mathcal{H}$ with core C. Define $\mathcal{H}' = (V, E')$ with

$$E' := E \setminus S \cup C$$

then $\#hs(\mathcal{H}, k) = \#hs(\mathcal{H}', k)$.

Proof. The definition of a sunflower implies that, given a sunflower S with $k+1$ petals, every cardinality k hitting set in $\mathcal{H}$ must have a nonempty intersection with the core of S. From this observation one can easily infer that every size k hitting set in $\mathcal{H}$ is a hitting set in $\mathcal{H}'$. For the backward direction note that every size k hitting set X in $\mathcal{H}'$ satisfies $C \cap X \neq \emptyset$ and therefore, by the definition of $\mathcal{H}'$, X is a hitting set of $\mathcal{H}$, as well. ✓

This claim shows that the kernelization phase of the algorithm (that is the **for**-loop) is correct and that with I and $\mathcal{H}' = (V', E')$ defined as in line 13 and 14 of the algorithm we have

$$\#hs(\mathcal{H}, k) = \#hs((V' \cup I, E'), k)$$

For the correctness of the computation in the last line of the algorithm, note that I is the set of all isolated vertices in $\mathcal{H}'$ and the computation is essentially the same as in the CountBussVC algorithm on page 17.

Time Complexity. Consider the kernelization phase of the algorithm. Note that, for each $i \in [d]$ the E_i and V_i before the **while**-loop can be computed in time $\mathcal{O}(\|\mathcal{H}\|)$. Furthermore, FindSunflower is called at most $\mathcal{O}(|E|)$ times and the sets V_i and E_i in lines 7 and 8 can be computed in time $\mathcal{O}(\|\mathcal{H}\|)$.

For the whole kernelization phase this implies a time bound of

$$\mathcal{O}(d \cdot (\|\mathcal{H}\| + |E|(d\|\mathcal{H}\| + \|\mathcal{H}\|))) \quad \leq \quad \mathcal{O}(d^2 \cdot |E| \cdot \|\mathcal{H}\|)$$

The computation of $\mathcal{H}'$ and I can be completed within the same time. For the last line of the algorithm, note that for the hypergraph $\mathcal{H}' = (V', E')$ the sunflower lemma implies $|E'| \leq k^d \cdot d! \cdot d$ and $|V'| \leq k^d \cdot d! \cdot d^2$. Thus we have $\|\mathcal{H}'\| \leq 2 \cdot k^d \cdot d! \cdot d^2$.

The last line of the algorithm induces at most k additions in the sum. By theorem 2.5, the running time of $\textsc{CountGHS}(G', i, \emptyset)$ is at most $\mathcal{O}(d^{2i} \cdot \|\mathcal{H}'\|)$. As before, the binomial coefficient can be computed in time $\mathcal{O}(k)$. In the following we omit constants hidden in the "big Oh" notation of the terms considered. Then, computing the value

$$\sum_{i=1}^{k'} \textsc{CountGHS}(G', i, \emptyset) \cdot \binom{|I|}{k' - i} \tag{2.2}$$

takes time

$$\begin{aligned} k + \sum_{i=0}^{k} (d^{2i} \cdot \|\mathcal{H}'\| + k) \quad &\leq \quad k + k \cdot (k+1) + k^d \cdot d! \cdot d^2 \cdot \sum_{i=0}^{k} d^{2i} \\ &= \quad k + k \cdot (k+1) + k^d \cdot d! \cdot d^2 \cdot \frac{d^{2k+2} - 1}{d^2 - 1} \\ &\in \quad \mathcal{O}(k^d \cdot d! \cdot d^{2k+2}) \end{aligned}$$

Thus, we get an overall time bound of $\mathcal{O}(d^2 \cdot |E| \cdot \|\mathcal{H}\| + k^d \cdot d! \cdot d^{2k+2})$. □

2.3. p-#UniqueHittingSet

Among the problems which we consider in this chapter the following problem and its solution exhibit the weakest similarity to the vertex cover problem. The decision problem corresponding to p-#UniqueHittingSet is described in [DF99] (see exercise 3.2.5).

p-#UniqueHittingSet

Instance: A hypergraph $\mathcal{H} = (V, E)$
Parameter: $|E|$
Problem: Compute the number of *unique* hitting sets sets in $\mathcal{H}$ that is all sets $X \subseteq V$ satisfying $\forall\, e \in E : |e \cap X| = 1$

Note that, in comparison to many other parameterized problems, the problem at hand does not include the cardinality of the sets under consideration into the parameter but it does so for the number of hyperedges in the instance.

The reason that an fpt algorithm is known for this problem is tightly connected to the property that the hitting sets have to be *unique*. This means that in addition to being a hitting set, a unique hitting set has to have *exactly* one vertex in common with each hyperedge. We will see the consequences this constraint has for the design of an fpt algorithm.

As with the previous problems we will outline the well known solution of the decision problem and show how it can be used to design an efficient algorithm for the counting problem.

Let $\mathcal{H} = (V, E)$ be a p-#UniqueHittingSet instance. Let $k := |E|$. For all $x, y \in V$ we define an equivalency relation by

$$x \sim y =_{def} \forall e \in E : \ x \in e \Leftrightarrow y \in e.$$

Clearly, this relation defines l nonempty equivalency classes for an $l \leq 2^k$. Let A_0 be the equivalency class of all isolated vertices and define $\tilde{\mathbf{A}} := \{A_1, \ldots, A_{l-1}\}$ as the family of all nonempty equivalency classes, except for A_0.

Furthermore, we define $\tilde{\mathcal{H}} := (\tilde{\mathbf{A}}, \tilde{E})$. This hypergraph will play the part of the kernel in our algorithm. Its set of hyperedges $\tilde{E}$ represents the hyperedges in E with respect to the equivalency classes in $\tilde{\mathbf{A}}$. To make this precise, we define two functions. Let $h : V \to \tilde{\mathbf{A}}$ be defined by $h(v) := A \in \tilde{\mathbf{A}}$ s.t. $v \in A$, i.e. we map vertices to their corresponding equivalency classes. A second function $f : 2^V \to 2^{\tilde{\mathbf{A}}}$ defined by $f(\{v_1, \ldots, v_b\}) := \{h(v_1), \ldots, h(v_b)\}$ does so analogously for sets of vertices. Note that h (and f) are undefined for isolated vertices (and sets containing them, respectively). The definition of $\tilde{E}$ is easy now:

$$\tilde{E} := \{f(e) \mid e \in E\}.$$

The set A_0 the family $\tilde{\mathbf{A}}$ and the hypergraphs $\mathcal{H}$ and $\tilde{\mathcal{H}}$ will play a central part throughout the whole section.

We will construct an algorithm that applies the following theorem.

Theorem 2.9. *Let $\mathcal{H}, \tilde{\mathcal{H}}$ and $\tilde{\mathbf{A}}$ be defined as above.*
Let $\tilde{\mathbf{U}}$ be the set of all unique hitting sets in $\tilde{\mathcal{H}}$, then the number $\#uhs(\mathcal{H})$ of unique hitting sets in $\mathcal{H}$ satisfies:

$$\#uhs(\mathcal{H}) = \sum_{\tilde{S} \in \tilde{\mathbf{U}}} 2^{|A_0|} \prod_{A \in \tilde{S}} |A| \tag{2.3}$$

Before we prove theorem 2.9, we will prove some facts that will be helpful in the actual proof.

Lemma 2.10. *(1) Let $A \in \tilde{\mathbf{A}}$. For every unique hitting set S of $\mathcal{H}$ we have $|A \cap S| \leq 1$.*

(2) Let $S \subseteq V \setminus A_0$ be a unique hitting set in $\mathcal{H}$. Then $\tilde{S} := f(S)$ is a unique hitting set in $\tilde{\mathcal{H}}$. Furthermore $|S| = |\tilde{S}|$.

(3) Let $\tilde{S}$ be a unique hitting set in $\tilde{\mathcal{H}}$. Then every set $S \in V \setminus A_0$ satisfying $f(S) = \tilde{S}$ is a unique hitting set in $\mathcal{H}$ if, and only if, $|S| = |\tilde{S}|$.

Proof. Recall that A_0 consists of all isolated vertices in $\mathcal{H}$.

To show (1), assume $A \cap S \neq \emptyset$ and, for contradiction, assume that $|A \cap S| \geq 2$. That is, there are distinct $w, v \in A \cap S$. As $A_0 \notin \tilde{\mathbf{A}}$, we know that there is a hyperedge $e \in E$ with $\{v, w\} \subseteq e$. Thus, $|e \cap S| \geq 2$ in contradiction to S being a unique hitting set.

For (2), note that (by (1)) any two distinct $v, w \in S$ belong to different equivalency classes. This implies $|S| = |\tilde{S}|$. Furthermore, to see that $\tilde{S}$ is a hitting set consider a hyperedge $f(e) \in \tilde{E}$. As $e \cap S = \{v\}$ for some $v \in V$, we have $h(v) \in f(e)$ by definition of f. To see that $\tilde{S}$ is a *unique* hitting set assume that $\tilde{S}$ contains an equivalency class $h(w) \neq h(v)$ with $h(w) \in f(e)$. $h(w) \neq h(v)$ implies $v \neq w$ and, by the construction of $\tilde{\mathcal{H}}$ we have $w \in e$. Contradiction.

Now, it remains to prove (3). For the forward direction, note that, by the definition of f, $|S| < |\tilde{S}|$ is not possible. Therefore assume $|S| > |\tilde{S}|$. This implies, that there are at least two distinct $v, w \in S$ and an $A \in \tilde{\mathbf{A}}$ such that $v, w \in A$. Thus there is an edge $e \in E$ with $\{v, w\} \subseteq S \cap e$ contradicting the uniqueness condition of S.

For the backward direction, let $S \in V \setminus A_0$ be a set satisfying $f(S) = \tilde{S}$ and $|S| = |\tilde{S}|$. Consider a hyperedge $e \in E$ in $\mathcal{H}$. We have to show that $|S \cap e| = 1$. To show $|S \cap e| > 0$, note that $f(e)$ satisfies $f(e) \cap A \neq \emptyset$ for some $A \in \tilde{S}$ in $\tilde{\mathcal{H}}$. As $f(S) = \tilde{S}$ there is a $v \in S$ such that $v \in A$ and therefore $v \in e$.

For $|S \cap e| \leq 1$, assume, for contradiction, $S \cap e \supseteq \{v, w\}$ for distinct v and w. We know that, $\{h(v), h(w)\} \subseteq \tilde{S}$ and as $|S| = |\tilde{S}|$, v and w belong to different equivalency classes, that is $h(v) \neq h(w)$. Moreover, by the definition of $\tilde{\mathcal{H}}$, $\{h(v), h(w)\} \subseteq f(e)$ contradicting the unique hitting set property of $\tilde{S}$. □

Now, we are ready to prove the theorem.

Proof (of Theorem 2.9). Note that any hitting set S in $\mathcal{H}$ can be partitioned into a set S_0 consisting of all isolated vertices in S and another set $S_1 := S \setminus S_0$. Conversely, for S_1 there are $2^{|A_0|}$ hitting sets in $\mathcal{H}$ that contain S_1. Therefore, we can restrict our attention to the non-isolated vertices. That is, we define $\mathcal{H}' := (V \setminus A_0, E)$ and we know that $\#uhs(\mathcal{H}) = \#uhs(\mathcal{H}') \cdot 2^{|A_0|}$. Hence, we still have to show that

$$\#uhs(\mathcal{H}') = \sum_{\tilde{S} \in \tilde{\mathbf{U}}} \prod_{A \in \tilde{S}} |A| \tag{2.4}$$

Note that, by lemma 2.10 (1) and (3), it is easy to see, that for a unique hitting set $\tilde{S} = \{A_{i_1}, \ldots, A_{i_m}\}$ in $\tilde{\mathcal{H}}$ there are exactly

$$\prod_{j=1}^{m} |A_{i_j}| \tag{2.5}$$

unique hitting sets S in $\mathcal{H}'$ with $f(S) = \tilde{S}$.

To see "$\geq$" (in (2.4)), note that by lemma 2.10 (1) and (2) every unique hitting set S in $\mathcal{H}'$ uniquely identifies its corresponding hitting set $\tilde{S}$ in $\tilde{\mathcal{H}}$. Therefore, S is counted only once in the right hand side expression.

To see "$\leq$", note that by lemma 2.10 (2) every unique hitting set S in $\mathcal{H}$ is represented by some $\tilde{S} \in \tilde{\mathbf{U}}$, thus this direction follows by (2.5). □

With this theorem, we can develop a very simple brute force algorithm, that solves the problem in $\tilde{\mathcal{H}}$ and performs the necessary computations according to equation (2.4).

To estimate the running time of the algorithm, we have to bound the size of the kernel $\tilde{\mathcal{H}} := (\tilde{\mathbf{A}}, \tilde{E})$. Obviously, by the construction of $\tilde{H}$ we have $|\tilde{E}| = |E| = k$ and $|\tilde{\mathbf{A}}| \leq 2^k$.

Theorem 2.11. *Given a hypergraph $\mathcal{H} = (V, E)$ with $|E| = k$. There is an algorithm that solves* p-#UniqueHittingSet *on $\mathcal{H}$ in time*

$$\mathcal{O}(\|\mathcal{H}\| + k^3 \cdot 2^{k+k^2})$$

Proof. Given $\mathcal{H}$ as above and let $|V| = n$, we assume that the hyperedges are ordered, to wit, $E = \{e_1, \ldots, e_k\}$. First, the algorithm computes the equivalency classes and their cardinalities. To achieve this, each vertex $v \in V$ is assigned a k-bit number a_v set to zero. Then for each hyperedge e_i the $(i-1)$-th bit entry of every vertex occurring in e_i is set to 1. This can be done in time $\mathcal{O}(k \cdot n)$.

Then each nonempty equivalency class A_i is identified by an integer $i \in [2^k]$. and the cardinality of each class can be determined by checking the numbers a_v for all vertices $v \in V$. As there are at most n different nonempty equivalency classes, this can be done in time $\mathcal{O}(n + k \cdot n)$.

We define $\mathcal{H}' = (V \setminus A_0, E)$, i.e. $\mathcal{H}'$ is the hypergraph obtained from $\mathcal{H}$ by deleting all isolated vertices.

Furthermore $\tilde{\mathcal{H}} = (\tilde{V}, \tilde{E})$ is constructed by setting $\tilde{V} := \{i \mid A_i \neq \emptyset,\ A_i \in \tilde{\mathbf{A}}\}$. As we have $|\tilde{V}| \leq n$ we know that $\tilde{V}$ can be constructed in polynomial time. Note that the representative for A_0 is excluded from this set, that is $0 \notin \tilde{V}$. $\tilde{E}$ can be obtained from E by substituting in each hyperedge e_i each occurrence of a vertex $v \in V$ with a_v and deleting multiple occurrences of these numbers. This does not increase the time bounds given so far.

Equation (2.4) shows how to compute $\#uhs(\mathcal{H}')$ by means of computing $\#uhs(\tilde{\mathcal{H}})$. In a last step $\#uhs(\mathcal{H})$ is determined by evaluating the product $2^{|A_0|} \cdot \#uhs(\mathcal{H}')$.

Note that, as $|E| = |\tilde{E}| = k$ a unique hitting set in $\mathcal{H}'$ (and likewise in $\tilde{\mathcal{H}}$), can be of cardinality at most k. This is due to the fact that every set of at least $k+1$ unisolated vertices contains at least two vertices that occur together in a hyperedge.

Enumeration phase. The actual work of the algorithm is done by enumerating subsets $\tilde{S}$ of $\tilde{V}$ of cardinality at most k. Then, the algorithm determines whether $\tilde{S}$ is a unique hitting set in $\tilde{\mathcal{H}}$ and the value according to equation (2.5) is computed. Eventually, all values for the unique hitting sets found are summed in a number u and the product $2^{|A_0|} \cdot u$ is returned.

By the fact that $|\tilde{V}| \leq 2^k$, for an $l \leq k$, at most $\binom{2^k}{l}$ sets have to be enumerated. Let S be a set of cardinality $l \leq k$. Determining whether S is a unique hitting set takes at most $k \cdot 2^k \cdot l$ steps. Computing the value according to equation (2.5) and adding it to the number of solutions found so far takes l steps.

Hence, for the enumeration phase, we obtain the following bound:

$$\begin{aligned}
\sum_{l=0}^{k} \binom{2^k}{l} \cdot (k \cdot 2^k \cdot l + l) &= (k \cdot 2^k + 1) \cdot \sum_{l=0}^{k} \binom{2^k}{l} \cdot l \\
&\leq (k \cdot 2^k + 1) \cdot k \cdot \sum_{l=0}^{k} \binom{2^k}{l} \\
&\leq (k \cdot 2^k + 1) \cdot k \cdot k \cdot (2^k)^k \\
&\leq (k \cdot 2^k + 1) \cdot k^2 \cdot 2^{k^2} \\
&\in \mathcal{O}(k^3 \cdot 2^{k+k^2})
\end{aligned}$$

Note that the estimate above is derived by using the fact that $\binom{n}{k} \leq n^k$.

Therefore, together with the time for the computation of the hypergraph $\tilde{\mathcal{H}}$ the algorithm completes in time $\mathcal{O}(k \cdot n + k^3 \cdot 2^{k+k^2})$ □

2.3.1. Applying the Logarithmic Cost Measure Once More

Recall our analysis of the CountBussVC algorithm using the logarithmic cost measure. The bound on the running time obtained there suggests that the part of the algorithm that has a running time exponential in k does not depend on n.

To refute the supposition that this might be the case for all kernelizations of parameterized counting problems we will give another analysis in application of the logarithmic cost measure. We will refine the analysis of the algorithm given in the proof of theorem 2.11.

Lemma 2.12. *Given a hypergraph $\mathcal{H} = (V, E)$ with $|E| = k$ and $|V| = n$. There is an algorithm that solves* p-#UniqueHittingSet *on $\mathcal{H}$ in time*

$$\mathcal{O}(\|\mathcal{H}\| + n \log n + k^3 \cdot 2^{k+k^2} + \log n \cdot \log\log n \cdot k^3 \cdot 2^{k^2})$$

if the logarithmic cost measure is applied.

Proof. Let $\mathcal{H} = (V, E)$ be a hypergraph with $|E| = k$ and define $n := |V|$. Consider the proof of theorem 2.11. Note that the impact of the logarithmic cost measure as opposed to the uniform cost measure comes into play mainly in the computation of the value according to equation (2.5):

$$\prod_{j=1}^{m} |A_{i_j}|.$$

Furthermore, let u be the variable which, in each step of the algorithm, contains the number of unique hitting sets found so far.

Recall that A_0 is the set of all isolated vertices in $\mathcal{H}$ and for $\mathcal{H}' := (V \setminus A_0, E)$ we have $\#uhs(\mathcal{H}) = 2^{|A_0|} \cdot \#uhs(\mathcal{H}')$. Therefore, after computing the value $\#uhs(\mathcal{H}')$ we can compute the product by a single computation. This has the advantage that the numbers in the computation of $\#uhs(\mathcal{H}')$ are bounded. Hence, we can derive an upper bound on this number

$$\#uhs(\mathcal{H}') \leq \sum_{i=0}^{k} \binom{n}{i} \leq k \cdot n^k$$

The inequality holds, by the fact that $\mathcal{H}'$ contains no isolated vertices which implies that only sets of vertices of cardinality at most k may be unique hitting sets.

Consider the computation of $\#uhs(\mathcal{H}')$. Recall that this value is computed by means of $\tilde{\mathcal{H}} = (\tilde{V}, \tilde{E})$ and that $|\tilde{V}| \leq 2^k$. The algorithm enumerates all sets $S \subseteq \tilde{V}$ of cardinality at most k. For a set of cardinality $l \leq k$ determining whether it is a unique hitting set still takes $k \cdot 2^k \cdot l$ steps.

Note that for all equivalency classes $A_i \in \tilde{\mathbf{A}}$ we have $|A_i| \in \mathcal{O}(n)$ which implies that they are represented by numbers of at most $\log n$ bits. Furthermore the product of two x-bit numbers can be represented by at most $2x$ bits. Likewise for the product of l x-bit numbers, we have at most $l \cdot x$ bits. Hence, computing the value according to equation (2.5) can be done in time at most $l \cdot t_{mul}(l \cdot \log n)$ (where t_{mul} denotes the time of multiplying two numbers according to fact 0.7).

As $\#uhs(\mathcal{H}') \leq k \cdot n^k$, the number u can be represented by at most $\log k + k \log n \in \mathcal{O}(k \log n)$ bits. Thus, adding values to u takes time at most $\mathcal{O}(k \log n)$. In the following we omit constants hidden in the "bigOh" notation. For the enumeration phase, we obtain the new time bound:

$$\begin{aligned}
& \sum_{l=0}^{k} \binom{2^k}{l} \cdot (k \cdot 2^k \cdot l + l \cdot t_{mul}(l \cdot \log n) + k \log n) \\
\leq\ & (k^2 \cdot 2^k + k \cdot t_{mul}(k \cdot \log n) + k \log n) \cdot \sum_{l=0}^{k} \binom{2^k}{l} \\
\leq\ & (k^2 \cdot 2^k + k \cdot t_{mul}(k \cdot \log n) + k \log n) \cdot k \cdot (2^k)^k \\
\leq\ & (k^2 \cdot 2^k + k^2 \cdot \log n \cdot \log(k \cdot \log n) + k \log n) \cdot k \cdot 2^{k^2} \\
=\ & (k^2 \cdot 2^k + k^2 \cdot \log n \cdot \log k + k^2 \cdot \log n \cdot \log\log n + k \log n) \cdot k \cdot 2^{k^2} \\
\in\ & \mathcal{O}(k^3 \cdot 2^{k+k^2} + \log n \cdot \log\log n \cdot k^3 \cdot 2^{k^2})
\end{aligned}$$

Note that the product $2^{|A_0|} \cdot \#uhs(\mathcal{H}')$ involves an $\mathcal{O}(n)$ bit and an $\mathcal{O}(k \log n)$ bit number and can be computed in time $\mathcal{O}(n \log n)$.

Hence the whole algorithm completes in time

$$\mathcal{O}(k \cdot n + n \log n + k^3 \cdot 2^{k+k^2} + \log n \cdot \log\log n \cdot k^3 \cdot 2^{k^2})$$

as claimed. □

2.4. p-#NearlyAPartition

The following problem has a kernelization that is based on a partial reduction to vertex cover. This means that the problem exhibits a structure similar to that of the p-#VertexCover problem and therefore some of the facts known about vertex covers can be exploited to design an algorithm for the problem at hand. We will see another application of this technique in the next section.

p-#NearlyAPartition

Instance: A hypergraph $\mathcal{H} = (V, E)$ and $k \in \mathbb{N}$
Parameter: k
Problem: Compute the number of sets $E' \subseteq E$ with $|E'| = k$ such that $E \setminus E'$ is a partition of V

Call the set E' in the definition above a *witness*. Given an instance $(\mathcal{H}, K)$ with $\mathcal{H} = (V, E)$ we compute a graph $\mathcal{G} = (E, F)$ with

$$F := \{\{e_1, e_2\} \mid e_1, e_2 \in E,\ e_1 \cap e_2 \neq \emptyset\}.$$

Lemma 2.13. *If in $\mathcal{H}$ there is a $E' \subseteq E$ with $|E'| = k$ such that $E \setminus E'$ is a partition of V then E' is a size k vertex cover in $\mathcal{G}$.*

Proof. The proof is immediate, because if $E \setminus E'$ is a partition of V, then $E \setminus E'$ consists exclusively of pairwise disjoint hyperedges and thus by the definition of $\mathcal{G}$, E' covers all edges in $\mathcal{G}$. □

Note that we can exclude isolated hyperedges from our consideration without changing the number of solutions. The lemma above allows us to apply Buss' kernelization to our problem: As every vertex e in $\mathcal{G}$ corresponds to a hyperedge e in $\mathcal{H}$, we have that if $d_{\mathcal{G}}(e) > k$ then e intersects more than k hyperedges in $\mathcal{H}$ then any witness E' of cardinality k necessarily contains e. Furthermore, if all $e \in E$ satisfy $d_{\mathcal{G}}(e) \leq k$ and $|F| > k^2$ then, by lemma 2.13, $\mathcal{H}$ has no witness.

Lemma 2.14. *Given a hypergraph $\mathcal{H} = (V, E)$ and $k \in \mathbb{N}$. There is an algorithm that solves* p-#NearlyAPartition *in time*

$$\mathcal{O}(\|\mathcal{H}\|^2 + (1 + \sqrt{3})^k k^2)$$

Proof. Note that any algorithm solving p-#VertexCover can be easily adapted to solve p-#NearlyAPartition within almost the same time bounds. This is true by the considerations above since CountBussVC can be adapted such that every solution is checked if it forms a partition of V. Thus the adapted algorithm completes within the time bounds of algorithm 2.

Furthermore, there is an overhead of $\mathcal{O}(\|\mathcal{H}\|^2)$ in preprocessing for computing from the input hypergraph $\mathcal{H}$ the graph $\mathcal{G}$ that forms the input of the vertex cover algorithm. The Buss kernelization does not increase this time bound. □

2.5. p-#BipartiteEdgeDomination

The last counting problem which we will show a kernelization for is the so-called Matrix Domination problem. The decision problem can be found in [DF99] and its corresponding counting problem is defined as follows:

p-#MatrixDomination

Instance: An $n \times n$ Matrix M with entries in $\{0, 1\}$ and $k \in \mathbb{N}$
Parameter: k
Problem: Compute the number sets C of exactly k nonzero entries that dominate all others, in the sense that each nonzero entry of M is in the same row or column as an entry from C

One well known solution of the decision version of this problem rephrases it as a problem on bipartite graphs. We will follow this approach and develop a kernelization that is appropriate for counting.

It is easy to see that an $n \times n$ matrix M as above can be regarded as a bipartite graph $\mathcal{B}(M) = (U, W, E)$ in such a sense that for each row and column a distinct vertex is introduced such that $U = \{r_1, \ldots, r_n\}$ represents all rows and $W = \{c_1, \ldots, c_n\}$ represents all columns in M. Then for each nonzero entry a_{ij} in M an edge $\{r_i, c_j\}$ is constructed in E:

$$E := \{\{r_i, c_j\} \mid a_{ij} = 1,\ a_{ij} \in M\}.$$

Then, two nonzero entries $a, b \in M$ occurring in the same row or column are represented in $\mathcal{B}(M)$ by two edges $e_a, e_b \in E$ that have a vertex in common (i.e. $e_a \cap e_b \neq \emptyset$). In this case, we say that the edges e_a and e_b *dominate* each other. Accordingly, for a set of edges $S \subset E$ we say that S *dominates* an edge $e \in E$ if there is an edge $f \in S$ such that $f \cap e \neq \emptyset$. If S dominates all edges in a graph $\mathcal{G}$ then S is called an *edge dominating set* of $\mathcal{G}$.

This implies that there is a one-to-one correspondence between solutions of the Matrix Domination problem in M and edge dominating sets in $\mathcal{B}(M)$. Therefore, we can solve p-#MatrixDomination by solving the following problem:

p-#BipartiteEdgeDomination

Instance: A bipartite graph $\mathcal{B} = (U, W, E)$ and $k \in \mathbb{N}$
Parameter: k
Problem: Compute the number of sets $S \subseteq E$ with $|S| = k$ s.t. S dominates all edges in $\mathcal{B}$

We use the technique of partially reducing the problem to vertex cover again.

Let $\mathcal{B} = (U, W, E)$ be a bipartite graph and $k \in \mathbb{N}$. By $V := U \cup W$ we denote the set of *all* vertices in $\mathcal{B}$. For a set $X \subseteq V$ of vertices define $X_U := X \cap U$ and analogously $X_W := X \cap W$.

Note that isolated vertices in $\mathcal{B}$ don't play any part in edge dominating sets. Hence, we can assume w.l.o.g. that in $\mathcal{B}$ each vertex occurs in at least one edge.

Edge dominating sets and vertex covers are correlated. Let $C \subseteq E$ be an edge dominating set of cardinality k in $\mathcal{B}$. Then the set $S = \bigcup C$ is a vertex cover in $\mathcal{B}$ with $|C| \leq 2k$. Thus, we could apply Buss' lemma to our problem but as $\mathcal{B}$ is bipartite, we can do even better, as is shown by the following claim.

Claim 1. Let $v \in V$ be a vertex with $d_{\mathcal{B}}(v) > k$. Then for every edge dominating set C of cardinality at most k in $\mathcal{B}$ there is an edge $e \in C$ such that $v \in e$.

Proof. Suppose that there is an edge dominating set C with $|C| \leq k$ such that $\forall\, e \in C : \; v \notin e$. Then, let $e_1, \ldots e_{k+1}$ be $k+1$ distinct edges incident with v. That is, for all $i \in [k+1] \quad e_i$ has the form $e_i = \{v, v_i\}$. Thus, as C dominates all edges e_i, and as $\mathcal{B}$ is bipartite, there has to be, for each e_i a distinct $f_i \in C$ such that $e_i \cap f_i \neq \emptyset$. Thus $|C| \geq k+1$ in contradiction to our assumption. ✓

Motivated by this claim, we form the set $R := \{v \in V \mid d_{\mathcal{B}}(v) > k\}$. By the above mentioned connection to vertex covers it is easy to see that if $|R| > 2k$ then there is no edge dominating set of cardinality k in $\mathcal{B}$. As $\mathcal{B}$ is bipartite, we can describe this upper bound on $|R|$ in more detail:

Claim 2. If $|R_U| > k$ or $|R_W| > k$ then there is no edge dominating set of cardinality k in $\mathcal{B}$.

Proof. Assume that $|R_U| > k$ (the case $|R_W| > k$ is symmetric). Let $v_1, \ldots, v_{k+1} \in R_U$ be $k+1$ distinct vertices. Then for each of these vertices there has to be a distinct edge in every edge dominating set. Thus there is no edge dominating set of cardinality k in $\mathcal{B}$. ✓

Let $\mathcal{B}|_R$ denote the induced subgraph of $\mathcal{B}$ on the vertices in R.

In the following, we consider a partition of the vertices in $\mathcal{B}$ that consists of three sets I, R and K:

- R is defined as above.
- I denotes the set of all vertices $x \in V \setminus R$ with $N_{\mathcal{B}}(x) \subseteq R$.
- $K := (V \setminus I) \setminus R$. Hence, K is the set of all vertices in $V \setminus R$ that have neighbors outside of R.

We will estimate the size of the subgraph of $\mathcal{B}$ induced by $K \cup R$. Observe that for all $v \in K$ we have $d_{\mathcal{G}}(v) \leq k$ and as $\mathcal{B}|_K$ admits no edge dominating set of cardinality at most k unless it has a vertex cover of size $2k$, there can be at most $2k^2$ edges in $\mathcal{B}|_K$. Hence, $|K| \leq 2k + 2k^2$ and as $K = K_U \cup K_W$ we can assume that there is an $l \in \{0, \ldots, 2k + 2k^2\}$ such that $|K_U| = 2k + 2k^2 - l$ and $|K_U| = l$. Thus, by claim 2, there are at most

$$(2k + 2k^2 - l) \cdot k + l \cdot k \quad \leq \quad 2k^2 + 2k^3$$

edges between K and R in $\mathcal{B}$. Furthermore, claim 2 implies as well that $\mathcal{B}|_R$ contains at most k^2 edges.

Thus, we have $|K \cup R| \leq 4k + 2k^2$ and $\mathcal{B}|_{K \cup R}$ has at most $5k^2 + 2k^3$ edges.

With these considerations in mind, we will develop an algorithm solving p-#BipartiteEdgeDomination.

Lemma 2.15. *There is an algorithm that, given a bipartite graph $\mathcal{B} = (U, W, E)$ and $k \in \mathbb{N}$, solves* p-#BipartiteEdgeDomination *in time*

$$\mathcal{O}(\|\mathcal{B}\| + k^4 \cdot 14^k \cdot k^{2k})$$

Proof. Note, that by the considerations above, a set $S \subseteq V$ in $\mathcal{B}$ with $|S| = k$ is an edge dominating set in $\mathcal{B}$ if, and only if, it is an edge dominating set in $\mathcal{B}|_K$ and for every vertex $v \in R$ there is an edge $e \in S$ such that $v \in e$.

Let $\mathcal{C}$ be the subgraph of $\mathcal{B}$ that contains all edges between R and I. Each edge dominating set can be partitioned into a set of edges from $\mathcal{C}$ and another set of edges from $\mathcal{B}|_{K \cup R}$.

Hence, to solve our problem, is suffices to enumerate all subsets L of edges in $\mathcal{B}|_{K \cup R}$ with $|L| \leq k$. Then for each of theses sets L we have to determine the number of ways in which edges from $\mathcal{C}$ can be added to L such that this results in a cardinality k edge dominating set of $\mathcal{B}$.

In order to do this, the algorithm checks for all of these subsets L, if it is an edge dominating set in $\mathcal{B}|_K$. If this is not the case, then adding edges from $\mathcal{C}$ cannot form a solution, as edges in $\mathcal{C}$ are not adjacent to edges in $\mathcal{B}|_K$.

Otherwise, if L with $|L| \leq k$ is an edge dominating set in $\mathcal{B}|_K$, let $U := \{v \in R \mid \forall e \in L : \ v \notin e\}$, that is the set of vertices in R that are not part of an edge in L. Let $l := k - |L| - |U|$.

Note that for each vertex $v \in U$ we have to add an edge e from $\mathcal{C}$ to L to obtain an edge dominating set of $\mathcal{B}$. Thus, if $l < 0$ or $d_{\mathcal{C}}(v) = 0$ for a $v \in U$ then we can form no solution from L.

So assume, that $l \geq 0$ and $d_{\mathcal{C}}(v) > 0$ for all $v \in U$. Define

$$a \ := \ \sum_{x \in R} d_{\mathcal{C}}(x)$$

Thus there are exactly $\binom{a-|U|}{l}$ edge dominating sets of cardinality k in $\mathcal{B}$ that contain L as a subset.

Time. In a preprocessing step the sets R, I, K and the graphs $\mathcal{B}|_K$, $\mathcal{B}|_{K \cup R}$ and $\mathcal{C}$ can be computed together with the values $d_{\mathcal{C}}(v)$ for all $v \in R$ and the value a in time $\mathcal{O}(\|\mathcal{B}\|)$.

For each $i \in \{0, \ldots, k\}$ all of the at most $\binom{5k^2+2k^3}{i}$ cardinality i sets of edges in $\mathcal{B}|_{K \cup R}$ are enumerated. For each of these sets L, checking if L is an edge dominating set in $\mathcal{B}|_K$ can be done in $i \cdot 2k^2$ steps. Forming the set U takes $i \cdot |R| \ \leq \ i \cdot 2k$ steps. Determining if there is a $v \in U$ with $d_{\mathcal{C}}(v) = 0$ can be done in time $\mathcal{O}(|R|)$ as the values $d_{\mathcal{C}}(v)$ are precomputed. Computing the binomial coefficient takes time $\mathcal{O}(l)$. Hence, for a set of cardinality i the necessary

computations can be carried out in time $c \cdot (i \cdot k^2 + (i+1) \cdot k + l)$ with c a constant sufficiently large. For the algorithm (without preprocessing) this results in time:

$$\begin{aligned}
& \sum_{i=0}^{k} \binom{5k^2 + 2k^3}{i} \cdot c \cdot (i \cdot k^2 + (i+1) \cdot k + l) \\
\leq \; & c \cdot k \cdot \binom{5k^2 + 2k^3}{k} \cdot (k^3 + k^2 + 2k) \\
\leq \; & c \cdot 4k^4 \cdot \left(\frac{e \cdot (5k^2 + 2k^3)}{k} \right)^k \\
\leq \; & c \cdot 4k^4 \cdot \left(e \cdot (5k + 2k^2) \right)^k \\
\leq \; & c \cdot 4k^4 \cdot \left(e \cdot 5k^2 \right)^k \\
\in \; & \mathcal{O}(k^4 \cdot 14^k \cdot k^{2k})
\end{aligned}$$

□

Observe that the algorithm given above is not restricted to bipartite graphs. Arguably on general graphs the runtime bound degenerates a little bit. However, it is easy to see that the following still holds

Corollary 2.16. *The problem* p-#EdgeDomination *which is essentially the* p-#BipartiteEdgeDomination *problem without restriction to bipartite graphs is fixed parameter tractable.*

Chapter 3.

Drawing Consequences

In the preceeding chapters we have examined the application of kernelization techniques to counting problems. In this chapter we will reflect on these results. Mainly, this reflection will be focused on creating a formal notion of kernelizations for counting problems. This notion will be justified in two ways. On the one hand, it will become clear that the algorithmic techniques from the preceding chapters can be described by this notion. On the other hand we will see that #FPT can be characterized completely by this concept.

Before we do this, however, we will draw some consequences that do not directly affect #FPT .

3.1. Redefining #W[P]

The fact that the problem p-#UniqueHittingSet is fixed parameter tractable reveals some inconsistencies in the definition of several complexity classes that were defined for parameterized counting problems. For the sake of simplicity we will restrict our discussion to the inconsistencies between #FPT and the class #W[P] .

The definition of #W[P] given by Flum and Grohe (cf. [FG06], p. 366) is tightly connected to certain results from the parameterized complexity theory of decision problems. The most important notions we will consider here are those underlying the class W[P]. As a detailed introduction of this class would be beyond the scope of this work, we refer the reader not familiar with it to [FG06]. Furthermore, we expect some familiarity with nondeterministic Turing machines. For an introduction see, for example, Appendix A of the book mentioned.

We give a brief overview of the necessary facts.

Definition 3.1. *(1) Let Σ be an alphabet and $\kappa : \Sigma^* \to \mathbb{N}$ a parameterization. A nondeterministic Turing machine $\mathbb{M}$ with input alphabet Σ is called* κ-restricted *if there are computable functions $f, h : \mathbb{N} \to \mathbb{N}$ and a polynomial $p \in \mathbb{N}_0[X]$ such that on every run with input $x \in \Sigma^*$ the machine $\mathbb{M}$ performs at most $f(\kappa(x)) \cdot p(|x|)$ steps, at most $h(\kappa(x)) \cdot \log|x|$ of them being nondeterministic.*

(2) W[P] *is the class of all parameterized problems (Q, κ) that can be decided by a κ-restricted Turing machine.*

For $\mathbb{M}$ a κ-restricted Turing machine with input alphabet Σ and $x \in \Sigma^*$ the value $\#acc_{\mathbb{M}}(x)$ denotes the number of accepting runs of $\mathbb{M}$ on x.

The class #W[P] can be regarded as a W[P] analog for counting problems (cf. [FG06] Definition 14.15 , p. 366). For reasons that will become clear soon, we refer to this class as $\#\mathrm{W[P]}_{bounded}$.

Definition 3.2. *A parameterized counting problem (F, κ) over the alphabet Σ is in $\#\mathrm{W[P]}_{bounded}$ if there is a κ-restricted nondeterministic Turing machine $\mathbb{M}$ such that for every $x \in \Sigma^*$, we have*

$$F(x) = \#acc_{\mathbb{M}}(x).$$

Note that, in the case of parameterized decision problems, FPT $\subseteq$ W[P] is easily seen to be true. For parameterized counting problems, one would like to have the analogous fact $\#\mathrm{FPT} \subseteq \#\mathrm{W[P]}_{bounded}$, as intuitively #FPT is a class of computationally easy problems and $\#\mathrm{W[P]}_{bounded}$ contains many intractable problems.

However, it is easily observable that there are fixed parameter tractable counting problems that are not contained in $\#\mathrm{W[P]}_{bounded}$. To see this consider a parameterized counting problem (F, κ) over the alphabet Σ that is contained in $\#\mathrm{W[P]}_{bounded}$. Let $\mathbb{M}$ be a κ-restricted Turing machine witnessing $(F, \kappa) \in \#\mathrm{W[P]}_{bounded}$ and let f, h, p be as in definition 3.1. Observe that there is a constant $c = c(\mathbb{M})$ such that every nondeterminitic step of $\mathbb{M}$ can be described by at most c bits. Hence for $x \in \Sigma^*$, $k := \kappa(k)$ and $n := |x|$ each accepting run of $\mathbb{M}$ on x can be described by $c \cdot h(k) \cdot \log n \leq 2c \cdot h(k) \cdot \lfloor \log n \rfloor$ bits. This implies that there are at most

$$2^{2c \cdot h(k) \cdot \lfloor \log n \rfloor} \leq n^{2c \cdot h(k)}$$

different accepting runs of $\mathbb{M}$ on x. Therefore, for all $x \in \Sigma^*$ we have

$$F(x) \leq |x|^{2c \cdot h(k)}.$$

With this in mind we can construct a counting problem that is fixed parameter tractable but not in $\#\mathrm{W[P]}_{bounded}$. Let (F', κ') be a counting problem over Σ with $\kappa'(x) := 1$ for all $x \in \Sigma^*$ and $F'(x) := 2^{|x|}$. Note that, $F'(x)$ can be computed in linear time. Assume, for contradiction, that (F', κ') is contained in $\#\mathrm{W[P]}_{bounded}$, then by the definition of κ' and the consideration above, there is a constant c such that $F'(x) \leq |x|^c$ for all $x \in \Sigma^*$. By the definition of F' there are large enough $x \in \Sigma^*$ such that $F'(x) = 2^{|x|} > |x|^c$, in contradiction to $(F', \kappa') \in \#\mathrm{W[P]}_{bounded}$.

The example above suggests a redefinition either of the class #W[P] or of #FPT. Up to now, there has been no reasonable evidence that motivates favoring either of these possibilities. With the discussion of the problems in the previous chapters, however, this situation has changed. Of course, almost all of the problems considered there suggest a redefinition of the class #FPT, as they are contained in $\#\mathrm{W[P]}_{bounded}$.

Nevertheless, we have seen that the problem p-#UniqueHittingSet is fixed parameter tractable as well. Furthermore, the argument for (F', κ') above can easily be adapted to p-#UniqueHittingSet if we restrict our dicussion to hypergraphs without hyperedges. Let $\mathcal{H} = (V, \emptyset)$ be such a hypergraph, then the number of unique hitting sets in $\mathcal{H}$ equals $2^{|V|}$. Hence, p-#UniqueHittingSet is

an example of a natural problem in #FPT that is not contained in $\#\mathrm{W[P]}_{bounded}$. Therefore, we will give a definition of #W[P] as a class explicitly closed under a certain type of reductions which will circumvent this inconsistency.

To define the reductions needed, we will deal with *oracle algorithms*. With regard to our purposes it suffices to note that for an alphabet Σ an algorithm with an oracle to a function $G : \Sigma^* \to \mathbb{N}$ is equipped with a certain *oracle query*. This query of the form "$G(y) =?$" can be posed by the algorithm for any $y \in \Sigma^*$ and the oracle returns the correct answer in unit time.

Definition 3.3. *Let (F, κ) and (G, λ) be parameterized counting problems over the alphabets Σ and Γ, respectively.*

An fpt Turing reduction *from (F, κ) to (G, λ) is an algorithm $\mathbb{A}$ with an oracle to G such that:*

a) $\mathbb{A}$ computes F.

b) $\mathbb{A}$ is an fpt algorithm (with respect to κ).

c) There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all oracle queries "G(y) = ?" posed by $\mathbb{A}$ on input x, we have $\lambda(y) \leq g(\kappa(x))$.

The concept of parameterized Turing reductions is very powerful. Therefore, in our redefinition of #W[P] we restrict our attention to parameterized Turing reductions *with at most one oracle call*. For parameterized counting problems (F, κ) and (G, λ) we write $(F, \kappa) \leq^{fpt-T!} (G, \lambda)$ if there is an fpt Turing reduction from (F, κ) to (G, λ) that calls the oracle only once.

For a class C of parameterized counting problems, we define

$$[C]^{fpt-T!} := \{(G, \lambda) \mid \text{ex. } (F, \kappa) \in C : \ (G, \lambda) \leq^{fpt-T!} (F, \kappa)\}$$

which we call the *closure* of C under parameterized Turing reductions with a single oracle call. This is the explicit closure we mentioned earlier and hence we can give the redefinition.

Definition 3.4. $\#\mathrm{W[P]} := [\#\mathrm{W[P]}_{bounded}\ ]^{fpt-T!}$

Note that it is easy to see that #FPT $\subseteq$ #W[P] , as every problem in #FPT is trivially fpt-T!-reducible to any counting problem. Furthermore, as Turing reductions are very powerful we want to restrict their application in defining our complexity classes as far as possible. This receives its motivation from the intention that we seek natural counting analogs of parameterized complexity classes for decision problems.

3.2. Counting Kernelization

Recently, there has been some interest in forming a notion of kernelizations for counting problems. For example Nishimura et al. (cf. [NRT05]) suggested the notion of *compactor enumeration*, which means to reduce the input instance such

that the resulting instance depends only on the parameter. Then by enumerating the solutions of the reduced instance, one can compute the number of solutions of the original instance.

However, this notion has not yet been thoroughly formalized. In particular, the informal definition of a compactor from [NRT05] is not applicable to counting problems in general.

In this section we will remedy this deficiency by formally describing kernelizations of counting problems. Note that in this purely theoretical context we regard computations always as computations of Turing machines. We begin with some preliminary definitions.

Definition 3.5. *Let $\kappa : \Sigma^* \to \mathbb{N}$ be a parameterization of Σ^*.*

- *A mapping $K : \Sigma^* \to \Sigma^*$ is* κ-bounded *if there is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \Sigma^*$:*

$$|K(x)| \leq g(\kappa(x)).$$

- *A relation $Y \subseteq \Sigma^* \times \{0,1\}^*$ is* K-aware *for a κ-bounded mapping K if there are computable functions $h, f : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \Sigma^*$ and every $y \in \{0,1\}^*$:*
  - *(1) Given $K(x)$, it can be decided in time $f(\kappa(x))$ whether $(K(x), y) \in Y$ holds.*
  - *(2) if $(K(x), y) \in Y$ then $|y| \leq h(\kappa(x))$.*

Recall that in the introduction we formed the requirement (requirement 0.6) that for a counting problem (F, κ) the precomputation on an instance $x \in \Sigma^*$ can be done in polynomial time such that the size of the reduced instance $K(x)$ is bounded by some function of the parameter. Intuitively a *κ-bounded* mapping K will play the role of this precomputation.

Furthermore, we stipulated that only the reduced instance $K(x)$ will be solved by a search algorithm. We did not clarify these notions, but in our formal concept of counting kernelizations we will define a K-aware relation Y to characterize the "solutions" of the reduced instance $K(x)$ that have to be found by such a search algorithm. Unfortunately, the formal definition of a counting problem does not mention what a solution might be. Therefore we will formalize this by the notion of *witnesses*:

Let $Y \subseteq \Sigma^* \times \{0,1\}^*$ be a relation. For $x \in \Sigma^*$ we define $w_Y(x) := \{y \,|\, (x,y) \in Y\}$ as the set of *witnesses* of x in Y.

The following lemma shows the relation between fixed parameter tractable counting problems and K-aware relations. In the proof of this lemma the general meaning of witnesses will become clear.

Lemma 3.6. *Let $(F, \kappa) \in \#FPT$ be a parameterized counting problem over Σ. Let $K : \Sigma^* \to \Sigma^*$ be a κ-bounded polynomial time computable mapping.*

There is a K-aware relation $Y \subseteq \Sigma^* \times \{0,1\}^*$ such that for all $x \in \Sigma^*$:

$$F(K(x)) = |w_Y(K(x))|.$$

Proof. Let $x \in \Sigma^*$, $n := |x|$ and $k := \kappa(x)$. As $(F, \kappa) \in$#FPT there is a computable function $f : \mathbb{N} \to \mathbb{N}$ and a polynomial $p(X)$ such that $F(x)$ is computable in time $f(k) \cdot p(n)$. Furthermore as K is κ-bounded, there is a computable function g such that we have

$$|K(x)| \leq g(k).$$

Observe that, since $\kappa : \Sigma^* \to \mathbb{N}$ is polynomial time computable, $\kappa(K(x))$ can be computed in time $|K(x)|^{\mathcal{O}(1)} \leq g(k)^{\mathcal{O}(1)}$. As κ computes natural numbers, we may assume w.l.o.g. that these are represented in binary. Hence, for a computable function $h(k) := 2^{g(k)^{\mathcal{O}(1)}}$ we know that

$$\kappa(K(x)) \leq h(\kappa(x)).$$

Thus, given $K(x)$ the value $F(K(x))$ can be computed in time

$$f(\kappa(K(x))) \cdot p(|K(x)|) \leq f(h(k)) \cdot p(g(k)) =: f^*(k)$$

which implies that there is a computable function r such that $F(K(x)) \leq r(k)$.

Before we construct the K-aware relation Y we describe a κ-restricted Turing machine $\mathbb{M}$ that computes the function $F'(x) := F(K(x))$. This will simplify our task of defining Y.

$\mathbb{M}$ computes $F'(x)$ as follows:

First $K(x)$ is computed in polynomial time. Then $\mathbb{M}$ computes $F(K(x))$ and a $\lceil \log r(k) \rceil$ bit number c is guessed nondeterministically. $\mathbb{M}$ accepts if and only if $0 < c \leq F(K(x))$.

It is easy to see that $\mathbb{M}$ is a valid κ-restricted Turing machine and that

$$F'(x) = F(K(x)) = \#acc_{\mathbb{M}}(x).$$

The definition of Y is very simple, now:

$$Y := \{(K(x), y) \in \Sigma^* \times \{0,1\}^* \mid \begin{array}{l} |y| = \lceil \log r(\kappa(x)) \rceil, \text{ y represents} \\ \text{the nondeterministic steps of an accept-} \\ \text{ing run of } \mathbb{M} \text{ on } x \ \} \end{array}$$

This directly implies $F'(x) = \#acc_{\mathbb{M}}(x) = |w_Y(K(x))|$.

We still have to show that Y is K-aware. Note that for $(K(x), y) \in Y$ condition (2) of definition 3.5 holds, that is, $|y| \leq r(k)$. To see that condition (1) holds, we show, given $K(x)$ and $y \in \{0,1\}^*$, how to decide $(K(x), y) \in Y$ in time depending only on k.

An algorithm deciding this deterministically simulates the computation of $\mathbb{M}$ after $K(x)$ has been computed. To simulate the nondeterministic steps of $\mathbb{M}$ the bits in y are used. As $\mathbb{M}$ performs at most $f^*(k)$ steps for the computation of $F(K(x))$ and the guessed number c satisfies $c \leq r(k)$, the simulation takes time only depending on k. Therefore, condition (1) holds and hence Y is K-aware. □

Note that in the proof above, it is not essential that (F, κ) is fixed parameter tractable. It is only necessary that (F, κ) can be computed by some algorithm. However as we are concerned only with #FPT this lemma satisfies our needs. We will use it it the next section to show that the concept of counting kernelizations, which we will define now, characterizes all counting problems in #FPT.

We have discussed already the intuition behind the concepts of κ-bounded mappings K and K-aware relations. The last ingredient we need is a function, say μ, that for each solution y of the reduced instance $K(x)$ computes the number of solutions of an instance $x \in \Sigma^*$ that are "represented" by y. In the case of p-#VertexCover, for example, this was done by computing a binomial coefficient.

In chapter 1 and 2 we discussed two running time analyses that applied the logarithmic cost measure. Particularly the analysis of the algorithm solving p-#UniqueHittingSet implies that we cannot hope to bound the running time of μ in terms of the parameter. This is even more so as in this chapter we consider computations exclusively as computations of Turing machines, hence we may assume no operations except for trivial ones to take unit time. Moreover, the function μ has to depend on the original instance x to be able to create the link between $K(x)$ and x.

Definition 3.7 (Counting Kernelization). *Let (F, κ) be a parameterized counting problem over Σ. A pair (μ, K) of polynomial time computable functions $K : \Sigma^* \to \Sigma^*$ and $\mu : \Sigma^* \times \Sigma^* \times \{0,1\}^* \to \mathbb{N}$ is called a* counting kernelization *of (F, κ) if there are computable functions $g, h : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \Sigma^*$ the following is satisfied:*

(1) K is κ-bounded

(2) There is a K-aware relation $Y \subseteq \Sigma^* \times \{0,1\}^*$ such that

$$F(x) = \sum_{y \in w_Y(K(x))} \mu(x, K(x), y) \tag{3.1}$$

Note that, if the set $w_Y(K(x))$ in the definition above is empty, we regard the sum as evaluating to zero. We have to stress that if $\mu(x, K(x), y)$ is polynomial time computable this means that it is polynomial time computable in $|x| + |K(x)| + |y| \leq |x| + g(k) + h(k)$. With respect only to $|x|$, we would have to admit that $\mu(x, K(x), y)$ is fpt-computable as $|K(x)|$ and $|y|$ can be arbitrarily large, depending only on $\kappa(x)$. In practical problems, however, $|K(x)|$ and $|y|$ are often bounded by $|x|$ and then polynomial time computability of μ implies that μ is polynomial time computable in $|x|$. Therefore, we claim that it is reasonable to impose this constraint on μ.

Nevertheless, this definition still seems somewhat arbitrary. Its motivation will become clear if we know how to apply it to the problems studied in the previous chapters. We will begin with p-#VertexCover.

The counting kernelization of p-#VertexCover. Recall the Buss kernelization and in particular the algorithm CountBussVC studied in the first chapter. Note

that CountVC is applied in the algorithm CountBussVC (see algorithm 2) to improve upon the time bound of theorem 1.1. Hence we have to regard CountBussVC as a combination of a bounded search tree algorithm and kernelization techniques. To isolate the kernelization techniques, we present algorithm 7 which consists of lines 1-14 of CountBussVC. Note that K_{Buss} computes a mapping that satisfies condition (1) of a counting kernelization.

```
   K_Buss(G, k) // G = (V, E) a graph, k ∈ ℕ
 1 V' ← V; E' ← E;
 2 k' ← k; G' ← (V', E');
 3 while there is a v ∈ V' with d_G'(v) > k' do
 4     E' ← E' \ {e ∈ E | ∃ w ∈ V' : e = {v, w}};
 5     V' ← V' \ {v}; G' ← (V', E');
 6     k' ← k' − 1;
 7 end
 8 if k' = 0 and E' = ∅ then G' = (V', E') ← G_1;
 9 if k' ≤ 0 or |E'| > (k')^2 or |V'| < k' then
       // i.e.  #vc(G', k') = 0
10     G' = (V', E') ← G_0;
11     k' ← 0 ;
12 end
13 I ← {v ∈ V' | d_G'(v) = 0};
14 V' ← V' \ I;
15 G' ← (V', E');
16 return (G', k');
```

Algorithm 7: Isolating the kernelization phase of CountBussVC

Let $(\mathcal{G}, k)$ with $\mathcal{G} = (V, E)$ be an instance of p-#VertexCover. The mapping K_{Buss} computes an instance $K_{Buss}(\mathcal{G}, k) = (\mathcal{G}', k')$ such that with $\mathcal{G}' = (V', E')$ we have $|V'| \leq 2k^2$ and $|E'| \leq k^2$.

We define a relation Y such that for all instances $(\mathcal{G}, k)$ of p-#VertexCover and $y \in \{0,1\}^*$ we have $((\mathcal{G}, k), y) \in Y$ if and only if y represents a vertex cover of size at most k in $\mathcal{G}$. Note that "represents" in this context denotes any reasonable representation of sets $S \subseteq V$ by binary numbers. Hence, $w_Y((\mathcal{G}', k'))$ can be enumerated in time $h(k)$ for some computable function h.

Let $(\mathcal{G}, k)$ and $(\mathcal{G}', k')$ be defined as above and consider a string $y \in \{0,1\}^*$ that represents a vertex cover C in $\mathcal{G}'$ with $|C| \leq k'$. Let $m := |V \setminus V'| - (k - k')$. Note that for the set I from line 13 of the algorithm K_{Buss} we have $m = |I|$. In the same way as in CountBussVC the set I consists of all vertices that can be combined with C to form a size k vertex cover of $\mathcal{G}$. Thus for every subset $X \subset I$ of cardinality $k' - |C|$ the union $X \cup C$ is a size k' vertex cover of $\mathcal{G}$. As implicitely the $k - k'$ vertices in $\mathcal{G}$ deleted in the *while*-loop of algorithm K_{Buss} are considered as being contained in every size k vertex cover, $X \cup C$ represents such a vertex cover of cardinality k in $\mathcal{G}$.

Hence, defining

$$\mu((\mathcal{G},k),(\mathcal{G}',k'),y) := \binom{m}{k'-|C|}$$

completes the counting kernelization.

Note that all problems we have given kernelization algorithms for can be described in an analogous manner by counting kernelization. Therefore we give only one further example:

The counting kernelization of p-#UniqueHittingSet. We show how the solution of p-#UniqueHittingSet presented in chapter 2 can be viewed as a counting kernelization (μ, K). Recall that for an input hypergraph $\mathcal{H} = (V,E)$ with $|E| := k$ the equivalency classes $\tilde{\mathbf{A}} := \{A_1,\ldots,A_{l-1}\}$ and the equivalency class A_0 of all isolated vertices in $\mathcal{H}$ were constructed. Furthermore the hypergraph $\tilde{\mathcal{H}} := (\tilde{\mathbf{A}},\tilde{E})$ was defined such that its size $\|\tilde{\mathcal{H}}\|$ depended only on k. The given algorithm enumerates all unique hitting sets in $\tilde{\mathcal{H}}$ are and for each unique hitting set $\tilde{S}$ the value

$$2^{|A_0|}\prod_{A\in\tilde{S}}|A|$$

according to equation (2.3) is computed, giving the number of unique hitting sets in $\mathcal{H}$ that are represented by $\tilde{S}$.

In the counting kernelization the mapping K performs the construction of $\tilde{\mathcal{H}}$, which was shown to take polynomial time. Computing the value according to equation (2.3) clearly takes polynomial time, therefore we give this piece of work over to μ. That is, for a unique hitting set $\tilde{S}$ in $\tilde{\mathcal{H}}$, we define

$$\mu(\mathcal{H},\tilde{\mathcal{H}},\tilde{S}) := 2^{|A_0|}\prod_{A\in\tilde{S}}|A|.$$

Define the relation Y such that for all instances $\mathcal{H}$ of p-#UniqueHittingSet that contain no isolated vertices and $y\in\{0,1\}^*$ we have $(\mathcal{H},y)\in Y$ if and only if y represents a unique hitting set in $\mathcal{H}$. As for every $\mathcal{H}$ the reduced graph $K(\mathcal{H}) = \tilde{\mathcal{H}}$ does not contain isolated vertices, we have found our counting kernelization.

3.3. Characterizing #FPT by Counting Kernelizations

Up to now, we have seen that counting kernelizations can be used to describe certain fpt algorithms for counting problems. In this section we will see that the notion of counting kernelizations is even more general in the sense that it provides a characterization of fixed parameter tractable counting problems.

Theorem 3.8. *Let (F,κ) be a parameterized counting problem. The following are equivalent:*

(1) $(F,\kappa)\in \#FPT$

(2) (F,κ) has a counting kernelization.

Proof. (2) ⇒ (1): Let (μ, K) be a counting kernelization of (F, κ) and let $Y \subseteq \Sigma^* \times \{0,1\}^*$ be a K-aware relation satisfying definition 3.7. Let $x \in \Sigma^*$, $n := |x|$, $k := \kappa(x)$ and $p(X)$ a polynomial bounding the time needed to compute μ and K. As K is κ-bounded there is a computable function g such that $|K(x)| \leq g(\kappa(x))$.

We give an fpt algorithm that computes $F(x)$. First, $K(x)$ is computed in time $p(n)$. By the definition of Y and $|K(x)| \leq g(k)$ the set $w_Y(K(x))$ can be computed in time $h(k)$ for some computable function h. Note that this implies $|y| \leq h(k)$ for all $y \in w_Y(K(x))$. Hence $\mu(x, K(x), y)$ can be computed in time $p(n+g(k)+h(k))$ and the whole sum from equation (3.1) takes times at most $h(k) \cdot p(n+g(k)+h(k))$.

Consequently, the whole computation can be performed in time

$$p(n) + h(k) + h(k) \cdot p(n + g(k) + h(k)) \leq (h(k) + 1) \cdot (1 + p(n + g(k) + h(k))).$$

We may assume that $p(X) = X^c$ for a constant c. Then, by the inequality $a + b \leq a \cdot (b + 1)$ (for $a, b \in \mathbb{N}$), we get

$$\begin{aligned}(h(k) + 1) \cdot (1 + p(n + g(k) + h(k))) &\leq (h(k) + 1) \cdot (1 + n^c \cdot (g(k) + h(k) + 1)^c) \\ &\leq (h(k) + 1) \cdot (1 + (g(k) + h(k) + 1)^c) \cdot n^c \\ &= f(k) \cdot n^c\end{aligned}$$

with $f(k) := (h(k)+1) \cdot (1+(g(k)+h(k)+1)^c)$. And thus the algorithm presented is an fpt algorithm.

(1) ⇒ (2): Let $\mathbb{A}$ be an algorithm that, for every $x \in \Sigma^*$, computes $F(x)$ in time $f(\kappa(x)) \cdot p(|x|)$ for some computable function f and a polynomial $p(X)$. W.l.o.g. we may assume that $p(n) \geq n$ for all $n \in \mathbb{N}$.

In the following, we construct a counting kernelization (μ, K) of (F, κ).

If $F(x) = 0$ for all $x \in \Sigma^*$, we define $K(x) := z$ for all $x \in \Sigma^*$ and an arbitrary $z \in \Sigma$. Furthermore, $\mu(x, K(x), y) := 0$ for all $x \in \Sigma^*$ and all $y \in \{0,1\}^*$. Then with $Y := \emptyset$ we obtain a valid counting kernelization.

Otherwise, we fix a $z \in \Sigma^*$ such that $F(z) = a_z \neq 0$. For $x \in \Sigma^*$ with $n := |x|$ and $k := \kappa(x)$. First, we define $K(x)$:

The algorithm $\mathbb{A}$ is simulated on input x for $p(n) \cdot p(n)$ steps and the following function is computed:

$$K(x) := \begin{cases} z & \text{, if } x = z \\ & \text{or } \mathbb{A} \text{ completes in } p(n) \cdot p(n) \text{ steps} \\ x & \text{, otherwise} \end{cases}$$

Set $f^*(k) := |z| + f(k)$. By definition, K is computable in polynomial time and for all $x \in \Sigma^*$ we have $|K(x)| \leq f^*(k)$. To see this, note that if $x = z$ or if $K(x)$ is computable in $p(n) \cdot p(n)$ steps we have $|K(x)| = |z| \leq f^*(k)$. Otherwise we know that $|x|^2 \leq p(n) \cdot p(n) \leq f(k) \cdot f(k)$ and hence $|K(x)| = |x| \leq f^*(k)$.

Let $Y \subseteq \Sigma^* \times \{0,1\}^*$ be a K-aware relation according to lemma 3.6. In particular, note that $F(K(x)) = |w_Y(K(x))|$ holds. Furthermore, as z is fixed, we may assume that $F(z) = |w_Y(z)|$ holds as well.

Observe that, by the definition of Y, given $K(x)$ and z the sets $w_Y(K(x))$ and $w_Y(z)$ can be enumerated in time $g(k)$ for some appropriate computable function g.

As $F(z) \neq 0$, we know that $w_Y(z)$ is nonempty and hence we may fix a $y_z \in w_Y(z)$.

We are now able to define the function $\mu : \Sigma^* \times \Sigma^* \times \{0,1\}^* \to \mathbb{N}$. Recall that μ has to be polynomial time computable and

$$F(x) = \sum_{y \in w_Y(K(x))} \mu(x, K(x), y) \tag{3.2}$$

has to be satisfied for all $x \in \Sigma^*$.

Consider an argument $(x, K(x), y)$ of μ.

For $x = z$, we define $\mu(z, K(z), y) := 1$ for all $y \in \{0,1\}^*$. Clearly equation (3.2) is satisfied.

Otherwise, if $x \neq z$ and $K(x) = z$, we know by the definition of K that the value $F(x)$ can be computed in polynomial time. Hence, define $\mu(x, K(x), y_z) := F(x)$ and for all $y \neq y_z$ we set $\mu(x, K(x), y) := 0$. Observe that equation (3.2) is satisfied here as well.

If $x \neq z$ and $K(x) \neq z$, the definition of K implies that $K(x) = x$. Then we define $\mu(x, K(x), y) := 1$ for all $y \in \{0,1\}^*$. As the sum in equation (3.2) is defined only over $y \in w_Y(x)$, the equation is satisfied.

We summarize the considerations above by a formal definition of μ:

$$\mu(x, K(x), y) := \begin{cases} 1 & \text{, if } x = z \\ & \quad \text{or } x \neq z \wedge K(x) \neq z \\ F(x) & \text{, if } x \neq z \wedge K(x) = z \wedge y = y_z \\ 0 & \text{, if } x \neq z \wedge K(x) = z \wedge y \neq y_z \end{cases}$$

It is easy to see that the given functions K and μ are polynomial time computable and that for all $x \in \Sigma^*$ equation (3.2) is satisfied. Hence (μ, K) is a counting kernelization of (F, κ). □

Part II.

The Intractability of Parameterized Counting Problems

Chapter 4.

An Introduction to Parameterized Intractability

Parameterized Complexity has its own theory of intractability which comprises new complexity classes. Among these are the classes of the A and W-hierarchy and their analogs defined for counting problems: the #A and #W hierarchy. We will restrict our attention to the class #A[1] which forms the first level of the #A hierarchy. As this class is defined by means of logical problems, we have to introduce some notions from *first order logic* (FO).

Recall that, for a set A, A^n denotes the n-term cartesian product of A, that is

$$A^n := \underbrace{A \times \ldots \times A}_{\text{n times}}.$$

A *relational vocabulary* τ is a finite set of *relation symbols*. Each relation symbol $R \in \tau$ has an arity $ar(R) \geq 1$. A $\tau-$*structure* (or *structure*) $\mathcal{A}$ is a tuple $(A, (R^{\mathcal{A}})_{R\in\tau})$ that consists of a finite set A, called the *universe* of $\mathcal{A}$ and of an *interpretation* $R^{\mathcal{A}} \subseteq A^{ar(R)}$ of each relation symbol $R \in \tau$. Furthermore, the *size* of a τ-structure $\mathcal{A}$ is defined as follows:

$$\|\mathcal{A}\| := |A| + \sum_{R\in\tau} ar(R) \cdot |R^{\mathcal{A}}| \tag{4.1}$$

Syntax of FO. We define the countably infinite set VAR $= \{x_1, x_2, \ldots\}$ as the set of *variables*. For convenience we often take lower case letters $x, y, \ldots$ to denote variables. Given a vocabulary τ and variables $x_1, x_2, \ldots$, an *atomic τ-formula* is of the form $Rx_1 \ldots x_{ar(R)}$ for $R \in \tau$, or $x_1 = x_2$.

First-order τ-formulas consist of atomic τ-formulas connected by Boolean connectives $(\neg, \wedge, \vee)$ and quantified by *existential* or *universal quantifiers* $\exists, \forall$. Instead of $\neg x = y$ we often write $x \neq y$. The set of all first order formulas is denoted by FO.

For our purposes it will be convenient to consider the *quantifier-free* fragment Π_0 of FO. We call a first order formula φ *quantifier-free* if it contains no quantifiers and by Π_0 we denote the set of all these formulas. For $\varphi \in \Pi_0$ we write $\varphi(x_1, \ldots, x_k)$ to denote that the variables in φ are among $x_1, \ldots, x_k$.

Note that we can consider τ-formulas in a straightforward manner as strings over some alphabet $\Sigma_{FO[\tau]} = \text{VAR} \cup \tau \cup \{\wedge, \vee, \neg, \exists, \forall, =, (,)\}$. Therefore, we define $|\varphi|$ for a $\varphi \in$ FO as the length of the *string* φ.

Semantics. Let τ be a vocabulary. We restrict our considerations to τ-formulas from Π_0. Let $\varphi(x_1, \ldots, x_k) \in \Pi_0$ and $\mathcal{A}$ be a τ-structure. The relation $\varphi(\mathcal{A}) \subseteq A^k$ is defined inductively:

- If $\varphi(x_1, \ldots, x_k) = Rx_{i_1} \ldots x_{i_r}$ with $ar(R) = r$ and $i_1, \ldots, i_r \in [k]$, then

$$\varphi(\mathcal{A}) := \{(a_1, \ldots, a_k) \in A^k \mid (a_{i_1}, \ldots, a_{i_r}) \in R^{\mathcal{A}}\}.$$

- If $\varphi(x_1, \ldots, x_k) = \neg\psi(x_{i_1}, \ldots, x_{i_l})$ with $i_1, \ldots, i_l \in [k]$, then

$$\varphi(\mathcal{A}) := \{(a_1, \ldots, a_k) \in A^k \mid (a_{i_1}, \ldots, a_{i_l}) \notin \psi(\mathcal{A})\}.$$

- If $\varphi(x_1, \ldots, x_k) = \psi(x_{i_1}, \ldots, x_{i_l}) \overset{\wedge}{\vee} \chi(x_{j_1}, \ldots, x_{j_m})$ with $i_1, \ldots, i_l, j_1, \ldots, j_m \in [k]$, then

$$\varphi(\mathcal{A}) := \{(a_1, \ldots, a_k) \in A^k \mid (a_{i_1}, \ldots, a_{i_l}) \in \psi(\mathcal{A}) \overset{\text{and}}{\text{or}} (a_{j_1}, \ldots, a_{j_m}) \in \chi(\mathcal{A})\}.$$

4.1. The Class #A[1]

The parameterized complexity class #A[1] is defined via a so-called *model counting* problem, based on the evaluation of Π_0 formulas. Formally, it is defined in the following way:

p-#MC(Π_0)
Instance: A structure $\mathcal{A}$ and a formula $\varphi \in \Pi_0$ *Parameter:* $\lvert\varphi\rvert$ *Problem:* Compute $\lvert\varphi(\mathcal{A})\rvert$.

Note that, for an instance $(\mathcal{A}, \varphi)$ of p-#MC(Π_0), we write $\varphi(x_1, \ldots, x_k)$ if all of these variables actually occur in φ. This is necessary, as otherwise the expression $|\varphi(\mathcal{A})|$ would be ambiguous.

We will deal with *oracle algorithms* again. Recall the definition of fpt Turing reductions from the previous chapter:

Definition 4.1. *Let (F, κ) and (G, λ) be parameterized counting problems over the alphabets Σ and Γ, respectively.*

An fpt Turing reduction *from (F, κ) to (G, λ) is an algorithm $\mathbb{A}$ with an oracle to G such that:*

a) $\mathbb{A}$ computes F.

b) $\mathbb{A}$ is an fpt algorithm (with respect to κ).

c) There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all oracle queries "G(y) = ?" posed by $\mathbb{A}$ on input x, we have $\lambda(y) \leq g(\kappa(x))$.

We write $(F,\kappa) \leq^{fpt-T} (G,\lambda)$, for parameterized counting problems (F,κ) and (G,λ), if there is an fpt Turing reduction from (F,κ) to (G,λ). Recall that we write $(F,\kappa) \leq^{fpt-T!} (G,\lambda)$ if an fpt Turing reduction utilizes at most one oracle call.

Note that $(F,\kappa) \leq^{fpt} (G,\lambda)$ implies $(F,\kappa) \leq^{fpt-T!} (G,\lambda)$ which in turn implies $(F,\kappa) \leq^{fpt-T} (G,\lambda)$.

We are now able to define the class #A[1]:

Definition 4.2. $\#\mathrm{A}[1] := [\mathsf{p}\text{-}\#\mathrm{MC}(\Pi_0)]^{fpt-T!}$

Observe that this definition entails that for a parameterized counting problem (F,κ), $(F,\kappa) \in \#\mathrm{A}[1]$ if, and only if, $(F,\kappa) \leq^{fpt-T!} \mathsf{p}-\#\mathrm{MC}(\Pi_0)$. However, by our considerations above we know that an fpt parsimonious reduction suffices to show the containment.

Note that in [FG06] the class #A[1] is defined by a closure under fpt parsimonious reductions. Our motivation to give an alternative definition is essentially the same as that for redefining #W[P] - we want to make sure that $\#\mathrm{FPT} \subseteq \#\mathrm{A}[1]$ holds. Hence we refer to the class defined in [FG06] by $\#\mathrm{A}[1]_{bounded}$. Thus we clearly have that $\#\mathrm{A}[1]_{bounded}$ is a proper subset of #A[1] as p-#UniqueHittingSet is not contained in $\#\mathrm{A}[1]_{bounded}$.

We say that a parameterized counting problem (F,κ) is *#A[1]-hard under fpt parsimonious reductions*, if $\mathsf{p}\text{-}\#\mathrm{MC}(\Pi_0) \leq^{fpt} (F,\kappa)$. Likewise, (F,κ) is *#A[1]-hard under fpt Turing reductions*, if $\mathsf{p}\text{-}\#\mathrm{MC}(\Pi_0) \leq^{fpt-T} (F,\kappa)$.

Hence if these hardness results go along with $(F,\kappa) \in \#\mathrm{A}[1]$, we say that (F,κ) is *#A[1]-complete* under the corresponding reductions. Note that all completeness results for $\#\mathrm{A}[1]_{bounded}$ straightforwardly extend to #A[1].

In the remainder of this section we state the existence of a #A[1]-complete parameterized counting problem. Recall that a clique $\mathcal{K}_k$ is the complete graph on k vertices, to wit, $\mathcal{K}_k = ([k], \binom{[k]}{2})$.

p-#Clique
Instance: A graph $\mathcal{G} = (V,E)$ and $k \in \mathbb{N}$ *Parameter:* k *Problem:* Compute the number of $\mathcal{K}_k$ in $\mathcal{G}$

This problem is the most important starting point for showing the #A[1]-hardness of certain problems. Therefore we state its completeness.

Fact 4.3. p-#Clique *is #A[1]-complete under fpt parsimonious reductions.*

A proof of this fact can be found in [FG06], therefore we omit it.

Chapter 5.

The Parameterized Complexity of Counting Bipartite Cliques

The first hardness result which we will show involves the problem of counting bipartite cliques in bipartite graphs. We will make this precise by some definitions.

Definition 5.1. *1. A* bipartite *clique is a bipartite graph* $\mathcal{K}_{k,l} = ([k], [l], F)$ *for* $k, l \in \mathbb{N}$ *such that* F *contains all edges between* $[k]$ *and* $[l]$.

2. A bipartite digraph *is a triple* $\mathcal{B} = (U, W, F)$ *with* $F \subseteq U \times W \cup W \times U$.

3. A trivially directed *bipartite graph is a bipartite digraph* $\mathcal{B} = (U, W, F)$ *such that* $F \subseteq U \times W$.

4. A trivially directed *bipartite clique is a trivially directed bipartite graph* $\vec{\mathcal{K}}_{k,l} = ([k], [l], [k] \times [l])$ *for* $k, l \in \mathbb{N}$.

Unless stated otherwise, we consider digraphs without loops, that is, edges of the form (v, v) are disallowed. Note that bipartite digraphs contain no loops by definition.

Now, we are able to state our problem formally:

p-#BipartiteClique
Instance: A bipartite graph $\mathcal{B} = (U, W, F)$ and $k, l \in \mathbb{N}$ *Parameter:* k, l *Problem:* Compute the number of $\mathcal{K}_{k,l}$ in $\mathcal{B}$

We will prove that p-#BipartiteClique is #A[1]-complete under fpt Turing reductions. The proof is carried out by a chain of reductions. The first of these reductions demonstrates the #A[1]-hardness of certain kinds of homomorphism problems.

Let us clarify our notion of homomorphisms and isomorphisms.

Definition 5.2. *Given a relational vocabulary* τ *and* τ*-structures* $\mathcal{A} = (A, (R^{\mathcal{A}})_{R \in \tau})$ *and* $\mathcal{B} = (B, (R^{\mathcal{B}})_{R \in \tau})$.

(1) A homomorphism *from* $\mathcal{A}$ *to* $\mathcal{B}$ *is a mapping* $h : A \to B$ *such that for every* $R \in \tau$ *with* $ar(R) = r$ *and every tuple* $\bar{a} = (a_1, \ldots, a_r) \in A^r$ *we have:*

$$\bar{a} \in R^{\mathcal{A}} \Rightarrow h(\bar{a}) := (h(a_1), \ldots, h(a_r)) \in R^{\mathcal{B}}.$$

(2) *An* isomorphism *from* $\mathcal{A}$ *to* $\mathcal{B}$ *is a* bijective *mapping* $f : A \to B$ *such that for every* $R \in \tau$ *with* $ar(R) = r$ *and every tuple* $\bar{a} = (a_1, \ldots, a_r) \in A^r$ *we have:*

$$\bar{a} \in R^{\mathcal{A}} \Leftrightarrow f(\bar{a}) \in R^{\mathcal{B}}.$$

By $\mathrm{HOM}(\mathcal{A}, \mathcal{B})$ *we denote the set of all homomorphisms from* $\mathcal{A}$ *to* $\mathcal{B}$.

An automorphism *of a structure* $\mathcal{A}$ *is an isomorphism from* $\mathcal{A}$ *to* $\mathcal{A}$.

Since digraphs can be considered as structures on the vocabulary

$$\tau_{digraph} := \{E\},$$

homomorphisms and isomorphism for digraphs are defined by the definition above. For bipartite digraphs $\mathcal{G} = (U^{\mathcal{G}}, W^{\mathcal{G}}, E^{\mathcal{G}})$ and $\mathcal{H} = (U^{\mathcal{H}}, W^{\mathcal{H}}, E^{\mathcal{H}})$ this follows in a similar manner, if we regard them as structures $\mathcal{A}_{\mathcal{G}} = (U^{\mathcal{G}} \dot\cup W^{\mathcal{G}}, U^{\mathcal{G}}, W^{\mathcal{G}}, E^{\mathcal{G}})$ and $\mathcal{A}_{\mathcal{H}} = (U^{\mathcal{H}} \dot\cup W^{\mathcal{H}}, U^{\mathcal{H}}, W^{\mathcal{H}}, E^{\mathcal{H}})$ over the vocabulary $\tau_{bi} := \{U, W, E\}$ with $ar(U) = ar(W) = 1$ and $ar(E) = 2$.

The first step of the reduction is shown in the next section. For classes $\mathbf{C}$ and $\mathbf{D}$ of graphs the problems we will deal with are defined as follows:

p-#Hom($\mathbf{C}$)
Instance: A digraph $\mathcal{A} \in \mathbf{C}$ and a digraph $\mathcal{B}$ *Parameter:* $\|\mathcal{A}\|$ *Problem:* Count the number of homomorphisms from $\mathcal{A}$ to $\mathcal{B}$

p-#Hom($\mathbf{C}, \mathbf{D}$)
Instance: A digraph $\mathcal{A} \in \mathbf{C}$ and a digraph $\mathcal{B} \in \mathbf{D}$ *Parameter:* $\|\mathcal{A}\|$ *Problem:* Count the number of homomorphisms from $\mathcal{A}$ to $\mathcal{B}$

5.1. The complexity of p-#Hom(C)

In this section we will see that p-#Hom($\mathbf{C}$) is #A[1]-hard for some special classes of digraphs. The proofs rely on some graph theoretic notions that originated in graph minor theory. We will begin with a brief overview of the necessary concepts and facts.

A *tree* $\mathcal{T} = (T, F)$ is a directed acyclic graph. Recall that *acyclic* means that $\mathcal{T}$ contains no cycle.

Definition 5.3. *Given a graph* $\mathcal{G} = (V, E)$*, a* tree-decomposition *of* $\mathcal{G}$ *is a pair* $(\mathcal{T}, (B_t)_{t \in \mathcal{T}})$ *such that* $\mathcal{T} = (T, F)$ *is a tree and* $(B_t)_{t \in \mathcal{T}}$ *a family of subsets of* V *satisfying the following conditions:*

1. *For every $v \in V$ the set $B^{-1}(v) := \{t \in T \mid v \in B_t\}$ is nonempty and connected in $\mathcal{T}$.*
2. *For every edge $\{v, w\} \in E$ there is a $t \in T$ such that $\{v, w\} \subseteq B_t$*

The sets B_t are the bags *of the tree-decomposition.*
We define the width *of a tree-decomposition $(\mathcal{T}, (B_t)_{t \in T})$ as $\max_{t \in T} |B_t| - 1$.*
The treewidth *$tw(\mathcal{G})$ of a graph $\mathcal{G}$ is the minimum width of all tree-decompositions of $\mathcal{G}$.*

A class **C** of graphs is of *bounded* treewidth, if there is a $w \in \mathbb{N}$ such that for every $\mathcal{G} \in \mathbf{C}$ we have $tw(\mathcal{G}) \leq w$. If a class **C** of graphs is not of bounded treewidth, we say that it is of *unbounded* treewidth.

Definition 5.4. *Let $\mathcal{G} = (V, E)$ and $\mathcal{H} = (H, F)$ be graphs. A* minor map *from $\mathcal{H}$ to $\mathcal{G}$ is a mapping $\mu : H \to 2^V$ with the following properties:*

(1) For every $v \in H$ the set $\mu(v)$ is nonempty and connected in $\mathcal{G}$.
(2) For all $v, w \in H$ with $v \neq w$ we have $\mu(v) \cap \mu(w) = \emptyset$.
(3) For all $\{v, w\} \in F$ there are $x \in \mu(v)$ and $y \in \mu(w)$ such that $\{x, y\} \in E$.

We say that μ is onto *if $\bigcup_{v \in H} \mu(v) = V$.*
Furthermore, $\mathcal{H}$ is called a minor *of $\mathcal{G}$ if and only if there is a minor map from $\mathcal{H}$ to $\mathcal{G}$.*

Note that "minor map" and "tree-decomposition" are notions that are defined exclusively for graphs. To obtain the corresponding notions for digraphs, we regard the graphs *underlying* these digraphs. Let $\mathcal{G} = (V, E)$ be a digraph and define $\mathcal{G}^* = (V, E^*)$ with $E^* := \{\{u, v\} \mid (u, v) \in E\}$. We call $\mathcal{G}^*$ the graph *underlying* $\mathcal{G}$. Then, minor maps and tree-decompositions for digraphs $\mathcal{G}$ and $\mathcal{H}$ are defined as minor maps and tree-decompositions for the underlying graphs.
The $(k \times l)$ *grid* is a graph $\mathcal{G} = ([k] \times [l], E)$ with

$$E := \{\{(i, j), (i', j')\} \mid (i, j), (i', j') \in [k] \times [l],\ |i - i'| + |j - j'| = 1\}$$

Grids play a central role in a deep theorem from graph minor theory. This so-called "Excluded Grid Theorem" will be needed in the following. As its proof is far beyond the scope of this work, we state it without proof.

Theorem 5.5 (Excluded Grid Theorem). *There is a computable function $w : \mathbb{N} \to \mathbb{N}$ such that the $(k \times k)$ grid is a minor of every graph of treewidth at least $w(k)$.*

5.1.1. Proving the Complexity

Theorem 5.6 (Dalmau, Jonsson). *If **C** is a recursively enumerable class of digraphs of unbounded treewidth, then* p-#Hom(**C**) *is #A[1]-hard under fpt Turing reductions.*

The proof given in [DJ04] relies on a construction by Grohe [FG06] who showed the analogous result for decision problems. We outline the construction and the proof, as some details of it are needed later on.

Let $\mathcal{A} = (A, E^{\mathcal{A}})$ be a connected digraph. Furthermore, let $k \geq 2$, $l := \binom{k}{2}$ and $\mu : [k] \times [l] \to 2^A$ a minor map from the $(k \times l)$ grid onto $\mathcal{A}$.

Fix a bijection $\beta : [l] \to \binom{[k]}{2}$.

Let $\mathcal{G} = (V, E)$ be a graph. We define a digraph $\mathcal{B} = \mathcal{B}(\mathcal{A}, \mu, \mathcal{G}) := (B, E^{\mathcal{B}})$ by

$$B := \{(v, e, i, p, a) \in V \times E \times [k] \times [l] \times A \mid (v \in e \leftrightarrow i \in \beta(p)), a \in \mu((i, p))\}.$$

Define a *projection* $\Pi : B \to A$ by $\Pi(v, e, i, p, a) := a$ for all $(v, e, i, p, a) \in B$. The edge relation $E^{\mathcal{B}}$ is constructed such that Π is a homomorphism from $\mathcal{B}$ to $\mathcal{A}$:

For all edges $(a_1, a_2) \in E^{\mathcal{A}}$, $E^{\mathcal{B}}$ contains all tuples $(b_1, b_2) \in \Pi^{-1}((a_1, a_2))$ with $b_1 = (v, e, i, p, a_1)$ and $b_2 = (v', e', i', p', a_2)$ such that:

(C1) if $i = i'$ then $v = v'$

(C2) if $p = p'$ then $e = e'$.

By the definition of Π it is easily observable that Π is a homomorphism from $\mathcal{B}$ to $\mathcal{A}$.

The proof of theorem 5.6 uses the following lemma. Its proof can be found in [DJ04].

Lemma 5.7. *The number of homomorphisms h from $\mathcal{A}$ to $\mathcal{B}$ such that $\Pi \circ h = \mathbf{id}$ equals the number of k cliques in $\mathcal{G}$ multiplied by $k!$.*

We are now able to comprehend the proof of the theorem.

Proof (of theorem 5.6). To prove the theorem it suffices to show that p-#Clique is fpt Turing reducible to p-#Hom($\mathbf{C}$) with $\mathbf{C}$ being a recursively enumerable class of digraphs of unbounded treewidth.

Let $(\mathcal{G}, k)$ be an instance of p-#Clique. We may assume that $k \geq 2$, and let $l := \binom{k}{2}$. By the Excluded Grid Theorem, there is some $\mathcal{A} \in \mathbf{C}$ such that the $(k \times l)$ grid is a minor of $\mathcal{A}$. With $\mathbf{C}$ being recursively enumerable, we can enumerate the elements of $\mathbf{C}$ until an appropriate $\mathcal{A} = \mathcal{A}(k)$ is found. That is, there is a minor map $\mu : [k] \times [l] \to A$ from the $(k \times l)$ grid to $\mathcal{A}$.

Let $\mathcal{A}_1, \ldots, \mathcal{A}_m$ be a list of all connected components of $\mathcal{A}$ and assume w.l.o.g. that μ is onto $\mathcal{A}_1$. Furthermore, let $\mathcal{B} = \mathcal{B}(\mathcal{A}, \mu, \mathcal{G})$ and Π be defined as above.

Define $\mathbf{N}$ to be the set of homomorphisms h from $\mathcal{A}_1$ to $\mathcal{B}$ with $\Pi \circ h = \mathbf{id}$. Note that, by lemma 5.7, we have $\#\text{clique}(\mathcal{G}, k) \cdot k! = |\mathbf{N}|$. Therefore, we have to show how to compute $|\mathbf{N}|$. To achieve this, we construct a digraph $\mathcal{D} = (D, E^{\mathcal{D}}) := \mathcal{B} \dot\cup \mathcal{A}_2 \dot\cup \ldots \dot\cup \mathcal{A}_m$.

Furthermore, we define a function $\Gamma : D \to A$ by

$$\Gamma(c) := \begin{cases} \Pi(c) & , \text{ if } c \in B \\ c & , \text{ otherwise} \end{cases}$$

It is easy to see that Γ defines a homomorphism from $\mathcal{D}$ to $\mathcal{A}$.

Let $\mathbf{N}'$ be the set of homomorphisms h from $\mathcal{A}$ to $\mathcal{D}$ satisfying $(\Gamma \circ h)(A) = A$ and define $\mathbf{I}$ as the set of all automorphisms of $\mathcal{A}$.

Claim 1. $|\mathbf{N}'| = |\mathbf{N}| \cdot |\mathbf{I}|$

Proof. First, we show that

$$\mathbf{N}' = \{f \circ g \mid f \in \mathbf{N},\ g \in \mathbf{I}\} \tag{5.1}$$

The backward direction is trivially true. For the forward direction, let $h \in \mathbf{N}'$ and define $g := \Gamma \circ h$. Then g and g^{-1} are automorphisms of $\mathcal{A}$. Furthermore $h \circ g^{-1}$ is a homomorphism from $\mathcal{A}$ to $\mathcal{D}$ and $\Gamma \circ h \circ g^{-1} = g \circ g^{-1} = \mathbf{id}$. Thus $h \circ g^{-1} \in \mathbf{N}$ and with $h \circ g^{-1} \circ g = h$ the proof of equation (5.1) is done.

To complete the proof of the claim, note that for every $f, f' \in \mathbf{N}$ and $g, g' \in \mathbf{I}$ we have that if either $f \neq f'$ or $g \neq g'$ then $f \circ g \neq f' \circ g'$. Thus, the claim follows. $\checkmark$

As $\mathbf{I}$ depends only on $\mathcal{A}$, the value of $|\mathbf{I}|$ can be computed efficiently. Therefore, to complete the proof of the theorem we need to show how to compute $|\mathbf{N}'|$.

Let $S \subseteq A$ and define m_S as the number of homomorphisms h from $\mathcal{A}$ to $\mathcal{D}$ such that $(\Gamma \circ h)(A) \subseteq S$. Let $\mathcal{D}|_{\Gamma^{-1}(S)}$ be the induced subgraph of $\mathcal{D}$ with vertex set $\Gamma^{-1}(S)$, then, clearly, m_S equals the number of homomorphisms from $\mathcal{A}$ to $\mathcal{D}|_{\Gamma^{-1}(S)}$. Thus m_S can be computed by a call to the p-#Hom($\mathbf{C}$) oracle with the instance $(\mathcal{A}, \mathcal{D}|_{\Gamma^{-1}(S)})$.

Finally, the value of $|\mathbf{N}'|$ can be computed by the principle of inclusion and exclusion:

$$|\mathbf{N}'| = \sum_{j=0}^{|A|} (-1)^j \sum_{\substack{S \subseteq A \\ |S|=j}} \#\hom(\mathcal{A}, \mathcal{D}|_{\Gamma^{-1}(S)}) \tag{5.2}$$

□

Note that in the proof by Dalmau and Jonsson this reduction was claimed to be parsimonious, but the application of the principle of inclusion and exclusion shows that it is not. So we have to stress that this is an fpt Turing reduction.

5.2. Counting Bipartite Cliques is #A[1]-complete

Let $\mathbf{C} := \{\, \vec{\mathcal{K}}_{k,l} \mid k, l \in \mathbb{N} \,\}$, to wit, $\mathbf{C}$ is the class of all trivially directed bipartite cliques. This class is easily seen to be of unbounded treewidth. Thus, theorem 5.6 holds for this class. Accordingly, for the given class $\mathbf{C}$, p-#Hom($\mathbf{C}$) is the problem of counting homomorphisms of bipartite cliques in arbitrary digraphs.

As we want to study the parameterized complexity of counting bipartite cliques in bipartite graphs, we have to make sure that the right hand side class of graphs can be restricted to bipartite digraphs.

Lemma 5.8. *Let* $\mathbf{C}$ *be the class of all trivially directed bipartite cliques and* $\mathbf{D}$ *the class of all trivially directed bipartite graphs. Then* p-#Hom($\mathbf{C}$,$\mathbf{D}$) *is #A[1]-hard.*

Proof. Note that, in the proof of theorem 5.6, for an $\mathcal{A} \in \mathbf{C}$ a graph $\mathcal{B} = (\mathcal{A}, \mu, \mathcal{G})$ is constructed. This graph is bipartite, as $\mathcal{A}$ is bipartite and trivially directed because $\mathcal{A}$ is. This is true, because the function Π is a homomorphism from $\mathcal{B}$ to $\mathcal{A}$.

Furthermore, as all $\mathcal{A} \in \mathbf{C}$ are connected the graph $\mathcal{D}$ constructed in the proof is identical to $\mathcal{B}$. And with all induced subgraphs of $\mathcal{D}$ being bipartite as well the oracle calls in equation (5.2) are valid for a p-#Hom($\mathbf{C}$,$\mathbf{D}$) oracle. □

Let $\mathcal{B} = (U, W, E)$ be a bipartite graph and call U the *first part* and W the *second part* of $\mathcal{B}$. For a function $f : A \to B$ let $im(f)$ denote the image of f. For a homomorphism h from a bipartite graph $\mathcal{B} = (U, W, E)$ to any graph, call $im_1(h) = \{y \mid h(u) = y, u \in U\}$ the image of the first part under h and define $im_2(h)$ analogously for the second part.

Before we can do the final reduction to obtain the desired result about the complexity of p-#BipartiteClique we perform some intermediate steps. For the first step consider the following problem:

p-#DiBipartiteClique
Instance: A trivially directed bipartite graph $\mathcal{B} = (U, W, F)$ and $k, l \in \mathbb{N}$
Parameter: k, l
Problem: Compute the number of $\vec{\mathcal{K}}_{k,l}$ in $\mathcal{B}$

Call an r-tuple $\bar{a} = (a_1, \ldots, a_r) \in [k]^r$ an *r-part partition* of $k \in \mathbb{N}$, or shortly a (k, r)-partition, if

$$\sum_{i=1}^{r} a_i = k.$$

Let $P(k, r)$ denote the set of all (k, r)-partitions. Observe that the cardinality of $P(k, r)$ depends only on k and r.

Lemma 5.9. *Let* $\mathbf{C}$ *be the class of all trivially directed bipartite cliques and* $\mathbf{D}$ *the class of all trivially directed bipartite graphs. Then*

$$\text{p-\#Hom}(\mathbf{C}, \mathbf{D}) \leq^{fpt-T} \text{p-\#DiBipartiteClique}$$

Proof. Let $(\vec{\mathcal{K}}_{k,l}, \mathcal{B})$ be an instance of p-#Hom$(\mathbf{C}, \mathbf{D})$ with $\mathcal{B} = (U, W, E)$. We have to show how to compute $|\mathrm{HOM}(\vec{\mathcal{K}}_{k,l}, \mathcal{B})|$ using a p-#DiBipartiteClique oracle.

For $r \in [k]$ and $s \in [l]$ define

$$\mathbf{T}_{r,s} := \{h \in \mathrm{HOM}(\vec{\mathcal{K}}_{k,l}, \mathcal{B}) : \ |im_1(h)| = r \wedge |im_2(h)| = s\}$$

Clearly, for any homomorphism h from $\vec{\mathcal{K}}_{k,l}$ to $\mathcal{B}$ there are unique $r \in [k]$ and $s \in [l]$ such that $h \in \mathbf{T}_{r,s}$. Thus the following observation holds.

Observation. The family $\{\mathbf{T}_{r,s} \mid r \in [k],\, s \in [l]\}$ is a partition of $\mathrm{HOM}(\mathcal{K}_{k,l}, \mathcal{B})$.

This directly implies:

$$|\mathrm{HOM}(\vec{\mathcal{K}}_{k,l}, \mathcal{B})| \;=\; \sum_{\substack{r\in[k],\\ s\in[l]}} |\mathbf{T}_{r,s}| \tag{5.3}$$

As there are $k \cdot l$ different classes $\mathbf{T}_{r,s}$ we can compute $|\mathrm{HOM}(\vec{\mathcal{K}}_{k,l}, \mathcal{B})|$ in fpt time if we can compute $|\mathbf{T}_{r,s}|$ for every $r \in [k]$ and $s \in [l]$ in fpt time. Hence to complete the proof, we need to show how to determine $|\mathbf{T}_{r,s}|$.

Fix $r \in [k]$, $s \in [l]$ and let $h \in T_{r,s}$. Note that the image of h is some bipartite clique $\vec{\mathcal{K}}^{\mathcal{B}}_{r,s} = (U^{\mathcal{B}}_r, W^{\mathcal{B}}_s, U^{\mathcal{B}}_r \times W^{\mathcal{B}}_s)$ in $\mathcal{B}$. Note furthermore, that as all graphs here are trivially directed, under every homomorphism the first part of $\vec{\mathcal{K}}_{k,l}$ always maps to the first part of $\mathcal{B}$ and the same holds for the second parts.

W.l.o.g. let $U^{\mathcal{B}}_r = \{u_1, \ldots, u_r\}$ and $W^{\mathcal{B}}_s = \{w_1, \ldots, w_s\}$. For all $i \in [r]$ define

$$y_i = |h^{-1}(u_i)|. \tag{5.4}$$

and for all $j \in [s]$

$$z_j = |h^{-1}(w_j)|. \tag{5.5}$$

Hence, for every $g \in \mathbf{T}_{r,s}$ there are unique $\bar{y} = (y_1, \ldots, y_r)$ and $\bar{z} = (z_1, \ldots, z_s)$ that satisfy equations (5.4) and (5.5). Observe that by these equations we have $\bar{y} \in P(k, r)$ and $\bar{z} \in P(l, s)$.

Let $\mathbf{T}_{r,s}(\bar{y}, \bar{z}) \subseteq T_{r,s}$ denote the set of homomorphisms $g \in T_{r,s}$ that are represented by $\bar{y}$ and $\bar{z}$ via the above mentioned correspondence.

Thus, we can derive another observation

Observation. The family $\{\mathbf{T}_{r,s}(\bar{y}, \bar{z}) \mid \bar{y} \in P(k, r),\, \bar{z} \in P(l, s)\}$ is a partition of $\mathbf{T}_{r,s}$.

which implies

$$|\mathbf{T}_{r,s}| \;=\; \sum_{\substack{\bar{y}\in P(k,r),\\ \bar{z}\in P(l,s)}} |\mathbf{T}_{r,s}(\bar{y}, \bar{z})| \tag{5.6}$$

As it is easy to see that there is a computable function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $|P(k, r)| = f(k, r)$ and $|P(l, s)| = f(l, s)$ the proof is complete if we can show how to compute $|\mathbf{T}_{r,s}(\bar{y}, \bar{z})|$ for $\bar{y} \in P(k, r)$ and $\bar{z} \in P(l, s)$.

Let $\bar{y} = (y_1, \ldots, y_r) \in P(k, r)$ and $\bar{z} = (z_1, \ldots, z_s) \in P(l, s)$ and for $i \in [r]$ define

$$\sigma_i := \sum_{j=1}^{i-1} y_j.$$

Analogously, for $i \in [s]$ define

$$\tau_i := \sum_{j=1}^{i-1} z_j.$$

Let $c_{r,s}$ denote the number of trivially directed bipartite cliques $\vec{\mathcal{K}}_{r,s}$ in $\mathcal{B}$.

Claim 1. For $\bar{y} = (y_1, \ldots, y_r) \in P(k, r)$ and $\bar{z} = (z_1, \ldots, z_s) \in P(l, s)$ we have

$$|\mathbf{T}_{r,s}(\bar{y}, \bar{z})| = c_{r,s} \cdot \prod_{i=1}^{r} \binom{k - \sigma_i}{y_i} \cdot \prod_{i=1}^{s} \binom{l - \tau_i}{z_i}. \tag{5.7}$$

Proof. The first factor $c_{r,s}$ on the right hand side accounts for the easily observable fact that for every copy of $\vec{\mathcal{K}}_{r,s}$ in $\mathcal{B}$ the number of homomorphisms from $\vec{\mathcal{K}}_{k,l}$ to this copy of $\vec{\mathcal{K}}_{r,s}$ is the same. Hence we may consider the homomorphisms in $\mathbf{T}_{r,s}(\bar{y}, \bar{z})$ independently of their actual image in $\mathcal{B}$, that is, we regard all homomorphisms $g \in \mathbf{T}_{r,s}(\bar{y}, \bar{z})$ as mappings from $\vec{\mathcal{K}}_{k,l} = ([k], [l], [k] \times [l])$ to $\vec{\mathcal{K}}_{r,s} = ([r], [s], [r] \times [s])$.

Thus creating these homomorphisms $g \in \mathbf{T}_{r,s}(\bar{y}, \bar{z})$ can be modelled in two steps.

Firstly, mapping $[l]$ to $[s]$ is modelled by putting l balls into s bins with sizes $z_1, \ldots, z_s$. This contributes the last product of the above mentioned term.

Secondly, mapping $[k]$ to $[r]$ can be modelled analogously by k balls and r bins of sizes $y_1, \ldots, y_r$. This contributes the last but one product above. ✓

Consider the right hand side of equation (5.7). Except for the value $c_{r,s}$ all values depend only on k and l and can be computed in time bounded by some function of these parameters. For every $r \in [k]$ and $s \in [l]$ the value $c_{r,s}$ can be computed via an oracle call. This completes the proof. □

Before we do the final reduction, we introduce yet another problem.

p-#DiSymmBipartiteClique
Instance: A trivially directed bipartite graph $\mathcal{B} = (U, W, F)$ and $k \in \mathbb{N}$ *Parameter:* k *Problem:* Compute the number of $\vec{\mathcal{K}}_{k,k}$ in $\mathcal{B}$

Lemma 5.10. p-#DiBipartiteClique $\leq^{fpt-T}$ p-#DiSymmBipartiteClique

Proof. Let $(\mathcal{B}, k, l)$ with $\mathcal{B} = (U, W, F)$ be an instance of p-#DiBipartiteClique. As the case that $k = l$ is trivial and the cases $k < l$ and $k > l$ can be handled symmetrically, it suffices to give a proof for $k > l$.

We define a graph $\mathcal{B}_m = (U, W_m, F_m)$ by

$$\begin{aligned} W_m &:= W \mathbin{\dot\cup} \{w_1, \ldots, w_m\} \\ F_m &:= F \cup \{(u, w_j) \mid j \in [m],\, u \in U\} \end{aligned}$$

Set $\mathcal{B}_0 := \mathcal{B}$ and let $x_{r,s}$ denote the number of $\vec{\mathcal{K}}_{r,s}$ in $\mathcal{B}$. Define $c_k(m)$ as the number of $\vec{\mathcal{K}}_{k,k}$ in $\mathcal{B}_m$. Then we can easily derive the following correspondence:

$$c_k(m) \;=\; x_{k,k} + \sum_{j=1}^{m} \binom{m}{j} x_{k,k-j}$$

Define $a_{ij} := \binom{i}{j}$. For $m = 0, \ldots, k-l$ we obtain the following system of linear equations:

$$\begin{pmatrix} c_k(0) \\ c_k(1) \\ \vdots \\ c_k(k-l) \end{pmatrix} = \begin{pmatrix} a_{11} & 0 & \ldots & 0 \\ a_{21} & a_{22} & \ldots & 0 \\ & \vdots & \ddots & \\ a_{(k-l)1} & a_{(k-l)2} & \ldots & a_{(k-l)(k-l)} \end{pmatrix} \cdot \begin{pmatrix} x_{k,k} \\ \vdots \\ x_{k,l} \end{pmatrix}$$

Note that for $j > i$ we have $\binom{i}{j} = 0 = a_{ij}$ and for $1 \leq j \leq i$ we have $\binom{i}{j} = a_{ij} > 0$ therefore the matrix above is lower triangular. Hence, we can obtain the value $x_{k,l}$ by solving this system of linear equations. □

Theorem 5.11. p-#BipartiteClique *is #A[1]-complete under fpt Turing reductions.*

The theorem follows by three simple lemmas which utilize the problem of counting symmetric bipartite cliques.

p-#SymmetricBipartiteClique
Instance: A bipartite graph $\mathcal{B} = (U, W, F)$ and $k \in \mathbb{N}$ *Parameter:* k *Problem:* Compute the number of $\mathcal{K}_{k,k}$ in $\mathcal{B}$

Lemma 5.12.

$$\text{p-\#DiSymmBipartiteClique} \leq^{fpt} \text{p-\#SymmetricBipartiteClique}$$

Proof. For an instance $(\mathcal{B}, k)$ with $\mathcal{B} = (U, W, F)$ of p-#DiSymmBipartiteClique define $\mathcal{B}^* = (U, W, F^*)$ with $F^* = \{\{u, w\} \mid (u, w) \in F\}$. It is easy to see that the number of $\vec{\mathcal{K}}_{k,k}$ in $\mathcal{B}$ equals the number of $\mathcal{K}_{k,k}$ in $\mathcal{B}^*$. □

Lemma 5.13. p-#SymmetricBipartiteClique $\leq^{fpt}$ p-#BipartiteClique

Proof. This is almost trivial by the mapping $(\mathcal{B}, k) \mapsto (\mathcal{B}, k, k)$ for an instance $(\mathcal{B}, k)$ of p-#SymmetricBipartiteClique. □

Lemma 5.14. p-#BipartiteClique $\in \#A[1]$.

Proof. Let $(\mathcal{B}, k, l)$ with $\mathcal{B} = (U, W, F)$ be an instance of p-#BipartiteClique. We shall give an fpt-T! reduction to p-#MC(Π_0).
Define a structure $\mathcal{A} = (A, R^{\mathcal{A}})$ with $ar(R) = 2$, $A := U \cup W$ and

$$R^{\mathcal{A}} := \{(u, w) \mid u \in U,\ w \in W,\ \{u, w\} \in F\}.$$

Define

$$\varphi(x_1,\ldots,x_k,y_1,\ldots,y_l) \;:=\; \bigwedge_{1\le i<j\le k}(x_i\neq x_j)\wedge\bigwedge_{1\le i<j\le l}(y_i\neq y_j) \\ \wedge\left(\bigwedge_{1\le i\le k}\bigwedge_{1\le j\le l}Rx_iy_j\vee\bigwedge_{1\le i\le k}\bigwedge_{1\le j\le l}Ry_jx_i\right)$$

One can easily see that $|\varphi(\mathcal{A})|$ equals the number of $\mathcal{K}_{k,l}$ in $\mathcal{B}$ times a factor of $k!l!$. We can correct this factor after the oracle call to obtain the desired value.□

Observe that lemma 5.13 and lemma 5.14 together imply the containment of p-#SymmetricBipartiteClique in #A[1] and hence this problem is #A[1]-complete under parameterized Turing reductions.

Chapter 6.

Counting Induced Cycles and Paths

In [FG06] Flum and Grohe have shown that counting cycles and paths is #A[1]-complete under fpt Turing reductions. In this chapter, we will show the analogous results for induced cycles and paths.

We define $\mathcal{C}_k$ as the cycle on k vertices that is $\mathcal{C}_k = ([k], E)$ with

$$E := \{\{i, j\} \mid j \equiv i + 1 \,(mod\, k),\ i, j \in [k]\}.$$

Recall that for a graph $\mathcal{G} = (V, E)$ and a set of vertices $U \subseteq V$ the *subgraph of* $\mathcal{G}$ *induced by* U is defined as $\mathcal{G}|_U = (U, E \cap \binom{U}{2})$.

The problem we are interested in is defined as follows:

p-#InducedCycle
Instance: A graph $\mathcal{G} = (V, E)$, $k \in \mathbb{N}$ *Parameter:* k *Problem:* Compute the number of induced subgraphs in $\mathcal{G}$ that are isomorphic to $\mathcal{C}_k$

For a graph $\mathcal{G} = (V, E)$ its *complement* is defined by $\mathcal{G}^c = (V, \binom{V}{2} \setminus E)$. Hence, by $\mathcal{C}_k^c$ we denote the complement of the cycle on k vertices.

We will demonstrate the #A[1]-hardness of p-#InducedCycle by means of its connection to the following problem.

p-#InducedCycleComplement
Instance: A graph $\mathcal{G} = (V, E)$, $k \in \mathbb{N}$ *Parameter:* k *Problem:* Compute the number of induced subgraphs in $\mathcal{G}$ that are isomorphic to $\mathcal{C}_k^c$

Lemma 6.1. p-#InducedCycle $\equiv^{fpt}$ p-#InducedCycleComplement

Proof. The bijective mapping $(\mathcal{G}, k) \mapsto (\mathcal{G}^c, k)$ satisfies this. □

As we are dealing with complements $\mathcal{C}_k^c$ of cycles it will be convenient to consider *cyclic orders* on the vertices of these complements. A *cyclic order* of a set S of cardinality k is defined by a function $\sigma : S \to S$ satisfying:

- For each $x \in S$, by defining $x_{+0} := x$ and $x_{+i+1} := \sigma(x_{+i})$, we have

$$S = \{x_{+0}, x_{+1}, \ldots, x_{+k-1}\} \text{ and } x = \sigma(x_{+k-1}).$$

Functions of this type are called *successor functions*. We consider cyclic orders exclusively by means of their successor function. In fact, we will treat them interchangeably. Hence, we say that two cyclic orders are *identical* if their successor functions are identical and they are *inverse* if their corresponding successor functions σ and τ satisfy $\sigma^{-1} = \tau$. Somewhat sloppily, we say that two elements $x, y \in S$ are *successors* if $x = \sigma(y)$ or $y = \sigma(x)$.

Note that there are $(k-1)!$ cyclic orders on a set of cardinality k. Furthermore, for a complement $\mathcal{C}_k^c = ([k], \bar{E})$ of a cycle $\mathcal{C}_k = ([k], E)$ there are exactly two successor functions σ and σ^{-1}, such that for every $x \in [k]$ we have $\{x_{+i}, x_{+j}\} \in E$ if, and only if, $j - i \equiv 1 \,(mod\, k)$. In this case, we say that σ and its inverse *define* $\mathcal{C}_k^c$ (and $\mathcal{C}_k$).

Lemma 6.2. p-#Clique $\leq^{fpt-T}$ p-#InducedCycleComplement

Proof. Let $(\mathcal{G}, k)$ with $\mathcal{G} = (V, E)$ be an instance of p-#Clique. As the case $k \leq 2$ is trivial, we may assume that $k > 2$.

We construct a Graph $\mathcal{G}' = (V \dot\cup W, E \cup F)$ where:

$$\begin{aligned} W &:= \{x_e \mid e \in E\} \\ F &:= \binom{W}{2} \cup \{\{v, x_e\} \mid v \in V,\, x_e \in W,\, v \notin e\} \end{aligned}$$

Observe that the newly added vertices form a clique and each $x_e \in W$ satisfies

$$N(x_e) \cap V = V \setminus e. \tag{6.1}$$

Let $z_{a,i}$ denote the number of sets C which induce a copy of $\mathcal{C}_a^c$ in $\mathcal{G}'$ such that $|C \cap V| = a - i$ and $|C \cap W| = i$.

Claim 1. The number of k-cliques in $\mathcal{G}$ equals $\dfrac{2}{(k-1)!} \cdot z_{2k,k}$.

Proof. For the forward direction, let K be a set of vertices that induces a k-clique in $\mathcal{G}$. Let τ be a cyclic order on K and let $v \in K$, we consider $K = \{v_{+0}, \ldots, v_{+k-1}\}$.

By equation (6.1) there is, for each edge $e_i = \{v_{+i}, v_{+j}\}$ with $j \equiv i + 1 \,(mod\, k)$, exactly one vertex $x_e \in W$ that is adjacent to all vertices in K except for v_{+i} and v_{+j}. Thus for the set $C := \{v_{+0}, x_{e_0}, v_{+1}, \ldots, x_{e_{k-2}}, v_{+k-1}, x_{e_{k-1}}\}$ the function $\sigma : C \to C$ defined by

$$\sigma(v) := \begin{cases} x_{e_i} & \text{, if } v = v_{+i} \in K \\ v_{+j} & \text{, if } v = x_{e_i},\ i \in [k],\ j \equiv i + 1 \,(mod\, k) \end{cases} \tag{6.2}$$

defines a copy of C_{2k}^c in $\mathcal{G}'$. Furthermore, each cyclic order of K and its inverse induce the same C_{2k}^c in $\mathcal{G}'$.

For the backward direction, let C with $|C \cap W| = k$ induce a copy of C^c_{2k} in $\mathcal{G}'$. Hence, there is a cyclic order σ on the vertices in C such that two vertices $x, y \in C$ are not adjacent iff they are successors in σ. As all vertices $x, y \in W$ are adjacent, this implies that they may not be successors in the order. Let $v \in C \cap V$, then we have that two vertices $x, y \in C \cap V = \{v_{+0}, v_{+2}, \ldots, v_{+2(k-1)}\}$, are not successors in σ. Thus $C \cap V$ forms a k-clique in $\mathcal{G}'$ and hence in $\mathcal{G}$. ✓

By this claim, it suffices to show how to obtain the value $z_{2k,k}$. In order to do this, we define the graph $\mathcal{G}_m = (V \dot\cup W_m, E \cup F_m)$ with

$$\begin{aligned} W_m &:= W \times [m] \\ F_m &:= \binom{W_m}{2} \cup \{\{v, (x_e, j)\} \mid v \in V,\, x_e \in W,\, j \in [m],\, v \notin e\} \end{aligned}$$

Furthermore, we define a *projection* $\pi : V \dot\cup W_m \to V \dot\cup W$ that maps vertices in $\mathcal{G}_m$ to vertices in $\mathcal{G}'$:

$$\pi(v) := \begin{cases} v & , \text{ if } v \in V \\ y & , \text{ if } v = (y, i) \in W_m \end{cases} \tag{6.3}$$

Likewise, for every set of vertices $C \subseteq V \dot\cup W_m$ we define its projection by

$$\pi(C) := \{\pi(v) \mid v \in C\}.$$

Claim 2. Let C induce a copy of C^c_{2k} in $\mathcal{G}_m$ then $\pi(C)$ induces a copy of C^c_{2k} in $\mathcal{G}'$.

Proof. Note that, by the construction of $\mathcal{G}_m$ it suffices to show that $|C| = |\pi(C)|$. Assume for contradiction that C contains a pair of vertices $(y, i), (y, j) \in W_m$ with $i \neq j$. This implies that $\pi(y, i) = \pi(y, j) = y$.

Consider a cyclic order σ on the vertices of C that defines this copy of C^c_{2k} in $\mathcal{G}_m$. Then we can assume that

$$C = \{v_{+0}, \ldots, v_{+a-1}, v_{+a}, v_{+a+1}, \ldots, v_{+b-1}, v_{+b}, v_{+b+1}, \ldots, v_{+2k-1}\}$$

for a $v \in S$ such that $v_{+a} = (y, i)$ and $v_{+b} = (y, j)$. Note that, as σ defines the cycle complement, we have that $\{v_{+a-1}, v_{+a+1}, v_{+b-1}, v_{+b+1}\} \subseteq V$. $k \geq 3$ implies that either $v_{+a+1} \neq v_{+b-1}$ or $v_{+a-1} \neq v_{+b+1}$ holds.

Let $v_{+a+1} \neq v_{+b-1}$ and assume that $y = x_e$ for $e = \{v_{+b-1}, v_{+b+1}\}$. Then $v_{+a} = (y, i)$ is adjacent to v_{+a+1} in contradiction to our assumption that σ defined the cycle complement.

The other cases are analogous. ✓

Claim 3. Let C induce a copy of C^c_{2k} in $\mathcal{G}$ with $|C \cap W| = i$. Then there are m^i sets D in $\mathcal{G}_m$ with $\pi(D) = C$ which induce a copy of C^c_{2k} in $\mathcal{G}_m$.

Proof. Let C be as above and let $v \in C \cap W$ by the definition of π there are m distinct vertices in $w \in W_m$ with $\pi(w) = v$. As this holds independently for all of the i vertices in $C \cap W$, the claim follows. ✓

Claim 4. For every set of vertices C that induces a copy of C^c_{2k} in $\mathcal{G}_m$ we have $|C \cap W_m| \leq k$.

To see this, observe that if $|C \cap W_m| > k$ then in every cyclic order of the vertices in C at least two vertices $x, y \in C \cap W_m$ are successors. By the definition of $\mathcal{G}_m$, $\{x, y\}$ is a edge in F_m. Thus, C is not an induced cycle complement. Contradiction.

Define $c(2k, m)$ as the number of C^c_{2k} in $\mathcal{G}_m$ and note that claim 4 implies $z_{2k,j} = 0$ for all $j > k$. Together with claims 2 and 3 we immediately obtain:

$$c(2k, m) = \sum_{i=0}^{k} z_{2k,i} \cdot m^i \tag{6.4}$$

This is a system of linear equations and if we consider all $m \in [k]$, we obtain

$$\begin{pmatrix} c(2k,1) \\ \vdots \\ c(2k,k) \end{pmatrix} = \begin{pmatrix} 1^1 & 1^2 & \dots & 1^k \\ & \vdots & & \\ k^1 & k^2 & \dots & k^k \end{pmatrix} \cdot \begin{pmatrix} z_{2k,1} \\ \vdots \\ z_{2k,k} \end{pmatrix}$$

The matrix of our system of linear equations is the transpose of a Vandermonde matrix. As Vandermonde matrices are well-known to be non-singular, we can obtain its inverse and thus, compute all values $z_{2k,i}$ for $i \in [k]$. □

Corollary 6.3. *The problem* p-#InducedCycle *is #A[1]-hard under fpt Turing reductions.*

Define $\mathcal{P}_k := ([k+1], \{\{i, j\} \mid i, j \in [k+1] \wedge j - i = 1\})$ as the *path of length k*.

p-#InducedPath
Instance: A graph $\mathcal{G} = (V, E)$, $k \in \mathbb{N}$ *Parameter:* k *Problem:* Compute the number of induced copies of $\mathcal{P}_k$ in $\mathcal{G}$

To prove the hardness, we show that a result from [FG06] (Lemma 14.33) for the not necessarily induced case can be adapted without much effort.

Lemma 6.4. p-#InducedCycle $\leq^{fpt-T}$ p-#InducedPath

Proof. Consider the following problem:

p-#InducedCycleThroughEdge
Instance: A graph $\mathcal{G} = (V, E)$, an edge $e \in E$,$k \in \mathbb{N}$ *Parameter:* k *Problem:* Compute the number of induced copies of $\mathcal{C}_k$ in $\mathcal{G}$ that contain the edge e

We show first that the following holds

$$\text{p-\#InducedCycle} \leq^{fpt-T} \text{p-\#InducedCycleThroughEdge}$$

Let $(\mathcal{G}, k)$ be an instance of p-#InducedCycle with $\mathcal{G} = (V, E)$. An algorithm witnessing the reduction chooses an $e \in E$ arbitrarily. By an oracle call to p-#InducedCycleThroughEdge it computes the number z_1 of length k induced cycles containing e. Then, recursively the number z_2 of length k induced cycles in $(V, E \setminus \{e\})$ is computed. Thus the number of induced cycles of length k in $\mathcal{G}$ is $z_1 + z_2$.

Now, the proof is complete if we can show that

$$\text{p-\#InducedCycleThroughEdge} \leq^{fpt-T} \text{p-\#InducedPath}$$

holds. Consider an instance consisting of $\mathcal{G} = (V, E)$, $e = \{u, w\} \in E$ and $k \in \mathbb{N}$. Furthermore, we assume that $k \geq 3$.

We define a graph $\mathcal{G}_e(l, m) = (V_{l,m}, E_{l,m})$ as follows:

$$\begin{aligned} V_{l,m} &:= V \dot\cup \{v_1, \ldots, v_l\} \dot\cup \{w_1, \ldots, w_m\} \\ E_{l,m} &:= E \setminus \{e\} \cup \{\{v, w_j\} \mid j \in [m]\} \cup \{\{w, v_i\} \mid i \in [l]\} \end{aligned}$$

Note that in $\mathcal{G}_e(l, m)$ the newly added vertices can only be endpoints of paths and as $k \geq 3$ each of the sets $\{v_1, \ldots, v_l\}$ and $\{w_1, \ldots, w_m\}$ can contain only one endpoint of a length k path.

Furthermore, as e is not contained in $\mathcal{G}_e(l, m)$ induced paths of length $k+1$ with both endpoints among the v_i and w_j correspond to induced cycles of length k in $\mathcal{G}$. Thus in $\mathcal{G}_e(1, 1)$ the number of induced paths of length $k + 1$ that have as endpoints v_1 and w_1 equals the number of length k induced cycles in $\mathcal{G}$.

Define, for paths of length $k + 1$ in $\mathcal{G}_e(1, 1)$:

- x_1 as the number of induced paths with endpoints v_1, w_1
- x_2 as the number of induced paths that contain v_1 but not w_1
- x_3 as the number of induced paths that contain w_1 but not v_1
- x_4 as the number of induced paths that contain neither w_1 nor v_1

Let $y_{l,m}$ be the number of length $k+1$ induced paths in $\mathcal{G}_e(l, m)$. Hence, we have

$$y_{l,m} = l \cdot m \cdot x_1 + l \cdot x_2 + m \cdot x_3 + x_4$$

and for $0 \leq l, m \leq 1$ we obtain the following system of linear equations:

$$\begin{pmatrix} y_{0,0} \\ y_{0,1} \\ y_{1,0} \\ y_{1,1} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

with a nonsingular matrix and hence, we can obtain the desired value x_1. □

Lemma 6.5. p-#InducedPath *and* p-#InducedCycle *are contained in* #*A[1].*

Proof. Let $\mathcal{G} = (V, E)$ and $k \in \mathbb{N}$. We assume that $k \geq 3$ and define a structure $\mathcal{A} := (V, E, \leq^{\mathcal{A}})$ with a total order $\leq^{\mathcal{A}}$ on V. We show how to reduce p-#InducedCycle with a single oracle call to p-#MC(Π_0). Consider the Π_0 formula:

$$\begin{aligned} \chi_k(x_1, \ldots, x_k) \quad := \quad & \bigwedge_{1 \leq i < j \leq k} x_i \neq x_j \\ & \wedge \bigwedge_{\substack{i,j \in [k] \\ j \equiv i+1 \,(mod\, k)}} Ex_ix_j \wedge \bigwedge_{\substack{i,j \in [k] \\ \neg j \equiv i+1 \,(mod\, k)}} \neg Ex_ix_j \end{aligned}$$

It is easy to see that we have $\bar{v} = (v_1, \ldots, v_k) \in \chi_k(\mathcal{A})$ for all tuples that, in this order, induce a cycle in $\mathcal{G}$. Note that this would not directly yield a parsimonious reduction as there are always $2k$ tuples inducing the same cycle. For our fpt-T! reduction, however, it suffices to note that we can correct this value by a simple integer division after the oracle call.

For p-#InducedPath the proof is analogous. □

Chapter 7.

A Digression: Further Tractability Results

In the remainder of this thesis we will make an effort to summarize some classification results from parameterized counting complexity and compare them to their analogs in classical complexity. This summary will be given in the following chapter. In this chapter we will mainly discuss some further tractability results that will prove illustrative later on.

In [Fri01] an fpt algorithm was given for counting independent *tuples* in digraphs with low degree. However, an explicit time bound for this algorithm was not mentioned there. We will see how the algorithm can be applied to counting independent *sets* in undirected graphs and the time bound of this algorithm will be given.

As we are acquainted with cliques, we can define *independent sets* via their connection to cliques. Given a graph $\mathcal{G} = (V, E)$, a set $S \subseteq V$ is an *independent set* in $\mathcal{G}$ if and only if it induces a clique in $\mathcal{G}^c$ (where $\mathcal{G}^c$ is the edge-complement of $\mathcal{G}$).

First note the definition of the counting independent set problem, considered here:

p-Δ-#IndependentSet
Instance: A graph $\mathcal{G} = (V, E)$ and $k \in \mathbb{N}$ *Parameter:* $k + \Delta(\mathcal{G})$ *Problem:* Compute the number of independent sets of size k in $\mathcal{G}$

In the same manner as for vertex cover and hitting set, let $\#is(\mathcal{G}, k)$ denote the number of independent sets of cardinality k in $\mathcal{G}$.

We need some new notation. For two graphs $\mathcal{A} = (V, E)$ and $\mathcal{B} = (V, F)$ with the same vertex set we write $\mathcal{A} \subset \mathcal{B}$ if $E \subset F$, i.e. if E is a proper subset of F.

Furthermore, given a graph $\mathcal{G} = (V, E)$ we define the *r-neighborhood* of a vertex $v \in V$ inductively:

$$\begin{aligned} N^1_{\mathcal{G}}(v) &:= N_{\mathcal{G}}(v) \cup \{v\} \\ N^{i+1}_{\mathcal{G}}(v) &:= \bigcup_{w \in N^i_{\mathcal{G}}(v)} N_{\mathcal{G}}(w) \end{aligned}$$

Theorem 7.1. p-Δ-#IndependentSet *is fixed parameter tractable. More precisely, there is an algorithm that solves* p-Δ-#IndependentSet *in time*

$$\mathcal{O}(k^4 \cdot (2 \cdot (d+1))^{k^2} \cdot |V| + 2^{2k^2}).$$

Proof. Given an instance $(\mathcal{G}, k)$ of p-Δ-#IndependentSet with $\mathcal{G} = (V, E)$, $k \in \mathbb{N}$ and $\Delta(\mathcal{G}) = d$. Assume w.l.o.g. that $V = \{v_1, \ldots, v_n\}$. We define a digraph $\vec{\mathcal{G}} = (V, \vec{E})$ by

$$\vec{E} := \{(v_i, v_j) \mid \{v_i, v_j\} \in E \ \wedge \ i, j \in [n] : \ i < j\} \cup \{(v, v) \mid v \in V\}.$$

Note that we have $\Delta(\vec{\mathcal{G}}) = d + 1$ and $\vec{\mathcal{G}}$ can be computed in time $\mathcal{O}((d+1) \cdot |V|)$. Let $\Gamma_k = \{\gamma_0, \ldots, \gamma_m\}$ be the set of all digraphs with self-loops allowed and with vertex set $[k]$, where $\gamma_0 = ([k], \emptyset)$ denotes the empty graph. For a digraph $\mathcal{H} = ([k], F)$, we define the sets

$$Hom(\mathcal{H}) := \{\bar{v} = (v_1, \ldots, v_k) \in V^k \mid \forall\, i, j \in [k] : \ (i, j) \in F \Rightarrow (v_i, v_j) \in \vec{E}\}$$

and

$$StHom(\mathcal{H}) := \{\bar{v} = (v_1, \ldots, v_k) \in V^k \mid \forall\, i, j \in [k] : \ (i, j) \in F \Leftrightarrow (v_i, v_j) \in \vec{E}\}$$

and let $h_{\mathcal{H}} := |Hom(\mathcal{H})|$ and $s_{\mathcal{H}} := |StHom(\mathcal{H})|$.

Intuitively, $Hom(\mathcal{H})$ is the set of all tuples that define homomorphisms from $\mathcal{H}$ to $\vec{\mathcal{G}}$ and analogously elements of $StHom(\mathcal{H})$ define strong homomorphisms from $\mathcal{H}$ to $\vec{\mathcal{G}}$.

As $\vec{\mathcal{G}}$ contains a self-loop for every $v \in V$ the value s_{γ_0} equals the number of k-tuples $\bar{v} = (v_1, \ldots, v_k)$ that are independent in such a way that $\{v_1, \ldots, v_k\}$ is an independent set of cardinality *exactly* k in $\vec{\mathcal{G}}$. This implies

$$\#is(\mathcal{G}, k) = \frac{1}{k!} \cdot s_{\gamma_0} \tag{7.1}$$

Thus we have to show how s_{γ_0} can be computed.

Claim 1. Given a digraph $\gamma \in \Gamma_k$ and $\vec{\mathcal{G}} = (V, \vec{E})$ as above, then for every tuple $\bar{v} = (v_1, \ldots, v_k) \in V^k$ the following holds:

$$\bar{v} \in StHom(\gamma) \ \Leftrightarrow \ \bar{v} \in Hom(\gamma) \ \wedge \bigwedge_{\substack{\gamma' \in \Gamma_k \\ \gamma' \supset \gamma}} \bar{v} \notin Hom(\gamma') \tag{7.2}$$

Proof. For the forward direction, let $\bar{v} \in StHom(\gamma)$ for $\gamma = ([k], F) \in \Gamma_k$. Then $\bar{v} \in Hom(\gamma)$ is true by definition, but assume, for contradiction, that there is a $\gamma' = ([k], F') \in \Gamma_k$ with $\gamma \subset \gamma'$ and $\bar{v} \in Hom(\gamma')$. Thus there is an edge $e = (i, j) \in F' \setminus F$ such that $(v_i, v_j) \in \vec{E}$. But as $\bar{v} \in StHom(\gamma)$ and $e \notin F$ we have $(v_i, v_j) \notin \vec{E}$.

The backward direction is analogous. ✓

Observe that this claim implies that

$$StHom(\gamma) = Hom(\gamma) \setminus \Big(\bigcup_{\substack{\gamma' \in \Gamma_k \\ \gamma' \supset \gamma}} Hom(\gamma') \Big)$$

Furthermore it is easy to see that for a $\gamma \in \Gamma_k$ we have

$$\bigcup_{\substack{\gamma' \in \Gamma_k \\ \gamma' \supset \gamma}} StHom(\gamma') = \bigcup_{\substack{\gamma' \in \Gamma_k \\ \gamma' \supset \gamma}} Hom(\gamma')$$

and hence

$$StHom(\gamma) = Hom(\gamma) \setminus \left(\bigcup_{\substack{\gamma' \in \Gamma_k \\ \gamma' \supset \gamma}} StHom(\gamma') \right) \tag{7.3}$$

Furthermore, for two distinct graphs $\eta, \theta \in \Gamma_k$ we have

$$StHom(\eta) \cap StHom(\theta) = \emptyset.$$

And if $\eta \subset \theta$ then $Hom(\eta) \supset Hom(\theta) \supseteq StHom(\theta)$. These two facts follow directly from the definitions of the corresponding sets. Hence, in combination with equation (7.3) we obtain the following equation which holds for all $\gamma \in \Gamma_k$:

$$s_\gamma = h_\gamma - \sum_{\substack{\gamma' \in \Gamma_k \\ \gamma' \supset \gamma}} s_{\gamma'} \tag{7.4}$$

This entails a way of computing the desired value s_{γ_0} which is described by algorithm 8.

```
   cardStHom_{γ0}(G⃗, k)
   // G⃗ = (V, E⃗) a digraph, k ∈ ℕ
   // calculate a table of the values h_ε for all ε ∈ Γ_k
 1 forall ε ∈ Γ_k do
 2     h_ε ← |Hom(ε)|;
 3 end
   // compute a table of the values s_ε for all ε ∈ Γ_k
 4 for l = k² downto 0 do
 5     forall ε = ([k], E^ε) ∈ Γ_k with |E^ε| = l do
 6         s_ε ← h_ε − Σ_{ε' ⊃ ε} s_ε';
 7     end
 8 end
 9 return s_{γ0};
```

Algorithm 8: Computing the value s_{γ_0}

Time Complexity. Let t denote the time needed to compute the value h_γ for a $\gamma \in \Gamma_k$. We will determine the exact value of t later.

Consider the first **for**-loop in algorithm 8. As there are 2^{k^2} digraphs with vertex set $[k]$ and each one of them can be constructed in time at most $\mathcal{O}(k^2)$, this loop completes in time at most $\mathcal{O}(k^2 \cdot 2^{k^2} \cdot t)$.

For the second **for**-loop fix l and let $x := k^2$ then there are $\binom{x}{l}$ graphs to be enumerated. Each one can be constructed in at most $l \leq k^2$ steps. For the expression in line 6 of the algorithm, note that there are at most $\sum_{i=l}^{x} \binom{x}{i}$ graphs that play a part in the sum.

Thus for the second loop on all values of l we have time at most

$$\begin{aligned}
\sum_{l=0}^{x} \binom{x}{l} \cdot (k^2 + \sum_{i=l}^{x} \binom{x}{i}) &= k^2 \cdot \sum_{l=0}^{x} \binom{x}{l} + \sum_{l=0}^{x} \binom{x}{l} \sum_{i=l}^{x} \binom{x}{i} \\
&= k^2 \cdot 2^x + \sum_{l=0}^{x} \binom{x}{l} \sum_{i=l}^{x} \binom{x}{i} \\
&\leq k^2 \cdot 2^x + 2^{2x} \\
&\in \mathcal{O}(2^{2k^2})
\end{aligned}$$

Thus with the first loop and $x = k^2$ the algorithm completes in time

$$\mathcal{O}(k^2 \cdot 2^{k^2} \cdot t + 2^{2k^2}).$$

We still have to show how to compute the values h_γ. First, we stress that, if γ consists of the connected components $\epsilon_1, \ldots \epsilon_r$ then we have $h_\gamma = h_{\epsilon_1} \cdot \ldots \cdot h_{\epsilon_r}$. Hence, we may assume that γ is connected.

Consider a k-tuple $\bar{v} = (v_1, \ldots, v_k)$ and let $V(\bar{v}) := \{v_1, \ldots, v_k\}$ then for a connected graph γ we know that $\bar{v} \in Hom(\gamma)$ holds only if $\vec{\mathcal{G}}|_{V(\bar{v})}$ (i.e. the subgraph of $\vec{\mathcal{G}}$ induced by $V(\bar{v})$), is connected as well. This implies that we can systematically find all k-tuples $\bar{v} \in Hom(\gamma)$ by enumerating for the first entry v_1 of $\bar{v}$ all $v \in V$. Then for each case $v_1 = v$ for the other $k-1$ entries of $\bar{v}$ we know, by the connectedness of $\vec{\mathcal{G}}|_{V(\bar{v})}$, that $\{v_2, \ldots, v_k\} \subseteq N_{\vec{\mathcal{G}}}^{k-1}(v)$. As $\vec{\mathcal{G}}$ has maximum degree $d+1$ we have

$$|N_{\vec{\mathcal{G}}}^{k-1}(v)| \leq (d+1)^{k-1}.$$

Thus, via a brute force algorithm, all tuples that are possibly contained in $Hom(\gamma)$ can be generated in time $\mathcal{O}(|V| \cdot ((d+1)^k)^k)$ and as, for a given tuple $\bar{v} \in V^k$ determining $\bar{v} \in Hom(\gamma)$ can be done in time $\mathcal{O}(k^2)$, we have

$$t \in \mathcal{O}(|V| \cdot k^2 \cdot (d+1)^{k^2}).$$

Recall that, $\vec{\mathcal{G}}$ can be computed from $\mathcal{G}$ in time $\mathcal{O}((d+1) \cdot |V|)$ and, given s_{γ_0} the value according to equation (7.1) can be computed in constant time.

Hence, the whole algorithm for solving p-Δ-#IndependentSet completes in time $\mathcal{O}(k^4 \cdot (2 \cdot (d+1))^{k^2} \cdot |V| + 2^{2k^2})$, as claimed. □

Note that by the tight connection of cliques and independent sets, we directly obtain the fact that the following problem is fixed parameter tractable.

p-$(n-\delta)$-#Clique
Instance: A graph $\mathcal{G} = (V, E)$ and $k \in \mathbb{N}$ *Parameter:* $k + \lvert V\rvert - \delta(\mathcal{G})$ *Problem:* Compute the number of cliques of size k in $\mathcal{G}$

7.1. Tractable Cases of Counting Matchings

In this section we will discuss some restricted versions of the parameterized problem of counting matchings. Consider the following problem:

p-Δ-#GeneralMatching
Instance: A graph $\mathcal{G} = (V, E)$ and $k \in \mathbb{N}$ *Parameter:* $k + \Delta(\mathcal{G})$ *Problem:* Compute the number of matchings of size k in $\mathcal{G}$

The technique that was applied to show that p-Δ-#IndependentSet is fixed parameter tractable can indeed be applied to show this for many different problems on graphs of bounded degree, for example, p-Δ-#GeneralMatching could be solved in the same way. However, that this problem is in #FPT can be seen as well by a very simple reduction.

Theorem 7.2. p-Δ-#GeneralMatching *is fixed parameter tractable.*

Proof. We reduce p-Δ-#GeneralMatching to p-Δ-#IndependentSet parsimoniously.

Let $(\mathcal{G}, k)$ be an instance of p-Δ-#GeneralMatching with $\mathcal{G} = (V, E)$. We define a graph $\mathcal{G}' := (E, F)$ with $F := \{\, \{e, f\} \mid e, f \in E \text{ and } e \cap f \neq \emptyset\}$. This graph is called the *line graph* of $\mathcal{G}$.

We show first that $2 \cdot \Delta(G) \geq \Delta(G')$. This guarantees that the parameter does not increase too much. Let e in $\mathcal{G}'$ be a vertex with maximum degree. e represents an edge $\{u, v\}$ in $\mathcal{G}$. By the construction of $\mathcal{G}'$ we have

$$d_{\mathcal{G}'}(e) \leq d_{\mathcal{G}}(u) + d_{\mathcal{G}}(v) \leq 2 \cdot \Delta(G).$$

To complete the proof of the theorem, we show, that there is a trivial one-to-one correspondence between matchings in $\mathcal{G}$ and independent sets in $\mathcal{G}'$.

Let M be a matching in $\mathcal{G}$. For any $e, f \in M$ with $e \neq f$ we have $e \cap f = \emptyset$. Therefore no two distinct e and f in M are adjacent in $\mathcal{G}'$, that is M is an independent set in $\mathcal{G}'$.

The backward direction is completely analogous. □

Define p-Δ-#Matching as the restriction of p-Δ-#GeneralMatching to bipartite graphs. Clearly, we have

Corollary 7.3. p-Δ-#Matching *is fixed parameter tractable.*

Let the problems p-#Matching and p-#GeneralMatching denote the counterparts of p-Δ-#Matching and p-Δ-#GeneralMatching that do not include $\Delta(\mathcal{G})$ into the parameter. For these unrestricted problems it is still unknown whether they are fixed parameter tractable or #A[1]-complete (Note that the containment in #A[1] is obvious).

Another version of the matching problem that can easily be seen to be in #FPT is the parametric dual of p-#Matching.

p-#DualOfMatching
Instance: A graph $\mathcal{G} = (V, E)$ and $k \in \mathbb{N}$ *Parameter:* k *Problem:* Compute the number of matchings of size $\|E\| - k$ in $\mathcal{G}$

This problem can be seen to be fixed parameter tractable by a reduction to vertex cover which follows directly from the reduction in theorem 7.2.

Lemma 7.4. p-#DualOfMatching $\leq^{fpt}$ p-#VertexCover

Proof. Let $\mathcal{G} = (V, E)$ and $k \in \mathbb{N}$ be an instance of p-#DualOfMatching. Again, we construct the linegraph $\mathcal{G}' := (E, F)$ with $F := \{ \{e, f\} \mid e, f \in E \text{ and } e \cap f \neq \emptyset\}$. Then it is easy to see that a set $M \subseteq V$ is a matching of cardinality $|E| - k$ in $\mathcal{G}$ if, and only if, $E \setminus M$ is a size k vertex cover in $\mathcal{G}'$. □

Conclusion

We can summarize the work of this thesis from two different viewpoints. From a structural perspective, we removed some inconsistencies between certain parameterized complexity classes, such as #A[1] and #W[P]. Moreover, the development of the concept of counting kernelizations strengthened the definition of #FPT, as these provide a complete characterization of fixed parameter tractable counting problems.

With a view that is devoted more to the counting problems themselves, one would emphasize the classification results. Instead of giving an exhaustive list of the results, we will summarize some of these by comparing them to their analogs from classical complexity theory. This will provide some intuition about the differences of these two theories.

Recall that we introduced the complexity classes that were defined by Valiant (cf. [Val79a]). The class #P comprises all functions that are representable by the accepting runs of a polynomially bounded nondeterministic Turing machine. Its deterministic counterpart FP contains all functions that are deterministically computable in polynomial time. For our purposes these informal definitions suffice.

In his seminal papers [Val79a] and [Val79b] Valiant showed that many problems such as #VertexCover, #Clique and #Matching are #P complete. These problems ask for the number of the corresponding structures irrespective of their size. For example #VertexCover is the problem of counting all vertex covers in a given graph $\mathcal{G}$. Although the parameterized problems we considered count only structures of a certain size k, these, namely p-#VertexCover, p-#Clique and p-#Matching, are their most natural parameterized peers.

Note that any comparison of parameterized complexity classes with classical ones is by no means trivial. The claim that #FPT is the parameterized analog of FP is unproblematic, but for the analog of #P a reasonable choice is not as simple. In this thesis we have considered at least two classes that might be candidates for this, to wit, #W[P] and #A[1]. With respect to the problems which we will consider in this chapter, we regard #A[1] as the analog of #P only because all of these problems are contained in #A[1]. In a more general context, however, this situation might change such that #W[P] is given pride of place.

We will restrict our discussion to the three problems mentioned above and some of their restricted variants. These restrictions include the clique problem in graphs of large minimum degree, for example #2$(n-\delta)$-Clique denotes the clique problem on graphs $\mathcal{G} = (V, E)$ where $|V| - \delta(\mathcal{G}) = 2$. Analogously, #4$\Delta$-Matching is the matching problem in bipartite graphs of maximum degree 4.

Table 7.1 summarizes the different classifications of some vertex cover, clique and matching problems. The classical result for #BipartiteClique was found by

Classical Problems	
#P-hard	#Clique #BipartiteClique #Matching #4Δ-Matching #VertexCover #5$(n-\delta)$-Clique
in FP	#2$(n-\delta)$-Clique #2Δ-Matching

Parameterized Problems	
p-#Clique p-#BipartiteClique	#A[1]-hard
p-#VertexCover p-$(n-\delta)$-#Clique p-Δ-#Matching p-#DualOfMatching	in #FPT

Table 7.1.: Complexity Classifications of Counting Cliques and Vertex Covers

Provan and Ball (cf. [PB83]), whereas the restricted versions of #clique and #Matching were classified by Vadhan in [Vad01].

The most eye-catching fact about parameterized complexity is that it renders the vertex cover problem tractable. As we have seen in the first part of this thesis, indeed the fixed parameter tractability of many problems is tightly connected to that of vertex cover.

A closer look, however, reveals some truly interesting details. The reader not familiar with parameterized complexity might ask why the p-#DualOfMatching problem is known to be in #FPT whereas p-#Matching has not been classified. In parameterized complexity, a problem can be of very different complexity if we consider a parameterization and its dual. For example, it can easily be seen that the parametric dual of p-#VertexCover is equivalent to p-#IndependentSet. This is so, because for a vertex cover S in a graph $\mathcal{G} = (V, E)$, the set $V \setminus S$ is an independent set in $\mathcal{G}$. Hence, as p-#IndependentSet is parametrically equivalent to p-#Clique, which is #A[1]-complete, the dual of p-#VertexCover is #A[1]-complete as well.

Something similar can be observed with the matching problem. The parametric dual of the matching problem p-#DualOfMatching is fixed parameter tractable, whereas this is unknown for p-#Matching itself and it has been conjectured (cf. [FG06]) that this problem is #A[1]-complete.

Another fact that is unique to parameterized complexity is revealed by the restricted problems given in table 7.1. Note that the restrictions of the matching problem to bounded degree graphs, namely #cΔ-Matching for a constant c are subject to a dichotomy, insofar as for $c \leq 2$ the problem is efficiently solvable, whereas for $c \geq 5$ it is #P hard. Contrarily, the analogous parameterized problem p-Δ-#Matching is in #FPT irrespective of the value of Δ. Note that this problem includes Δ into the parameter instead of keeping it constant but this implies that, if Δ would be constant, the fixed parameter tractability would still be given. Similar results hold for #$c(n-\delta)$-Clique and p-$(n-\delta)$-#Clique.

These problems reveal a property of parameterized problems which can be observed directly by the definition of fixed parameter tractability, that is, the notion

of the parameter is a unifying scheme, as containment in #FPT is given irrespective of the actual parameter value. Therefore, in parameterized complexity restricted case analysis will reveal no dichotomies as long as the restrictions are expressed in terms of the parameter.

Guidelines For Future Research

There are a lot of questions that formed during the work on this thesis. We will conclude in giving a short impression about some of these.

Are the counting problems corresponding to p-SetSplitting (for a definition see [DFRS04]) and p-DisjointTriangle (cf. [MPS04]) solvable by some extension of the crown rule? If not, maybe some other algorithms not known to the author that solve p-SetSplitting or p-DisjointTriangle without applying crowns can be adapted easily to solve the corresponding counting problems.

It is not clear if the definition of extremal versions of parameterized counting problems is reasonable. At least problems asking for optimal soultions, such as p-#MinimumVertexCover, might be of interest as either the optimum is smaller than k and hence we count all solutions, or no optimal solution is found at all. In the case of structures that are *minimal* or *maximal* (with respect to inclusion) this seems not as reasonable. However, it would be interesting if the definition of a problem like p-#MinimalVertexCover can be motivated in some way.

As far as the intractability of counting problems is concerned, the classification of p-#Matching is still open. This problem might well be #A[1]-complete, which would imply the intractability of p-#DisjointTriangle and similar problems.

Another nice result would be the classification of the counting problem associated with p-DualOfColoring (this problem is defined in [DF99], p. 46). Although p-DualOfColoring is fixed parameter tractable, we conjecture that p-#DualOfColoring is #A[1]-complete.

With respect to the problem of counting induced substructures, we have seen that counting induced paths and cycles is #A[1]-complete. This is not very surprising, as the two extreme problems corresponding to counting induced substructures, namely p-#IndependentSet and p-#Clique, are #A[1]-hard. Furthermore, induced cycles and paths are the complements of graphs of high treewidth. Therefore, we conjecture that the problem of counting induced substructures is #A[1]-hard irrespective of the substructures under consideration.

Bibliography

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[AR04] Susanne Albers and Tomasz Radzik, editors. *Algorithms - ESA 2004, 12th Annual European Symposium, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3221 of *Lecture Notes in Computer Science*. Springer, 2004.

[BJS05] Hajo Broersma, Matthew Johnson, and Stefan Szeider, editors. *Algorithms and Complexity in Durham 2005 - Proceedings of the First ACiD Workshop, 8-10 July 2005, Durham, UK*, volume 4 of *Texts in Algorithmics*. King's College, London, 2005.

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Springer, 1999.

[DFD04] Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne, editors. *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3162 of *Lecture Notes in Computer Science*. Springer, 2004.

[DFRS04] Frank K. H. A. Dehne, Michael R. Fellows, Frances A. Rosamond, and Peter Shaw. Greedy localization, iterative compression, modeled crown reductions: New fpt techniques, an improved algorithm for set splitting, and a novel 2k kernelization for vertex cover. In Downey et al. [DFD04], pages 271–280.

[DJ04] Victor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004.

[DLOS05] Frank K. H. A. Dehne, Alejandro López-Ortiz, and Jörg-Rüdiger Sack, editors. *Algorithms and Data Structures, 9th International Workshop, WADS 2005, Waterloo, Canada, August 15-17, 2005, Proceedings*, volume 3608 of *Lecture Notes in Computer Science*. Springer, 2005.

[DR02] Krzysztof Diks and Wojciech Rytter, editors. *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*. Springer, 2002.

[DST04] Josep Díaz, Maria J. Serna, and Dimitrios M. Thilikos. Fixed parameter algorithms for counting and deciding bounded restrictive list h-colorings. In Albers and Radzik [AR04], pages 275–286.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Springer, 2006.

[Fri01] M. Frick. *Easy Instances for Model Checking.* PhD thesis, Universität Freiburg, 2001.

[Gre00] Catherine S. Greenhill. The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. *Computational Complexity*, 9(1):52–72, 2000.

[Knu81] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition.* Addison-Wesley, 1981.

[LS05] Daniel Lokshtanov and Christian Sloper. Fixed parameter set splitting, linear kernel and improved running time. In Broersma et al. [BJS05], pages 105–113.

[McC02] Catherine McCartin. Parameterized counting problems. In Diks and Rytter [DR02], pages 556–567.

[MPS04] Luke Mathieson, Elena Prieto, and Peter Shaw. Packing edge disjoint triangles: A parameterized view. In Downey et al. [DFD04], pages 127–137.

[NRT05] Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Parameterized counting algorithms for general graph covering problems. In Dehne et al. [DLOS05], pages 99–109.

[PB83] J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.

[Rot96] Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.

[Vad01] Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.*, 31(2):398–427, 2001.

[Val79a] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[Val79b] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Ich bin damit einverstanden, dass die vorliegende Arbeit in der Bibliothek des Instituts für Informatik der Humboldt-Universität zu Berlin ausgestellt wird.

Berlin, den