Counting Patterns in Strings and Graphs

A dissertation submitted towards the degree
Doctor of Natural Sciences
of the Faculty of Mathematics and Computer Science
of Saarland University

Philip Wellnitz

Saarbrücken / 2021

Colloquium Information

Date	December 2, 2021; 15:00 CET
Dean	Prof. Dr. Thomas Schuster
Chairman	Prof. Dr. Markus Bläser
Reporters	Prof. Dr. Dr. h.c. mult. Kurt Mehlhorn
	Prof. Dr. Gad M. Landau
	Prof. Dr. Martin Grohe
	Prof. Dr. Karl Bringmann
Academic Assistant	Dr. Roohani Sharma

Acknowledgments

First, I am deeply indebted to Kurt Mehlhorn for both funding and supervising me. In particular, I felt honored to hear both his words of encouragement, and even more so his words of wisdom.

Second, I want to thank Karl Bringmann, for introducing me to the field of *Fine-Grained Complexity Theory* and to the problem of *Approximate String Matching*, as well as his supervision. Further, I want to thank him for the fruitful collaborations [18, 20, 19, 21] and the many things I learned during them.

Third, I am extremely grateful to Marc Roth for introducing me to the field of *Parameterized (Counting) Complexity Theory* and for the fruitful collaborations [38, 101, 40, 102, 103]. Even more I want to extend my gratitude for his never ending enthusiasm and motivation that helped me go on at times, and for his patience to clear up some of my confusions at other times.

Fourth, I wish to extend my sincere thanks to Tomasz Kociumaka for hosting me for a very productive two-week-stay at Bar-Ilan University and for the fruitful and inspiring collaboration [26].

Fifth, I am very grateful to Martin Grohe and Gad M. Landau for dedicating their time to read and assess this thesis.

Additionally, I want to thank the other co-authors of the aforementioned collaborations—Panagiotis Charalampopoulos, Holger Dell, Julian Dörfler, Nick Fischer, Danny Hermelin, Marvin Künnemann, Dvir Shabtay, and Johannes Schmitt—for the countless tricks and techniques that I learned from them. Also, I want to thank André, Attila, Bhaskar, Edgar, Kai, and Leo, for interesting, inspiring, and insightful discussions over the years.

Finally, this thesis would not exist without my trusty LuaLaTeX compiler, that *almost* never let me down, no matter the `smashing`, `breaking`, and all the other cruelties it had to cope with.

Abriss

Wir untersuchen Probleme im Zusammenhang mit dem Finden und Zählen von Mustern in Strings und Graphen.

Im Stringbereich ist die Aufgabe, alle Teilstrings eines Strings T zu bestimmen, die eine Hamming- (oder Editier-) Distanz von höchstens k zu einem Pattern P haben. Unter anderem sind wir am voll-komprimierten *Setting interessiert, in dem sowohl T, als auch P in komprimierter Form gegeben sind. Für beide Abstandsbegriffe entwickeln wir die ersten Algorithmen mit einer (fast) linearen Laufzeit in der Größe der komprimierten Darstellungen. Die Algorithmen nutzen neue strukturelle Einsichten in die Lösungsstruktur der Probleme.*

Im Graphenbereich betrachten wir Probleme im Zusammenhang mit dem Zählen von Homomorphismen zwischen Graphen. Im Besonderen betrachten wir das Problem $\#\textsc{IndSub}(\Phi)$, bei dem alle induzierten Subgraphen mit k Knoten zu zählen sind, die die Eigenschaft Φ haben. Basierend auf einer Theorie von Lovász, Curticapean, Dell, and Marx drücken wir $\#\textsc{IndSub}(\Phi)$ als Linearkombination von Homomorphismen-Zahlen aus um $\#\mathsf{W}[1]$-Härte und fast scharfe konditionale untere Laufzeitschranken zu erhalten für Φ, die monoton sind oder nur auf der Kantenanzahl der Graphen basieren. Somit beweisen wir eine Vermutung von Jerrum and Meeks.

Weiterhin beschäftigen wir uns mit der Komplexität des Problems $\#\textsc{Hom}(\mathcal{H} \to \mathcal{G})$ für Graphklassen $\mathcal{H}$ und $\mathcal{G}$. Im Besonderen zeigen wir, dass es für jedes Problem P in $\#\mathsf{W}[1]$ Graphklassen $\mathcal{H}_P$ und $\mathcal{G}_P$ gibt, sodass P äquivalent *zu $\#\textsc{Hom}(\mathcal{H}_P \to \mathcal{G}_P)$ ist.*

Summary

We study problems related to finding and counting patterns in strings and graphs.

In the string-regime, we are interested in counting how many substring of a text T are at Hamming (or edit) distance at most k to a pattern P. Among others, we are interested in the *fully-compressed* setting, where both T and P are given in a compressed representation. For both distance measures, we give the first algorithm that runs in (almost) linear time in the size of the compressed representations. We obtain the algorithms by new and tight structural insights into the solution structure of the problems.

In the graph-regime, we study problems related to counting homomorphisms between graphs. In particular, we study the parameterized complexity of the problem $\#\textsc{IndSub}(\Phi)$, where we are to count all k-vertex induced subgraphs of a graph that satisfy the property Φ. Based on a theory of Lovász, Curticapean et al., we express $\#\textsc{IndSub}(\Phi)$ as a linear combination of graph homomorphism numbers to obtain $\#W[1]$-hardness and almost tight conditional lower bounds for properties Φ that are monotone or that depend only on the number of edges of a graph. Thereby, we prove a conjecture by Jerrum and Meeks.

In addition, we investigate the parameterized complexity of the problem $\#\textsc{Hom}(\mathcal{H} \to \mathcal{G})$ for graph classes $\mathcal{H}$ and $\mathcal{G}$. In particular, we show that for any problem P in the class $\#W[1]$, there are classes $\mathcal{H}_P$ and $\mathcal{G}_P$ such that P is *equivalent* to $\#\textsc{Hom}(\mathcal{H}_P \to \mathcal{G}_P)$.

Introduction to this Thesis

This thesis consists of two parts: Algorithms for counting patterns in strings and lower bounds for counting patterns in graphs. Due to limits in current technology, one part has to appear before the other—this is by no means a statement regarding neither quantity nor quality of the contained content. In both measures, the author considers both parts to be on equal footing with each other. While both parts may be similar in name, they nevertheless are completely *independent* of each other. Both parts assume as little prior knowledge as possible to accommodate readers familiar only with concepts from at most one of the parts.

Collaborations and Publications

This thesis is based on the conference publications [26, 102, 101], where the publication [26] subsumes and extends the author's earlier publication [20]. All of the aforementioned publications are the result of fruitful collaborations of the author with many researchers. An overview of all chapters of this thesis follows that lists the details and extent of the collaborations and the corresponding publications. The list doubles as a quick overview over the novelty of the content of a chapter: If a chapter contains mostly known results (not by the author), the chapter is marked with a ⬜ White Square; a chapter is marked with a 🟥 Pink Square if the contained results are obtained via easy modifications of known results (not by the author); and a chapter is marked with a 🟨 Yellow Square if the contained results are to be considered as the main contribution of this thesis. The same color scheme has been adopted for theorems and other statements.

Part 1. Counting Patterns in Strings

1 This chapter introduces basic terminology used in this part.

2 This chapter introduces the abstract PILLAR model that encapsulates common operations on "string-like" objects such as computing longest common prefixes. While the PILLAR model was introduced by Panagiotis Charalampopoulos, Tomasz Kociumaka, and the author in [26], the underlying results on "string-like" objects were known beforehand and are in particular not the author's work.

3 This chapter provides new structural insights into the solution structure of Pattern Matching with Mismatches.
A weaker version of the main structural result was first obtained by Karl Bringmann, Marvin Künnemann, and the author in [20], from which select lemmas were included for technical reasons. The improved (and optimal) structural result was shown by Panagiotis Charalampopoulos, Tomasz Kociumaka, and the author in [26]. All authors contributed equally.

4 This chapter implements the insights from 3 into a PILLAR model algorithm. The results are joint work by Panagiotis Charalampopoulos, Tomasz Kociumaka, and the author [26]. All authors contributed equally.

5 In this chapter, the structural result from 3 is generalized to Pattern Matching with Edits. The results are joint work by Panagiotis Charalampopoulos, Tomasz Kociumaka, and the author [26]. All authors contributed equally.

6 This chapter implements the insights from 5 into a PILLAR model algorithm. The results are joint work by Panagiotis Charalampopoulos, Tomasz Kociumaka, and the author [26]. All authors contributed equally.

Part 2. Counting Patterns in Graphs

7 This chapter introduces graph theoretic concepts used in this part.

8 This chapter introduces parameterized (counting) complexity theoretic concepts used in this part.

9 This chapter presents first, easy applications of the concepts introduced in 7 and 8. The results are joint work of Marc Roth and the author [101]. All authors contributed equally.

10 This chapter presents new hardness results for the problem #IndSub(Φ) for (large) classes of properties Φ. The results are joint work of Marc Roth, Johannes Schmitt, and the author [102]. All authors contributed equally.

11 This chapter presents a proof that any problem in the complexity class #$W[1]$ is equivalent to a problem of counting homomorphisms. The result is joint work of Marc Roth and the author [101]. All authors contributed equally.

All figures in this thesis have been produced by the author. Figures also appearing in a conference publication of the author are indicated with a Blue Turnip that includes a reference to said publication. In that case, the figure has been revised by the author (multiple times) based on the valuable feedback of the corresponding co-authors and applicable reviewer comments.

Finally, note that this thesis—apart from the cited results—is not based on the authors other works [38, 40, 21, 19, 103] that were published during their Ph.D.

Contents

II Counting Patterns in Graphs

List of Figures and Algorithms

Part I

Counting Patterns in Strings

1 2 3 4 5 6

Introduction to Part I

In the first part of this thesis, we focus on *counting patterns in strings*. In particular, we are interested in counting *approximate* occurrences of a pattern in a text—Think of counting or finding DNA or RNA sequences in (large) protein databases (possibly under so-called "sequencing-errors") [see 108, 91], common search functionality on webpages that also accounts for spelling mistakes.[1] Consult also the very good overview in [90] for a more detailed discussion about the plethora of applications.

1. Think of a certain large online encyclopedia as a concrete example.

As an illustrative example, say you are interested in reading the latest news about *turnips*—In your daily routine you check the local newspaper for any news on *turnips*. Alas, sometimes you miss very interesting articles, because they are about *turmips* or *tunips* instead—unacceptable.

Thus, instead of searching for *exact* occurrences, we are satisfied with finding (or counting) substrings of the text T, that are *close* to a given pattern P. Many different measures of "closeness" have been studied (again, see [90] for an overview)—we focus on two very prominent measures, the *Hamming distance* and the *edit distance*. In particular, we are interested in algorithms for very large strings or data sets—just as DNA sequences or (collections of) large webpages. To obtain said algorithms, we take a closer look at the rich solution structure of the approximate pattern matching problems.

Highlights from Approximate Pattern Matching

Naturally, many (classical) results are known about approximate pattern matching problems. We give a short summary of the most prominent results next. To that and, write n for the length of the text T and m for the length of the pattern P.

Pattern Matching with Mismatches

First, we consider the *Hamming distance* as a similarity measure, that is, the number of positions where two strings differ. In our example, we have $\delta_H(\texttt{TURNIP}, \texttt{TURMIP}) = 1$ and $\delta_H(\texttt{TURNIP}, \texttt{TUNIP}) = 4$. Our main question then becomes *which substrings of T are at Hamming distance at most k to P*?—We call this problem the *Pattern Matching with Mismatches* problem.

Already in the late 1980s, Abrahamson [2] and Kosaraju [74] independently gave an $O(n\sqrt{m \log m})$-time algorithm for computing the Hamming distance of P and all length-m substrings of T. While their algorithms can indeed be used to solve pattern matching with mismatches, the first algorithm to be tailored to this problem was given by Landau and Vishkin [77]: Based on so-called "kangaroo jumping", they obtained an $O(kn)$-time[2] algorithm—a faster algorithm even for moderately large k. Amir, Lewenstein, and Porat [5] improved Landau and Vishkin's algorithm to obtain both an $O(n\sqrt{k \log k})$-time algorithm and an $\tilde{O}(n + k^3 n/m)$-time algorithm; the latter algorithm was then improved further by Clifford, Fontaine, Porat, Sach, and Starikovskaya [32] to obtain an $\tilde{O}(n + k^2 n/m)$-time algorithm. Finally, Gawrychowski and Uznański [51] bridged the running times of both algorithms with a smooth tradeoff by designing an $\tilde{O}(n + kn/\sqrt{m})$-time algorithm. Very recently, Chan, Golan, Kociumaka, Kopelowitz, and Porat [23] reduced the polylog n factors in the latter solution at the cost of (Monte-Carlo) randomization, achieving a running time of

$$O(n + \min(k^2 n/m, kn\sqrt{\log m}/\sqrt{m})).$$

2. Recall that k is the number of allowed mismatches.

Further, Gawrychowski and Uznański [51] demonstrated that a significantly faster "combinatorial" algorithm would have (unexpected) consequences for the complexity of combinatorial matrix multiplication.

We can thus conclude that pattern matching with mismatches on strings is essentially fully understood. Nevertheless, for other settings (that is, when the text and the pattern are not given as ordinary strings) a similar understanding, in terms of both upper and lower bounds, is yet to be obtained. One of the main contributions of this thesis is to improve the known upper bounds for two such settings, obtaining algorithms with running times analogous to the algorithm of Clifford et al. [32].

Pattern Matching with Errors

As is evident from the example, the Hamming distance—while easy to compute—is only moderately useful for counter-acting simple spelling mistakes, such as missing (or duplicate) characters. Hence, we also consider the *edit distance* (also called Levenshtein distance) as a similarity measure, that is, the number of *insertions*, *deletions*, or *replacements* necessary to transform one string to another. Returning to our example, we have $\delta_E(\texttt{TURNIP}, \texttt{TURMIP}) = 1$ (replace `N` with `M`) and $\delta_E(\texttt{TURNIP}, \texttt{TUNIP}) = 1$ (delete `R`). As before, our main question then becomes *which substrings of T are at edit distance at most k to P*?—We call this problem the *Pattern Matching with Edits* (or Errors) problem.

Again, there is a by now classical $O(kn)$-time algorithm by Landau and Vishkin [78] that; a line of research [105, 33] improved this running time (for some range of parameters) to $O(n + k^4 n/m)$. From a lower-bound perspective, the classical $O(n^2)$-time dynamic programming-based algorithm for computing the edit distance of two strings of size $O(n)$ is essentially optimal: Backurs and Indyk [7] recently proved that a significantly faster algorithm would translate to a major breakthrough for the Satisfiability problem. For the pattern matching with edits problem, this means that there is no hope for obtaining an algorithm that is significantly faster than $O(kn)$ or $O(n + k^2 n/m)$; however, apart from that "trivial" lower bound and the 20-year-old conjecture of Cole and Hariharan [33] that an $O(n + k^3 n/m)$ algorithm *should be possible*, nothing is known that would close this gap. While we do not manage to decrease the size of this gap, we do believe that the structural insights that we obtain for pattern matching with edits may be useful for doing so. What we do manage, however, is to significantly improve the running time of the known algorithms in two settings where T and P are not given as ordinary strings, thereby obtaining running times similar to the running time of the algorithm of Cole and Hariharan [33].

Results in the Fully-Compressed Setting

Recall the example of finding (parts of) DNA sequences in each other. In this case, both text and pattern may be *huge*—in particular, we would greatly appreciate if we could work on *compressed strings* instead.[3]

3. While DNA sequences are not really well compressible, *any* speed-ups are welcome in that setting.

Hence, the first setting that we consider is the so-called *fully-compressed* setting; specifically, when both the text T and the pattern P are given as straight-line programs (SLP), that is as context-free grammars that encode exactly one string.[4] Our goal is to solve the approximate string matching problems *without* decompressing the SLPS first—as to not loose the gains from the compression.

4. We work with SLPS, as they are equivalent (up to logarithmic factors) to widely-used compression schemes [104, 70, 69] such as the LZ77 parsing [121] and the run-length-encoded Burrows-Wheeler transform [22]. Even more schemes can be expressed as SLPS: byte-pair encoding [109], members of the Lempel-Ziv family [81, 119], Sequitur [92], Re-Pair [79], to name but a few.

In this setting, we obtain the first algorithms running in (almost) linear time.

Main Theorem 1. *Let $\mathcal{G}_T$ denote an SLP of size n generating a text T, let $\mathcal{G}_P$ denote an SLP of size m generating a pattern P, let k denote a threshold, and set $N := |T| + |P|$.*

We can compute the number of occurrences of P in T with up to k mismatches in time $O(m \log N + n k^2 \log^3 N)$.

We can compute the number of occurrences of P in T with up to k errors in time $O(m \log N + n k^4 \log^3 N)$.

Further, in both settings we can report the starting positions of the occurrences in constant time per occurrence.

Finally, observe that as a special case, we also solve the case if *only* the text is compressed—think of searching in a compressed file without wanting to decompress it first.

Results in the Dynamic Setting

Recall the example of the search functionality of a large webpage. It may be required to regularly update the database behind such a webpage—recomputing answers to search queries after each update naively, we possibly recompute queries whose answer did not change (or did not change much).

Hence, we also consider a setting that is tailored to large, constantly changing strings. In particular, we want to maintain an initially empty collection of non-empty persistent strings $\mathcal{X}$ that can be modified via the following "update" operations:

- `makestring`(U): Insert a non-empty string U to $\mathcal{X}$.
- `concat`(U, V): Insert UV to $\mathcal{X}$, for $U, V \in \mathcal{X}$.
- `split`(U,i): Insert $U[\,0\,..\,i\,)$ and $U[\,i\,..\,|U|\,)$ in $\mathcal{X}$, for $U \in \mathcal{X}$ and $i \in [\,0\,..\,|U|\,)$.

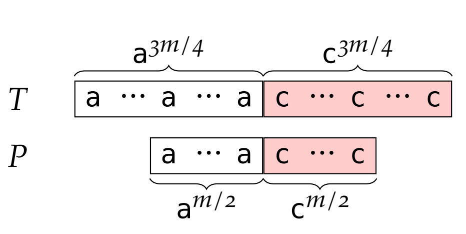
Note that persistence here means that `concat` and `split` do not destroy their arguments.

Our main goal in this setting (and for dynamic algorithms in general) is to provide algorithms that are faster than recomputing the answer from scratch after every update. Building on top of a data structure by Gawrychowski, Karczmarz, Kociumaka, Lacki, and Sankowski [52][5], we achieve this goal and obtain the following result.

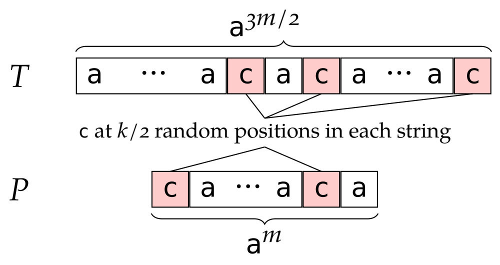
5. This data structure is in itself, the (final) result of a long line of research [112, 88, 4].

MAIN THEOREM 2. *We can maintain a collection $\mathcal{X}$ of non-empty persistent strings of total length N under the operations* `makestring`(U), `concat`(U,V), `split`(U,i) *requiring time $O(\log N + |U|)$, $O(\log N)$ and $O(\log N)$, respectively, so that given two strings $P, T \in \mathcal{X}$ with $|P| = m$ and $|T| = n$ and a threshold k, we can compute the starting positions of all k-mismatch occurrences of P in T in time $O(n/m \cdot k^4 \log^2 N)$ and of all k-error occurrences in time $O(n/m \cdot k^4 \log^2 N)$.*[6]

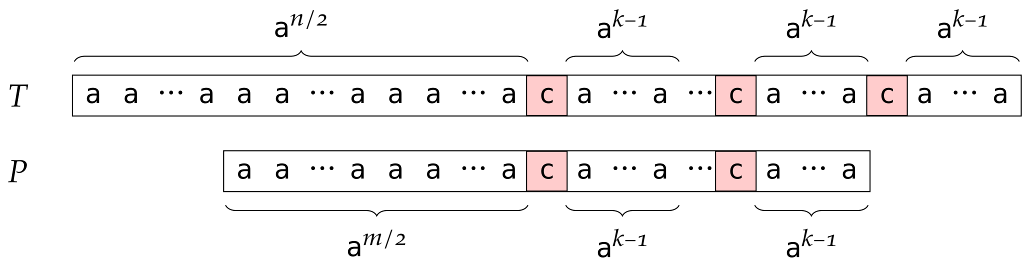
6. All running time bounds hold w.h.p.



(a) Both P and T are not periodic, but there are $2k$ k-mismatch occurrences of P in T.



(b) There are $O(m)$ k-mismatch occurrences of P in T, but both P and T are not perfectly periodic.


$a^{n/2}$
a^{k-1}
a^{k-1}
a^{k-1}
T
a a ⋯ a a a ⋯ a a a ⋯ a c a ⋯ a ⋯ c a ⋯ a c a ⋯ a
P
a a ⋯ a a a ⋯ a c a ⋯ a ⋯ c a ⋯ a
$a^{m/2}$
a^{k-1}
a^{k-1}

(c) Both P of length m and T of length $n := m + 2k^2$ are not periodic, but there are $\Omega(k^2)$ k-error occurrences of P in T.

26 FIGURE 1.1. Examples 1 to 3 illustrated, showing that Main Theorems 3 and 4 are tight.

Insights into the Solution Structure of Approximate Pattern Matching

We obtain our main results by a gaining a better understanding in the solution structure of the approximate pattern matching problems. In particular, we prove the following result.

Main Theorem 3. *Given a pattern P of length m, a text T of length $n \le {}^3/_2\, m$, and a threshold $k \le m$, at least one of the following holds.*

- *The number of k-mismatch occurrences of P in T is bounded by $O(k)$.*
- *The pattern is very close to being periodic.*[7]

[7] More concretely, there is a primitive string Q of length $O(m/k)$, such that P has a Hamming distance of at most $2k$ to any prefix of an infinite repetition of Q.

Our characterization is tight. Consider the following examples.

1. For $T := \mathsf{a}^{3m/4}\mathsf{c}^{3m/4}$ and $P := \mathsf{a}^{m/2}\mathsf{c}^{m/2}$, there is one exact occurrence of P in T. However, shifting P by at most k characters in either direction still yields a k-mismatch occurrence. Hence, in order to obtain periodicity for P, we need to account for $O(k)$ k-mismatch occurrences. Consult Figure 1.1a for an illustration of this example.
2. Now consider $T := \mathsf{a}^{3m/2}$ and $P := \mathsf{a}^m$ and change both T and P at $k/2$ random positions to c. Then, we still have $O(m)$ k-mismatch occurrences of P in T, but, with high probability, both T and P are not perfectly periodic. Hence, we need to relax the periodicity notion to allow for up to $O(k)$ mismatches. Consult Figure 1.1b for an illustration of this example.

We generalize Main Theorem 3 to the edit distance case:

Main Theorem 4. *Given a pattern P of length m, a text T of length $n \le {}^3/_2\, m$, and a threshold $k \le m$, at least one of the following holds.*

- *The starting positions of all k-error occurrences of P in T lie in $O(k)$ intervals of length $O(k)$ each.*
- *The pattern is very close to being periodic.*[8]

[8] More concretely, there is a primitive string Q of length $O(m/k)$, such that P has a edit distance of at most $2k$ to any prefix of an infinite repetition of Q.

Note that this characterization is tight as well: First, observe that the examples from the Hamming distance case are still valid as any k-mismatch occurrence is also a k-error occurrence. However, as the edit distance allows insertions and deletions of characters, we can construct an example where both P and T are not periodic, and the number of k-error occurrences is $\Omega(k^2)$:

3 Consider the text $T := \mathsf{a}^{n/2}(\mathsf{a}^{k-1}\mathsf{c})^{n/2k}$ and the pattern $P := \mathsf{a}^{m/2}(\mathsf{a}^{k-1}\mathsf{c})^{m/2k}$ for $n := m + 2k^2$.
Now, at every position $n/2 - m/2 + i \cdot k, i \in [-k\,..\,k]$ in T, a k-mismatch occurrence of P in T starts, and these are all k-mismatch occurrences of P in T. Any such occurrence has less than k mismatches—thus, there are also k-error occurrences at all positions $n/2 - m/2 + i \cdot k + j, j \in (-k + i\,..\,k - i)$. Thus there are at least $\Omega(k^2)$ k-error occurrences of P in T. Consult Figure 1.1c for an illustration of this example.

1 Almost Everything You Need to Know About Strings

Let us start with basic definitions and concepts used throughout this part. We write $[\,i\,.\,.\,j\,]$ to denote the set $\{i, \dots, j\}$ and $[\,i\,.\,.\,j\,)$ to denote the set $\{i, \dots, j-1\}$; the sets $(\,i\,.\,.\,j\,]$ and $(\,i\,.\,.\,j\,)$ are defined accordingly.

For a finite set S, we write $\{a+j\cdot d\}_S := \{a+j\cdot d \mid j \in S\}$ to denote the (finite) *arithmetic progression* with starting value a, difference d, and length $|S|$. Whenever we use arithmetic progressions in an algorithm, we store them as a triple of the first value, the difference, and the length.

For a set X, we write kX to denote the set containing all elements of X multiplied by k, that is, $kX := \{k \cdot x \mid x \in X\}$. Similarly, we define $\lfloor X/k \rfloor := \{\lfloor x/k \rfloor \mid x \in X\}$ and $k\lfloor X/k \rfloor := \{k \cdot \lfloor x/k \rfloor \mid x \in X\}$.

1.1 Strings, Fragments, and Periods

We write $T = T[\,0\,]\,T[\,1\,]\cdots T[\,n-1\,]$ to denote a *string* of length $|T| = n$ over an alphabet Σ. The elements of Σ are called *characters*. We write ε to denote the *empty string*.

For a string T, we denote the *reverse string* of T by T^R, that is, $T^R := T[\,n-1\,]T[\,n-2\,]\cdots T[\,0\,]$. For two positions i and j in T, we write $T[\,i\,.\,.\,j+1\,) := T[\,i\,.\,.\,j\,] := T[\,i\,]\cdots T[\,j\,]$ to denote the *fragment* of T that starts at position i and ends at position j. We set $T[\,i\,.\,.\,j\,] := \varepsilon$ whenever $j < i$.

A string P of length m with $0 < m \leq n$ is a *substring* of T if there is a fragment $T[\,i\,.\,.\,i+m\,)$ equal to P; we write $P \preccurlyeq T$. In this case, we say that there is an *exact occurrence* of P at position i in T, or, more simply, that P *exactly occurs in* T. We write $\mathrm{Occ}(P, T)$ for the set of all starting positions of exact matches of P in T.

A *prefix* of a string T is a fragment that starts at position 0 (that is, a prefix is a fragment of the form $T[\,0\,.\,.\,j\,]$ for some $j \geq 0$). A *suffix* of a string T is a fragment that ends at position $|T| - 1$ (that is, a suffix is a fragment of the form $T[\,i\,.\,.\,|T|\,)$ for some $i < |T|$). For two strings U and V, the longest *common* prefix of U and V is the length of the longest string that is a prefix of both U and V, that is,

$$lcp(U, V) := 1 + max\{j \mid U[\,0\,.\,.\,j\,] = V[\,0\,.\,.\,j\,]\}.$$

The longest common suffix of U and V is the length of the longest string that is a suffix of both U and V, that is,

$$lcp^R(U, V) := |T| - min\{i \mid U[\,i\,.\,.\,|T|\,) = V[\,i\,.\,.\,|T|\,)\}.$$

For two strings U and V, we write UV to denote their concatenation. We also write $U^k := U \cdots U$ to denote the concatenation of k copies of the string U. Furthermore, U^∞ denotes the string obtained by concatenating infinitely many copies of U. A string T is called *primitive* if for no string U and integer $k > 1$, the string T can be expressed as U^k.

A positive integer p is called a *period* of a string T if $T[i] = T[i+p]$ for all $i = 1, \ldots, n - p$. We refer to the smallest period as *the period* of the string and denote it by $\text{per}(T)$. A string is called *periodic* if its period is no more than half of its length. If a string is not periodic, we say that it is *aperiodic*.

For a string T, we define the following operations that *rotate* a string. The operation $\text{rot}(\star)$ takes as input a string, and moves its last character to the front; that is,

$$\text{rot}(T) := T[\,n-1\,]T[\,0\,.\,.\,n-2\,].$$

The inverse operation $\text{rot}^{-1}(\star)$ takes as input a string and moves its first character to the end; that is,

$$\text{rot}^{-1}(T) := T[\,1\,.\,.\,n-1\,]T[\,0\,].$$

Note that a primitive string T does not match any of its non-trivial rotations, that is, we have $T = \mathrm{rot}^j(T)$ if and only if $j \equiv 0 \pmod{|T|}$.

Finally, the *run-length encoding* (RLE) of a string T is a decomposition of T in maximal blocks such that each block is a power of a single character. (For instance, the RLE of the string `aaabbabbbb` is $\mathtt{a}^3\mathtt{b}^2\mathtt{ab}^3$.) Note that each block of the RLE can be represented in $O(1)$ space.

1.2 String Measures and Approximate String Matching

Hamming Distance and Pattern Matching with Mismatches

For two strings S and T of the same length n, we define the set of *mismatches* between S and T as

$$\mathrm{Mis}(S,T) := \{i \in [\,0\,..\,n\,) \mid S[\,i\,] \neq T[\,i\,]\}.$$

Now, the *Hamming distance* of S and T is the number of mismatches between S and T, that is, $\delta_H(S,T) := |\mathrm{Mis}(S,T)|$. The Hamming distance has a triangle inequality:

⌜ Lemma 1.1 (Triangle Inequality, Hamming distance). *Any three strings A, B, and C of the same length satisfy*

$$\delta_H(A,C) + \delta_H(C,B) \geq \delta_H(A,B) \geq |\delta_H(A,C) - \delta_H(C,B)|.$$

▭ Proof. Observe that for any $\pi \in \mathrm{Mis}(A,B)$, and any string C, we have at least one and at most both of $\pi \in \mathrm{Mis}(A,C)$ and $\pi \in \mathrm{Mis}(C,B)$; yielding the inequalities, respectively. ⌟

As we are often concerned with the Hamming distance of a string S and a prefix of T^∞ for a string T, we write

$$\mathrm{Mis}(S,T^*) := \mathrm{Mis}(S,T^\infty[\,0\,..\,|S|\,))$$

and $\delta_H(S,T^*) := |\mathrm{Mis}(S,T^*)|$.

Now, for two strings P (also called *pattern*) and T (also called *text*), and a positive integer k (also called *threshold*), we say that there is a k-mismatch occurrence of P in T at position i if

$$\delta_H(P, T[\,i\,..\,i+m\,)) \le k.$$

We write $\mathrm{Occ}_k^H(P,T)$ for the set of all positions of k-mismatch occurrences of P in T, that is,

$$\mathrm{Occ}_k^H(P,T) := \{i \mid \delta_H(P, T[\,i\,..\,i+m\,) \le k)\}.$$

Lastly, we define the *pattern matching with mismatches* problem.

⌜ PROBLEM 1.2 (PATTERN MATCHING WITH MISMATCHES). *Given a pattern P, a text T, and a threshold k, compute the set* $\mathrm{Occ}_k^H(P,T)$. ⌟

Note that depending on the use case, we may want to compute only the size $|\mathrm{Occ}_k^H(P,T)|$ or the leftmost position in the set $\mathrm{Occ}_k^H(P,T)$; especially if the set $\mathrm{Occ}_k^H(P,T)$ is "huge". However, our algorithms easily adapt to these settings.

Edit Distance and Pattern Matching with Edits

The *edit distance* (also known as *Levenshtein distance*) between two strings S and T, denoted by $\delta_E(S,T)$, is the minimum total cost of a sequence of unit cost edit operations (insertions, deletions, substitutions) required to transform S into T. The edit distance has a triangle inequality as well:

⌜ LEMMA 1.3 (TRIANGLE INEQUALITY, EDIT DIS.). *Any three strings A, B, and C of the same length satisfy*

$$\delta_E(A,C) + \delta_E(C,B) \ge \delta_E(A,B) \ge |\delta_E(A,C) - \delta_E(C,B)|.$$

▭ PROOF. The edit sequence from A to C to B is a valid edit sequence from A to B; the lower bound follows by rearranging. ⌟

Similarly to the Hamming distance, we write

$$\delta_E(S, T^*) := min\{\delta_E(S, T^\infty[\,0\,..\,j\,]) \mid j \in \mathbb{Z}\}$$

to denote the minimum edit distance between a string S and any prefix of a string T^∞. Further, we write

$$\delta_E(S, {}^*T^*) := min\{\delta_E(S, T^\infty[\,i\,..\,j\,]) \mid i, j \in \mathbb{Z}\}$$

for the minimum edit distance between S and any substring of T^∞.

Now, for two strings P (also called *pattern*) and T (also called *text*), and a positive integer k (also called *threshold*), we say that there is a k-error or k-edits occurrence of P in T at position i if

$$\delta_E(P, T[\,i\,..\,j\,)) \leq k \text{ for some } j \geq i.$$

We write $\mathrm{Occ}^E_k(P, T)$ to denote the set of all positions of k-error occurrences of P in T, that is,

$$\mathrm{Occ}^E_k(P, T) := \{i \mid \exists j \geq i : \delta_E(P, T[\,i\,..\,j\,) \leq k)\}.$$

Lastly, we define the *pattern matching with edits* problem.

⌜ PROBLEM 1.4 (PATTERN MATCHING WITH EDITS). *Given a pattern P, a text T, and a threshold k, compute the set* $\mathrm{Occ}^E_k(P, T)$. ⌟

We may want to compute only the size $|\mathrm{Occ}^E_k(P, T)|$ or the leftmost position in the set $\mathrm{Occ}^E_k(P, T)$; especially if the set $\mathrm{Occ}^E_k(P, T)$ is “huge”. Again, our algorithms easily adapt to these settings.

The PILLAR Model

In order to unify the implementations of our approach in the different considered settings for pattern matching, we introduce the PILLAR model.[9] The PILLAR model captures certain primitive operations which can be implemented efficiently in all considered settings. Thus, in the algorithmic chapters of this part, 4 and 6, we bound the running times in terms of PILLAR operations—if the algorithm uses more time than PILLAR operations, we also specify the extra running time.

9. The name stems from the basic operations used in the model.

Further, we implement the PILLAR model in the static, fully-compressed and dynamic settings. For each setting, we first show how to implement each of the primitive PILLAR operations. Note that for our main algorithms, it is not relevant how specific operations are implemented as long as the implementations meet the running time bounds. In the interest of keeping this part short, we hence defer most of the concrete implementations to the literature.[10]

10. In particular, a non-expert reader need not worry about the details of the implementation; an expert reader may already know most of the implementation details.

2.1 Basic Building Blocks of the PILLAR Model

In the PILLAR model, we are given a family of strings $\mathcal{X}$ for preprocessing. The elementary objects are fragments $X[\,\ell\,..\,r\,)$ of strings $X \in \mathcal{X}$. Initially, the model provides access to each $X \in \mathcal{X}$ interpreted as $X[0\,..\,|X|)$. Other fragments can be obtained through an Extract operation.

- Extract(S, ℓ, r): Given a fragment S and positions $0 \le \ell \le r \le |S|$, extract the (sub)fragment $S[\,\ell\,..\,r\,)$. If $S = X[\,\ell'\,..\,r'\,)$ for $X \in \mathcal{X}$, then $S[\,\ell\,..\,r\,)$ is defined as $X[\,\ell' + \ell\,..\,\ell' + r\,)$.

Further, the following primitive operations are supported in the `PILLAR` model:

- `LCP`(S, T): Compute the length of the longest common prefix of S and T.
- `LCP`$^R(S, T)$: Compute the length of the longest common suffix of S and T.
- `IPM`(P, T): Assuming $|T| \leq 2|P|$, compute $\mathrm{Occ}(P, T)$ (represented as an arithmetic progression with difference $\mathrm{per}(P)$).
- `Access`(S, i): Retrieve the character $S[\,i\,]$.
- `Length`(S): Compute the length $|S|$ of the string S.

We now collect known results into a toolbox that is to be used in 4 and 6. We start with an operation that allows us to compute the period of a given string.

FACT 2.1 (`Period`(S), [73, 72]). *Given a string S, we can compute $\mathrm{per}(S)$ or declare that $\mathrm{per}(S) > |S|/2$ in $O(1)$ time in the `PILLAR` model.* ◻

Next, we have an operation to "revert" a rotation of a string.

FACT 2.2 (`Rotations`(S, T), [73, 72]). *Given strings S and T, we can find all integers j such that $T = \mathrm{rot}^j(S)$ in $O(1)$ time in the `PILLAR` model. The output is represented as an arithmetic progression.* ◻

Next, we generalize the primitive `LCP` and `LCP`^{R} operations to also support fragments of infinite repetitions of a string.

FACT 2.3 (`LCP`(S, Q^∞), [72, FACT 2.5.2]; [6]). *Given strings S and Q, we can compute `LCP`(S, Q^∞) in $O(1)$ time in the `PILLAR` model.* ◻

COROLLARY 2.4 (`LCP`$(S, Q^\infty[\,\ell\,..\,))$). *Given strings S and Q, and an integer $\ell > 0$, we can compute `LCP`$(S, Q^\infty[\,\ell\,..\,))$ in $O(1)$ time in the `PILLAR` model.*

PROOF. We first compute $\texttt{LCP}(S, Q[\,\ell \bmod |Q| \mathinner{.\,.} |Q|\,))$ by a primitive operation. If we reach the end of the string Q, we continue with an $\texttt{LCP}$ operation from Fact 2.3. ◻

Next, we provide an equality check.

FACT 2.5 (EQUALITY, [72, FACT 2.5.2]). *Given strings S and T, we can check $S = T$ in $O(1)$ time in the* $\texttt{PILLAR}$ *model.* ◻

Lastly, we discuss an operation to find all exact occurrences of a given string P in another given string T.

LEMMA 2.6 ($\texttt{ExactMatches}(P, T)$). *Let T denote a string of length n and let P denote a string of length m. We can compute the set* $\mathrm{Occ}(P, T)$ *using $O(n/\operatorname{per}(P))$ time and $O(n/m)$* $\texttt{PILLAR}$ *operations.*

PROOF. We perform an $\texttt{IPM}(P, T_i)$ query with

$$T_i := T[\,\lfloor i \cdot m/2 \rfloor \mathinner{.\,.} min\{n, \lfloor (i+3) \cdot m/2 \rfloor - 1\}\,)$$

for each $0 \le i < \lfloor 2n/m \rfloor$; that is a total of $O(n/m)$ $\texttt{PILLAR}$ operations. Each occurrence of P in T corresponds to a single occurrence of P in a single T_i. Further, each $\texttt{IPM}(P, T_i)$ query returns an arithmetic progression with difference $\operatorname{per}(P)$, which thus consists of $O(m/\operatorname{per}(P))$ elements. Hence, the total number of elements of all arithmetic progressions is $O(n/\operatorname{per}(P))$. ◻

Another important concept for us are *generators*.

DEFINITION 2.7 (GENERATOR OF A SET). *For an (ordered) set S, an* $(O(P), O(Q))$-time generator of *S is a data structure that after $O(P)$-time initialization in the* $\texttt{PILLAR}$ *model, supports the following operation:*

- $\texttt{Next}$*: In the i-th call of* $\texttt{Next}$*, return the i-th largest element of the set S or return $\perp$ if $i > |S|$, using $O(Q(i))$ time in the* $\texttt{PILLAR}$ *model.* ◻

Note that a generator is not specific to the `PILLAR` model. We use generators to obtain positions where two strings differ—either by a mismatch or by an edit. Consult Sections 4.1 and 6.1 for the details, as well as other `PILLAR` operations that are specific to pattern matching with mismatches or edits, respectively.

2.2 Implementing the `PILLAR` Model in the Standard Setting

Next, we present implementations of the `PILLAR` model. We start with the standard setting. Unsurprisingly, this turns out to be a straight-forward application of known tools for strings.

Let us denote the total length of all strings in $\mathcal{X}$ by n. In the standard setting, the implementations of $\texttt{Access}(S,i)$, $\texttt{Length}(S)$ is trivial as we explicitly store all strings in $\mathcal{X}$. Further, we can retrieve a pointer to the array storing each string in $O(1)$ time using a perfect hashmap [48].

Next, we efficiently implement $\texttt{LCP}(S,T)$ queries as follows. We construct the generalized suffix tree for the elements of $\mathcal{X}$ in $O(n)$ time [45] and preprocess it within the same time complexity so that we can support $O(1)$-time lowest common ancestor queries [9].

Next, for efficient $\texttt{IPM}(P,T)$ queries, we build the data structure of Kociumaka et al. [73], Kociumaka [72]. This data structure is encapsulated in the following fact for the concatenation of the elements of $\mathcal{X}$.

Fact 2.8 ([73, 72]). *For every string S of length n, there is a data structure of size $O(n)$, which can be constructed in $O(n)$ time and answers $\texttt{IPM}(P,T)$ queries in $O(1)$ time for fragments P and T of S.* ◿

We summarize the above discussion in Lemma 2.9.

Lemma 2.9. *After processing a collection of strings for $O(n)$-time, we can perform every `PILLAR` operation in $O(1)$ time each.* ◿

2.3 Implementing the PILLAR Model in the Fully-Compressed Setting

Next, we focus on the fully-compressed setting, where we want to solve approximate pattern matching when both the text and the pattern are given as a straight-line programs—that is, in this setting, we maintain a collection $\mathcal{X}$ of straight-line programs and show how to implement the primitive PILLAR operations on this collection. We start with a short exposition on straight-line programs and related concepts.

Straight-Line Programs

For a context-free grammar $\mathcal{G}$, we write $N_{\mathcal{G}}$ to denote the set of non-terminals of $\mathcal{G}$ and call the elements of $\mathcal{S}_G := N_{\mathcal{G}} \cup \Sigma$ the symbols of $\mathcal{G}$. A *straight line program* (SLP) $\mathcal{G}$ is a context-free grammar that consists of a set $N_{\mathcal{G}} = \{A_1, \dots, A_n\}$ of non-terminals, such that each $A \in N_{\mathcal{G}}$ is associated with a unique production rule $A \to f_{\mathcal{G}}(A)$, where $f_{\mathcal{G}}(A) \in \mathcal{S}_{\mathcal{G}}^*$. For SLPs given as input, we can assume without loss of generality that each production rule is of the form $A \to BC$ for some symbols B and C.[11]

11. That is, the given SLP is in Chomsky normal form.

Every symbol $A \in \mathcal{S}_{\mathcal{G}}$ generates a unique string, which we call $gen(A) \in \Sigma^*$. We can obtain the string $gen(A)$ from A by repeatedly replacing each non-terminal with its production. Sometimes we also need an *uncompressed* representation of (the part of) a grammar rooted at A; we call this representation the *parse tree* $PT(A)$. Formally, we define $PT(A)$ as a tree that has a root labeled with A and that has zero or more subtrees:

- If A is a terminal, the root has no subtrees.
- If A is a non-terminal $A \to B_1 \cdots B_p$, then the subtrees of A are $PT(B_i)$, in increasing order of i.

Observe that the leaves of $PT(A)$ from left to right spell out $gen(A)$.

The parse tree $PT_{\mathcal{G}}$ of $\mathcal{G}$ is the parse tree of the starting symbol $A_n \in N_{\mathcal{G}}$; we have $gen(A_n) = S$, where S is the unique string gener-

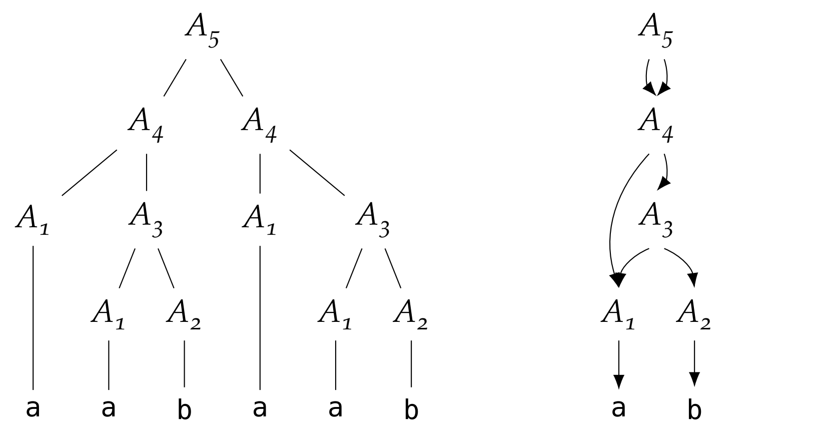
$A_5 \rightarrow A_4A_4$

$A_4 \rightarrow A_1A_3$

$A_3 \rightarrow A_1A_2$

$A_2 \rightarrow \mathtt{b}$

$A_1 \rightarrow \mathtt{a}$



(a) The SLP $\mathcal{G}$. (b) The parse tree $PT_{\mathcal{G}}$. (c) The acyclic graph $H_{\mathcal{G}}$.

FIGURE 2.1. An SLP $\mathcal{G}$ generating `aabaab` and the corresponding parse tree $PT_{\mathcal{G}}$ and directed acyclic graph $H_{\mathcal{G}}$.

ated by $\mathcal{G}$. We write $gen(\mathcal{G}) := S$. Finally, an SLP can be represented naturally as a directed acyclic graph $H_{\mathcal{G}}$ of size at most $2|gen(\mathcal{G})|$. Consult Figure 2.1 for an example of an SLP, its parse tree, and the corresponding acyclic graph.

The *value* $val(v)$ of a node v in $PT_{\mathcal{G}}$ is the fragment $S[\,a\,..\,b\,]$ corresponding to the leaves $S[\,a\,], \dots, S[\,b\,]$ in the subtree of v.[12]

12. Note that $val(v)$ is an occurrence of $gen(A)$ in $gen(\mathcal{G})$, where A is the label of v.

Given an SLP $\mathcal{G}$ of size n, we can compute $|gen(\mathcal{G})|$ in $O(n)$ time using a straight-forward bottom-up dynamic programming approach. Similarly, Bille, Landau, Raman, Sadakane, Satti, and Weimann [12] have shown that we can efficiently access any character in $gen(\mathcal{G})$.

FACT 2.10 (BILLE ET AL. [12]). *An SLP $\mathcal{G}$ of size n, generating a string S of size N, can be preprocessed in time $O(n \log(N/n))$ so that, for any $i \in [\,0\,..\,N\,)$, we can access $gen(\mathcal{G})[\,i\,]$ in $O(\log N)$ time.* ⌟

To answer `LCP` queries, we use the algorithm by I [60].

⌜ FACT 2.11 (I [60]). *An* SLP $\mathcal{G}$ *of size* n*, generating a string* S *of size* N*, can be preprocessed in time* $O(n \log(N/n))$ *so that for any* i *and* j*, we can compute* $\texttt{LCP}(S[i\,..\,N), S[j\,..\,N))$ *in* $O(\log N)$ *time.* ⌟

Finally, we discuss how to "concatenate" two SLPs. Given two SLPs $\mathcal{G}_1$ and $\mathcal{G}_2$, with $gen(\mathcal{G}_1) = S_1$ and $gen(\mathcal{G}_2) = S_2$, we can construct an SLP generating S_1S_2 in $O(|\mathcal{G}_1| + |\mathcal{G}_2|)$ time as follows. We first rename the non-terminals in $N_{\mathcal{G}_2}$ to make sure that they are disjoint from the non-terminals in $N_{\mathcal{G}_1}$. Next, let R_1 and R_2 denote the starting non-terminals of $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. We construct a new SLP $\mathcal{G}$ with $N_{\mathcal{G}} := N_{\mathcal{G}_1} \cup N_{\mathcal{G}_2} \cup \{R\}$, where R has production rule $R \to R_1R_2$. Note that this procedure can be applied to more than two strings at once: We first apply a global renaming, and then repeatedly "concatenate" two strings in the collection. Note further that we can place distinct sentinel characters not in Σ between concatenated strings that cannot be "crossed" by longest common prefix/suffix queries.

Let us now denote the total size of all SLPs in $\mathcal{X}$ by n, and the total length of all strings generated by those SLPs by N. We can access each SLP in $O(1)$ time using a perfect hashmap.

The above discussion on computing the length of the string generated by an SLP and Fact 2.10 imply that we can preprocess $\mathcal{X}$ in $O(n \log(N/n))$ time so that for any SLP $\mathcal{G} \in \mathcal{X}$, a $\texttt{Length}(gen(\mathcal{G}))$ operation requires $O(1)$ time, while an $\texttt{Access}(gen(\mathcal{G}), i)$ operation requires $O(\log N)$ time, for any $i \in [0\,..\,|gen(\mathcal{G})|)$.

For efficiently answering LCP queries, we rely on Fact 2.11. We build I's data structure for an SLP that generates the concatenation of all elements in the multi-set $\{gen(\mathcal{G}) \mid \mathcal{G} \in \mathcal{X}\}$, with distinct sentinel characters not in Σ between them. Therefore, after an $O(n \log(N/n))$-time preprocessing, each LCP operation requires $O(\log N)$ time.

To implement the IPM operation efficiently, we rely on the following known result.

Fact 2.12 ([69, Section 6.1 (full version)]). *An* SLP $\mathcal{G}$ *of size* n*, generating a string* S *of size* N*, can be preprocessed in time* $O(n \log N)$ *so that, given a fragment* $T = S[j\,..\,j+\nu)$*, and a fragment* $P = S[i\,..\,i+\mu)$ *with* $|T| \le 2|P|$*, we can compute* $\mathrm{Occ}(P, T)$ *in the time required by* $O(log^2 N)$ `LCP` *and* $\mathtt{LCP}^R$ *operations on fragments of* S*.*[13]

13. Internally, this operation is implemented using the recompression technique due to Jeż [65, 66].

In total, we have thus proved the following result.

Theorem 2.13. *Given a collection of* SLPs *of total size* n*, generating strings of total length* N*, each* `PILLAR` *operation can be performed in* $O(log^3 N)$ *time after an* $O(n \log N)$*-time preprocessing.*

2.4 Implementing the PILLAR Model in the Dynamic Setting

Lastly, we consider the dynamic setting. In particular, we want to maintain a collection of non-empty persistent strings $\mathcal{X}$ that is initially empty and undergoes the following updates:

- `makestring`(U): Insert a non-empty string U to $\mathcal{X}$.
- `concat`(U, V): Insert UV to $\mathcal{X}$, for $U, V \in \mathcal{X}$.
- `split`(U, i): Insert $U[0\,..\,i)$ and $U[i\,..\,|U|)$ in $\mathcal{X}$, for $U \in \mathcal{X}$ and $i \in [0\,..\,|U|)$.

Let N denote an upper bound on the total length of all strings in $\mathcal{X}$ throughout the execution of the algorithm. Gawrychowski et al. [52] presented a data structure that efficiently maintains such a collection and allows for efficient longest common prefix queries.

Fact 2.14 ([52]). *A collection* $\mathcal{X}$ *of non-empty persistent strings of total length* N *can be dynamically maintained with update operations* `makestring`(U), `concat`(U, V), `split`(U, i) *requiring time* $O(log N + |U|)$*,* $O(log N)$*, and* $O(log N)$*,*[14] *respectively, so that* `LCP`(U, V) *queries for* $U, V \in \mathcal{X}$ *can be answered in time* $O(1)$*.*

14. These running times hold w.h.p.

We maintain the lengths of the strings in $\mathcal{X}$ explicitly. Upon a `makestring` operation, we naively compute the length of U, while upon a `concat` or a `split` operation, we can compute the lengths of the strings that are inserted in $\mathcal{X}$ in constant time from the arguments of the operation.

We can compute the `LCP` and $\mathtt{LCP}^R$ operations for arbitrary fragments of elements of $\mathcal{X}$ in time $O(\log N)$ by first performing a constant number of `split` operations to add the corresponding fragments to the collection, and then using a `LCP` query on the fragments.

Next, we wish to obtain fast `Access` and `IPM` operations. To that end, we need to take a (slightly) closer look at the data structure of Gawrychowski et al. [52]. For each string of the collection $\mathcal{X}$, the data structure of [52] maintains a recompressed SLP that is of depth $O(\log N)$ w.h.p.

We now show that $\mathtt{Access}(X, i)$ for $X \in \mathcal{X}$ can be performed efficiently. Given a string $X \in \mathcal{X}$, a pointer to the root of the parse tree of X can be retrieved in $O(1)$ time. Even though the parse trees are not maintained explicitly, given a pointer to any node v in the parse tree of X, we can obtain in constant time the endpoints a, b of the fragment $val(v) = X[a\,..\,b]$, the degree of v, a pointer to the parent of v, and a pointer to the j-th child of v, provided that such a child exists. Thus, we can implement the $\mathtt{Access}(X, i)$ operation in time proportional to the height of the parse tree, that is, in $O(\log N)$ time w.h.p.

Finally, we need to show that `IPM` operations can be performed efficiently. To that end, one can show that Fact 2.12 also holds in this setting;[15] in particular, combining Facts 2.12 and 2.14, we can answer $\mathtt{IPM}(P, T)$ queries in $O(\log^2 N)$ time (w.h.p.) by performing $O(\log N)$ `split` operations and $O(\log^2 N)$ `LCP` queries.

15. Consult [25, Section 4] for details.

We summarize the above discussion in Theorem 2.15.

THEOREM 2.15. *We can maintain a collection $\mathcal{X}$ of non-empty persistent strings of total length N under* $\mathtt{makestring}(U)$, $\mathtt{concat}(U, V)$, $\mathtt{split}(U, i)$ *requiring time $O(\log N + |U|)$, $O(\log N)$ and $O(\log N)$, respectively, so that* `PILLAR` *operations take time $O(\log^2 N)$.*[16]

16. All running time bounds hold w.h.p.

Structural Insights into Pattern Matching with Mismatches

In this chapter, we provide insight into the structure of k-mismatch occurrences of a pattern P in a text T. In particular, we show the following tight characterization.

Theorem 3.1. *Given a pattern P of length m, a text T of length n, and a threshold $k \leq m$, at least one of the following holds:*

- *We have $|\mathrm{Occ}^H_k(P,T)| \leq 576 \cdot n/m \cdot k$.*
- *There is a primitive Q with $|Q| \leq m/128k$ and $\delta_H(P, Q^*) < 2k$.*

3.1 Characterization of the Periodic Case

In order to prove Theorem 3.1, we first need to discuss in more detail the (approximately) periodic case, that is, the case when we have $\delta_H(P, Q^*) < 2k$. In particular, we prove the following statement.

Lemma 3.2. *Let P denote a string of length m, let T denote a string of length $n \leq {}^3/_2\, m$, and let $k \leq m$ denote a non-negative integer. Suppose that both $T[\,0\,..\,m\,)$ and $T[\,n-m\,..\,n\,)$ are k-mismatch occurrences of P, that is, $\{0, n-m\} \subseteq \mathrm{Occ}^H_k(P,T)$. If there are a positive integer $d \geq 2k$ and a primitive string Q with $|Q| \leq m/8d$ and $\delta_H(P,Q^*) \leq d$, then each of following holds:*

1. *Every position in $\mathrm{Occ}^H_k(P,T)$ is a multiple of $|Q|$.*
2. *The string T satisfies $\delta_H(T, Q^*) \leq 3d$.*
3. *The set $\mathrm{Occ}^H_k(P,T)$ can be decomposed into $3d(d+1)$ arithmetic progressions with difference $|Q|$.*
4. *If $\delta_H(P,Q^*) = d$, then $|\mathrm{Occ}^H_k(P,T)| \leq 6d$.*

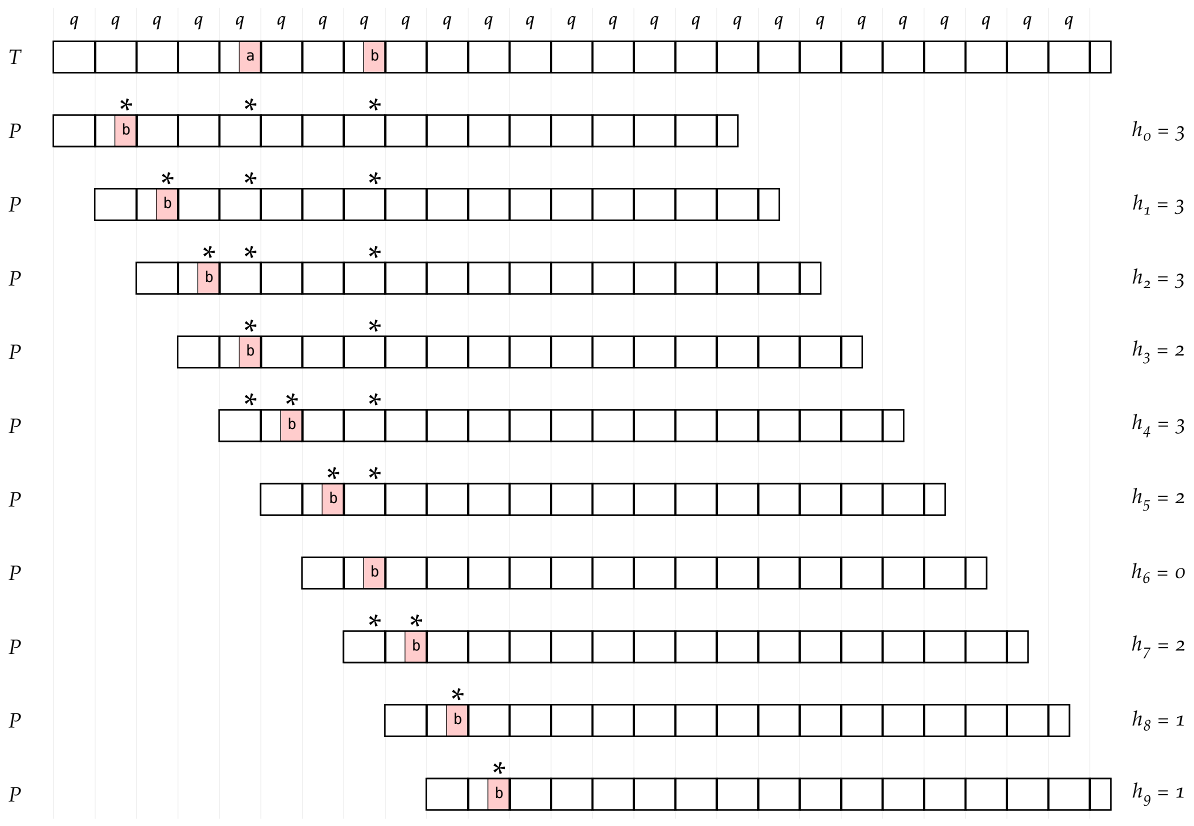


FIGURE 3.1. Examples for a string T of length n, a string P of length m and their alignments at positions jq for $0 \le j \le (n - m)/q$. Each of the strings T and P decomposes into blocks of size q (and a possibly shorter last block); in each block the mismatches to the string Q are denoted by a colored box. We have $d_T = 2$ and $d_P = 1$. Mismatches between P and T are denoted by asterisks above the corresponding positions in P; the number of mismatches for each alignment (that is h_j) is written after the pattern in each alignment.

Observe that mismatches in T translate to mismatches in P and vice versa—unless they overlap and possibly even neutralize each other. In total, there are only $d_T(2d_P + 1) = 6$ changes in adjacent values of the sequence h_j and a single entry with $h_j = 0 \le d_P/2$.

Before proving Lemma 3.2, we characterize the values

$$\delta_H(T[\, j|Q| \,..\, j|Q| + m\,), P)$$

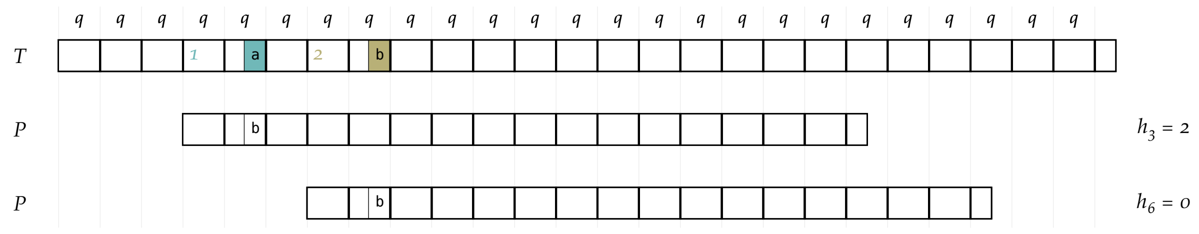
in the case that *both* $\delta_H(P, Q^*)$ and $\delta_H(T, Q^*)$ are small.

Lemma 3.3. *Let P denote a pattern of length m and let T denote a text of length $n \leq {}^3\!/_2\, m$. Further, let Q denote a string of length q and set $d_P := \delta_H(P, Q^*)$ and $d_T := \delta_H(T, Q^*)$. Then, the sequence of values $h_j := \delta_H(T[\, jq \,..\, jq + m\,), P)$ for $0 \leq j \leq (n - m)/q$ contains at most $d_T(2d_P + 1)$ entries h_j with $h_j \neq h_{j+1}$ and, unless $d_P = 0$, at most $2d_T$ entries h_j with $h_j \leq d_P/2$.*

Proof. First, consult Figure 3.1 for an illustration of an example. Intuitively, first consider a pattern with $d_P = 0$. In this case, the values h_j directly correspond to the mismatches between Q^∞ and the corresponding fragment $T[\, jq \,..\, jq + m\,)$. Hence, h_j may differ from h_{j+1} only if there is a mismatch between Q and $T[\, jq \,..\, (j + 1)q\,)$ or between Q and $T[\, jq + m \,..\, (j + 1)q + m\,)$—Hence, in total at most d_T times. Now, each mismatch between P and Q^∞ usually just increases the corresponding values h_j for all j—unless such a mismatch overlaps with a mismatch between T and Q^∞; in such a case the corresponding value h_j may differ from both the preceding value and the following value in the sequence of the h_j values. This gives the claimed bound of $d_T + 2d_H d_T$.

Further, if $d_H > 0$, then in any alignment with $h_j \leq d_P/2$ at least $d_P/2$ mismatches between P and Q^∞ are "neutralized" by mismatches between T and Q^∞. As every such mismatch between T and Q^∞ may "neutralize" every mismatch between P and Q^∞ at most once and there are at most $d_P d_T$ possible "neutralizations" in total, there are at most $d_P d_T/(d_P/2) = 2d_T$ values $h_j \leq d_P/2$.

Formally, for every $\tau \in \mathrm{Mis}(T, Q^*)$ and $\pi \in \mathrm{Mis}(P, Q^*)$, let us put $(2 - \delta_H(P[\,\pi\,], T[\,\tau\,]))$ marks at position $\tau - \pi$ in T. For each



🟨 FIGURE 3.2. The marks that we place in the example from Figure 3.1—The number of marks are written at the corresponding position in the text with the same color as the corresponding mismatch in T. We have

$$\mu_3 = 1 = 1 + 2 - 2 = \delta_H(P, Q^*) + \delta_H(T[\,3q\,..\,3q+m\,), Q^*) - h_3$$

and

$$\mu_3 = 2 = 1 + 1 - 0 = \delta_H(P, Q^*) + \delta_H(T[\,6q\,..\,6q+m\,), Q^*) - h_6.$$

$0 \le j \le (n-m)/q$, let $\mu_j(\tau, \pi)$ denote the number of marks placed at position jq in T due to the positions τ in T and π in P, that is,

$$\mu_j(\tau, \pi) := \begin{cases} 2 - \delta_H(P[\,\pi\,], T[\,\tau\,]) & \text{if } \pi \in \mathrm{Mis}(P, Q^*) \text{ and} \\ & \tau = jq + \pi \in \mathrm{Mis}(P, Q^*), \\ 0 & \text{otherwise.} \end{cases}$$

Further, define $\mu_j := \sum_{\tau,\pi} \mu_j(\tau, \pi)$ as the total number of marks at position jq in T. Consult Figure 3.2 for a visualization for the previous example.

Next, for every $0 \le j \le (n-m)/q$, we relate the Hamming distance $h_j := \delta_H(T[\,jq\,..\,jq+m\,), P)$ to the number of marks μ_j at position jq and the Hamming distances $\delta_H(T[\,jq\,..\,jq+m\,), Q^*)$ and $\delta_H(P, Q^*)$.

CLAIM 3.4. *For each $0 \le j \le (n-m)/q$, we have $h_j = \delta_H(P, Q^*) + \delta_H(T[\,jq\,..\,jq+m\,), Q^*) - \mu_j$.*

PROOF. By definition, it suffices to prove

$$|\mathrm{Mis}(T[jq\,..\,jq+m), P)| \\ = |\mathrm{Mis}(P, Q^*)| + |\mathrm{Mis}(T[jq\,..\,jq+m), Q^*)| - \sum_{\tau,\pi} \mu_j(\tau, \pi). \tag{3.1}$$

By construction, we have $\mu_j(\tau,\pi) = 0$ whenever $\tau \neq \pi + jq$. Hence, we can prove Equation (3.1) by showing that for every position $0 \le \pi < m$ in P and every position $\tau := jq + \pi$ in T, we have:

$$\delta_H(T[\tau], P[\pi]) \\ = \delta_H(P[\pi], Q^\infty[\pi]) + \delta_H(T[\tau], Q^\infty[\tau]) - \mu_j(\tau, \pi).$$

We proceed by case distinction on whether $\pi \in \mathrm{Mis}(P, Q^*)$ and whether $\tau \in \mathrm{Mis}(T, Q^*)$.

1 If $\pi \notin \mathrm{Mis}(P, Q^*)$ and $\tau \notin \mathrm{Mis}(T, Q^*)$, then we have $P[\pi] = Q^\infty[\pi] = Q^\infty[\tau] = T[\tau]$ and thus

$$\delta_H(T[\tau], P[\pi]) = 0 = 0 + 0 - 0 \\ = \delta_H(P[\pi], Q^\infty[\pi]) + \delta_H(T[\tau], Q^\infty[\tau]) - \mu_j(\tau, \pi).$$

2 If $\pi \in \mathrm{Mis}(P, Q^*)$ and $\tau \notin \mathrm{Mis}(T, Q^*)$, then we have $P[\pi] \neq Q^\infty[\pi] = Q^\infty[\tau] = T[\tau]$ and thus

$$\delta_H(T[\tau], P[\pi]) = 1 = 1 + 0 - 0 \\ = \delta_H(P[\pi], Q^\infty[\pi]) + \delta_H(T[\tau], Q^\infty[\tau]) - \mu_j(\tau, \pi).$$

3 If $\pi \notin \mathrm{Mis}(P, Q^*)$ and $\tau \in \mathrm{Mis}(T, Q^*)$, then we have $P[\pi] = Q^\infty[\pi] = Q^\infty[\tau] \neq T[\tau]$ and thus

$$\delta_H(T[\tau], P[\pi]) = 1 = 0 + 1 - 0 \\ = \delta_H(P[\pi], Q^\infty[\pi]) + \delta_H(T[\tau], Q^\infty[\tau]) - \mu_j(\tau, \pi).$$

4 If $\pi \in \operatorname{Mis}(P, Q^*)$ and $\tau \in \operatorname{Mis}(T, Q^*)$, then we have $P[\pi] \neq Q^\infty[\pi] = Q^\infty[\tau] \neq T[\tau]$ and thus

$$\begin{aligned}\delta_H(T[\tau], P[\pi]) &= 1 + 1 - (2 - \delta_H(T[\tau], P[\pi])) \\ &= \delta_H(P[\pi], Q^\infty[\pi]) + \delta_H(T[\tau], Q^\infty[\tau]) - \mu_j(\tau, \pi).\end{aligned}$$

Combining the equations obtained for every pair of positions π and τ, we derive Equation (3.1). ◿

In particular, Claim 3.4 yields

$$\begin{aligned}h_{j+1} - h_j = {} & |\operatorname{Mis}(T, Q^*) \cap [jq + m \,..\, (j+1)q + m)| \\ & - |\operatorname{Mis}(T, Q^*) \cap [jq \,..\, (j+1)q)| - \mu_{j+1} + \mu_j.\end{aligned}$$

Hence, in order for h_{j+1} not to equal h_j, at least one of the four terms on the right hand side of the equation above must be non-zero. Let us analyze when this is possible. To that end, we first observe that the set $\operatorname{Mis}(T, Q^*) \cap [jq+m \,..\, (j+1)q+m)$ contains only elements $\tau \in \operatorname{Mis}(T, Q^*)$ with $\tau \geq m$, and that the set $\operatorname{Mis}(T, Q^*) \cap [jq \,..\, (j+1)q)$ contains only elements $\tau \in \operatorname{Mis}(T, Q^*)$ with $\tau < n-m$. Using $n \leq {}^3/_2\, m$, we observe that h_{j+1} can be different from h_j due to the first or second term at most d_T times. Further, each non-zero value in one of the terms μ_{j+1} and μ_j can be attributed to a marked position (jq or $(j+1)q$, respectively). The total number of marked positions is $d_P d_T$, so h_{j+1} can be different from h_j due one of the terms μ_{j+1} or μ_j at most $2d_P d_T$ times. In total, we conclude that the number of entries h_j with $h_j \neq h_{j+1}$ is at most $d_T(2d_P + 1)$.

Next, observe that

$$\mu_j \leq 2|\operatorname{Mis}(T, Q^*) \cap [jq \,..\, jq + m)| = 2\delta_H(T[jq \,..\, jq + m), Q^*),$$

and therefore

$$h_j = \delta_H(P, Q^*) + \delta_H(T[jq \,..\, jq + m), Q^*) - \mu_j \geq d_P - \mu_j/2.$$

Hence, $h_j \le d_P/2$ yields $\mu_j \ge d_P$, that is, that there are at least d_P marks at position jq. As the total number of marks is at most $2d_Pd_T$, the number of entries h_j with $h_j \le d_P/2$ is at most $2d_T$. ◢

We are now ready to consider a more general case, where only $\delta_H(P, Q^*)$ is small, that is, we are ready to prove Lemma 3.2.

◤ LEMMA 3.2. *Let P denote a string of length m, let T denote a string of length $n \le {}^3\!/_2\, m$, and let $k \le m$ denote a non-negative integer. Suppose that both $T[\,0\,..\,m\,)$ and $T[\,n-m\,..\,n\,)$ are k-mismatch occurrences of P, that is, $\{0, n-m\} \subseteq \mathrm{Occ}^H_k(P,T)$. If there are a positive integer $d \ge 2k$ and a primitive string Q with $|Q| \le m/8d$ and $\delta_H(P,Q^*) \le d$, then each of following holds:*

1 *Every position in $\mathrm{Occ}^H_k(P,T)$ is a multiple of $|Q|$.*
2 *The string T satisfies $\delta_H(T, Q^*) \le 3d$.*
3 *The set $\mathrm{Occ}^H_k(P,T)$ can be decomposed into $3d(d+1)$ arithmetic progressions with difference $|Q|$.*
4 *If $\delta_H(P,Q^*) = d$, then $|\mathrm{Occ}^H_k(P,T)| \le 6d$.*

▬ PROOF. Consider any position $\ell \in \mathrm{Occ}^H_k(P,T)$. By definition of a k-mismatch occurrence, we have

$$\delta_H(T[\,\ell\,..\,\ell+m\,), P) \le k \le d/2.$$

Combining this inequality with $\delta_H(P,Q^*) \le d$ via the triangle inequality (Lemma 1.1) yields $\delta_H(T[\,\ell\,..\,\ell+m\,), Q^*) \le {}^3\!/_2\, d$. Note that similarly for the position $0 \in \mathrm{Occ}^H_k(P,T)$, we obtain

$$\delta_H(T[\,0\,..\,m\,), Q^*) \le {}^3\!/_2\, d,$$

which lets us compare the overlapping parts of Q^∞. Replacing strings by superstrings and applying the triangle inequality yields

$$\begin{aligned}
&\delta_H(Q^\infty[\,\ell\,..\,m\,), Q^\infty[\,0\,..\,m-\ell\,))\\
&\quad\le \delta_H(T[\,\ell\,..\,m\,), Q^\infty[\,\ell\,..\,m\,)) + \delta_H(T[\,\ell\,..\,m\,), Q^\infty[\,0\,..\,m-\ell\,))\\
&\quad\le \delta_H(T[\,0\,..\,m\,), Q^\infty[\,0\,..\,m\,)) + \delta_H(T[\,\ell\,..\,\ell+m\,), Q^\infty[\,0\,..\,m\,))\\
&\quad= \delta_H(T[\,0\,..\,m\,), Q^*) + \delta_H(T[\,\ell\,..\,\ell+m\,), Q^*)\\
&\quad\le 3d.
\end{aligned}$$

Towards a proof by contradiction, suppose that ℓ is not an integer multiple of $|Q|$. As Q is primitive, we have

$$3d \ge \delta_H(Q^\infty[\,\ell\,..\,m\,), Q^\infty[\,0\,..\,m-\ell\,)) \ge \left\lfloor \frac{m-\ell}{|Q|} \right\rfloor \ge \left\lfloor \frac{m/2}{m/8d} \right\rfloor = 4d,$$

where the second bound follows from $\ell \le m/2$ and $|Q| \le m/8d$. The contradiction yields Item 1.

In order to prove Item 2, note that $n-m \in \mathrm{Occ}_k^H(P,T)$ is a multiple of $|Q|$. Consequently,

$$\begin{aligned}
\delta_H(T, Q^*) &= \delta_H(T[\,0\,..\,n-m\,), Q^*) + \delta_H(T[\,n-m\,..\,n\,), Q^*)\\
&\le \delta_H(T[\,0\,..\,m\,), Q^*) + {}^3\!/_2\, d \le 3d,
\end{aligned}$$

which concludes the proof of Item 2.

For a proof of Items 3 and 4, we apply Lemma 3.3. Due to Item 1, each position in $\mathrm{Occ}_k^H(P,T)$ corresponds to an entry h_j with $h_j \le k$. In particular, each block of consecutive entries $h_j, \cdots, h_r$ not exceeding k yields an arithmetic progression (with difference $|Q|$) in $\mathrm{Occ}_k^H(P,T)$. The number of entries h_j with $h_j < k \le h_{j+1}$ or $h_j > k \ge h_{j+1}$ is in total at most $3d(2d+1)$, so the number of arithmetic progressions is at most $1 + 1/2 \cdot 3d(2d+1) \le 3d(d+1)$, which proves Item 3.

In order to prove Item 4, we observe that if $d = \delta_H(P, Q^*)$, then each position in $\mathrm{Occ}^H_k(P, T)$ corresponds to an entry h_j with $h_j \le k \le d/2$. Hence, $|\mathrm{Occ}^H_k(P, T)| \le 2 \cdot 3d \le 6d$. ◿

Corollary 3.5. *Let P denote a pattern of length m, let T denote a string of length n, and let $k \le m$ denote a non-negative integer. Suppose that there are a positive integer $d \ge 2k$ and a primitive string Q with $|Q| \le m/8d$ and $\delta_H(P, Q^*) \le d$. Then, the set $\mathrm{Occ}^H_k(P, T)$ can be decomposed into $6{\cdot}n/m{\cdot}d(d{+}1)$ arithmetic progressions with difference $|Q|$. Moreover, if $\delta_H(P, Q^*) = d$, then $|\mathrm{Occ}^H_k(P, T)| \le 12 \cdot n/m \cdot d$.*

Proof. Partition the string T into $\lfloor 2n/m \rfloor$ blocks $T_0, \dots, T_{\lfloor 2n/m \rfloor - 1}$ of length less than $3/_2\, m$ each, where the i-th block starts at position $\lfloor i \cdot m/2 \rfloor$, that is, $T_i := T[\, \lfloor i \cdot m/2 \rfloor \, .. \, min\{n, \lfloor (i+3) \cdot m/2 \rfloor - 1\}\,)$. If $\mathrm{Occ}^H_k(P, T_i) \ne \{\}$, we define T'_i to be the shortest fragment of T_i containing all k-mismatch occurrences of P in T_i. As a result, T'_i satisfies the assumptions of Lemma 3.2. Hence, $\mathrm{Occ}^H_k(P, T'_i)$ can be decomposed into $3d(d+1)$ arithmetic progressions with difference $|Q|$, and $|\mathrm{Occ}(P, T'_i)| \le 6d$ if $\delta_H(P, Q^*) = d$.

In total, we conclude that $\mathrm{Occ}^H_k(P, T'_i)$ can be decomposed into $6 \cdot n/m \cdot d(d+1)$ arithmetic progressions with difference $|Q|$ and that $|\mathrm{Occ}(P, T_i)| \le 12 \cdot n/m \cdot d$ if $\delta_H(P, Q^*) = d$. ◿

3.2 Bounding the Number of Occurrences in the Non-Periodic Case

Having dealt with the (approximately) periodic case, we now turn to the general case. In particular, we show that whenever the string P is sufficiently far from being periodic, there are at most $O(k)$ occurrences of P in any other string T of length at most $3/_2\, m$.

Intuitively, we proceed (and thereby prove Theorem 3.1) as follows: We first analyze the string P for useful structure that helps bounding the number of occurrences of P in any string T. If we fail to find any special structure in P, then we observe that the string P is already very close to a periodic string with a small period (compared to $|P|$)—a case we understand thanks to the previous section.

We start by investigating the structure of any string P.

ALGORITHM 3.3 A constructive proof of Lemma 3.6.

```
$\mathcal{B} \leftarrow \{\}; \mathcal{R} \leftarrow \{\};$
**while true do**
    Consider fragment $P' = P[j \mathinner{.\,.} j + \lfloor m/8k \rfloor)$ of the next $\lfloor m/8k \rfloor$ unprocessed
      characters of $P$;
    **if** $\mathrm{per}(P') > m/128k$ **then**
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{P'\};$
        **if** $|\mathcal{B}| = 2k$ **then return** *breaks* $\mathcal{B}$;
    **else**
        $Q \leftarrow P[j \mathinner{.\,.} j + \mathrm{per}(P'));$
        Search for prefix $R$ of $P[j \mathinner{.\,.} m)$ with $\delta_H(R, Q^*) = \lceil 8k/m \cdot |R| \rceil$ and $|R| > |P'|$;
        **if** *such* $R$ *exists* **then**
            $\mathcal{R} \leftarrow \mathcal{R} \cup \{(R, Q)\};$
            **if** $\sum_{(R,Q) \in \mathcal{R}} |R| \geq 3/8 \cdot m$ **then**
                **return** *repetitive regions (and their corresponding periods)* $\mathcal{R}$;
        **else**
            Search for suffix $R'$ of $P$ with $\delta_H(R', \mathrm{rot}^{|R'|-m+j}(Q)^*) = \lceil 8k/m \cdot |R'| \rceil$ and
              $|R'| \geq m - j$;
            **if** *such* $R'$ *exists* **then return** *repetitive region* $(R', \mathrm{rot}^{|R'|-m+j}(Q))$;
            **else return** *approximate period* $\mathrm{rot}^{j}(Q)$;
```

LEMMA 3.6. *Let P denote a string of length m and let $k \le m$ denote a positive integer. Then, at least one of the following holds:*

1 *The string P contains $2k$ disjoint* breaks $B_1, \dots, B_{2k}$ *each having periods* $\operatorname{per}(B_i) > m/128k$ *and length* $|B_i| = \lfloor m/8k \rfloor$.
2 *The string P contains disjoint* repetitive regions $R_1, \dots, R_r$ *of total length* $\sum_{i=1}^{r} |R_i| \ge 3/8 \cdot m$ *such that each region R_i satisfies* $|R_i| \ge m/8k$ *and has a primitive* approximate period Q_i *with* $|Q_i| \le m/128k$ *and* $\delta_H(R_i, Q_i^*) = \lceil 8k/m \cdot |R_i| \rceil$.
3 *The string P has a primitive* approximate period Q *that satisfies* $|Q| \le m/128k$ *and* $\delta_H(P, Q^*) < 8k$.

PROOF. We prove the claim constructively, that is, we either construct a set $\mathcal{B}$ of $2k$ breaks, a set $\mathcal{R}$ of repetitive regions, or if we fail to do so, detect that the string P has an approximate period Q with the desired properties.

We process the string P from left to right as follows: If the fragment P' of the next $\lfloor m/8k \rfloor$ (unprocessed) characters of P has a long period, we have found a new break and continue (or return the found set of $2k$ breaks). Otherwise, if P' has a short period Q, we try to extend the fragment P' (to the right) into a repetitive region. If we succeed, we have found a new repetitive region and continue (or return the found set of repetitive regions if the total length of all repetitive regions found so far is at least $3/8 \cdot m$). If we fail to construct a new repetitive region, then we conclude that the suffix of P starting with P' has an approximate period Q. We try to construct a repetitive region by extending this suffix to the left, dropping all other repetitive regions computed so far. If we fail again, we declare that Q is an approximate period of the string P. Consider Algorithm 3.3 for a detailed description.

Note that by construction, all breaks in the set $\mathcal{B}$ and repetitive regions in the set $\mathcal{R}$ returned by the algorithm are disjoint and satisfy the claimed properties. To prove that the algorithm is also correct when it fails to find a new repetitive region, we start by upper bounding the length of the processed prefix of P.

Claim 3.7. *Any new fragment $P[\,j\,..\,j+\lfloor m/8k\rfloor\,)$ of $\lfloor m/8k\rfloor$ unprocessed characters of P that we consider starts at a position $j < 5/8\cdot m$.*

Proof. Whenever we turn to a new fragment $P[\,j\,..\,j+\lfloor m/8k\rfloor\,)$, the string $P[\,0\,..\,j\,)$ has been partitioned into breaks and repetitive regions. The total length of breaks is less than $2k\lfloor m/8k\rfloor \le 2/8\cdot m$, and the total length of repetitive regions is less than $3/8\cdot m$. Hence, $j < 5/8\cdot m$, yielding the claim.

Note that Claim 3.7 also shows that whenever we consider a new fragment P' of $\lfloor m/8k\rfloor$ characters, there is indeed such a fragment, that is, P' is well-defined.

Now consider the following case: For a fragment $P' = P[\,j\,..\,j+\lfloor m/8k\rfloor\,)$ (that is not a break) and its corresponding period $Q = [\,j\,..\,j+\operatorname{per}(P')\,)$, we fail to obtain a new repetitive region R. In this case, we search for a repetitive region R' of length $|R'|\ge m-j$ that is a suffix of P and has an approximate period $Q' := \operatorname{rot}^{|R'|-m+j}(Q)$. If we find such an R', then $|R'|\ge m-j \ge m-5/8\cdot m = 3/8\cdot m$ by Claim 3.7, so R' is long enough to be reported on its own. However, if we fail to find such an R', we need to show that $\operatorname{rot}^j(Q)$ can be reported as an approximate period of P, that is, $\delta_H(P,\operatorname{rot}^j(Q)^*) < 8k$.

We first show that $\delta_H(P[\,j\,..\,m\,),Q^*) < \lceil 8k/m\cdot(m-j)\rceil$. For this, we inductively prove that the values

$$\Delta_\rho := \lceil 8k/m\cdot\rho\rceil - \delta_H(P[\,j\,..\,j+\rho\,),Q^*)$$

for $|P'|\le\rho\le m-j$ are all at least 1. In the base case of $\rho = |P'|$, we have $\Delta_\rho = 1-0$ because Q is the string period of P'. To carry out an inductive step, suppose that $\Delta_{\rho-1}\ge 1$ for some $|P'|<\rho\le m-j$. Notice that $\Delta_\rho \ge \Delta_{\rho-1}-1\ge 0$: The first term in the definition of Δ_ρ has not decreased, and the term $\delta_H(P[\,j\,..\,j+\rho\,),Q^*)$ may have increased by at most one compared to $\Delta_{\rho-1}$. Moreover, $\Delta_\rho \ne 0$ because $R = P[\,j\,..\,j+\rho\,)$ could not be reported as a repetitive region. Since Δ_ρ is an integer, we conclude that $\Delta_\rho\ge 1$. This inductive reasoning ultimately shows that $\Delta_{m-j} > 0$, that is, $\delta_H(P[\,j\,..\,m\,),Q^*) < \lceil 8k/m\cdot(m-j)\rceil$.

A symmetric argument holds for values

$$\Delta'_\rho := \lceil 8k/m \cdot \rho \rceil - \delta_H(P[\, m-\rho \,..\, m\,), \mathrm{rot}^{\rho-m+j}(Q)^*)$$

for $m - j \leq \rho \leq m$ because no repetitive region R' was found as an extension of $P[\, j \,..\, m\,)$ to the left. Hence, $\delta_H(P, \mathrm{rot}^j(Q)^*) < 8k$, that is, $\mathrm{rot}^j(Q)$ is an approximate period of P. ◢

In the next steps, we discuss how to exploit the structure obtained by Lemma 3.6. First, we discuss the case that a string P contains $2k$ disjoint breaks.

LEMMA 3.8. *Let P denote a pattern of length m, let T denote a text of length m, and let $k \leq m$ denote a positive integer. Suppose that P that contains $2k$ disjoint breaks $B_1, \dots, B_{2k} \preccurlyeq P$ each satisfying* $\mathrm{per}(B_i) \geq$ *$m/128k$. Then, $|\mathrm{Occ}^H_k(P, T)| \leq 256 \cdot n/m \cdot k$.*

PROOF. For every break $B_i = P[\, b_i \,..\, b_i + |B_i|\,)$ we mark a position j in T if $j + b_i \in \mathrm{Occ}(B_i, T)$.

CLAIM 3.9. *We place at most $256 \cdot n/m \cdot k^2$ marks in T.*

PROOF. Fix a break B_i. Notice that positions in $\mathrm{Occ}(B_i, T)$ are at distance at least $\mathrm{per}(B_i)$ from each other. Hence, for the break B_i, we place at most $128 \cdot n/m \cdot k$ marks in T. In total, we therefore place at most $2k \cdot 128n/m \cdot k = 256 \cdot n/m \cdot k^2$ marks in T. ◢

In a next step, we show that every k-mismatch occurrence of P in T starts at a position with at least k marks.

CLAIM 3.10. *Each position $\ell \in \mathrm{Occ}^H_k(P, T)$ has at least k marks in T.*

PROOF. Fix an $\ell \in \mathrm{Occ}^H_k(P, T)$. Out of the $2k$ breaks, at least k breaks are matched exactly, as not matching a break exactly incurs at least one mismatch. If a break B_i is matched exactly, then we have $\ell + b_i \in \mathrm{Occ}(B_i, T)$. Hence, we have placed a mark at position ℓ. Thus, there is a mark at position ℓ for every break B_i matched exactly in the corresponding occurrence of P in T. In total, there are at least k marks at position ℓ in T. ◢

By Claims 3.9 and 3.10, we have $|\mathrm{Occ}^H_k(P,T)| \le (256 \cdot n/m \cdot k^2)/k = 256 \cdot n/m \cdot k$.

Second, we discuss how to use repetitive regions in the string P to bound $|\mathrm{Occ}^H_k(P,T)|$.

LEMMA 3.11. *Let P denote a string of length m, let T denote a string of length n, and let $k \le m$ denote a positive integer. Suppose that P contains disjoint repetitive regions $R_1, \dots, R_r$ with a total length of at least $\sum_{i=1}^r |R_i| \ge 3/8 \cdot m$ such that each region R_i satisfies $|R_i| \ge m/8k$ and has a primitive approximate period Q_i with $|Q_i| \le m/128k$ and $\delta_H(R_i, Q_i^*) = \lceil 8k/m \cdot |R_i| \rceil$. Then, $|\mathrm{Occ}^H_k(P,T)| \le 576 \cdot n/m \cdot k$.*

PROOF. Set $m_R := \sum_{i=1}^r |R_i|$. For every repetitive region $R_i = P[\, r_i \,..\, r_i + |R_i| \,)$, we define $k_i := \lfloor 4 \cdot |R_i|/m \cdot k \rfloor$, and place $|R_i|$ marks at every position j such that $j + r_i \in \mathrm{Occ}^H_{k_i}(R_i, T)$.

CLAIM 3.12. *We place at most $192 \cdot n/m \cdot k \cdot m_R$ marks.*

PROOF. We use Corollary 3.5 to bound $|\mathrm{Occ}^H_{k_i}(R_i,T)|$. For this, we set $d_i := \delta_H(R_i, Q_i^*)$ and notice that $d_i = \lceil 8k/m \cdot |R_i| \rceil \le 16k/m \cdot |R_i|$ since $|R_i| \ge m/8k$. Moreover, $d_i \ge 2k_i$ and $|Q_i| \le m/128k \le |R_i|/8d_i$ due to $d_i \le 16k/m \cdot |R_i|$. Hence, the assumptions of Corollary 3.5 are satisfied. Consequently, $|\mathrm{Occ}^H_{k_i}(R_i,T)| \le 12 \cdot n/|R_i| \cdot d_i \le 192 \cdot n/m \cdot k$ where the last inequality is due to $d_i \le 16k/m \cdot |R_i|$.

Therefore, due to R_i, we place at most $192 \cdot n/m \cdot k \cdot |R_i|$ marks. Across all repetitive regions, this sums up to $192 \cdot n/m \cdot k \cdot m_R$, yielding the claim.

In a next step, we show that every k-mismatch occurrence of P in T starts at a position with many marks.

CLAIM 3.13. *Each $\ell \in \mathrm{Occ}^H_k(P,T)$ has at least $m_R - m/4$ marks.*

PROOF. Fix an $\ell \in \mathrm{Occ}^H_k(P,T)$ and let $k'_i := \delta_H(R_i, T[\, \ell + r_i \,..\, \ell + r_i + |R_i| \,))$ denote the number of mismatches incurred by repetitive region R_i. Further, let $I := \{i \mid k'_i \le k_i\} = \{i \mid k'_i \le 4 \cdot |R_i|/m \cdot k\}$ denote the set of indices of all repetitive regions that are k_i-mismatch occurrences at the corresponding positions in T. By construction, for each $i \in I$, we have placed $|R_i|$ marks at position ℓ. Hence, the total number of marks at position ℓ is at least $\sum_{i \in I} |R_i| = m_R - \sum_{i \notin I} |R_i|$. It remains to bound the term $\sum_{i \notin I} |R_i|$. Intuitively, $i \notin I$ implies that there are at least $4k/m$ mismatches on average incurred per position of R_i. Formally, using the definition of I, we obtain

$$\begin{aligned}\sum_{i \notin I} |R_i| &= \sum_{i \notin I} \frac{4mk}{4mk} \cdot |R_i| = \frac{m}{4k} \cdot \sum_{i \notin I} (4 \cdot |R_i|/m \cdot k) \\ &< \frac{m}{4k} \cdot \sum_{i \notin I} k'_i \le \frac{m}{4k} \cdot \sum_{i=1}^{r} k'_i \le \frac{m}{4},\end{aligned}$$

where the last bound holds because, in total, all repetitive regions incur at most $\sum_{i=1}^{r} k'_i \le k$ mismatches (since all repetitive regions are pairwise disjoint). Hence, the number of marks placed is at least $m_R - m/4$, completing the proof of the claim. ◿

In total, by Claims 3.12 and 3.13, the number of k-mismatch occurrences of P in T is at most

$$\mathrm{Occ}^H_k(P,T) \le \frac{192 \cdot n/m \cdot k \cdot m_R}{m_R - m/4}.$$

As this bound is a decreasing function in m_R, the bound $m_R \ge 3/8 \cdot m$ yields the upper bound

$$\mathrm{Occ}^H_k(P,T) \le \frac{192 \cdot n/m \cdot k \cdot 3/8 \cdot m}{3/8 \cdot m - m/4} = 576 \cdot n/m \cdot k,$$

completing the proof. ◿

LEMMA 3.14. *Let P denote a string of length m, let T denote a string of length n, and let $k \le m$ denote a positive integer. If there is a primitive string Q of length at most $|Q| \le m/128k$ that satisfies $2k \le \delta_H(P, Q^*) \le 8k$, then $|\mathrm{Occ}^H_k(P,T)| \le 96 \cdot n/m \cdot k$.*

PROOF. We apply Corollary 3.5 with $d = \delta_H(P, Q^*)$. Observe that $d \ge 2k$ and that $|Q| \le m/128k \le m/8d$ due to $d \le 8k$. Hence, the assumptions of Corollary 3.5 are met. Thus, $|\mathrm{Occ}_k^H(P,T)| \leq 12 \cdot n/m \cdot d \leq 96 \cdot n/m \cdot k$. ◻

Gathering Lemma 3.6 and Lemmas 3.8, 3.11 and 3.14, we are now ready to prove Theorem 3.1.

Theorem 3.1. *Given a pattern P of length m, a text T of length n, and a threshold $k \leq m$, at least one of the following holds:*

- *We have $|\mathrm{Occ}_k^H(P,T)| \leq 576 \cdot n/m \cdot k$.*
- *There is a primitive Q with $|Q| \leq m/128k$ and $\delta_H(P,Q^*) < 2k$.*

Proof. We apply Lemma 3.6 on the string P and proceed depending on the structure found in P.

If the string P contains $2k$ disjoint breaks $B_1,\ldots,B_{2k}$ (in the sense of Lemma 3.6), we apply Lemma 3.8 and obtain that $|\mathrm{Occ}_k^H(P,T)| \leq 256 \cdot n/m \cdot k$.

If the string P contains r disjoint repetitive regions $R_1,\ldots,R_r$ (again, in the sense of Lemma 3.6), we apply Lemma 3.11 and obtain that $|\mathrm{Occ}_k^H(P,T)| \leq 576 \cdot n/m \cdot k$.

Otherwise, Lemma 3.6 ensures that there is a primitive string Q of length at most $|Q| \leq m/128k$ that satisfies $\delta_H(P,Q^*) < 8k$. If $\delta_H(P,Q^*) \geq 2k$, then Lemma 3.14 yields $|\mathrm{Occ}_k^H(P,T)| \leq 96 \cdot n/m \cdot k$. If, however, $\delta_H(P,Q^*) < 2k$, then we are in the second alternative of the theorem statement. ◻

Algorithm: Pattern Matching with Mismatches in the `PILLAR` Model

4

In this chapter, we implement the structural result (Theorem 3.1) from the previous chapter to obtain the following result.

Theorem 4.1. *Given a pattern P of length m, a text T of length n, and a positive integer $k \leq m$, we can compute (a representation of) the set* $\mathrm{Occ}^H_k(P,T)$ *using $O(n/m \cdot k^2 \log\log k)$ time plus $O(n/m \cdot k^2)$* `PILLAR` *operations.*

The algorithm follows the outline given by the proof of Theorem 3.1: We first show how to implement Lemma 3.6 to preprocess the given pattern P. Then, depending on the structure of P, we (construct and) use algorithms implementing the insights from the corresponding lemmas from the previous chapter.

4.1 Auxiliary `PILLAR` Model Operations for Pattern Matching with Mismatches

We start by introducing some commonly used operations for pattern matching with mismatches and show how to implement them efficiently in the `PILLAR` model.

Lemma 4.2 (`MismGen`(S, Q^*), `MismGen`$^R(S, {}^*Q)$). *For every pair of strings S and Q, the sets* $\mathrm{Mis}(S, Q^*)$ *and* $\mathrm{Mis}(S^R, (Q^R)^*)$ *admit $(O(1), O(1))$-time generators.*

Proof. We develop only the `MismGen` generator; `MismGen`^{R} can be obtained similarly.

Given strings S and Q, the generator itself just stores S, Q, and an index i of the position *after* the last returned value by `Next`; initially, we set i to 0.

☐ ALGORITHM **4.1** A generator for the set $\mathrm{Mis}(S, Q^*)$.

```
MismGen($S, Q^*$)
    **return** G ← $\{S \leftarrow S; Q \leftarrow Q; i \leftarrow 0\}$;
Next(G = $\{S; Q; i\}$)
    **if** $i \geq |S|$ **then return** $\perp$;
    $\pi \leftarrow$ LCP($S[i\,..\,|S|), Q^\infty[i\,..\,)$);
    $i \leftarrow i + \pi + 1$;
    **if** $i > |S|$ **then return** $\perp$;
    **else return** $i - 1$;
```

We implement the Next operation by using Corollary 2.4 to compute π = LCP($S[i\,..\,|S|), Q^\infty[i\,..\,)$). If we observe that $i + \pi = |S|$, that is, if we reached the end of the string S, then we return $\perp$. Otherwise, we report $i + \pi$ and update the index i to $i + \pi + 1$. See Algorithm 4.1 for a pseudo-code.

For the correctness, we observe that due to storing the index i, we are able to retrieve the suffixes of S and Q to be compared, so the correctness follows.

For the running time, we observe that the creation of a generator is only bookkeeping, which takes constant time. Further, the Next operation uses one call to the primitive LCP operation and a single call to the LCP operation from Corollary 2.4, which uses $O(1)$ PILLAR operations. Thus in total, the Next operation also uses $O(1)$ PILLAR operations, completing the proof. ⌟

⌜ COROLLARY 4.3 (Mismatches(S, Q^*)). *Given strings S and Q, we can compute the set* $\mathrm{Mis}(S, Q^*)$*, using* $O(\delta_H(S, Q^*) + 1)$ *primitive operations in the* PILLAR *model.*

▭ PROOF. We use a MismGen from Lemma 4.2 and call its Next operation until the Next operation returns $\perp$. The claim follows. ⌟

LEMMA 4.4 (`Verify(`S`, `T`, `k`)`). *Let S and T denote strings with $|S| = |T| = m$, and let $k \leq m$ denote a positive integer. Using $O(k)$* `PILLAR` *operations, we can check whether $\delta_H(S, T) \leq k$.*

PROOF. We use a `MismGen` from Lemma 4.2 and call its `Next` operation until either the `Next` operation returns $\perp$ (in which case we return `true`) or until we obtain the $(k+1)$st mismatch between S and T (in which case we return `false`). The claim follows.

4.2 COMPUTING STRUCTURE IN THE PATTERN

In this chapter, we show how to implement Lemma 3.6. While the proof of Lemma 3.6 is already constructive, we still need to fill in some implementation details.

LEMMA 4.5 (`Analyze(`P`, `k`)`: IMPLEMENTATION OF LEMMA 3.6). *Let P denote a string of length m and let $k \leq m$ denote a positive integer. Then, there is an algorithm that computes one of the following:*

1. *$2k$ disjoint breaks $B_1, \dots, B_{2k} \preccurlyeq P$ that each satisfy $\operatorname{per}(B_i) > m/128k$ and $|B_i| = \lfloor m/8k \rfloor$;*
2. *disjoint repetitive regions $R_1, \dots, R_r \preccurlyeq P$ of total length $\sum_{i=1}^{r} |R_i| \geq 3/8 \cdot m$ such that each region R_i satisfies $|R_i| \geq m/8k$ and is constructed along with a primitive approximate period Q_i such that $|Q_i| \leq m/128k$ and $\delta_H(R_i, Q_i^*) = \lceil 8k/m \cdot |R_i| \rceil$; or*
3. *a primitive approximate period Q of P that satisfies $|Q| \leq m/128k$ and $\delta_H(P, Q^*) < 8k$.*

The algorithm uses $O(k)$ time plus $O(k)$ `PILLAR` *operations.*

PROOF. We follow Algorithm 3.3 from the proof of Lemma 3.6: Recall that P is processed from left to right and split into breaks and repetitive regions. In each iteration, the algorithm first considers a fragment of length $\lfloor m/8k \rfloor$. This fragment either becomes the next break (if its shortest period is long enough) or is extended to the right to a repetitive region (otherwise). Having constructed sufficiently many breaks or repetitive regions of sufficiently large total

Algorithm 4.2 A PILLAR model implementation of Algorithm 3.3.

```
Analyze($P, k$)
    $j \leftarrow 0; r \leftarrow 1; b \leftarrow 1;$
    while true do
        $j' \leftarrow j + \lfloor m/8k \rfloor;$
        if Period($P[j \mathinner{.\,.} j')$) $> m/128k$ then
            $B_b \leftarrow P[j \mathinner{.\,.} j');$
            if $b = 2k$ then return breaks $B_1, \dots, B_{2k};$
            $b \leftarrow b + 1; j \leftarrow j';$
        else
            $q \leftarrow$ Period($P[j \mathinner{.\,.} j')$);
            $Q_r \leftarrow P[j \mathinner{.\,.} j + q); \delta \leftarrow 0;$
            generator G $\leftarrow$ MismGen($P[j \mathinner{.\,.} m), Q_r{}^*$);
            while $\delta < 8k/m \cdot (j' - j)$ and ($\pi \leftarrow$ Next(G)) $\neq \perp$ do
                $j' \leftarrow j + \pi + 1; \delta \leftarrow \delta + 1;$
            if $\delta \geq 8k/m \cdot (j' - j)$ then
                $R_r \leftarrow P[j \mathinner{.\,.} j');$
                if $\sum_{i=1}^{r} |R_i| \geq 3/8 \cdot m$ then
                    return repetitive regions $R_1, \dots, R_r$ with periods $Q_1, \dots, Q_r;$
                $r \leftarrow r + 1; j \leftarrow j';$
            else
                $Q \leftarrow Q_r; j'' \leftarrow j;$
                generator G' $\leftarrow$ MismGen$^R$($P[0 \mathinner{.\,.} j), {}^*Q$);
                while $\delta < 8k/m \cdot (m - j'')$ and ($\pi \leftarrow$ Next(G')) $\neq \perp$ do
                    $j'' \leftarrow \pi; \delta \leftarrow \delta + 1;$
                $Q \leftarrow P[j + (j'' - j) \bmod q \mathinner{.\,.} j + (j'' - j) \bmod q + q);$ // $Q \leftarrow \mathrm{rot}^{j-j''}(Q)$
                if $\delta \geq 8k/m \cdot (m - j'')$ then
                    return repetitive region $P[j'' \mathinner{.\,.} m)$ with period $Q$
                else return approximate period $Q;$
```

length, the algorithm stops. Processing the string P in this manner guarantees disjointness of breaks and repetitive regions. As in the proof of Lemma 3.6, a slightly different approach is needed if the algorithm encounters the end of P while growing a repetitive region. If this happens, the region is also extended to the left. This way, the algorithm either obtains a single repetitive region (which is not necessarily disjoint with the previously created ones, so it is returned on its own) or learns that the whole string P is close to being periodic.

Next, we fill in missing details of the implementation of the previous steps in the `PILLAR` model. To that end, first note that the `PILLAR` model includes a `Period` operation of checking if the period of a string S satisfies $\operatorname{per}(S) \leq |S|/2$ and computing $\operatorname{per}(S)$ in case of a positive answer. Since our threshold $m/128k$ satisfies $\lfloor m/128k \rfloor \leq \lfloor m/8k \rfloor /2$, no specific work is required to obtain the period of an unprocessed fragment of $\lfloor m/8k \rfloor$ characters of P.

To compute a repetitive region starting from a fragment $P' = P[\,j\,..\,j + \lfloor m/8k \rfloor\,)$ with string period $Q = P'[\,0\,..\,\operatorname{per}(P')\,)$, we use a $\texttt{MismGen}(P[\,j\,..\,m\,), Q^*)$ generator from Lemma 4.2: We extend P' up to the next mismatch between P' and Q^∞ until we either reach the end of P or the number $\delta = \delta_H(P', Q^*)$ reaches the bound $8k/m \cdot |P'|$. If we reach the end of P, we similarly extend $P' = P[\,j\,..\,m\,)$ to the left using a $\texttt{MismGen}^R(P[\,0\,..\,j\,), {}^*Q)$ generator from Lemma 4.2: Again, we always extend P' up to the next mismatch until we reach the start of P or the number $\delta = \delta_H(P', \overline{Q}^*)$ reaches the bound $8k/m \cdot |P'|$ (where $\overline{Q} = \operatorname{rot}^{|P'|-m+j}(Q)$ is the corresponding cyclic rotation of Q). If we reach the start of the string, we return a suitable cyclic rotation of Q; otherwise we found a long repetitive region, which we then return. Consider Algorithm 4.2 for a detailed pseudo-code of the implementation.

For the correctness, since our algorithm follows the proof of Lemma 3.6, we need to show only that our implementation of finding repetitive regions correctly implements the corresponding step in Algorithm 3.3. However, this is easy, as with each extension of P', the number δ may increase by at most 1. As we start with $\delta = \delta_H(P[\,j\,..\,j + \lfloor m/8k \rfloor\,), Q^*) = 0$, we thus never skip over a repetitive region. Further, the fragment $P' = P[\,j\,..\,j + \lfloor m/8k \rfloor\,)$ by construction contains at least two repetitions of the period Q, so we can obtain each cyclic rotation of Q as a fragment of P. In particular we indeed compute a cyclic rotation of Q in Line 25 of Algorithm 4.2. Consequently, Algorithm 4.2 indeed correctly implements Algorithm 3.3.

For the running time analysis, observe that each iteration of the outer while loop processes at least $\lfloor m/8k \rfloor$ characters of P, so there are at most $O(k)$ iterations of the outer while loop. In each iteration, we perform one `Period` operation, a constant number of `Access` operations, and at most $8k/m \cdot (j' - j)$ calls to the generator `MismGen`. Each of these calls uses $O(1)$ `PILLAR` operations, which is $O(8k/m \cdot m) = O(k)$ in total across all iterations. Similarly, we bound the running time of the calls to the generator $\texttt{MismGen}^R$: As we find at most $8k/m \cdot m = 8k$ mismatches, $\texttt{MismGen}^R$ uses at most $O(k)$ operations. Thus, Algorithm 4.2 uses $O(k)$ `PILLAR` operations.

The remaining running time is bounded by $O(k)$ in the same way, completing the proof. ◿

4.3 Computing Occurrences in the Periodic Case

We continue with detailed implementations for the two main cases of Theorem 3.1; we start with the (almost) periodic case.

◸ Lemma 4.6 ($\texttt{FindRotation}(k,\ Q,\ S)$). *Let k denote a positive integer, let Q denote a primitive string, and let S denote a string with $|S| \geq (2k+1)|Q|$. Then, we can compute a unique integer $j \in [\,0\,..\,|Q|\,)$ such that $\delta_H(S, \mathrm{rot}^j(Q)^*) \leq k$, or report $\perp$ if no such integer exists, using $O(k)$ time plus $O(k)$ `PILLAR` operations.*

▭ Proof. For every $0 \leq i \leq 2k$, define $S_i := S[\,i|Q|\,..\,(i+1)\,|Q|\,)$. We compute the majority of $S_0, \dots, S_{2k}$ (using Fact 2.5 for checking equality of fragments). If no majority exists, then we return $\perp$. Otherwise, we set $\overline{Q}$ to be the majority string of $S_0, \dots, S_{2k}$ and check if $\delta_H(S, \overline{Q}^*) \leq k$ using a `MismGen` from Lemma 4.2. If this test succeeds, we use a `Rotations` operation to retrieve all $j \in [\,0\,..\,|Q|\,)$

with $\overline{Q} = \mathrm{rot}^j(Q)$ and return any such j. If the test fails or if no such j is found, then we return $\bot$.

For the correctness, observe that if we have $\delta_H(S, \overline{Q}^*) \le k$, then at least $k+1$ fragments S_i match $\overline{Q}$ exactly, so $\overline{Q}$ must be the majority of $S_0, \dots, S_{2k}$. Moreover, since Q is primitive, there is at most one $j \in [\,0\,..\,|Q|\,)$ with $\overline{Q} = \mathrm{rot}^j(Q)$.

For the running time, note that we can compute the majority of $O(k)$ elements with a classic linear-time algorithm by Boyer and Moore [17] using $O(k)$ equality tests; as (by Fact 2.5) each equality test takes $O(1)$ time in the `PILLAR` model, we obtain the claimed running time and hence the claim. ⌟

LEMMA 4.7 (`FindRelevantFragment(`P`,` T`,` d`,` Q`)`). *Let P denote a pattern of length m and let T denote a text of length $n \le {}^3\!/_2\, m$. Further, let d denote a positive integer and let Q denote a primitive string that satisfies $|Q| \le m/8d$ and $\delta_H(P, Q^*) \le d$.*

Then, using $O(d)$ time plus $O(d)$ `PILLAR` operations, we can report a fragment $T' = T[\,\ell\,..\,r\,)$ such that $\delta_H(T', Q^*) \le 3d$ and, for every $k \le d/2$, the set $\mathrm{Occ}^H_k(P, T') = \{p - \ell \mid p \in \mathrm{Occ}^H_k(P, T)\}$ contains only multiples of $|Q|$.

PROOF. First, we call `FindRotation` from Lemma 4.6 to find the unique integer j such that $\delta_H(T[\,n-m\,..\,m\,), \mathrm{rot}^j(Q)^*) \le {}^3\!/_2\, d$. If no such j exists, then we return the empty string ε. Otherwise, we proceed by computing the rightmost position r such that $\delta_H(T[\,n-m+j\,..\,r\,), Q^*) \le {}^3\!/_2\, d$ and the leftmost position ℓ (with $\ell \equiv (n-m+j) \pmod{|Q|}$) such that $\delta_H(T[\,\ell\,..\,n-m+j\,), Q^*) \le {}^3\!/_2\, d$; afterwards, we return the fragment $T[\,\ell\,..\,r\,)$. Consider Algorithm 4.3 for implementation details.

For the correctness, triangle inequality (Lemma 1.1) yields

$$\delta_H(T[\,p\,..\,p+m\,), Q^*) \le k + \delta_H(P, Q^*) \le {}^3\!/_2\, d.$$

Since $p \le n - m$ and $p + m \ge m$, this yields

$$\delta_H(T[\,n-m\,..\,m\,), \mathrm{rot}^{p-n+m}(Q)) \le {}^3\!/_2\, d.$$

Algorithm 4.3 A `PILLAR` algorithm computing a *relevant* fragment T: a fragment T' such that all k-mismatch occurrences (for any $k \le d/2$) of P in T start at a position in T' which is a multiple of $|Q|$.

```
FindRelevantFragment($P, T, d, Q$)
    $j \leftarrow$ FindRotation($\lfloor {}^3\!/_2 d \rfloor, Q, T[n-m\,..\,m]$);
    if $j = \bot$ then return $\varepsilon$;
    $\delta \leftarrow 0$; $r \leftarrow n - m + j$;
    generator G $\leftarrow$ MismGen($T[n-m+j\,..\,n), Q^*$);
    while $\delta \le {}^3\!/_2 d$ and $(\pi \leftarrow$ Next(G)$) \ne \bot$ do
        $r \leftarrow n - m + j + \pi$;
        $\delta \leftarrow \delta + 1$;
    if $\delta \le {}^3\!/_2 d$ then $r \leftarrow m$;
    $\delta' \leftarrow 0$; $\ell \leftarrow n - m + j$; $\ell' \leftarrow (n - m + j) \bmod |Q|$;
    generator G$'$ $\leftarrow$ MismGen$^R$($T[\ell'\,..\,n-m+j), {}^*Q$);
    while $\delta' \le {}^3\!/_2 d$ and $(\pi \leftarrow$ Next(G$'$)$) \ne \bot$ do
        $\ell \leftarrow \ell' + |Q| \cdot \lceil (\pi+1)/|Q| \rceil$;
        $\delta' \leftarrow \delta' + 1$;
    if $\delta' \le {}^3\!/_2 d$ then $\ell \leftarrow \ell'$;
    return $T[\ell\,..\,r)$;
```

Further, we have

$$|T[n-m\,..\,m)| = 2m - n \ge m/2 \ge 4d|Q| \ge (2 \cdot \lfloor {}^3\!/_2 d \rfloor + 1)|Q|,$$

so the call to `FindRotation` is valid.

Hence, if the call to `FindRotation` returns $\bot$, then $\mathrm{Occ}_k^H(P,T) = \{\}$ (for each $k \le d/2$). Otherwise, each position $p \in \mathrm{Occ}_k^H(P,T)$ satisfies $p \equiv n - m + j \equiv \ell \pmod{|Q|}$. Further, we have

$$\begin{aligned} \delta_H(T[n-m+j\,..\,p+m), Q^*) &\le \delta_H(T[p\,..\,p+m), Q^*) \le {}^3\!/_2 d, \\ \delta_H(T[p\,..\,n-m+j), Q^*) &\le \delta_H(T[p\,..\,p+m), Q^*) \le {}^3\!/_2 d. \end{aligned}$$

Hence, the fragment $T' = T[\ell\,..\,r)$ contains all k-mismatch occurrences of P in T (for any $k \leq d/2$), and all these occurrences start at multiples of $|Q|$ in T'. Further, we have,

$$\delta_H(T', Q^*) = \delta_H(T[\ell\,..\,n-m+j), Q^*) + \delta_H(T[n-m+j\,..\,r), Q^*) \leq 3d.$$

For the running time (and the number of `PILLAR` operations used), the call to `FindRotation` uses $O(d)$ time plus $O(d)$ `PILLAR` operations; the same is true for the usage of `MismGen` and `MismGen`^{R}. Thus, the algorithm uses $O(d)$ time plus $O(d)$ `PILLAR` operations in total, completing the proof. ⌟

⌜ LEMMA 4.8 (`DistancesRLE(`P`, `T`, `Q`)`: IMPL. OF LEMMA 3.3). *Let P denote a pattern of length m and let T denote a text of length $n \leq {}^3\!/_2\, m$. Further, let d denote a positive integer and let Q denote a string that satisfies $\delta_H(P, Q^*) = O(d)$ and $\delta_H(T, Q^*) = O(d)$.*

Then, using $O(d^2 \log\log d)$ time plus $O(d)$ `PILLAR` operations, we can compute a run-length encoded sequence of $h_j := \delta_H(T[j|Q|\,..\,j|Q| + m), P)$ for $0 \leq j \leq (n-m)/|Q|$.

PROOF. Observe that Claim 3.4 already gives rise to an algorithm: Starting with h_0, we can obtain the value h_{j+1} from h_j by adding the value

$$\begin{aligned} h_{j+1} - h_j = {} & |\mathrm{Mis}(T, Q^*) \cap [\,jq + m\,..\,(j+1)q + m\,)| \\ & - |\mathrm{Mis}(T, Q^*) \cap [\,jq\,..\,(j+1)q\,)| - \mu_{j+1} + \mu_j, \end{aligned}$$

where μ_{j+1} and μ_j are defined as in Lemma 3.3.

We implement this idea in two steps: In a first step, we compute the values μ_j (using marking) and the positions of mismatches in $\mathrm{Mis}(T, Q^*)$ (using `Mismatches` from Corollary 4.3). In a second step, we use a sliding-window approach (with the positions computed in the first step interpreted as events) to output the sequence of values of h_j. Consider the pseudo-code (Algorithm 4.4) for implementation details.

Algorithm 4.4 A PILLAR algorithm for Lemma 3.3.

```
DistancesRLE($P, T, Q$)
    // Marking phase
    $M \leftarrow \{\}$;
    foreach $\tau \in$ Mismatches($T, Q^*$) do
        $M \leftarrow M \cup \{(\tau - m, 1), (\tau, -1)\}$;
        foreach $\pi \in$ Mismatches($P, Q^*$) do
            $M \leftarrow M \cup \{(\tau - \pi - 1, \delta_H(P[\pi], T[\tau]) - 2), (\tau - \pi, 2 - \delta_H(P[\pi], T[\tau]))\}$;
    // Sliding-window phase
    sort $M$;
    $h \leftarrow |$Mismatches($P, Q^*$)$|$;
    foreach $(i', w) \in M$ with $i' < 0$ do $h \leftarrow h + w$;
    $i \leftarrow 0$;
    foreach $(i', w) \in M$ with $0 \le i' < n - m$ sorted by $i'$ do
        Output a block of $\lceil (i' + 1)/q \rceil - \lceil i/q \rceil$ values $h$;
        $i \leftarrow i' + 1$;
        $h \leftarrow h + w$;
    Output a block of $\lceil (n - m + 1)/q \rceil - \lceil i/q \rceil$ values $h$;
```

For the correctness, in the marking phase, the algorithm constructs a multiset M of pairs (i, w) (where i can be interpreted as a position in T and w as the weight) so that $h_j - \delta_H(P, Q^*) = w(M, j|Q|)$, where

$$w(M, i) := \sum_{\{(i', w) \in M | i' < i\}} w.$$

Specifically, for each $\tau \in \mathrm{Mis}(T, Q^*)$, the algorithm first inserts to M pairs $(\tau - m, 1)$ and $(\tau, -1)$. As a result, for each position i with $0 \le i \le n - m$, we have $w(M, i) = |\mathrm{Mis}(T, Q^*) \cap [i \mathinner{.\,.} i + m)|$. In particular, if $i = j|Q|$, then $w(M, i) = \delta_H(T[j|Q| \mathinner{.\,.} j|Q| + m), Q^*)$. Next, for each $\tau \in \mathrm{Mis}(T, Q^*)$ and each $\pi \in \mathrm{Mis}(P, Q^*)$, the algorithm inserts to M pairs $(\tau - \pi - 1, \delta_H(P[\pi], T[\tau]) - 2)$ and $(\tau - \pi, 2 - \delta_H(P[\pi], T[\tau]))$. As a result, the values $w(M, i)$ with $i \neq \tau - \pi$ are not altered, whereas $w(M, i)$ for $i = \tau - \pi$ is decreased by the number of marks placed in the proof of Lemma 3.3 at position $i = \tau - \pi$ of T due to positions τ in T and π in P. Consequently, we have $w(M, j|Q|) = \delta_H(T[\, j|Q| \,..\, j|Q| + m\,), Q^*) - \mu_j$, which yields $h_j = \delta_H(P, Q^*) + w(M, j|Q|)$ due to Claim 3.4.

Hence, in order to construct the sequence h_j, the algorithm sorts the pairs in M and determines the partial sums $w(M, i)$. In each block of $[\, i \,..\, i'\,]$ of equal partial sums, the algorithm reports a block with all $\lceil (i' + 1)/q \rceil - \lceil i/q \rceil$ entries h_j for $j|Q| \in [\, i \,..\, i'\,)$, which is indeed correct.

The running time is $O(d^2 \log\log d)$ (dominated by sorting the set M, which contains $O(d^2)$ integer pairs) plus $O(d)$ `PILLAR` operations (for the calls to `Mismatches(`P, Q^*`)` and `Mismatches(`T, Q^*`)` and for accessing the mismatching positions of P and T). ⌟

⌜ LEMMA 4.9 (`PerMat(`P, T, k, d, Q`)`: IMPL. OF COROLLARY 3.5). *Let P denote a pattern of length m and let T denote a text of length n. Further, let $k \leq m$ denote a non-negative integer, let $d \geq 2k$ denote a positive integer, and let Q denote a primitive string Q that satisfies $|Q| \leq n/8d$ and $\delta_H(P, Q^*) \leq d$.*

There is an algorithm that computes the set $\mathrm{Occ}^H_k(P, T)$, represented as $O(n/m \cdot d^2)$ arithmetic progressions with difference $|Q|$ (or as $O(n/m \cdot d)$ individual positions if $\delta_H(P, Q^*) = d$). The algorithm uses $O(n/m \cdot d^2 \log\log d)$ time plus $O(n/m \cdot d)$ `PILLAR` operations.

PROOF. First, we split the string T into $\lfloor 2n/m \rfloor$ blocks

$$T_i := T[\, \lfloor i \cdot m/2 \rfloor \,..\, \min\{n, \lfloor (i+3) \cdot m/2 \rfloor - 1\}\,); \quad 0 \leq i < \lfloor 2n/m \rfloor.$$

For each block T_i, we call `FindRelevantFragment(`P, T_i, d, Q`)` from Lemma 4.7 to obtain a fragment $T'_i = T[\, \ell_i \,..\, r_i\,)$ containing all k-mismatch occurrences of P in T_i. Next, we call `DistancesRLE(`P, T'_i, Q`)` from Lemma 4.8, yielding a run-length encoded sequence of values $h_t := \delta_H(T'_i[\, t|Q| \,..\, t|Q| + m\,), P)$ for $0 \leq t \leq (|T'_i| - m)/|Q|$. For each run $h_t = \cdots = h_{t'} \leq k$, we add the arithmetic progression $\{\ell_i + j \cdot |Q|\}_{[t\,..\,t']}$ to $\mathrm{Occ}_k^H(P,T)$. In the end, we return the set $\mathrm{Occ}_k^H(P,T)$.

For the correctness, observe that we essentially follow the proof of Corollary 3.5. In particular, each k-mismatch occurrence of P in T is contained in exactly one of the fragments T_i. By Lemma 4.7, we see that T_i' contains all the k-mismatch occurrences of P in T_i. Moreover, as $\mathrm{Occ}_k^H(P,T_i')$ contains only multiples of $|Q|$, each $p \in \mathrm{Occ}_k^H(P,T_i')$ corresponds to an entry $h_j \le k$. Hence, all k-mismatch occurrences of P in T are found.

Next, observe that we have

$$h_j = \delta_H(T[\,\ell_i + j|Q|\,..\,\ell_i + j|Q| + m\,), P) \le k$$

whenever $\ell_i + j|Q|$ is reported. Hence, there are no false positives.

If $\delta_H(P,Q^*) = d$, then for each i, the number of entries h_j with $h_j \le k$ is $O(d)$ by Lemma 3.3, so the corresponding positions $\ell_i + j|Q|$ can be added to $\mathrm{Occ}_k^H(P,T)$ individually.

We obtain the bounds on the overall running time from Lemmas 4.7 and 4.8 as we have $\delta_H(P,Q^*) \le d$ and $\delta_H(T_i,Q^*) \le 3d$ for each i by Lemma 4.8. ◼

4.4 Computing Occurrences in the Non-Periodic Case

We discuss algorithms for the non-periodic case next.

Lemma 4.10 (`BreakMatches`$(P, T, \{B_1,\dots,B_{2k}\}, k)$: Impl. of Lemma 3.8). *Let P denote a string of length m having $2k$ disjoint breaks $B_1,\dots,B_{2k} \preccurlyeq P$ each satisfying $\mathrm{per}(B_i) \ge m/128k$. Further, let T denote a string of length $n \le {}^3\!/_2\, m$.*

Then, we can compute the set $\mathrm{Occ}_k^H(P,T)$ using $O(k^2 \log\log k)$ time plus $O(k^2)$ `PILLAR` operations.

Proof. The implementation of (the marking in the proof of) Lemma 3.8 is straightforward: For each break $B_i = P[\,b_i\,..\,b_i + |B_i|\,)$, we use a call to `ExactMatches`$(B_i, T)$ from Lemma 2.6 to find all exact occurrences $\mathrm{Occ}(B_i,T)$. For each occurrence $\pi \in \mathrm{Occ}(B_i,T)$, we mark position $\pi - b_i$ in T. Having placed all marks, we run

ALGORITHM 4.5 A PILLAR model algorithm for Lemma 3.8.

```
BreakMatches($P, T, \{B_1 = P[b_1 \mathinner{.\,.} b_1 + |B_1|), \dots, B_{2k} = P[b_{2k} \mathinner{.\,.} b_{2k} + |B_{2k}|)\}, k$)
    multi-set $M \leftarrow \{\}$; $\mathrm{Occ}^H_k(P,T) \leftarrow \{\}$;
    for $i \leftarrow 1$ to $2k$ do
        foreach $\tau \in$ ExactMatches($B_i, T$) do
            $M \leftarrow M \cup \{\tau - b_i\}$;                    // Mark position $\tau - b_i$ in $T$
    sort $M$;
    foreach $\pi \in [0 \mathinner{.\,.} n-m]$ that appears at least $k$ times in $M$ do
        if Verify($P, T[\pi \mathinner{.\,.} \pi + m), k$) then $\mathrm{Occ}^H_k(P,T) \leftarrow \mathrm{Occ}^H_k(P,T) \cup \{\pi\}$;
    return $\mathrm{Occ}^H_k(P,T)$;
```

Verify from Lemma 4.4 for every position $\pi \in [0 \mathinner{.\,.} n-m]$ in T that has at least k marks. In the end, we return all positions where Verify confirmed a k-mismatch occurrence. See Algorithm 4.5 for a pseudo-code.

For the correctness, note that we placed marks as in the proof of Lemma 3.8; in particular, by Claim 3.10, any $\pi \in \mathrm{Occ}^H_k(P,T)$ has at least k marks. As we verify every possible candidate using Verify, we report no false positives, and thus the algorithm is correct.

We continue with analyzing the number of PILLAR operations used. As every break B_i has period $\mathrm{per}(B_i) > m/128k$, every call to ExactMatches uses $O(k)$ basic PILLAR operations; thus, all calls to ExactMatches use $O(k^2)$ basic operations in total. As there are at most $O(k^2/k) = O(k)$ positions that we verify, and every call to Verify uses $O(k)$ PILLAR operations, the verifications use $O(k^2)$ PILLAR operations in total.

Finally, for the running time, by Claim 3.9, we place at most $O(k^2)$ marks in T, so the marking step uses $O(k^2)$ operations in total. Further, finding all positions in T with at least k marks can be done via a linear scan over the multiset M of all marks after sorting M, which can be done in time $O(k^2 \log\log k)$. Overall, Algorithm 4.5 runs in time $O(k^2 \log\log k)$ plus $O(k^2)$ PILLAR operations. ◿

Algorithm 4.6 A PILLAR model algorithm for Lemma 3.11.

```
RepMat($P, T, \{(R_1 = P[r_1 \mathinner{.\,.} r_1 + |R_1|), Q_1) \dots, (R_r = P[r_r \mathinner{.\,.} r_r + |R_r|), Q_r)\}, k$)
    multi-set $M \leftarrow \{\}$; $\mathrm{Occ}^H_k(P,T) \leftarrow \{\}$;
    for $i \leftarrow 1$ to $r$ do
        foreach $\tau \in$ PerMat($R_i, T, \lfloor 4 \cdot k/m \cdot |R_i| \rfloor, \lceil 8 \cdot k/m \cdot |R_i| \rceil, Q_i$) do
            $M \leftarrow M \cup \{(\tau - r_i, |R_i|)\}$; // Place $|R_i|$ marks at position $\tau - r_i$ in
              $T$
    sort $M$ by positions;
    foreach $\pi \in [0 \mathinner{.\,.} n-m]$ appearing at least $\sum_{(\pi,v)\in M} v \geq \sum_{i=1}^{r}|R_i| - m/4$ times in
      $M$ do
        if Verify($P, T[\pi \mathinner{.\,.} \pi + m), k$) then $\mathrm{Occ}^H_k(P,T) \leftarrow \mathrm{Occ}^H_k(P,T) \cup \{\pi\}$;
    return $\mathrm{Occ}^H_k(P,T)$;
```

Lemma 4.11 (RepMat(P, T, $\{(R_1,Q_1),\dots,(R_r,Q_r)\}$, k): Impl. of Lemma 3.11). *Let P denote a string of length m, let T denote a string of length $n \leq {}^3/_2\, m$, and let $k \leq m$ denote a positive integer. Suppose that P contains disjoint repetitive regions $R_1,\dots,R_r$ of total length at least $\sum_{i=1}^{r}|R_i| \geq 3/8 \cdot m$ such that each region R_i satisfies $|R_i| \geq m/8k$ and has a primitive approximate period Q_i with $|Q_i| \leq m/128k$ and $\delta_H(R_i, Q_i^*) = \lceil 8k/m \cdot |R_i| \rceil$.*

Then, we can compute the set $\mathrm{Occ}^H_k(P,T)$ using $O(k^2 \log\log k)$ time plus $O(k^2)$ PILLAR operations.

Proof. As in the proof of Lemma 3.11, set

$$m_R := \sum_{i\in[1\mathinner{.\,.}r]} |R_i| \geq 3/8 \cdot m$$

and define for every $1 \leq i \leq r$ the values

$$k_i := \lfloor 4 \cdot k/m \cdot |R_i| \rfloor;$$
$$d_i := \lceil 8 \cdot k/m \cdot |R_i| \rceil = |\mathrm{Mis}(R_i, Q_i^*)|.$$

Further, write $R_i = P[\,r_i \,..\, r_i + |R_i|\,)$.

We directly implement the marking of the proof of Lemma 3.11: For every repetitive region R_i, we call `PerMat`(R_i, T, k_i, d_i, Q_i) from Lemma 4.9 to obtain the set $\mathrm{Occ}^H_{k_i}(R_i, T)$. Next, for each position $\tau \in \mathrm{Occ}^H_{k_i}(R_i, T)$, we place $|R_i|$ marks at position $\tau - r_i$. Note that for performance reasons, instead of placing $|R_i|$ unweighted marks, we place a single mark of weight $|R_i|$ at position $\tau - r_i$.

Having placed all marks, we run `Verify` from Lemma 4.4 for every position $\pi \in [\,0 \,..\, n - m\,]$ in T that has marks of total weight at least $m_r - m/4$. We return all positions where `Verify` confirmed a k-mismatch occurrence. See Algorithm 4.6 for a pseudo-code.

For the correctness, first note that in every call to `PerMat` from Lemma 4.9, we have

$$16k/m \cdot |R_i| \geq d_i = \lceil 8k/m \cdot |R_i| \rceil = \delta_H(R_i, Q_i^*) \geq 2k_i,$$

so $|Q_i| \leq m/128k \leq |R_i|/8d_i$; hence, we can indeed call `PerMat` in this case. Further, note that we placed the marks as in the proof of Lemma 3.11; in particular, by Claim 3.13, any $\pi \in \mathrm{Occ}^H_k(P, T)$ has at least $m_R - m/4$ marks. As we verify every possible candidate using `Verify`, we report no false positives, and thus the algorithm is correct.

For the number of `PILLAR` operations, observe that during the marking step, for every repetitive region R_i, we call `PerMat` once, and the call uses $O(n/|R_i| \cdot d_i) = O(m/|R_i| \cdot k/m \cdot |R_i|) = O(k)$ `PILLAR` operations. Hence, the marking step uses $O(r \cdot k) = O(k^2)$ `PILLAR` operations in total. Next, during the verification step, by Claims 3.12 and 3.13, we call `Verify` at most $O(k)$ times. As each call to `Verify` uses $O(k)$ `PILLAR` operations, the verification step in total uses $O(k^2)$ `PILLAR` operations. Overall, Algorithm 4.6 uses $O(k^2)$ `PILLAR` operations.

Algorithm 4.7 A `PILLAR` model algorithm for Theorem 3.1.

```
MismatchOccurrences($P, T, k$)
    ( $B_1, \dots, B_{2k}$ or $(R_1, Q_1), \dots, (R_r, Q_r)$ or $Q$ ) $\leftarrow$ Analyze($P, k$);
    $\mathrm{Occ}_k^H(P,T) \leftarrow \{\}$;
    for $i \leftarrow 0$ to $\lfloor 2n/m \rfloor$ do
        $T_i \leftarrow T[\, \lfloor i \cdot m/2 \rfloor \,..\, min\{n, \lfloor (i+3) \cdot m/2 \rfloor - 1\}\,)$;
        if breaks $B_1, \dots, B_{2k}$ exist then
            $\mathrm{Occ}_k^H(P,T_i) \leftarrow$ BreakMatches($P, T_i, \{B_1, \dots, B_{2k}\}, k$);
        else if repetitive regions $(R_1, Q_1), \dots, (R_r, Q_r)$ exist then
            $\mathrm{Occ}_k^H(P,T_i) \leftarrow$ RepMat($P, T_i, \{(R_1, Q_1), \dots, (R_r, Q_r)\}, k$);
        else $\mathrm{Occ}_k^H(P,T_i) \leftarrow$ PerMat($P, T_i, k, 8k, Q$);
        $\mathrm{Occ}_k^H(P,T) \leftarrow \mathrm{Occ}_k^H(P,T) \cup \{\ell + im/2 : \ell \in \mathrm{Occ}_k^H(P,T_i)\}$;
    return $\mathrm{Occ}_k^H(P,T)$;
```

Finally, for the running time, with similar calculations as for the number of `PILLAR` operations, we see that the marking step, including calls to `PerMat`, takes time

$$\sum_i O(n/|R_i| \cdot d_i^2 \log\log d_i) = \sum_i O(|R_i|/m \cdot k^2 \log\log k) = O(k^2 \log\log k).$$

Further, for every R_i, we place at most $|\mathrm{Occ}_{k_i}^H(R_i,T)|$ (weighted) marks. Using Corollary 3.5, we have

$$|\mathrm{Occ}_{k_i}^H(R_i,T)| = O(n/|R_i| \cdot d_i) = O(k).$$

Thus, we place $|M| = O(k^2)$ (weighted) marks in total. Therefore, we can sort M (by positions) in time $O(k^2 \log\log k)$; afterwards, we can find the elements with total weight at least $m_R - m/4$ via a linear scan over M in time $O(k^2)$. Hence, Algorithm 4.6 runs in $O(k^2 \log\log k)$ overall time, completing the proof. ◿

4.5 A PILLAR Model Algorithm for Pattern Matching with Mismatches

Piecing together the algorithms for the various cases, we obtain a PILLAR model algorithm for Pattern Matching with Mismatches.

Theorem 4.1. *Given a pattern P of length m, a text T of length n, and a positive integer $k \leq m$, we can compute (a representation of) the set* $\mathrm{Occ}^H_k(P,T)$ *using* $O(n/m \cdot k^2 \log\log k)$ *time plus* $O(n/m \cdot k^2)$ *PILLAR operations.*

Proof. First, we split T into overlapping parts $T_1,\dots,T_{\lfloor 2n/m \rfloor}$ of length less than $^3/_2\, m$ each. In order to compute $\mathrm{Occ}^H_k(P,T_i)$ for each i, we follow the structure of the proof of Theorem 3.1: We first call `Analyze(`P,k`)` from Lemma 4.5. If the call to `Analyze(`P,k`)` yields $2k$ disjoint breaks $B_1,\dots,B_{2k}$ in P, we call `BreakMatches(`P, T_i,$\{B_1,\dots,B_{2k}\}$, k`)` from Lemma 4.10. If the call to `Analyze(`P,k`)` yields disjoint repetitive regions $R_1,\dots,R_r$ (and corresponding approximate periods $Q_1,\dots,Q_r$), then we call use the implementation of Lemma 4.11 `RepMat(`P, T_i, $\{(R_1,Q_1),\dots,(R_r,Q_r)\}$, k`)` from Lemma 4.11. Finally, if the call to `Analyze(`P,k`)` yields an approximate period Q, then we call `PerMat(`P, T_i, k, $8k$, Q`)` from Lemma 4.9. We obtain $\mathrm{Occ}^H_k(P,T)$ by combining the sets $\mathrm{Occ}^H_k(P,T_i)$. Consider Algorithm 4.7 for a visualization as pseudo-code.

For the correctness, first, we do not lose any occurrences by splitting T, since every length-m fragment of T is contained in one of the fragments T_i. Second, by Lemma 4.5 and due to $|T_i| \leq {}^3/_2\, m$, the parameters in the calls to `BreakMatches` and `RepMat` each satisfy the requirements. Lastly, if we use `PerMat`, notice that again by Lemma 4.5 the string Q satisfies $\delta_H(P,Q^*) \leq 8k$ and $|Q| \leq m/128k \leq m/(8 \cdot 8k)$; hence, we can indeed call `PerMat` in this case.

For the number of PILLAR operations used, the call to `Analyze` uses $O(k)$ operations, each call to `BreakMatches` and `RepMat` uses $O(k^2)$ operations, and each call to `PerMat` uses $O(k)$ operations. As there are at most $O(n/m)$ calls to `BreakMatches`, `RepMat`, and `PerMat`, we use $O(n/m \cdot k^2)$ PILLAR operations in total.

For the running time, the call to `Analyze` takes $O(k)$ time, where each call to `BreakMatches`, `RepMat`, and `PerMat` takes $O(k^2 \log\log k)$ time. As there are at most $O(n/m)$ calls to `BreakMatches`, `RepMat`, and `PerMat` each, and we can combine the sets $\mathrm{Occ}_k^H(P, T_i)$ into the set $\mathrm{Occ}_k^H(P, T)$ in total time $O(n/m \cdot k^2)$, we can bound the total running time by $O(n/m \cdot k^2 \log\log k)$. ◪

Combining Lemma 2.9 and Theorem 4.1, we obtain an algorithm for pattern matching with mismatches with the same running time as the algorithm of Clifford et al. [32], that is essentially optimal when $k = O(\sqrt{m})$.[17]

17. Strictly speaking, the algorithm in [32] runs in time $O(n \operatorname{polylog}(m) + n/m \cdot k^2 \log k)$, so our algorithm is slightly faster. However, an even better improvement in the logarithmic factors was already obtained recently in [23].

◸ Corollary 4.12. *Given a text T of length n, a pattern P of length m and a threshold k, we can compute the set $\mathrm{Occ}_k^H(P, T)$ in time $O(n + n/m \cdot k^2 \log\log k)$.* ◿

Next, we discuss an efficient algorithm for the fully-compressed setting. We are given an SLP $\mathcal{G}_T$ of size n with $T = gen(\mathcal{G}_T)$, an SLP $\mathcal{G}_P$ of size m with $P = gen(\mathcal{G}_T)$ and a threshold k and are required to compute the k-mismatch or k-error occurrences of P in T.

Set $N := |T| + |P|$ and $\mathcal{X} := \{\mathcal{G}_T, \mathcal{G}_P\}$. The overall structure of our algorithm is as follows: We first preprocess the collection $\mathcal{X}$ in $O((n + m) \log N)$ time according to Theorem 2.13. Next, we traverse $\mathcal{G}_T$ and compute for every non-terminal A of $\mathcal{G}_T$ the approximate occurrences of P in T that "cross" A. Now, we combine Theorem 2.13 with Theorem 4.1 to compute the occurrences. Finally, we combine the computed occurrences using dynamic programming.

Formally, for each non-terminal $A \in N_{\mathcal{G}_T}$, with production rule $A \to BC$, we wish to compute all approximate occurrences of P in the string

$$gen(B)[\,|gen(B)| - |P| + 1 \,..\, |gen(B)|\,)gen(C)[\,0 \,..\, |P| - 1\,),$$

which is indeed a fragment of $gen(\mathcal{G}_T)$ and is of length $|2P| - 2$. We can compute these approximate occurrences in time $O(k^2 log^3 N)$ by combining Theorems 2.13 and 4.1.

Other approximate occurrences in $gen(A)$ lie entirely in $gen(B)$ or $gen(C)$; hence we compute them when considering B and C. (Compare [20, Theorem 4.1] for a similar algorithm.)

Now, $|\mathrm{Occ}_k^H(P,T)|$ can be computed by dynamic programming that is analogous to the dynamic programming used to compute the length of a string generated by an SLP. Further, all approximate occurrences can be reported in time proportional to their number by performing a traversal of $PT_{\mathcal{G}}$, avoiding to explore subtrees that correspond to fragments of T that do not contain any approximate occurrences.

We hence obtain the following algorithm for pattern matching with mismatches in the fully-compressed setting.

COROLLARY 4.13. *Let $\mathcal{G}_T$ denote an SLP of size n generating a string T, let $\mathcal{G}_P$ denote an SLP of size m generating a string P, let k denote a threshold, and set $N := |T| + |P|$.*

Then, we can compute $|\mathrm{Occ}_k^H(P,T)|$ in time $O(m \log N + n\, k^2 log^3 N)$.

The elements of $\mathrm{Occ}_k^H(P,T)$ can be reported with $|\mathrm{Occ}_k^H(P,T)|$ extra time.

Finally, combining Theorems 2.15 and 4.1, we obtain an algorithm for approximate pattern matching for dynamic strings.

COROLLARY 4.14. *We can maintain a collection $\mathcal{X}$ of non-empty persistent strings of total length N under the operations* `makestring`(U), `concat`(U,V), `split`(U,i) *requiring time $O(\log N + |U|)$, $O(\log N)$ and $O(\log N)$, respectively, so that given two strings $P,T \in \mathcal{X}$ with $|P| = m$ and $|T| = n$ and a threshold k, we can compute the set $\mathrm{Occ}_k^H(P,T)$ in time $O(n/m \cdot k^2 log^2 N)$.*[18]

18. All running time bounds hold w.h.p.

Structural Insights into Pattern Matching with Edits

5

In this chapter, we develop insights into the structure of k-error occurrences of a pattern P in a text T. We prove the following result, which is analogous to Theorem 3.1.

Theorem 5.1 (Compare Theorem 3.1). *Given a pattern P of length m, a text T of length n, and a positive integer $k \le m$, then at least one of the following holds.*

- *We have $|\lfloor \mathrm{Occ}^E_k(P,T)/k \rfloor| \le 642045 \cdot n/m \cdot k$.*
- *There is a primitive Q with $|Q| \le m/128k$ and $\delta_E(P, {}^*Q^*) < 2k$.*

5.1 Characterization of the Periodic Case

Similarly to 3, we start with the (approximately) periodic case. Unsurprisingly, the proofs are (much) more involved.[19]

19. For non-expert readers, it may be a wise decision to first read the other sections and return to this section *after* you read how we use the results from this section.

Lemma 5.2 (Compare Lemma 3.2). *Let P denote a pattern of length m, let T denote a text of length n, and let $k \le m$ denote a non-negative integer such that $n < {}^3\!/_2\, m + k$. Suppose that the k-error occurrences of P in T include a prefix of T and a suffix of T. If there are a positive integer $d \ge 2k$ and a primitive string Q with $|Q| \le m/8d$ and $\delta_E(P, Q^*) = \delta_E(P, {}^*Q^*) \le d$, then each of following holds:*

1. *For every $p \in \mathrm{Occ}^E_k(P,T)$, we have $p \bmod |Q| \le 3d$ or $p \bmod |Q| \ge |Q| - 3d$.*
2. *The string T satisfies $\delta_E(T, {}^*Q^*) \le 3d$.*
3. *If $\delta_E(P, {}^*Q^*) = d$, then $|\lfloor \mathrm{Occ}^E_k(P,T)/d \rfloor| \le 304d$.*
4. *The set $\mathrm{Occ}^E_k(P,T)$ can be decomposed into $617d^3$ arithmetic progressions with difference $|Q|$.*

Lemma 5.3. *Let k denote a positive integer, let Q denote a primitive string, and let S denote a string of length $|S| \geq (2k+1)|Q|$. If there are integers $\ell \leq r$ and $\ell' \leq r'$ such that $\delta_E(S, Q^\infty[\ell \mathinner{..} r)) \leq k$ and $\delta_E(S, Q^\infty[\ell' \mathinner{..} r')) \leq k$, then there are integers j, j' and a decomposition $S = S_L\, S_R$ that satisfy*

$$\begin{aligned}&\delta_E(S, Q^\infty[\ell \mathinner{..} r))\\&\quad = \delta_E(S_L, Q^\infty[\ell \mathinner{..} j|Q|)) + \delta_E(S_R, Q^\infty[j|Q| \mathinner{..} r)) \quad \textit{and}\\&\delta_E(S, Q^\infty[\ell' \mathinner{..} r'))\\&\quad = \delta_E(S_L, Q^\infty[\ell' \mathinner{..} j'|Q|)) + \delta_E(S_R, Q^\infty[j'|Q| \mathinner{..} r')).\end{aligned}$$

If $|Q| = 1$, then the assumption $|S| \geq (2k+1)|Q|$ is not required.

Proof. If $|Q| = 1$, we can set $S_L := S$, $S_R := \varepsilon$, $j := r$, and $j' := r'$.

Now assume that we have $|S| \geq (2k+1)|Q|$ and define $S_i := S[i|Q| \mathinner{..} (i+1)|Q|)$ for $0 \leq i \leq 2k$. Further, fix optimal alignments between S and $Q^\infty[\ell \mathinner{..} r)$ and between S and $Q^\infty[\ell' \mathinner{..} r')$.

Observe that at least one of the fragments S_i is aligned without errors in *both* alignments. Let us fix such a fragment S_i and observe that S_i is a length-$|Q|$ substring of Q^∞, so $S_i = \mathrm{rot}^p(Q)$ for some $p \in [0 \mathinner{..} |Q|)$. Based on this value, we set $S_L := S[0 \mathinner{..} i|Q| + p)$ and $S_R := S[i|Q| + p \mathinner{..} |S|)$.

Next, consider the fragment Q' of $Q^\infty[\ell \mathinner{..} r)$ that is aligned to S_i in the alignment fixed earlier. The fragment Q' matches $\mathrm{rot}^p(Q)$. As Q is primitive, Q' is thus of the form $Q' = Q^\infty[j|Q| - p \mathinner{..} (j+1)|Q| - p)$ for some integer j. Hence,

$$\delta_E(S, Q^\infty[\ell \mathinner{..} r)) = \delta_E(S_L, Q^\infty[\ell \mathinner{..} j|Q|)) + \delta_E(S_R, Q^\infty[j|Q| \mathinner{..} r)).$$

A similar argument shows that for some integer j', we also have

$$\begin{aligned}&\delta_E(S, Q^\infty[\ell' \mathinner{..} r'))\\&\quad = \delta_E(S_L, Q^\infty[\ell' \mathinner{..} j'|Q|)) + \delta_E(S_R, Q^\infty[j'|Q| \mathinner{..} r')).\end{aligned}$$

This completes the proof. ◻

Lemma 5.4. *Let T denote a text of length n, let k denote a positive integer, and let Q denote a primitive string. Suppose that for integers $p \le q$, $x \le y$, and $x' \le y'$ we have*

$$\begin{aligned} \delta_E(T[\,0\,..\,q\,), Q^\infty[\,x\,..\,y\,)) &\le k \quad \textit{and} \\ \delta_E(T[\,p\,..\,n\,), Q^\infty[\,x'\,..\,y'\,)) &\le k. \end{aligned}$$

If $|Q| = 1$ or $q - p \ge (2k+1)|Q|$, then for some $x'' \equiv x' \pmod{|Q|}$ and $y'' \equiv y' \pmod{|Q|}$, and $(p + x - x' + 2k) \bmod |Q| \le 4k$, we have

$$\delta_E(T, Q^\infty[\,x''\,..\,y\,)) = \delta_E(T, Q^\infty[\,x\,..\,y''\,)) \le 2k.$$

Proof. Observe that, for some integer $z \in [\,x\,..\,y\,]$, we have

$$\begin{aligned} &\delta_E(T[\,0\,..\,q\,), Q^\infty[\,x\,..\,y\,)) \\ &\quad = \delta_E(T[\,0\,..\,p\,), Q^\infty[\,x\,..\,z\,)) + \delta_E(T[\,p\,..\,q\,), Q^\infty[\,z\,..\,y\,)). \end{aligned}$$

Similarly, for some integer $z' \in [\,x'\,..\,y'\,]$, we have

$$\begin{aligned} &\delta_E(T[\,p\,..\,n\,), Q^\infty[\,x'\,..\,y'\,)) = \\ &\quad \delta_E(T[\,p\,..\,q\,), Q^\infty[\,x'\,..\,z'\,)) + \delta_E(T[\,q\,..\,n\,), Q^\infty[\,z'\,..\,y'\,)). \end{aligned}$$

Now, Lemma 5.3 applied to $S := T[\,p\,..\,q\,)$ yields an integer $r \in [\,p\,..\,q\,]$ and integers j, j' such that

$$\begin{aligned} &\delta_E(T[\,p\,..\,q\,), Q^\infty[\,z\,..\,y\,)) \\ &\quad = \delta_E(T[\,p\,..\,r\,), Q^\infty[\,z\,..\,j|Q|\,)) \\ &\qquad + \delta_E(T[\,r\,..\,q\,), Q^\infty[\,j|Q|\,..\,y\,)), \text{ and} \\ &\delta_E(T[\,p\,..\,q\,), Q^\infty[\,x'\,..\,z'\,)) \\ &\quad = \delta_E(T[\,p\,..\,r\,), Q^\infty[\,x'\,..\,j'|Q|\,)) \\ &\qquad + \delta_E(T[\,r\,..\,q\,), Q^\infty[\,j'|Q|\,..\,z'\,)). \end{aligned}$$

This implies that

$$\begin{aligned}&\delta_E(T[0\,..\,r), Q^\infty[x\,..\,j|Q|))\\&\quad= \delta_E(T[0\,..\,p), Q^\infty[x\,..\,z))\\&\qquad+ \delta_E(T[p\,..\,r), Q^\infty[z\,..\,j|Q|)) \le k, \text{ and}\\&\delta_E(T[r\,..\,n), Q^\infty[j'|Q|\,..\,y'))\\&\quad= \delta_E(T[r\,..\,q), Q^\infty[j'|Q|\,..\,z'))\\&\qquad+ \delta_E([q\,..\,n), Q^\infty[z'\,..\,y')) \le k.\end{aligned}$$

Combining the equations yields

$$\begin{aligned}&\delta_E(T, Q^\infty[x + (j' - j)|Q|\,..\,y'))\\&\ = \delta_E(T, Q^\infty[x\,..\,y' + (j - j')|Q|))\\&\ \le \delta_E(T[0\,..\,r), Q^\infty[x\,..\,j|Q|)) + \delta_E(T[r\,..\,n), Q^\infty[j'|Q|\,..\,y'))\\&\ \le 2k.\end{aligned}$$

We deduce $|j|Q| - x - r| \le k$ and $|j'|Q| - x' - r + p| \le k$, which yields $|p+x-x'-(j-j')|Q|| \le 2k$, and thus $(p+x-x'+2k) \bmod |Q| \le 4k$. ◼

Definition 5.5. *Let S denote a string and let Q denote a primitive string. We say that a fragment L of S is* locked *(with respect to Q) if at least one of the following holds:*

- *For some integer α, we have $\delta_E(L, {}^*Q^*) = \delta_E(L, Q^\alpha)$.*
- *The fragment L is a suffix of S and $\delta_E(L, {}^*Q^*) = \delta_E(L, Q^*)$.*
- *The fragment L is a prefix of S and $\delta_E(L, {}^*Q^*) = \delta_E(L, {}^*Q)$.*
- *We have $L = S$.*

The notion of locked fragments was also used in [33]. In order to develop some intuition, let us consider the following example: Consider a string $U = Q^{k+1}SQ^{k+1}$ such that $\delta_E(U, {}^*Q^*) \le k$ and Q is primitive. Then, in any optimal alignment of U with a substring of Q^∞, at least one of the leading (or trailing) $k+1$ occurrences of Q

in U is matched exactly and hence also all occurrences preceding it (or succeeding it). Thus U is locked with respect to Q.

Next, we show that we can identify short locked fragments covering all errors with respect to $^*Q^*$. Intuitively, our strategy is to start with at most k fragments of length $|Q|$ of S that contain all the errors and extend and/or merge them, so that the resulting fragments contain sufficiently many copies of Q.

LEMMA 5.6. *Let S denote a string and let Q denote a primitive string. There are disjoint locked fragments $L_1, \dots, L_\ell \preceq S$ with $\delta_E(L_i, {}^*Q^*) > 0$ such that*

$$\delta_E(S, {}^*Q^*) = \sum_{i=1}^{\ell} \delta_E(L_i, {}^*Q^*) \quad \textit{and} \quad \sum_{i=1}^{\ell} |L_i| \leq (5|Q|+1)\delta_E(S, {}^*Q^*).$$

PROOF. Let us choose integers $x \leq y$ that satisfy

$$\delta_E(S, {}^*Q^*) = \delta_E(S, Q^\infty[x\,..\,y)).$$

Without loss of generality, we may assume that $x \in [0\,..\,|Q|)$. If $y \leq |Q|$, then $|S| \leq |Q| + \delta_E(S, {}^*Q^*)$; thus setting the whole string S as the sole locked fragment satisfies the claimed conditions. Hence, we may assume that $y > |Q|$.

An arbitrary optimum alignment of S and $Q^\infty[x\,..\,y)$ yields a partition $S = S_0^{(0)} \cdots S_{s^{(0)}}^{(0)}$ with $s^{(0)} = \lfloor (y-1)/|Q| \rfloor$ such that

$$\delta_E(S, Q^\infty[x\,..\,y)) = \sum_{i=0}^{s^{(0)}} \Delta_i^{(0)},$$

where

$$\Delta_i^{(0)} = \begin{cases} \delta_E(S_0^{(0)}, Q[x\,..\,|Q|)) & \text{if } i = 0, \\ \delta_E(S_i^{(0)}, Q) & \text{if } 0 < i < s^{(0)}, \\ \delta_E(S_{s^{(0)}}^{(0)}, Q[0\,..\,y - s^{(0)}|Q|)) & \text{if } i = s^{(0)}. \end{cases}$$

We start with this partition and then coarsen it by exhaustively applying the merging rules specified below, where each rule is applied only if the previous rules cannot be applied. In each case, we re-index the unchanged fragments $S_i^{(t)}$ to obtain a new partition $S = S_0^{(t+1)} \cdots S_{s^{(t+1)}}^{(t+1)}$ and re-index the corresponding values $\Delta_i^{(t)}$ accordingly. We say that a fragment $S_i^{(t)}$ is *interesting* if $i = 0$, $i = s^{(t)}$, $S_i^{(t)} \neq Q$, or $\Delta_i^{(t)} > 0$.

1 If subsequent fragments $S_i^{(t)}$ and $S_{i+1}^{(t)}$ are both interesting, then we merge $S_i^{(t)}$ and $S_{i+1}^{(t)}$, obtaining $S_i^{(t+1)} := S_i^{(t)} S_{i+1}^{(t)}$ and $\Delta_i^{(t+1)} := \Delta_i^{(t)} + \Delta_{i+1}^{(t)}$.
2 If $0 < i < s^{(t)}$ and $\Delta_i^{(t)} > 0$, then we merge the subsequent fragments $S_{i-1}^{(t)} = Q$, $S_i^{(t)}$, and $S_{i+1}^{(t)} = Q$, obtaining $S_{i-1}^{(t+1)} := S_{i-1}^{(t)} S_i^{(t)} S_{i+1}^{(t)}$, and set $\Delta_{i-1}^{(t+1)} := \Delta_i^{(t)} - 1$.
3 If $0 < i = s^{(t)}$ and $\Delta_i^{(t)} > 0$, then we merge the subsequent fragments $S_{i-1}^{(t)} = Q$ and $S_i^{(t)}$, obtaining $S_{i-1}^{(t+1)} := S_{i-1}^{(t)} S_i^{(t)}$, and set $\Delta_{i-1}^{(t+1)} := \Delta_i^{(t)} - 1$.
4 If $0 = i < s^{(t)}$ and $\Delta_i^{(t)} > 0$, then we merge the subsequent fragments $S_i^{(t)}$ and $S_{i+1}^{(t)} = Q$, obtaining $S_i^{(t+1)} := S_i^{(t)} S_{i+1}^{(t)}$, and set $\Delta_i^{(t+1)} := \Delta_i^{(t)} - 1$.

Let $S = S_0 \cdots S_s$ denote the obtained final partition. We select as locked fragments all the fragments S_i with $\delta_E(S_i, {}^*Q^*) > 0$. Below, we show that this selection satisfies the desired properties. We start by proving that we indeed picked locked fragments.

CLAIM 5.7. *Each fragment $S_i^{(t)}$ of each partition $S = S_0^{(t)} \cdots S_{s^{(t)}}^{(t)}$ satisfies at least one of the following:*
1 $\delta_E(S_i^{(t)}, Q^\alpha) \le \delta_E(S_i^{(t)}, {}^*Q^*) + \Delta_i^{(t)}$ *for some integer α;*
2 $i = s^{(t)}$ *and* $\delta_E(S_i^{(t)}, Q^*) \le \delta_E(S_i^{(t)}, {}^*Q^*) + \Delta_i^{(t)}$*;*

3 *$i = 0$ and $\delta_E(S_i^{(t)}, {}^*Q) \le \delta_E(S_i^{(t)}, {}^*Q^*) + \Delta_i^{(t)}$;*

4 *$i = 0 = s^{(t)}$.*

Proof. We proceed by induction on t. The base case follows from the definition of the values $\Delta_i^{(t)}$.

As for the inductive step, we assume that the claim holds for all fragments $S_i^{(t)}$ and we prove that it holds for all fragments $S_i^{(t+1)}$. We consider several cases based on the merge rule applied.

1 For a type-1 merge of interesting fragments $S_i^{(t)}$ and $S_{i+1}^{(t)}$ into $S_i^{(t+1)}$, it suffices to prove that $S_i^{(t+1)}$ satisfies the claim.

a If $0 < i < s^{(t+1)}$, then $\delta_E(S_i^{(t)}, Q^\alpha) \le \delta_E(S_i^{(t)}, {}^*Q^*) + \Delta_i^{(t)}$ and $\delta_E(S_{i+1}^{(t)}, Q^{\alpha'}) \le \delta_E(S_{i+1}^{(t)}, {}^*Q^*) + \Delta_{i+1}^{(t)}$ hold by the inductive assumption for some integers α, α'. Hence,

$$\begin{aligned}
&\delta_E(S_i^{(t+1)}, Q^{\alpha+\alpha'})\\
&\quad = \delta_E(S_i^{(t)} S_{i+1}^{(t)}, Q^\alpha Q^{\alpha'}) \le \delta_E(S_i^{(t)}, Q^\alpha) + \delta_E(S_{i+1}^{(t)}, Q^{\alpha'})\\
&\quad \le \delta_E(S_i^{(t)}, {}^*Q^*) + \Delta_i^{(t)} + \delta_E(S_{i+1}^{(t)}, {}^*Q^*) + \Delta_{i+1}^{(t)}\\
&\quad \le \delta_E(S_i^{(t+1)}, {}^*Q^*) + \Delta_i^{(t+1)}.
\end{aligned}$$

b If $0 < i = s^{(t+1)}$, then $\delta_E(S_i^{(t)}, Q^\alpha) \le \delta_E(S_i^{(t)}, {}^*Q^*) + \Delta_i^{(t)}$ and $\delta_E(S_{i+1}^{(t)}, Q^*) \le \delta_E(S_{i+1}^{(t)}, {}^*Q^*) + \Delta_{i+1}^{(t)}$ hold by the inductive assumption for some integer α. Hence,

$$\begin{aligned}
&\delta_E(S_i^{(t+1)}, Q^*)\\
&\quad = \delta_E(S_i^{(t)} S_{i+1}^{(t)}, Q^*) \le \delta_E(S_i^{(t)}, Q^\alpha) + \delta_E(S_{i+1}^{(t)}, Q^*)\\
&\quad \le \delta_E(S_i^{(t)}, {}^*Q^*) + \Delta_i^{(t)} + \delta_E(S_{i+1}^{(t)}, {}^*Q^*) + \Delta_{i+1}^{(t)}\\
&\quad \le \delta_E(S_i^{(t+1)}, {}^*Q^*) + \Delta_i^{(t+1)}.
\end{aligned}$$

c The analysis of the case that $0 = i < s^{(t+1)}$ is symmetric to that of the above case—this can be seen by reversing all strings in scope.

d If $0 = i = s^{(t+1)}$, then the claim holds trivially.

2 For a type-2 merge of $S_{i-1}^{(t)}$, $S_i^{(t)}$, and $S_{i+1}^{(t)}$ into $S_{i-1}^{(t+1)}$, it suffices to prove that $S_{i-1}^{(t+1)}$ satisfies the claim.

a If $\delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) > \delta_E(S_i^{(t)}, {}^*Q^*)$, observe that $\delta_E(S_i^{(t)}, Q^\alpha) \leq \delta_E(S_i^{(t)}, {}^*Q^*) + \Delta_i^{(t)}$ holds by inductive assumption for some integer α. Hence,

$$\begin{aligned}\delta_E(S_{i-1}^{(t+1)}, Q^{\alpha+2}) &= \delta_E(QS_i^{(t)}Q, QQ^\alpha Q)\\ &\leq \delta_E(S_i^{(t)}, Q^\alpha) \leq \delta_E(S_i^{(t)}, {}^*Q^*) + \Delta_i^{(t)}\\ &\leq \delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) - 1 + \Delta_i^{(t)}\\ &= \delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) + \Delta_{i-1}^{(t+1)}.\end{aligned}$$

b If $\delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) = \delta_E(S_i^{(t)}, {}^*Q^*)$, then let $x' \leq y'$ denote integers that satisfy $\delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) = \delta_E(S_{i-1}^{(t+1)}, Q^\infty[x' \mathinner{.\,.} y'])$. This also yields integers x'', y'' with $x' \leq x'' \leq y'' \leq y'$ such that

$$\begin{aligned}&\delta_E(S_{i-1}^{(t+1)}, Q^\infty[x' \mathinner{.\,.} y'])\\ &\quad = \delta_E(Q, Q^\infty[x' \mathinner{.\,.} x''])\\ &\qquad + \delta_E(S_i^{(t)}, Q^\infty[x'' \mathinner{.\,.} y''])\\ &\qquad + \delta_E(Q, Q^\infty[y'' \mathinner{.\,.} y']).\end{aligned}$$

Due to

$$\begin{aligned}\delta_E(S_i^{(t)}, {}^*Q^*) &\leq \delta_E(S_i^{(t)}, Q^\infty[x'' \mathinner{.\,.} y''])\\ &\leq \delta_E(S_{i-1}^{(t+1)}, Q^\infty[x' \mathinner{.\,.} y'])\\ &= \delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) = \delta_E(S_i^{(t)}, {}^*Q^*),\end{aligned}$$

we have $\delta_E(Q, Q^\infty[x'..x'']) = 0 = \delta_E(Q, Q^\infty[y''..y'])$. As the string Q is primitive, this means that x', x'', y'', y' are all multiples of $|Q|$. Hence,

$$\begin{aligned}\delta_E(S_{i-1}^{(t+1)}, Q^{(y'-x')/|Q|}) &= \delta_E(S_{i-1}^{(t+1)}, Q^\infty[x'..y'])\\ &= \delta_E(S_{i-1}^{(t+1)}, {}^*Q^*)\\ &\leq \delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) + \Delta_{i-1}^{(t-1)}.\end{aligned}$$

3 For a type-3 merge of $S_{i-1}^{(t)}$ and $S_i^{(t)}$ into $S_{i-1}^{(t+1)}$, it suffices to prove that $S_{i-1}^{(t+1)}$ satisfies the claim.

a If $\delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) > \delta_E(S_i^{(t)}, {}^*Q^*)$, observe that $\delta_E(S_i^{(t)}, Q^*) \leq \delta_E(S_i^{(t)}, {}^*Q^*) + \Delta_i^{(t)}$ holds by inductive assumption. Hence,

$$\begin{aligned}&\delta_E(S_{i-1}^{(t+1)}, Q^*)\\ &\quad= \delta_E(QS_i^{(t)}, Q^*) \leq \delta_E(S_i^{(t)}, Q^*)\\ &\quad\leq \delta_E(S_i^{(t)}, {}^*Q^*) + \Delta_i^{(t)} \leq \delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) - 1 + \Delta_i^{(t)}\\ &\quad= \delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) + \Delta_{i-1}^{(t+1)}.\end{aligned}$$

b If $\delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) = \delta_E(S_i^{(t)}, {}^*Q^*)$, then let $x' \leq y'$ denote integers that satisfy $\delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) = \delta_E(S_{i-1}^{(t+1)}, Q^\infty[x'..y'])$. This also yields an integer x'' with $x' \leq x'' \leq y'$ such that

$$\begin{aligned}&\delta_E(S_{i-1}^{(t+1)}, Q^\infty[x'..y'])\\ &\quad= \delta_E(Q, Q^\infty[x'..x'']) + \delta_E(S_i^{(t)}, Q^\infty[x''..y']).\end{aligned}$$

Due to

$$\begin{aligned}\delta_E(S_i^{(t)}, {}^*Q^*) &\leq \delta_E(S_i^{(t)}, Q^\infty[x''..y'])\\ &\leq \delta_E(S_{i-1}^{(t+1)}, Q^\infty[x'..y'])\\ &= \delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) = \delta_E(S_i^{(t)}, {}^*Q^*),\end{aligned}$$

we have $\delta_E(Q, Q^\infty[x' \mathinner{.\,.} x'']) = 0$. As the string Q is primitive, this means that x', x'' are both multiples of $|Q|$. Hence,

$$\begin{aligned}\delta_E(S_{i-1}^{(t+1)}, Q^*) &= \delta_E(S_{i-1}^{(t+1)}, Q^\infty[x' \mathinner{.\,.} y'])\\ &= \delta_E(S_{i-1}^{(t+1)}, {}^*Q^*)\\ &\le \delta_E(S_{i-1}^{(t+1)}, {}^*Q^*) + \Delta_{i-1}^{(t-1)}.\end{aligned}$$

4 The analysis of type 4 merges is symmetrical to that of type 3 merges—this can be seen by reversing all strings in scope.

This completes the proof of the inductive step.

Observe that if no merge rule can be applied to a partition $S = S_0^{(t)} \cdots S_{s^{(t)}}^{(t)}$, then $s^{(t)} = 0$ or $\Delta_0^{(t)} = \cdots = \Delta_{s^{(t)}}^{(t)} = 0$. Hence, Claim 5.7 implies that all fragments S_i in the final partition $S = S_0 \cdots S_s$ are locked.

Claim 5.8. *For each partition $S = S_0^{(t)} \cdots S_{s^{(t)}}^{(t)}$, the total length $\lambda^{(t)}$ of interesting fragments satisfies*

$$\lambda^{(t)} + 2|Q| \sum_{i=0}^{s^{(t)}} \Delta_i^{(t)} \le (5|Q| + 1)\delta_E(S, {}^*Q^*).$$

Proof. We proceed by induction on t. In the base case of $t = 0$, each interesting fragment other than $S_0^{(0)}$ and $S_{s^{(0)}}^{(0)}$ satisfies $\Delta_i^{(0)} > 0$. Hence, the number of interesting fragments is at most

$$2 + \sum_{i=0}^{s^{(0)}} \Delta_i^{(0)} = 2 + \delta_E(S, {}^*Q^*).$$

Further, the length of each fragment $S_i^{(0)}$ does not exceed $|Q| + \Delta_i^{(0)}$. Hence,

$$\begin{aligned}\lambda^{(0)} + 2|Q| \sum_{i=0}^{s^{(0)}} \Delta_i^{(0)} &\le (2 + \delta_E(S, {}^*Q^*))|Q| + (2|Q| + 1) \sum_{i=0}^{s^{(0)}} \Delta_i^{(0)} \\ &\le (5|Q| + 1)\, \delta_E(S, {}^*Q^*).\end{aligned}$$

This completes the proof in the base case.

As for the inductive step, it suffices to prove that

$$\lambda^{(t+1)} + 2|Q| \sum_{i=0}^{s^{(t+1)}} \Delta_i^{(t+1)} \le \lambda^{(t)} + 2|Q| \sum_{i=0}^{s^{(t)}} \Delta_i^{(t)}$$

.

1 For a type-1 merge (where we merge two interesting fragments), we have

$$\lambda^{(t+1)} + 2|Q| \sum_{i=0}^{s^{(t+1)}} \Delta_i^{(t+1)} = \lambda^{(t)} + 2|Q| \sum_{i=0}^{s^{(t)}} \Delta_i^{(t)}.$$

2 For a type-2, type-3, or type-4 merge (where we merge a fragment with its one or two non-interesting neighbors), we have

$$\begin{aligned}\lambda^{(t+1)} + 2|Q| \sum_{i=0}^{s^{(t+1)}} \Delta_i^{(t+1)} &\le \lambda^{(t)} + 2|Q| + 2|Q| \sum_{i=0}^{s^{(t+1)}} \Delta_i^{(t+1)} \\ &= \lambda^{(t)} + 2|Q| \sum_{i=0}^{s^{(t)}} \Delta_i^{(t)}.\end{aligned}$$

Overall, we obtain the claimed bound. ◻

We conclude that the total length of interesting fragments S_i does not exceed $(5|Q| + 1)\delta_E(S, {}^*Q^*)$.

Claim 5.9. *We have $\delta_E(S, {}^*Q^*) = \sum_{i=0}^{s} \delta_E(S_i, {}^*Q^*)$.*

Proof. The claim is immediate if $s = 0$; hence, assume that $s \geq 1$. Observe that the inequality $\sum_{i=0}^{s} \delta_E(S_i, {}^*Q^*) \leq \delta_E(S, {}^*Q^*)$ easily follows from disjointness of fragments S_i; thus, we focus on proving $\delta_E(S, {}^*Q^*) \leq \sum_{i=0}^{s} \delta_E(S_i, {}^*Q^*)$.

For $0 \leq i \leq s$, let Q_i denote a substring of Q^∞ that satisfies $\delta_E(S_i, {}^*Q^*) = \delta_E(S_i, Q_i)$. Since each S_i is locked (by Claim 5.7), we may assume that for $0 < i < s$ the substring Q_i is a power of Q, the substring Q_s is a prefix of a power of Q, and the substring Q_0 is a suffix of a power of Q. Hence, $Q_0 \cdots Q_s$ is a substring of Q^∞, and we have

$$\begin{aligned}\delta_E(S, {}^*Q^*) &\leq \delta_E(S_0 \cdots S_s, Q_0 \cdots Q_s) \\ &\leq \sum_{i=0}^{s} \delta_E(S_i, Q_i) = \sum_{i=0}^{s} \delta_E(S_i, {}^*Q^*),\end{aligned}$$

thus completing the proof.

The locked fragments created satisfy

$$\delta_E(S, {}^*Q^*) = \sum_{i=1}^{\ell} \delta_E(L_i, {}^*Q^*)$$

due to Claim 5.9. Further, since we have $\delta_E(S_i, {}^*Q^*) > 0$ only for interesting fragments, Claim 5.8 yields

$$\sum_{i=1}^{\ell} |L_i| \leq (5|Q| + 1)\, \delta_E(S, {}^*Q^*),$$

completing the proof.

To prove Theorem 5.1, we do not need the definition and lemma that follow, as well as Item 4 of Lemma 5.2. A reader interested only in Theorem 5.1 may safely skip them. They, however, provide additional structural insights that we exploit in 6.

DEFINITION 5.10. *Let S denote a string, let Q denote a primitive string, and let $k \geq 0$ denote an integer. We say that a prefix L of S is k-*locked *(with respect to Q) if at least one of the following holds:*

- *For every $p \in [\,0\,..\,|Q|\,)$, if we have $\delta_E(L, \mathrm{rot}^p(Q)^*) \leq k$, then we also have $\delta_E(L, \mathrm{rot}^p(Q)^*) = \delta_E(L, Q^\infty[\,|Q| - p\,..\,j|Q|\,))$ for an integer j.*
- *We have $L = S$.*

LEMMA 5.11. *Let S denote a string, let Q denote a primitive string, and let $k \geq 0$ be an integer. There are disjoint locked fragments $L_1, \ldots, L_\ell \preceq S$, such that L_1 is a k-locked prefix of S, L_ℓ is a suffix of S, $\delta_E(L_i, {}^*Q^*) > 0$ for $1 < i < \ell$,*

$$\begin{aligned}\delta_E(S, {}^*Q^*) &= \sum_{i=1}^{\ell} \delta_E(L_i, {}^*Q^*), \quad \textit{and} \\ \sum_{i=1}^{\ell} |L_i| &\leq (5|Q| + 1)\delta_E(S, {}^*Q^*) + 2(k+1)|Q|.\end{aligned}$$

PROOF. We proceed as in the proof of Lemma 5.6 except that $\Delta_0^{(0)}$ is artificially increased by $k + 1$, the prefix S_0 in the final partition is included as L_1 among the locked fragments even if we have $\delta_E(S_0, {}^*Q^*) = 0$, and the suffix S_s is included as L_ℓ among the locked fragments even if $\delta_E(S_s, {}^*Q^*) = 0$.

It is easy to see that Claims 5.7 and 5.9 remain satisfied, whereas the upper bound in Claim 5.8 is increased by $2(k+1)|Q|$. We need to prove only that S_0 is a k-locked prefix of S. For this, we prove the following claim using induction.

CLAIM 5.12. *For each partition $S = S_0^{(t)} \cdots S_{s^{(t)}}^{(t)}$, at least one of the following holds:*

1 *For every $p \in [\,0\,..\,|Q|\,)$, if $\delta_E(S_0^{(t)}, \mathrm{rot}^p(Q)^*) \le k - \Delta_0^{(t)}$, then*

$$\delta_E(S_0^{(t)}, Q^\infty[\,|Q|-p\,..\,j|Q|\,)) \le \delta_E(S_0^{(t)}, \mathrm{rot}^p(Q)^*) + \Delta_0^{(t)}$$

for an integer j.

2 *We have $S_0^{(t)} = S$.*

Proof. We proceed by induction on t. In the base case of $t = 0$, the claim holds trivially since

$$\delta_E(S_0^{(0)}, \mathrm{rot}^p(Q)^*) \ge 0 > k - \Delta_0^{(0)}$$

holds for every p due to $\Delta_0^{(0)} \ge k + 1$.

As for the induction step, we assume that the claim holds for $S_0^{(t)}$ and we prove that it holds for $S_0^{(t+1)}$. The claim holds trivially if the merge rule applied did not affect $S_0^{(t)}$. Given that $S_0^{(t)}$ is interesting by definition, the merges that might affect $S_0^{(t)}$ are of type 1 (if $S_1^{(t)}$ is interesting) or 4 (otherwise).

1 Consider a type-1 merge of $S_0^{(t)}$ and $S_1^{(t)}$. If $s^{(t)} = 1$, then $S_0^{(t+1)} = S$ satisfies the claim trivially. Hence, we may assume that $1 < s^{(t)}$ so that Claim 5.7 yields $\delta_E(S_1^{(t)}, Q^\alpha) \le \delta_E(S_1^{(t)}, {}^*Q^*) + \Delta_1^{(t)}$ for some integer α. Let us fix $p \in [\,0\,..\,|Q|\,)$ with

$$\delta_E(S_0^{(t+1)}, \mathrm{rot}^p(Q)^*) \le k - \Delta_0^{(t+1)}.$$

Due to $\Delta_0^{(t+1)} \ge \Delta_0^{(t)}$, this yields $\delta_E(S_0^{(t)}, \mathrm{rot}^p(Q)^*) \le k - \Delta_0^{(t)}$, so the inductive assumption implies for an integer j

$$\delta_E(S_0^{(t)}, Q^\infty[\,|Q|-p\,..\,j|Q|\,)) \le \delta_E(S_0^{(t)}, \mathrm{rot}^p(Q)^*) + \Delta_0^{(t)}.$$

Hence,

$$\begin{aligned}
&\delta_E(S_0^{(t+1)}, Q^\infty[\,|Q|-p\,..\,(j+\alpha)|Q|\,)) \\
&\quad= \delta_E(S_0^{(t)}S_1^{(t)}, Q^\infty[\,|Q|-p\,..\,j|Q|\,)Q^\alpha) \\
&\quad\le \delta_E(S_0^{(t)}, Q^\infty[\,|Q|-p\,..\,j|Q|\,)) + \delta_E(S_1^{(t)}, Q^\alpha)
\end{aligned}$$

$$\begin{aligned}&\le \delta_E(S_0^{(t)}, \mathrm{rot}^p(Q)^*) + \Delta_0^{(t)} + \delta_E(S_1^{(t)}, {}^*Q^*) + \Delta_1^{(t)} \\ &\le \delta_E(S_0^{(t+1)}, \mathrm{rot}^p(Q)^*) + \Delta_0^{(t+1)}.\end{aligned}$$

2 Consider a type-4 merge of $S_0^{(t)}$ and $S_1^{(t)}$. Let us fix $p \in [\,0\,..\,|Q|\,)$ with $\delta_E(S_0^{(t+1)}, \mathrm{rot}^p(Q)^*) \le k - \Delta_0^{(t+1)}$.

a If $\delta_E(S_0^{(t+1)}, \mathrm{rot}^p(Q)^*) > \delta_E(S_0^{(t)}, \mathrm{rot}^p(Q)^*)$, then

$$\begin{aligned}\delta_E(S_0^{(t)}, \mathrm{rot}^p(Q)^*) &\le \delta_E(S_0^{(t+1)}, \mathrm{rot}^p(Q)^*) - 1 \\ &\le k - \Delta_0^{(t+1)} - 1 = k - \Delta_0^{(t)}.\end{aligned}$$

Hence, the inductive assumption implies

$$\delta_E(S_0^{(t)}, Q^\infty[\,|Q| - p\,..\,j|Q|\,)) \le \delta_E(S_0^{(t)}, \mathrm{rot}^p(Q)^*) + \Delta_0^{(t)}$$

for an integer j. Hence,

$$\begin{aligned}&\delta_E(S_0^{(t+1)}, Q^\infty[\,|Q| - p\,..\,(j+1)|Q|\,)) \\ &\quad = \delta_E(S_0^{(t)}Q, Q^\infty[\,|Q| - p\,..\,j|Q|\,)Q) \\ &\quad \le \delta_E(S_0^{(t)}, Q^\infty[\,|Q| - p\,..\,j|Q|\,)) \\ &\quad \le \delta_E(S_0^{(t)}, \mathrm{rot}^p(Q)^*) + \Delta_0^{(t)} \\ &\quad \le \delta_E(S_0^{(t+1)}, \mathrm{rot}^p(Q)^*) - 1 + \Delta_0^{(t)} \\ &\quad = \delta_E(S_0^{(t+1)}, \mathrm{rot}^p(Q)^*) + \Delta_0^{(t+1)}.\end{aligned}$$

b If $\delta_E(S_0^{(t+1)}, \mathrm{rot}^p(Q)^*) = \delta_E(S_0^{(t)}, \mathrm{rot}^p(Q)^*)$, then let y' denote an arbitrary integer that satisfies $\delta_E(S_0^{(t+1)}, \mathrm{rot}^p(Q)^*) = \delta_E(S_0^{(t+1)}, Q^\infty[\,|Q| - p\,..\,y'\,))$. This also yields an integer y'' with $|Q| - p \le y'' \le y'$ such that

$$\begin{aligned}&\delta_E(S_0^{(t+1)}, Q^\infty[\,|Q| - p\,..\,y'\,)) \\ &\quad = \delta_E(S_0^{(t)}, Q^\infty[\,|Q| - p\,..\,y''\,)) + \delta_E(Q, Q^\infty[\,y''\,..\,y'\,)).\end{aligned}$$

Due to

$$\begin{aligned}
&\delta_E(S_0^{(t)}, \mathrm{rot}^p(Q)^*)\\
&\quad \le \delta_E(S_0^{(t)}, Q^\infty[\,|Q|-p\,..\,y''\,))\\
&\quad \le \delta_E(S_0^{(t+1)}, Q^\infty[\,|Q|-p\,..\,y'\,))\\
&\quad = \delta_E(S_0^{(t+1)}, \mathrm{rot}^p(Q)^*) = \delta_E(S_0^{(t)}, \mathrm{rot}^p(Q)^*),
\end{aligned}$$

we have $\delta_E(Q, Q^\infty[\,y''\,..\,y''\,))$. As the string Q is primitive, this means that y'', y' are both multiples of $|Q|$. Hence,

$$\begin{aligned}
&\delta_E(S_0^{(t+1)}, Q^\infty[\,|Q|-p\,..\,y'\,))\\
&\quad = \delta_E(S_0^{(t+1)}, \mathrm{rot}^p(Q)^*)\\
&\quad \le \delta_E(S_0^{(t+1)}, \mathrm{rot}^p(Q)^*) + \Delta_0^{(t+1)}.
\end{aligned}$$

This completes the proof of the inductive step. ◪

Given that the final partition $S = S_0^{(t)} \cdots S_{s^{(t)}}^{(t)}$ satisfies $s^{(t)} = 0$ or $\Delta_0^{(t)} = 0$, we conclude that S_0 is indeed k-locked. ◪

We are now ready to prove Lemma 5.2.

LEMMA 5.2 (COMPARE LEMMA 3.2). *Let P denote a pattern of length m, let T denote a text of length n, and let $k \le m$ denote a non-negative integer such that $n < {}^3\!/_2\, m + k$. Suppose that the k-error occurrences of P in T include a prefix of T and a suffix of T. If there are a positive integer $d \ge 2k$ and a primitive string Q with $|Q| \le m/8d$ and $\delta_E(P, Q^*) = \delta_E(P, {}^*Q^*) \le d$, then each of following holds:*

1 *For every $p \in \mathrm{Occ}_k^E(P,T)$, we have $p \bmod |Q| \le 3d$ or $p \bmod |Q| \ge |Q| - 3d$.*
2 *The string T satisfies $\delta_E(T, {}^*Q^*) \le 3d$.*
3 *If $\delta_E(P, {}^*Q^*) = d$, then $|\lfloor \mathrm{Occ}_k^E(P,T)/d \rfloor| \le 304d$.*
4 *The set $\mathrm{Occ}_k^E(P,T)$ can be decomposed into $617d^3$ arithmetic progressions with difference $|Q|$.*

- Proof. Consider any k-error occurrence $T[\ell\,..\,r)$ of P. By definition, $\delta_E(T[\ell\,..\,r), P) \le k \le d/2$. Combining this inequality with $\delta_E(P, Q^*) \le d$ via the triangle inequality (Lemma 1.3), we obtain the bound $\delta_E(T[\ell\,..\,r), Q^*) \le {}^3\!/_2\, d$. In particular, this inequality is true for the k-error occurrence of P as a prefix of T. Hence, for some integer $m' \in [\, m-k\,..\,m+k\,]$, we have $\delta_E(T[\,0\,..\,m'\,), Q^*) \le {}^3\!/_2\, d$, and thus also $\delta_E(T[\,0\,..\,min(r,m')\,), Q^*) \le {}^3\!/_2\, d$.

Next, we apply Lemma 5.4 on the fragment $T[\,0\,..\,r)$, whose prefix $T[\,0\,..\,min(r,m')\,)$ satisfies $\delta_E(T[\,0\,..\,min(r,m')\,), Q^*) \le {}^3\!/_2\, d$ and whose suffix $T[\,\ell\,..\,r\,)$ satisfies $\delta_E(T[\,\ell\,..\,r\,), Q^*) \le {}^3\!/_2\, d$. Further, if $|Q| > 1$, we also have $|T[\,\ell\,..\,min(r,m')\,)| \ge (3d+1)|Q|$: Due to $r - \ell \ge m - k$ and $m' \ge m - k$, we have $min(r,m') - \ell \ge 2(m-k) - n > m/2 - 3k \ge 4d|Q| - 3k$. Hence, it suffices to prove that $(d-1)|Q| \ge 3k - 1$. This equality holds trivially if $k = 0$. For $k \ge 1$, we have $(d-1)|Q| \ge (2k-1)\cdot 2 = 4k - 2 \ge 3k - 1$ due to $d \ge 2k$ and $|Q| \ge 2$. Thus, we can indeed use Lemma 5.4.

In particular, Lemma 5.4 implies $(\ell + 3d) \bmod |Q| \le 6d$. Since $T[\,\ell\,..\,r\,)$ was an arbitrary k-error occurrence of P in T, we conclude that Item 1 holds.

Further, Lemma 5.4 implies $\delta_E(T[\,0\,..\,r\,), Q^*) \le 3d$. If we choose $T[\,\ell\,..\,r\,)$ to be a k-error occurrence of P that is a suffix of T, we have $r = n$ and therefore $\delta_E(T, Q^*) \le 3d$, which proves Item 2.

We proceed to the proof of Item 3. Let $L_1, \dots, L_\ell$ denote locked fragments of P obtained from Lemma 5.6. Note that we thus have

$$\sum_{i=1}^{\ell} \delta_E(L_i, {}^*Q^*) = \delta_E(P, {}^*Q^*) = d; \quad \ell \le d,$$

and $\sum_{i=1}^{\ell} |L_i| \le (5|Q| + 1)d$.

Further, let us fix an optimal alignment between T and a substring Q' of Q^*, and define $d' := \delta_E(T, Q^*)$. This yields partitions $T = T_0 \cdots T_{2d'}$ and $Q' = Q'_0 \cdots Q'_{2d'}$ such that:

- $T_i = Q'_i$ for even i,
- $T_i \ne Q'_i$ and $|T_i|, |Q'_i| \le 1$ for odd i.

We create a multi-set $E = \{\sum_{i'<i} |T_{i'}| : i \text{ is odd}\}$ of size d'. Its elements can be interpreted as positions in T which incur errors in an optimal alignment with Q'. In particular, we show the following:

Claim 5.13. *For every fragment $T[\,x\,..\,y\,)$, we have*

$$\delta_E(T[\,x\,..\,y\,), {}^*Q^*) \le |\{e \in E \mid x \le e < y\}|.$$

Proof. Observe that the alignment between T and Q' yields an alignment between $T[\,x\,..\,y\,)$ and a fragment $Q'[\,x'\,..\,y'\,)$ with $\delta_E(T[\,x\,..\,y\,), Q'[\,x'\,..\,y'\,)) \le |\{e \in E \mid x \le e < y\}|$ edits.

We split $\mathbb{Z}$ into disjoint blocks of the form $[\,jd\,..\,(j+1)d\,)$ for $j \in \mathbb{Z}$. We say that a block $[\,jd\,..\,(j+1)d\,)$ is *synchronized* if it contains a position p such that $(p + 3d) \bmod |Q| \le 6d$. For every locked fragment $L_i = P[\,\ell_i\,..\,r_i\,)$ and every $e \in E$, we mark a synchronized block $[\,jd\,..\,(j+1)d\,)$ if $e \in [\,jd + \ell_i - k\,..\,(j+1)d - 1 + r_i + k\,)$.

Claim 5.14. *If $[\,jd\,..\,(j+1)d\,) \cap \mathrm{Occ}^E_k(P,T) \ne \{\}$, then $[\,jd\,..\,(j+1)d\,)$ has at least $d-k$ marks.*

Proof. Consider a k-error occurrence of P in T starting at position $p \in [\,jd\,..\,(j+1)d\,)$ and fix its arbitrary optimal alignment with P. For each locked fragment L_i, let L'_i be the fragment of T aligned with L_i in this alignment. Further, $L'_i = T[\,\ell'_i\,..\,r'_i\,)$ for

$$[\,\ell'_i\,..\,r'_i\,) \subseteq [\,p+\ell_i-k\,..\,p+r_i+k\,) \subseteq [\,jd+\ell_i-k\,..\,(j+1)d-1+r_i+k\,).$$

Also, by Claim 5.13, we have

$$\begin{aligned}\delta_E(L'_i, {}^*Q^*) &\le |\{e \in E \mid \ell'_i \le e < r'_i\}| \\ &\le |\{e \in E \mid jd + \ell_i - k \le e < (j+1)d - 1 + r_i + k\}|.\end{aligned}$$

Hence, the number μ of marks at $[\,jd\,..\,(j+1)d\,)$ is at least

$$\mu \ge \sum_{i=1}^{\ell} \delta_E(L'_i, {}^*Q^*).$$

As the regions L_i are disjoint, we have

$$\sum_{i=1}^{\ell} \delta_E(L'_i, L_i) \le k.$$

By the triangle inequality (Lemma 1.3), this yields

$$\sum_{i=1}^{\ell} \delta_E(L_i, {}^*Q^*) \le k + \mu.$$

Since

$$\sum_{i=1}^{\ell} \delta_E(L_i, {}^*Q^*) = \delta_E(P, {}^*Q^*) = d,$$

we conclude that $\mu \ge d - k$.

Claim 5.15. *We place at most $152d^2$ marks.*

Proof. Let us fix $e \in E$ and a locked fragment $L_i = P[\ell_i \mathinner{..} r_i)$. Recall that a mark is placed at a synchronized block $[jd \mathinner{..} (j+1)d)$ if $e \in [jd + \ell_i - k \mathinner{..} (j+1)d - 1 + r_i + k)$, or, in other words,

$$[jd \mathinner{..} (j+1)d) \cap [e - r_i - k \mathinner{..} e - \ell_i + k) \ne \{\}.$$

The length of the interval $I_{i,e} := [e - r_i - k \mathinner{..} e - \ell_i + k)$ satisfies $|I_{i,e}| = 2k + |L_i| \le d + |L_i|$.

We now consider two cases depending on whether or not the inequality $|Q| < 9d$ is satisfied. If $|Q| < 9d$, it suffices to observe that any interval I overlaps with at most $2 + |I|/d$ blocks. Hence, $I_{i,e}$ overlaps with at most $3 + |L_i|/d$ blocks, and thus the number of marks we have placed due to e and L_i is bounded by $3 + |L_i|/d$. The overall number of marks is therefore at most

$$\begin{aligned}|E| \cdot \sum_{i=1}^{\ell} (3 + |L_i|/d) &\le 9d^2 + 3\sum_{i=1}^{\ell} |L_i| \\ &\le 9d^2 + 3(5|Q| + 1)d \le 9d^2 + 135d^2 = 144d^2.\end{aligned}$$

On the other hand, if $|Q| \geq 9d$, we utilize the fact that only synchronized blocks are marked. For this, observe that any interval I overlaps at most $2 + |I|/|Q|$ intervals of the form $[\,(j'-1)|Q| - 4d\,..\,j'|Q| + 4d\,)$ each of which overlaps with at most 7 synchronized blocks (covering $[\,j'|Q| - 3d\,..\,j'|Q| + 3d\,]$, which is of length $6d+1$). Hence, the total number of blocks marked due to e and L_i is bounded by $7(2 + (d + |L_i|)/|Q|)$. The overall number of marks is therefore at most

$$\begin{aligned}
&|E| \cdot \textstyle\sum_{i=1}^{\ell} 7(2 + (d + |L_i|)/|Q|)i \\
&\quad \leq 42d^2 + 21d^2/|Q| + 21d/|Q| \cdot \textstyle\sum_{i=1}^{\ell} |L_i| \\
&\quad \leq 42d^2 + 21d^2/|Q| + 21d^2 \cdot (5|Q| + 1)/|Q| \\
&\quad \leq 42d^2 + 21d^2/|Q| + 105d^2 + 21d^2/|Q| < 152d^2.
\end{aligned}$$

This completes the proof of the claim. ⌟

Hence, the number of blocks with at least $d/2$ marks is at most $304d$, completing the proof of Item 3.

We proceed to the proof of Item 4. Let $L_1^P, \dots, L_{\ell^P}^P$ denote the locked fragments of P obtained from Lemma 5.11 (so that L_1^P is a k-locked prefix of P), and let $L_1^T, \dots, L_{\ell^T}^T$ denote locked fragments of T obtained from Lemma 5.6. Set

$$\begin{aligned}
L_i^P &:= P[\,\ell_i^P\,..\,r_i^P\,) \quad \text{for } i \in [\,1\,..\,\ell^P\,] \text{ and} \\
L_j^T &:= T[\,\ell_j^T\,..\,r_j^T\,) \quad \text{for } j \in [\,1\,..\,\ell^T\,].
\end{aligned}$$

We say that a position $p \in [\,0\,..\,n\,)$ is *marked* if

$$\begin{aligned}
&p \in [\,n-m-k\,..\,n-m+k\,] \text{ or} \\
&[\,p + \ell_i^P - k\,..\,p + r_i^P + k\,) \cap [\,\ell_j^T\,..\,r_j^T\,) \neq \{\}
\end{aligned}$$

holds for some $i \in [\,1\,..\,\ell^P\,]$ and $j \in [\,1\,..\,\ell^T\,]$.[20] Further, we say that p is *synchronized* if $p \bmod |Q| \leq 3d$ or $p \bmod |Q| \geq |Q| - 3d$.

20. The positions in $[\,n-m-k\,..\,n-m+k\,]$ are marked for a technical reason that becomes clear in the proof of Claim 5.17.

Informally, if there is a k-occurrence at an unmarked position p, then no locked region of P can overlap a locked region of T in any corresponding optimal alignment and we can exploit this structure. Hence, we now show that there are only a few synchronized marked positions.

CLAIM 5.16. *The set of marked positions can be decomposed into at most $10d^2$ integer intervals. Further, the number of synchronized marked positions is at most $547d^3$.*

PROOF. First, consider the interval $[\, n-m-k \,..\, n-m+k \,]$. Observe that each pair $i \in [\, 1 \,..\, \ell^P \,]$ and $j \in [\, 1 \,..\, \ell^T \,]$ yields marked positions $p \in (\, \ell_j^T - r_i^P - k \,..\, r_j^T - \ell_i^P + k \,)$. Hence, the marked positions can be decomposed into $1 + \ell^P \cdot \ell^T$ integer intervals. Due to

$$\ell^P \leq \delta_E(P, {}^*Q^*) + 2 \leq d + 2 \leq 3d \text{ and } \ell^T \leq \delta_E(T, {}^*Q^*) \leq 3d,$$

the number of intervals is at most $10d^2$.

The interval of positions marked due to i and j is of length

$$2k + |L_i^P| + |L_j^T| - 1 \leq d + |L_i^P| + |L_j^T| - 1.$$

Out of any $|Q|$ consecutive positions, at most $6d+1 \leq 7d$ are synchronized. Hence, in any such interval I, the number of synchronized positions is at most $7d(|I| - 7d + |Q|)/|Q|$. Thus, the total number of synchronized marked positions does not exceed $2k + 1 \leq d^3$ (for $[\, n - m - k \,..\, n - m + k \,]$) plus

$$\begin{aligned}
&\sum_{i=1}^{\ell^P} \sum_{j=1}^{\ell^T} 7d \frac{|L_i^P| + |L_j^T| - 7d + |Q|}{|Q|} \\
&\leq \frac{7d\ell^T}{|Q|} \sum_{i=1}^{\ell^P} |L_i^P| + \frac{7d\ell^P}{|Q|} \sum_{i=1}^{\ell^T} |L_i^T| + \frac{7d\ell^P\ell^T(|Q| - 7d)}{|Q|} \\
&\leq \frac{21d^2}{|Q|} \left((5|Q| + 1)d + 2(k+1)|Q| + (5|Q| + 1)3d + 3d(|Q| - 7d) \right) \\
&\leq (21d^2/|Q|) \left((5|Q| + 1)d + 3d|Q| + (5|Q| + 1)3d + 3d(|Q| - 7d) \right) \\
&\leq (21d^2/|Q|)\,(26d|Q| + 4d - 21d^2) \leq 21 \cdot 26d^3 = 546d^3.
\end{aligned}$$

This completes the proof.

Next, we characterize unmarked positions $p \in \mathrm{Occ}^E_k(P,T)$. The unmarked positions can be decomposed into at most $10d^2$ integer intervals by Claim 5.16 and by the fact that $p < n - m - k$. Consider any such interval I.

Claim 5.17. *For any $p, p' \in I$ such that $p \equiv p' \pmod{|Q|}$ we have $p \in \mathrm{Occ}^E_k(P,T)$ if and only if $p' \in \mathrm{Occ}^E_k(P,T)$. In particular, $I \cap \mathrm{Occ}^E_k(P,T)$ can be decomposed into at most $6d+1$ arithmetic progressions with difference $|Q|$.*

Proof. By our marking scheme, for any i, j, for any pair $x \in [\,\ell^P_i \,..\, r^P_i\,)$ and $y \in [\,\ell^P_j \,..\, r^P_j\,)$, we see that $|(p+x) - y| > k$. Consider an unmarked position $p \in \mathrm{Occ}^E_k(P,T) \cap I$ and fix any alignment of any prefix $T[\,p \,..\, t\,)$ of $T[\,p \,..\, n\,)$ with P that has at most k errors. Then, for all i, the fragment T_i of T that is aligned with L^P_i is disjoint from all locked fragments of T. Hence, T_i is a substring of Q^∞. Now, recall that L^P_1 is a prefix of P and $L^P_{\ell^P}$ is a suffix of P. Hence, the locked fragments of T that are considered are exactly those L^T_j's with $p + k < \ell_j < r_j < p + m - k$; assume that this holds for $j \in [\,j_1 \,..\, j_2\,]$. For all $j \in [\,j_1 \,..\, j_2\,]$, the fragment P_j of P aligned with L^T_j is disjoint from all locked fragments of P and is a substring of Q^∞. In addition, since $I = [\,i_1 \,..\, i_2\,]$ is an interval of unmarked positions, $T[\,i_1 \,..\, i_2 + |L^P_1| + k\,)$ is disjoint from all locked fragments of T and hence is equal to a substring of Q^∞. Thus, for some r we see that:

$$\begin{aligned}
&\delta_E(P, T[\,p \,..\, t\,)) \\
&\quad \geq \delta_E(L^P_1, \mathrm{rot}^r(Q)^*) + \sum_{i=2}^{\ell^P} \delta_E(L^P_i, Q_i) + \sum_{j=j_1}^{j_2} \delta_E(\ell^T_j, P_j) \\
&\quad \geq \delta_E(L^P_1, \mathrm{rot}^r(Q)^*) + \sum_{i=2}^{\ell^P} \delta_E(L^P_i, {}^*Q^*) + \sum_{j=j_1}^{j_2} \delta_E(\ell^T_j, {}^*Q^*). \qquad (5.1)
\end{aligned}$$

Note that since L^P_1 is k-locked and $\delta_E(L^P_1, \mathrm{rot}^r(Q)^*) \le k$ there is a j such that

$$\delta_E(L^P_1, \mathrm{rot}^r(Q)^*) = \delta_E(L^P_1, Q^\infty[\,|Q| - r \,..\, j|Q|\,)).$$

Set $b = |Q^\infty[\,|Q| - r \,..\, j|Q|\,)|$.

Consider any position $p' \in I$ with $p' \equiv p \pmod{|Q|}$. We have $T[\,p' \,..\, p' + b\,) = T[\,p \,..\, p + b\,)$ since both fragments lie in $T[\,i_1 \,..\, i_2 + |L^P_1| + k\,]$ and start a multiple of $|Q|$ positions apart. In addition, we have

$$P[\,|L^P_1| \,..\, m\,) = Q^{\alpha_1} L^P_2 Q^{\alpha_2} \cdots Q^{\alpha_{\ell^P - 1}} L^P_{\ell^P}$$

for non-negative integers α_i. Further, we have

$$T[\,p' + b \,..\, p' + m + k\,) = Q^{\beta_{j_1 - 1}} L^T_{j_1} Q^{j_1} \cdots L^T_{j_2} Q^{\beta_{j_2}} Q'$$

for non-negative integers β_j and a prefix Q' of Q.[21] We claim that there is a t' that satisfies

21. Note that $p' + m + k < n$ since $p' < n - m - k$.

$$\begin{aligned}
&\delta_E(P, T[\,p' \,..\, t'\,)) \\
&\quad \le \delta_E(L^P_1, T[\,p' \,..\, p' + b\,)) + \delta_E(P[\,|L^P_1| \,..\, m\,), T[\,p' + b \,..\, t'\,)) \\
&\quad = \delta_E(L^P_1, \mathrm{rot}^r(Q)^*) + \sum_{i=2}^{\ell^P} \delta_E(L^P_i, {}^*Q^*) + \sum_{j=j_1}^{j_2} \delta_E(\ell^T_j, {}^*Q^*) \\
&\quad \le \delta_E(P, T[\,p \,..\, t\,)).
\end{aligned}$$

In order to prove this, let us consider the following greedy alignment of $P[\,|L^P_1| \,..\, m\,)$ and $T[\,p' + b \,..\, t'\,)$. We start at the leftmost position in both strings. We will maintain the invariant that the remainder of each string starts with either Q or a locked fragment, except possibly for the case that the remainder of P is $L^P_{\ell^P}$, in which case the remainder of T can be Q'. While we have not reached the end of $P[\,|L^P_1| \,..\, m\,)$ we repeat the following procedure. If both strings have a prefix equal to Q, we align those prefixes exactly. Else, the prefix of one of the strings is a locked fragment L. Let us first assume that $L \neq L^P_{\ell P}$. Then, since p' is unmarked, $Q^\infty[\,0\,..\,|L| + k - k'\,)$ is a prefix of the other string, where k' is the number of edits already performed by our greedy alignment. Since L is locked, $\delta_E(L, {}^*Q^*) = \delta_E(L, Q^\alpha)$ for some integer α. We have $|\alpha|Q| - |L|| \leq k - k'$ due to the fact that otherwise Equation (5.1) would imply that $\delta_E(P, T[\,p\,..\,t\,)) > k$, a contradiction. Hence, Q^α is a prefix of $Q^\infty[\,0\,..\,|L| + k - k'\,)$; we optimally align those two fragments. If $L = L^P_{\ell P}$, an analogous argument shows that there exists a prefix Q'' of the remainder of $T[\,p' + b\,..\,t'\,)$ such that $\delta_E(L, {}^*Q^*) = \delta_E(L, Q'')$. Upon termination of this greedy alignment, the equality in the above equation holds.

We have thus proved that if $p \in I \cap \mathrm{Occ}^E_k(P,T)$ then all $p' \in I$ such that $p' \equiv p \pmod{|Q|}$ are also in $\mathrm{Occ}^E_k(P,T)$. Thus, for any fixed $0 \leq j < |Q|$, for $U = \{i \cdot |Q| + j \in I \mid i \in \mathbb{Z}\}$ either $U \subseteq \mathrm{Occ}^E_k(P,T)$ or $U \cap \mathrm{Occ}^E_k(P,T) = \{\}$. By Item 1, we can restrict our attention to synchronized positions. We can thus decompose $I \cap \mathrm{Occ}^E_k(P,T)$ into at most $6d + 1$ arithmetic progressions. ◿

Combining Claims 5.16 and 5.17, we conclude that $\mathrm{Occ}^E_k(P,T)$ can be decomposed into at most $547d^3 + 10d^2(6d+1) \leq 617d^3$ arithmetic progressions with difference $|Q|$. ◿

Corollary 5.18 (Compare Corollary 3.5). *Let P denote a pattern of length m, let T denote a string of length n, and let $k \leq m$ denote a non-negative integer. Suppose that there are a positive integer $d \geq 2k$ and a primitive string Q with $|Q| \leq m/8d$ and $\delta_E(P, {}^*Q^*) = d$. Then $|\lfloor \mathrm{Occ}^E_k(P,T)/d \rfloor| \leq 1216 \cdot n/m \cdot d$.*

Proof. Partition the string T into $\lfloor 2n/m \rfloor$ blocks $T_0, \dots, T_{\lfloor 2n/m \rfloor - 1}$ of length at most $3/2\, m + k - 1$ each, where the ith block starts at position $i \cdot m/2$, that is, $T_i := T[\,\lfloor i \cdot m/2 \rfloor\,..\,min\{n, \lfloor (i+3) \cdot m/2 \rfloor + k - 1\}\,)$. Observe that each k-error occurrence of P in T is contained in at least one of the fragments T_i: Specifically, T_i covers all the occurrences starting in $[\,\lfloor i \cdot m/2 \rfloor\,..\,\lfloor (i+1) \cdot m/2 \rfloor\,)$. If $\mathrm{Occ}^E_k(P, T_i) \neq \{\}$, we define $T'_i := T[\,t'_i\,..\,t'_i + |T'_i|\,)$ to be the shortest fragment of T_i containing all k-error occurrences of P in T_i. As a result, T'_i satisfies the assumptions of Lemma 5.2, so $|\lfloor \mathrm{Occ}^E_k(P, T'_i) \rfloor / k| \leq 304d$.

Each block $[\,j'k\,..\,(j'+1)k\,)$ of positions in T_i corresponds to a block $[\,t'_i + j'k\,..\,t'_i + (j'+1)k\,)$ of positions in T, which intersects at most two blocks of the form $[\,jk\,..\,(j+1)k\,)$. In total, we conclude that $|\lfloor \mathrm{Occ}(P,T_i)\rfloor/k| \le \lfloor 2n/m\rfloor \cdot 2 \cdot 304d \le 1216 \cdot n/m \cdot d$. ◿

5.2 Characterization of the Non-Periodic Case

Next, we turn to the non-periodic case again. While this case is again more complicated compared to the Hamming distance setting, it is only moderately so—*fortunately*.

Lemma 5.19 (Compare Lemma 3.6). *Let P denote a string of length m and let $k \le m$ denote a positive integer. Then, at least one of the following holds:*

1. *The string P contains $2k$ disjoint* breaks *$B_1, \dots, B_{2k}$ each having periods* $\mathrm{per}(B_i) > m/128k$ *and length $|B_i| = \lfloor m/8k\rfloor$.*
2. *The string P contains disjoint* repetitive regions *$R_1, \dots, R_r$ of total length $\sum_{i=1}^{r} |R_i| \ge 3/8 \cdot m$ such that each region R_i satisfies $|R_i| \ge m/8k$ and has a primitive* approximate period *Q_i with $|Q_i| \le m/128k$ and $\delta_E(R_i, {}^*Q_i^*) = \lceil 8k/m \cdot |R_i|\rceil$.*
3. *The string P has a primitive* approximate period *Q that satisfies $|Q| \le m/128k$ and $\delta_E(P, {}^*Q^*) < 8k$.*

Proof. We use essentially the same algorithm as in the proof of Lemma 3.6: We replace all checks for a specific Hamming distance with the corresponding counterpart for the edit distance. Further, as we are interested only in (approximate) periods under an arbitrary rotation, we do not need to explicitly rotate the string Q in the algorithm anymore. Consider Algorithm 5.1 for a visualization; the changes to Algorithm 3.3 are highlighted.

In particular, we directly get an analogue of Claim 3.7:

Claim 5.20 (See Claim 3.7). *Whenever we consider a new fragment $P[\,j\,..\,j+\lfloor m/8k\rfloor\,)$ of $\lfloor m/8k\rfloor$ unprocessed characters of P, such a fragment starts at a position $j < 5/8 \cdot m$.* ◿

🟨 **Algorithm 5.1** A constructive proof of Lemma 5.19. Changes to Algorithm 3.3 are highlighted.

```
$\mathcal{B} \leftarrow \{\}; \mathcal{R} \leftarrow \{\};$
while true do
    Consider fragment $P' = P[\,j\,..\,j + \lfloor m/8k \rfloor\,)$ of the next $\lfloor m/8k \rfloor$ unprocessed
      characters of $P$;
    if $\mathrm{per}(P') > m/128k$ then
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{P'\};$
        if $|\mathcal{B}| = 2k$ then return breaks $\mathcal{B}$;
    else
        $Q \leftarrow P[\,j\,..\,j + \mathrm{per}(P')\,);$
        Search for prefix $R$ of $P[\,j\,..\,m\,)$ with $\delta_E(R, {}^*Q^*) = \lceil 8k/m \cdot |R| \rceil$ and $|R| > |P'|$;
        if such $R$ exists then
            $\mathcal{R} \leftarrow \mathcal{R} \cup \{(R, Q)\};$
            if $\sum_{(R,Q)\in\mathcal{R}} |R| \geq 3/8 \cdot m$ then
                return repetitive regions (and their corresponding periods) $\mathcal{R}$;
        else
            Search for suffix $R'$ of $P$ with $\delta_E(R', {}^*Q^*) = \lceil 8k/m \cdot |R'| \rceil$ and $|R'| \geq m - j$;
            if such $R'$ exists then return repetitive region $(R', Q)$;
            else return approximate period $Q$;
```

Again, note that Claim 5.20 also shows that whenever we consider a new fragment P' of $\lfloor m/8k \rfloor$ characters, there is indeed such a fragment, that is, P' is well-defined.

Now consider the following case: For a fragment $P' = P[\,j\,..\,j + \lfloor m/8k \rfloor\,)$ (that is not a break) and its corresponding period $Q = [\,j\,..\,j + \operatorname{per}(P')\,)$, we fail to obtain a new repetitive region R. Recall that in this case, we search for a repetitive region R' of length $|R'| \geq m - j$ that is a suffix of P and has an approximate period Q. If we indeed find such a region R', then $|R'| \geq m - j \geq m - 5/8 \cdot m = 3/8 \cdot m$ by Claim 5.20, so R' is long enough to be reported on its own. However, if we fail to find such R', we need to show that Q can be reported as an approximate period of P, that is, $\delta_E(P, {}^*Q^*) < 8k$.

Similar to Lemma 3.6, we first show that

$$\delta_E(P[\,j\,..\,m\,), {}^*Q^*) < \lceil 8k/m \cdot (m - j) \rceil.$$

For this, we inductively prove that the values

$$\Delta_\rho := \lceil 8k/m \cdot \rho \rceil - \delta_E(P[\,j\,..\,j + \rho\,), {}^*Q^*)$$

for $|P'| \leq \rho \leq m - j$ are all at least 1. In the base case of $\rho = |P'|$, we have $\Delta_\rho = 1 - 0$ because Q is the string period of P'. To carry out an inductive step, suppose that $\Delta_{\rho-1} \geq 1$ for some $|P'| < \rho \leq m - j$. Notice that $\Delta_\rho \geq \Delta_{\rho-1} - 1 \geq 0$: The first term in the definition of Δ_ρ has not decreased, and the term $\delta_E(P[\,j\,..\,j + \rho\,), {}^*Q^*)$ may have increased by at most 1 compared to $\Delta_{\rho-1}$. Further, $\Delta_\rho \neq 0$ as $R = P[\,j\,..\,j + \rho\,)$ could not be reported as a repetitive region. Since Δ_ρ is an integer, we conclude that $\Delta_\rho \geq 1$. This inductive reasoning ultimately shows that $\Delta_{m-j} > 0$, that is,

$$\delta_E(P[\,j\,..\,m\,), {}^*Q^*) < \lceil 8k/m \cdot (m - j) \rceil.$$

A symmetric argument holds for values

$$\Delta'_\rho := \lceil 8k/m \cdot \rho \rceil - \delta_E(P[\,m - \rho\,..\,m\,), {}^*Q^*)$$

for $m - j \leq \rho \leq m$, as no repetitive region R' was found as an extension of $P[\,j\,..\,m\,)$ to the left. Note that in contrast to the proof of Lemma 3.6, the rotation of Q is implicit. This completes the proof that $\delta_E(P, {}^*Q^*) < 8k$, that is, Q is an approximate period of P.

Lemma 5.21 (Compare Lemma 3.8). *Let P denote a pattern of length m, let T denote a text of length n, and let $k \leq m$ denote a positive integer. Suppose that P that contains $2k$ disjoint breaks $B_1, \dots, B_{2k} \preccurlyeq P$ each satisfying* $\operatorname{per}(B_i) \geq m/128k$. *Then, $|\lfloor \mathrm{Occ}^E_k(P,T)/k \rfloor| \leq 1024 \cdot n/m \cdot k$.*

Proof. The proof proceeds similarly to the proof of Lemma 3.8. The only major difference is that we obtain length-k *blocks* of possible starting positions instead of single starting positions. This is because the edit distance allows for deletions and insertions of characters.

Hence, we split $\mathbb{Z}$ into disjoint blocks of the form $[\,jk\,..\,(j+1)k\,)$ for $j \in \mathbb{Z}$. Now for every break $B_i = P[\,b_i\,..\,b_i + |B_i|\,)$, we mark a block $[\,jk\,..\,(j+1)k\,)$ if

$$[\,(j-1)\cdot k + b_i\,..\,(j+2)\cdot k + b_i\,) \cap \mathrm{Occ}(B_i, T) \neq \{\}.$$

Similarly to the proof of Lemma 3.8, we proceed to show that we place at most $O(n/m \cdot k^2)$ marks and that every k-error occurrence starts in a block with at least k marks.

Claim 5.22. *We place at most $1024 \cdot n/m \cdot k^2$ marks on blocks.*

Proof. Fix a break B_i. Notice that positions in $\mathrm{Occ}(B_i, T)$ are at distance at least $\operatorname{per}(B_i)$ from each other. Further, note that for every occurrence in $\mathrm{Occ}(B_i, T)$ we mark at most 4 blocks. Hence, for the break B_i, we place at most $512 \cdot n/m \cdot k$ marks. In total, we thus place at most $2k \cdot 512n/m \cdot k = 1024 \cdot n/m \cdot k^2$ marks.

Next, we show that every k-error occurrence of P in T starts in a block with at least k marks.

CLAIM 5.23. *If* $[\,jk\,..\,(j+1)k\,) \cap \mathrm{Occ}^E_k(P,T) \neq \{\}$, *then* $[\,jk\,..\,(j+1)k\,)$ *has at least* k *marks.*

PROOF. Consider a k-error occurrence of P in T starting at position $\ell \in [\,jk\,..\,(j+1)k\,)$ and fix an arbitrary optimal alignment of it with P. Out of the $2k$ breaks, at least k breaks are matched exactly, as not matching a break exactly incurs at least one error. If a break B_i is matched exactly, then for at least one $s \in [\,-k\,..\,k\,]$, we have $\ell + b_i + s \in \mathrm{Occ}(B_i, T)$. Since $jk \le \ell < (j+1)k$, we conclude that $[\,(j-1)\cdot k + b_i\,..\,(j+2)\cdot k + b_i\,) \cap \mathrm{Occ}(B_i,T) \neq \{\}$, that is, that the block $[\,jk\,..\,(j+1)k\,)$ has been marked for B_i. In total, there are at least k marks for the at least k breaks matched exactly.

By Claims 5.22 and 5.23, the number of blocks where k-error occurrences of P in T may start is $|\lfloor \mathrm{Occ}^E_k(P,T)/k \rfloor| \le (1024 \cdot n/m \cdot k^2)/k = 1024 \cdot n/m \cdot k$.

LEMMA 5.24 (COMPARE LEMMA 3.11). *Let* P *denote a string of length* m, *let* T *denote a string of length* n, *and let* $k \le m$ *denote a positive integer. Suppose that* P *contains disjoint repetitive regions* $R_1, \dots, R_r$ *of total length at least* $\sum_{i=1}^{r} |R_i| \ge 3/8 \cdot m$ *such that each region* R_i *satisfies* $|R_i| \ge m/8k$ *and has a primitive approximate period* Q_i *with* $|Q_i| \le m/128k$ *and* $\delta_E(R_i, {}^*Q_i^*) = \lceil 8k/m \cdot |R_i| \rceil$. *Then,* $|\lfloor \mathrm{Occ}^E_k(P,T)/k \rfloor| \le 642045 \cdot n/m \cdot k$.

PROOF. Again, the proof is similar to its counterpart from the Hamming distance setting. As before, a major difference is that we obtain only length-k *blocks* of possible starting positions instead of single starting positions.

As in the proof of Lemma 5.21, we split $\mathbb{Z}$ into disjoint blocks of the form $[\,jk\,..\,(j+1)k\,)$ for $j \in \mathbb{Z}$. Further, we set $m_R := \sum_r |R_i|$ and define $k_i := \lfloor 4 \cdot |R_i|/m \cdot k \rfloor$ for every $1 \le i \le r$.

For every repetitive region $R_i = P[\,r_i\,..\,r_i + |R_i|\,)$, we place $|R_i|$ marks on block $[\,jk\,..\,(j+1)k\,)$ if

$$[\,(j-1)\cdot k + r_i\,..\,(j+2)\cdot k + r_i\,) \cap \mathrm{Occ}^E_{k_i}(R_i, T) \neq \{\}.$$

Similarly to the proof Lemma 3.11, we proceed to show that we placed at most $O(n/m \cdot k \cdot m_R)$ marks and that every k-error occurrence of P in T starts in a block with at last $m_R - m/4$ marks.

Claim 5.25. *We place at most $214015 \cdot n/m \cdot k \cdot m_R$ marks.*

Proof. We use Corollary 5.18 to analyze $\mathrm{Occ}^E_{k_i}(R_i, T)$. For this, we set $d_i := \delta_E(R_i, {}^*Q_i^*) = \lceil 8k/m \cdot |R_i| \rceil$ and notice that $d_i \le 16k/m \cdot |R_i|$ since $|R_i| \ge m/8k$. Further, $d_i \ge 2k_i$ and $|Q_i| \le m/128k \le |R_i|/8d_i$. Hence, the assumptions of Corollary 5.18 are satisfied, so $\lfloor \mathrm{Occ}^E_{k_i}(R_i, T)/d_i \rfloor \le 1216 \cdot n/|R_i| \cdot d_i \le 19456 \cdot k \cdot n/m$.

For a block $[\, j'd_i \,.\,.\, (j'+1)d_i \,)$ intersecting $\mathrm{Occ}^E_{k_i}(R_i, T)$, we mark a block $[\, jk \,.\,.\, (j+1)k \,)$ only if

$$[\, (j-1) \cdot k + r_i \,.\,.\, (j+2) \cdot k + r_i \,) \cap [\, j'd_i \,.\,.\, (j'+1)d_i \,) \ne \{\},$$

which holds only if $jk \in [\, j'd_i - r_i - 2k \,.\,.\, (j'+1)d_i - r_i + k \,)$. The length of the latter interval is $d_i + 3k = \lceil 8k/m \cdot |R_i| \rceil + 3k \le 11k$, so the interval contains at most 11 multiples of k. Hence, the total number of marks placed due to R_i is bounded by $11 \cdot 19456 \cdot n/m \cdot k \cdot |R_i|$. Across all repetitive regions, this sums up to no more than $214015 \cdot n/m \cdot k \cdot m_R$, yielding the claim.

Next, we show that every k-error occurrence of P in T starts in a block with many marks.

Claim 5.26. *If $[\, jk \,.\,.\, (j+1)k \,) \cap \mathrm{Occ}^E_k(P, T) \ne \{\}$, then $[\, jk \,.\,.\, (j+1)k \,)$ has at least $m_R - m/4$ marks.*

Proof. Consider a k-error occurrence of P in T starting at position $\ell \in [\, jk \,.\,.\, (j+1)k \,)$ and fix an arbitrary optimal alignment of it with P. For each repetitive region R_i, let R_i' be the fragment of T aligned with R_i in this alignment. Define $k_i' = \delta_E(R_i, R_i')$ and observe that $R_i' = T[\, r_i' \,.\,.\, r_i' + |R_i'| \,)$ for some $r_i' \in [\, \ell + r_i - k \,.\,.\, \ell + r_i + k \,] \subseteq [\, (j-1) \cdot k + r_i \,.\,.\, (j+2) \cdot k + r_i \,)$. Hence,

$$[\, (j-1) \cdot k + r_i \,.\,.\, (j+2) \cdot k + r_i \,) \cap \mathrm{Occ}^E_{k_i'}(R_i, T) \ne \{\}.$$

Further, let $I := \{i \mid k'_i \le k_i\} = \{i \mid k'_i \le 4 \cdot |R_i|/m \cdot k\}$ denote the set of indices i for which R'_i is a k_i-error occurrence of R_i. By construction, for each $i \in I$, we have placed $|R_i|$ marks at the block $[\,jk\,..\,(j+1)k\,)$.

Hence, the total number of marks at the block $[\,jk\,..\,(j+1)k\,)$ is at least

$$\sum_{i\in I} |R_i| = m_R - \sum_{i\notin I} |R_i|.$$

It remains to bound the term $\sum_{i\notin I} |R_i|$. Using the definition of I, we obtain

$$\sum_{i\notin I} |R_i| = \frac{m}{4k} \cdot \sum_{i\notin I} (4 \cdot |R_i|/m \cdot k) < \frac{m}{4k} \cdot \sum_{i\notin I} k'_i \le \frac{m}{4k} \cdot \sum_{i=1}^{r} k'_i \le \frac{m}{4},$$

where the last bound holds because, in total, all repetitive regions incur at most $\sum_{i=1}^{r} k'_i \le k$ errors (since the repetitive regions are disjoint). Hence, the number of marks placed is at least $m_r - m/4k$, completing the proof of the claim. ◿

In total, by Claims 5.25 and 5.26, the number of k-error occurrences of P in T is at most

$$\mathrm{Occ}^E_k(P,T) \le \frac{214015 \cdot n/m \cdot k \cdot m_R}{m_R - m/4}.$$

As this bound is a decreasing function in m_R, the assumption $m_R \ge 3/8 \cdot m$ yields

$$\mathrm{Occ}^E_k(P,T) \le \frac{214015 \cdot n/m \cdot k \cdot 3/8 \cdot m}{3/8 \cdot m - m/4} = 642045 \cdot n/m \cdot k,$$

completing the proof. ◿

◸ LEMMA 5.27 (COMPARE LEMMA 3.14). *Let P denote a string of length m, let T denote a string of length n, and let $k \le m$ denote a positive integer. If there is a primitive string Q of length at most $|Q| \le m/128k$ that satisfies $2k \le \delta_E(P, {}^*Q^*) \le 8k$, then $|\lfloor \mathrm{Occ}^E_k(P,T)/k \rfloor| \le 87551 \cdot n/m \cdot k$.*

Proof. We apply Corollary 5.18 with $d = \delta_E(P, {}^*Q^*)$. Observe that $d \geq 2k$ and that $|Q| \leq m/128k \leq m/8d$ due to $d \leq 8k$. Hence, the assumptions of Corollary 5.18 are met.

Hence, $|\lfloor \mathrm{Occ}_k^E(P,T)/d \rfloor| \leq 1216 \cdot n/m \cdot d \leq 9728 \cdot n/m \cdot k$. Every block $[\,j'd\,..\,(j'+1)d\,)$ is of length at most $8k$, and thus may intersect at most 9 blocks of the form $[\,jk\,..\,(j+1)k\,)$. Hence, $\lfloor \mathrm{Occ}_k^E(P,T)/k \rfloor \leq 9 \cdot 9728 \cdot n/m \cdot k$, completing the proof.

Theorem 5.1 (Compare Theorem 3.1). *Given a pattern P of length m, a text T of length n, and a positive integer $k \leq m$, then at least one of the following holds.*

- *We have $|\lfloor \mathrm{Occ}_k^E(P,T)/k \rfloor| \leq 642045 \cdot n/m \cdot k$.*
- *There is a primitive Q with $|Q| \leq m/128k$ and $\delta_E(P, {}^*Q^*) < 2k$.*

Proof. The proof proceeds just as the proof of Theorem 3.1: We apply Lemma 5.19 on the string P and proceed depending on the structure found in P.

If the string P contains $2k$ disjoint breaks $B_1, \dots, B_{2k}$ (in the sense of Lemma 5.19), we apply Lemma 5.21 and obtain that

$$|\lfloor \mathrm{Occ}_k^E(P,T)/k \rfloor| \leq 1024 \cdot n/m \cdot k.$$

If the string P contains disjoint repetitive regions $R_1, \dots, R_r$ (in the sense of Lemma 5.19), we apply Lemma 5.24 and obtain that

$$|\lfloor \mathrm{Occ}_k^E(P,T)/k \rfloor| \leq 642045 \cdot n/m \cdot k.$$

Otherwise, Lemma 5.19 yields that there is a primitive string Q of length at most $|Q| \leq m/128k$ that satisfies $\delta_E(P, {}^*Q^*) < 8k$. If $\delta_E(P, {}^*Q^*) \geq 2k$, then Lemma 5.27 yields $|\lfloor \mathrm{Occ}_k^E(P,T)/k \rfloor| \leq 87551 \cdot n/m \cdot k$. If, however, $\delta_E(P, {}^*Q^*) < 2k$, then we are in the second alternative of the theorem statement.

Algorithm: Pattern Matching with Edits in the `PILLAR` Model

6

In this chapter, we discuss how to solve pattern matching with edits in the `PILLAR` model. Specifically, we prove the following result.

Theorem 6.1. *Given a pattern P of length m, a text T of length n, and a positive integer $k \le m$, we can compute (a representation of) the set $\mathrm{Occ}_k^E(P, T)$ using $O(n/m \cdot k^4)$ time in the `PILLAR` model.*

The overall structure of the algorithm is similar to the Hamming distance case: We first introduce useful tools for the algorithms later. Then, we implement `Analyze`, which is then followed by a discussion of the case when the pattern is periodic. Finally, we discuss the easier non-periodic case and conclude with combining the various auxiliary algorithms.

6.1 Auxiliary `PILLAR` Model Operations for Pattern Matching with Edits

As in the Hamming distance setting, we start the discussion of the algorithms with general tools that we use as auxiliary procedures in the remaining algorithms. Specifically, we discuss a generator that computes the "next" error between two strings. Further, we discuss a procedure to verify whether there is an occurrence of the pattern at a given (interval of) position(s) in the text.

Lemma 6.2 (`EditGen`(S, Q), `EditGen`$^R(S, Q)$). *Let S denote a string and let Q denote a string (that is possibly given as a cyclic rotation $Q' = \mathrm{rot}^j(Q)$). Then, there is an $(O(1), O(k))$-time generator that in the k-th call to* `Next` *returns the length of the longest prefix (suffix) S'*

Algorithm 6.1 An analogue of Algorithm 4.1 for the edit distance: We adapt the algorithm by Landau and Vishkin [78] to make its execution "resumable".

```
EditGen($S, Q' = \mathrm{rot}^j(Q)$)
    return {
        $S \leftarrow S; Q' \leftarrow Q'; k \leftarrow 0; j \leftarrow j; end \leftarrow$ false;
        $(\ell^{(-1)}_{-1}, \dots, \ell^{(-1)}_{1}) \leftarrow (-\infty, -\infty, -\infty)$;
        $A[\,-2\,..\,2\,] \leftarrow [\,(), \dots, ()\,]$;
    };
Next(G = $\{S;\ Q';\ k;\ j;\ (\ell^{(k-1)}_{-k-1}, \dots, \ell^{(k-1)}_{k+1});\ A;\ end\}$)
    if $k = 0$ then
        $(\ell^{(0)}_{-2}, \dots, \ell^{(0)}_{2}) \leftarrow (-\infty, -\infty, \mathrm{LCP}(S, Q^{\infty}), -\infty, -\infty)$;
        replace $\ell^{(k-1)}$ with $\ell^{(k)}$; $k \leftarrow k + 1$;
        return $\ell^{(0)}_{0}$;
    $(\ell^{(k)}_{-k-2}, \dots, \ell^{(k)}_{k+2}) \leftarrow (-\infty, \dots, -\infty)$;
    $A'[\,-k-2\,..\,k+2\,] \leftarrow [\,(), \dots, ()\,]; r \leftarrow -\infty; a_r \leftarrow -1$;
    for $i \leftarrow -k-1$ to $k+1$ do
        // Compute new prefix lengths as long as we did not reach the end of $S$
        if not $end$ then
            $\ell_{insert} \leftarrow \ell^{(k-1)}_{i-1} + 1 + \mathrm{LCP}(S[\,\ell^{(k-1)}_{i-1} - i\,..\,|S|\,), Q'^{\infty}[\,j + \ell^{(k-1)}_{i} + 1\,..\,))$;
            $\ell_{replace} \leftarrow \ell^{(k-1)}_{i} + 1 + \mathrm{LCP}(S[\,\ell^{(k-1)}_{i} - i + 1\,..\,|S|\,), Q'^{\infty}[\,j + \ell^{(k-1)}_{i} + 1\,..\,))$;
            $\ell_{delete} \leftarrow \ell^{(k-1)}_{i+1} + \mathrm{LCP}(S[\,\ell^{(k-1)}_{i} - i + 1\,..\,|S|\,), Q'^{\infty}[\,j + \ell^{(k-1)}_{i}\,..\,))$;
            $\ell^{(k)}_{i} \leftarrow max(\ell_{insert}, \ell_{replace}, \ell_{delete})$;
        else $\ell^{(k)}_{i} \leftarrow \ell^{(k-1)}_{i}$;
        $r \leftarrow max(r, \ell^{(k)}_{i})$;
        if $r = \ell^{(k)}_{i} - i$ then $a_r \leftarrow \ell^{(k)}_{i} + i$;
        if $end$ then continue;

        // Store witness for Alignment
        if $\ell^{(k)}_{i} = \ell_{insert}$ then
            $A'[\,i\,] \leftarrow (A[\,i-1\,], (\perp, j + \ell^{(k-1)}_{i}))$;
        if $\ell^{(k)}_{i} = \ell_{replace}$ then
            $A'[\,i\,] \leftarrow (A[\,i\,], (\ell_i - i, j + \ell^{(k-1)}_{i}))$;
        if $\ell^{(k)}_{i} = \ell_{delete}$ then
            $A'[\,i\,] \leftarrow (A[\,i+1\,], (\ell^{(k-1)}_{i} - i, \perp))$;
    replace $\ell^{(k-1)}$ with $\ell^{(k)}$; $k \leftarrow k + 1; A \leftarrow A'$;
    if $r \geq |S|$ then $end \leftarrow$ true;
    return $(r, a_r)$;
Alignment(G = $\{S;\ Q';\ k;\ j;\ (\ell^{(k-1)}_{-k-1}, \dots, \ell^{(k-1)}_{k+1});\ A;\ end\}$)
    $r \leftarrow -\infty; a_r \leftarrow -1$;
    for $i \leftarrow -k-1$ to $k+1$ do
        $r \leftarrow max(r, \ell^{(k-1)}_{i})$;
        if $r = \ell^{(k-1)}_{i}$ then $a_r \leftarrow i$;
    return $A[\,a_r\,]$;
```

of S and the length of the corresponding prefix (suffix) Q' of Q^∞ such that $\delta_E(S', Q') \le k$.[22]

Further, both generators support an additional operation `Alignment`*, that outputs a witness for the result returned by k-th call to* `Next` *that is,* `Alignment` *outputs a sequence of edits ((i,j) for a replacement, $(i,\perp)$ for an insertion in S, and $(\perp,i)$ for an insertion in Q). The operation* `Alignment` *takes $O(k)$ time in the* `PILLAR` *model.*

22. Note that $k \ge 0$, that is, the initial call to `Next` is the zeroth call.

PROOF. We focus on the generator `EditGen(`S`, `Q`)`; we obtain the generator `EditGen`^{R}`(`S`, `Q`)` in a symmetric manner.

We construct the generator as follows: We run the dynamic programming algorithm by Landau and Vishkin [78] for one additional error (per call to `Next`) at a time, storing the dynamic programming table as a state in the generator. In particular, we maintain a sequence ℓ that after k calls to `Next` stores at a position $i \in [-k\,..\,k]$ the length of the longest prefix S' of S that satisfies[23]

$$\delta_E(S', Q^\infty[0\,..\,|S'|+i]) \le k.$$

23. Intuitively, we store how far each of the "diagonals" of the dynamic programming table extends.

In each call to `Next`, we update the values stored in the sequence ℓ by using three calls to `LCP` from Corollary 2.4 (one call for each of the `insert`, `replace`, `delete` cases of the edit distance) to compute each new entry. We then obtain the result as the maximum value of the newly computed sequence.

In order to support the `Alignment` operation, we additionally store every diagonal represented as list of pairs of `insert`, `replace`, and `delete` operations and the position(s) in the strings S and Q^∞ where the edits happened. In each call to `Next`, we also update the representations of the diagonals.[24]

24. Note that, for performance reasons, we need to avoid copying whole diagonals; this can be done by storing all diagonals together as a (directed) graph.

Consider Algorithm 6.1 for a visualization of the generator as pseudo-code; note that we simplified how we store the diagonals for the `Alignment` operation to improve readability.

For the correctness, we show by induction that the values computed in the array ℓ are indeed correct, that is, after k calls to the `Next` operation, for all $i \in [-k\,..\,k]$ we have

$$\ell_i^{(k)} = max_r\{r \mid \delta_E(S[0\,..\,r], Q^\infty[0\,..\,r+i]) \le k\},$$

or we reached the end of the string S (in which case the output does not change anymore after calling `Next`). For the zeroth call to next, we explicitly return

$$\ell_0^{(0)} = \mathtt{LCP}(S, Q^\infty) = max_r\{r \mid \delta_E(S[\,0\,..\,r\,], Q^\infty[\,0\,..\,r+0\,]) = 0\},$$

which is thus correct. Now consider the k-th call to `Next` and assume that the values computed so far are correct. In particular, for all $i \in [\,-k+1\,..\,k-1\,]$, we have

$$\ell_i^{(k-1)} = max_r\{r \mid \delta_E(S[\,0\,..\,r\,], Q^\infty[\,0\,..\,r+i\,]) \le k-1\}.$$

Now, fix a $j \in [\,k\,..\,-k\,]$ and consider the longest prefix $S' = S[\,0\,..\,r\,]$ with $\delta_E(S', Q^\infty[\,0\,..\,r+j\,]) = k'$, for some $0 < k' \le k$. By definition of the edit distance, there is an integer r' that satisfies $S'(\,r'\,..\,r\,] = Q^\infty(\,r'+j\,..\,r+j\,]$ and at least one of the following

- $k' = \delta_E(S'[\,0\,..\,r'\,), Q^\infty[\,0\,..\,r'+j\,)) + 1$ and $S[\,r'\,] \neq Q^\infty[\,r'\,]$ (when changing the character $S[\,r'\,]$ to the character $Q^\infty[\,r'+j\,]$);
- $k' = \delta_E(S'[\,0\,..\,r'\,), Q^\infty[\,0\,..\,r'+j-1\,)) + 1$ (when inserting the character $Q^\infty[\,r'+j\,]$); or
- $k' = \delta_E(S'[\,0\,..\,r'\,], Q^\infty[\,0\,..\,r'+j\,)) + 1$ (when deleting the character $Q^\infty[\,r'+j\,]$).

Note that (as $S'(\,r'\,..\,r\,] = Q^\infty(\,r'+j\,..\,r+j\,]$) we may assume that r' is maximal (that is there is no larger integer with the same properties as r'). In particular, we have

$$r' = max(\ell_j^{(k-1)} + 1, \ell_{j-1}^{(k-1)} + 1, \ell_{j+1}^{(k-1)}),$$

and hence `Next` computes $\ell_j^{(k)}$ indeed correctly.

Using the computed values $\ell^{(k)}$, we can compute the length $|S'|$ of the longest prefix S' of S and the length $|Q'|$ of the corresponding prefix Q' of Q^∞ such that $\delta_E(S', Q') \le k$: For $|S'|$, we have

$$|S'| = max_{j,r}\{r \mid \delta_E(S[\,0\,..\,r\,], Q^\infty[\,0\,..\,r+y\,]) \le k\} = max_j \ell_j^{(k)},$$

as k edits allow only for up to k insertions or deletions (that is, operations that can change the shift between Q^∞ and S). Hence, the computation of $|S'|$ in `Next` is correct. For $|Q'|$, observe that if $|S'| = \ell_j^{(k)}$ for some $j \in [-k\,.\,.\,k]$, by construction, we have $|Q'| = \ell_j^{(k)} + j$. Hence, also the computed value for $|Q'|$ is correct.

For the correctness of `Alignment`, observe that we store information computed in `Next` (which is correct); further we always synchronize the information stored, hence also `Alignment` is correct.

For the running time, observe that in the k-th call to `Next` we call `LCP` three times for each of the $2k+3$ values $\ell_i^{(k)}$. Further, all other operations are essentially book-keeping that can be implemented in $O(1)$ time. Hence in total, the k-th call to `Next` takes $O(k)$ time in the `PILLAR` model.

For the `Alignment` operation, observe that we traverse the sequence $\ell^{(k)}$ exactly once, hence `Alignment` uses $O(k)$ time in the `PILLAR` model, completing the proof. ⌟

⌜ Lemma 6.3 (`Verify(`P`, `T`, `k`, `I`)`, [33, Section 5]). *Let P denote a string of length m, let T denote a string, and let $k \le m$ and denote a positive integer. Further, let I denote an interval of positive integers. Using $O(k(k+|I|))$ `PILLAR` operations, we can compute*[25]

$$\{(\ell, min_r \delta_E(P, T[\,\ell\,.\,.\,r\,])) \mid \ell \in \mathrm{Occ}_k^E(P,T) \cap I\}.$$

25. Note that we can also call `Verify(`P`, `T`, `k`, `I`)` for strings $T = Q^\infty$ (for some primitive Q), as, by Corollary 2.4, we can also efficiently compute `LCP`(P, Q^∞).

▭ Proof. Observe that the algorithm in [33, Section 5] uses only `LCP` operations, as it mainly implements [78]. In particular, the algorithm in [33, Section 5] uses $O(k(k+|I|))$ `LCP` operations. The claim follows. ⌟

6.2 Computing Structure in the Pattern

We proceed to discuss the implementation of Lemma 5.19, that is, the analysation of the pattern. While the algorithm itself is similar to the Hamming distance case, the analysis requires more involved arguments. We start with an auxiliary combinatorial lemma:

Lemma 6.4. *Let S denote a string, let k denote a positive integer, and let Q denote a primitive string such that $|Q| = 1$ or $|S| \geq (2k+1)|Q|$. Suppose that for integers $x \leq y$, we have*

$$\delta_E(S, {}^*Q^*) = \delta_E(S, Q^\infty[x\,..\,y)) \leq k.$$

Then, for any string S',

- *if $\delta_E(SS', {}^*Q^*) \leq k$, then $\delta_E(SS', {}^*Q^*) = \delta_E(SS', \mathrm{rot}^{-x}(Q)^*)$, and*
- *if $\delta_E(S'S, {}^*Q^*) \leq k$, then $\delta_E(S'S, {}^*Q^*) = \delta_E(S'S, {}^*\mathrm{rot}^{-y}(Q))$.*

Proof. Suppose that $\delta_E(SS', {}^*Q^*) = \delta_E(SS', Q^\infty[x'\,..\,z')) \leq k$ for some integers $x' \leq z'$. Then, there is a position $y' \in [x'\,..\,z']$ such that

$$\delta_E(SS', {}^*Q^*) = \delta_E(S, Q^\infty[x'\,..\,y')) + \delta_E(S', Q^\infty[y'\,..\,z')) \leq k.$$

Due to $\delta_E(S, Q^\infty[x\,..\,y)) \leq k$ and $\delta_E(S, Q^\infty[x'\,..\,y')) \leq k$, we may apply Lemma 5.3, which yields a decomposition $S = S_L S_R$ and integers j, j' such that

$$\begin{aligned}
&\delta_E(S, Q^\infty[x\,..\,y)) \\
&\quad = \delta_E(S_L, Q^\infty[x\,..\,j|Q|)) + \delta_E(S_R, Q^\infty[j|Q|\,..\,y)) \quad \text{and} \\
&\delta_E(S, Q^\infty[x'\,..\,y')) \\
&\quad = \delta_E(S_L, Q^\infty[x'\,..\,j'|Q|)) + \delta_E(S_R, Q^\infty[j'|Q|\,..\,y')).
\end{aligned}$$

In particular, we have

$$\begin{aligned}
&\delta_E(S_L, Q^\infty[x\,..\,j|Q|)) + \delta_E(S_R, Q^\infty[j|Q|\,..\,y)) \\
&\quad = \delta_E(S, Q^\infty[x\,..\,y)) = \delta_E(S, {}^*Q^*) \\
&\quad \leq \delta_E(S, Q^\infty[x'\,..\,y + (j'-j)|Q|)) \\
&\quad \leq \delta_E(S_L, Q^\infty[x'\,..\,j'|Q|)) + \delta_E(S_R, Q^\infty[j|Q|\,..\,y)),
\end{aligned}$$

and hence $\delta_E(S_L, Q^\infty[x\,..\,j|Q|)) \le \delta_E(S_L, Q^\infty[x'\,..\,j'|Q|))$. Thus,

$$\begin{aligned}
&\delta_E(SS', \mathrm{rot}^{-x}(Q)^*)\\
&\quad\le \delta_E(SS', Q^\infty[x\,..\,z' + (j-j')|Q|))\\
&\quad\le \delta_E(S_L, Q^\infty[x\,..\,j|Q|)) + \delta_E(S_R, Q^\infty[j'|Q|\,..\,y'))\\
&\qquad + \delta_E(S', Q^\infty[y'\,..\,z'))\\
&\quad\le \delta_E(S_L, Q^\infty[x'\,..\,j'|Q|)) + \delta_E(S_R, Q^\infty[j'|Q|\,..\,y'))\\
&\qquad + \delta_E(S', Q^\infty[y'\,..\,z'))\\
&\quad= \delta_E(SS', {}^*Q^*),
\end{aligned}$$

which implies $\delta_E(SS', {}^*Q^*) = \delta_E(SS', \mathrm{rot}^{-x}(Q)^*)$.

Using a symmetric proof, we obtain the claim for $\delta_E(S'S, {}^*Q^*)$ and $\delta_E(S'S, {}^*\mathrm{rot}^{-y}(Q))$. ⌟

⌜ LEMMA 6.5 (`Analyze(`P`, `k`)`: IMPLEMENTATION OF LEMMA 5.19). *Let P denote a string of length m and let $k \le m$ denote a positive integer. Then, there is an algorithm that computes one of the following:*

1 *$2k$ disjoint breaks $B_1, \dots, B_{2k} \preccurlyeq P$, that each satisfy $\mathrm{per}(B_i) > m/128k$ and $|B_i| = \lfloor m/8k \rfloor$;*
2 *disjoint repetitive regions $R_1, \dots, R_r \preccurlyeq P$ of total length $\sum_{i=1}^{r} |R_i| \ge 3/8 \cdot m$ such that each region R_i satisfies $|R_i| \ge m/8k$ and is constructed along with a primitive approximate period Q_i such that $|Q_i| \le m/128k$ and $\delta_E(R_i, {}^*Q_i^*) = \lceil 8k/m \cdot |R_i| \rceil$; or*
3 *a primitive approximate period Q of P that satisfies $|Q| \le m/128k$ and $\delta_E(P, {}^*Q^*) < 8k$.*

The algorithm uses $O(k^2)$ time plus $O(k^2)$ `PILLAR` operations.

▬ PROOF. Similarly to the Hamming distance case, our implementation follows Algorithm 5.1 from the proof of Lemma 5.19.

Recall that P is processed from left to right and split into breaks and repetitive regions. In each iteration, the algorithm first considers a fragment of length $\lfloor m/8k \rfloor$. This fragment either becomes the next break (if its shortest period is long enough) or is extended to the right to a repetitive region (otherwise). Having constructed

■ ALGORITHM **6.2** Analyzing the pattern: A PILLAR model implementation of Algorithm 5.1.

```
Analyze($P, k$)
    $j \leftarrow 0; r \leftarrow 1; b \leftarrow 1;$
    while true do
        $j' \leftarrow j + \lfloor m/8k \rfloor;$
        if Period($P[j\,..\,j')$) $> m/128k$ then
            $B_b \leftarrow P[j\,..\,j');$
            if $b = 2k$ then return breaks $B_1, \dots, B_{2k}$;
            $b \leftarrow b + 1; j \leftarrow j';$
        else
            $q \leftarrow$ Period($P[j\,..\,j')$); $Q_r \leftarrow P[j\,..\,j+q);$
            generator G $\leftarrow$ EditGen($P[j\,..\,m), Q_r$);
            $\delta \leftarrow 0;$
            while $\delta < 8k/m \cdot (j' - j)$ and $j' \leq m$ do
                $(\pi, \pi') \leftarrow$ Next(G);
                $j' \leftarrow j + \pi + 1; \delta \leftarrow \delta + 1;$
            if $j' \leq m$ then
                $R_r \leftarrow P[j\,..\,j');$
                if $\sum_{i=1}^{r} |R_i| \geq 3/8 \cdot m$ then
                    return repetitive regions $R_1, \dots, R_r$ with periods $Q_1, \dots, Q_r$;
                $r \leftarrow r + 1; j \leftarrow j';$
            else
                $Q \leftarrow Q_r;$
                generator G' $\leftarrow$ EditGen$^R$($P$, $\mathrm{rot}^{-\pi'}(Q)$);
                $j'' = m; \delta \leftarrow 0;$
                while $(j'' \geq j$ or $\delta < 8k/m \cdot (m - j''))$ and $j'' \geq 0$ do
                    $(\pi, \_) \leftarrow$ Next(G');
                    $j'' \leftarrow m - \pi - 1; \delta \leftarrow \delta + 1;$
                if $j'' \geq 0$ then return repetitive region $P[j''\,..\,m)$ with period $Q$;
                else return approximate period $Q$;
```

sufficiently many breaks or repetitive regions of sufficiently large total length, the algorithm stops. Processing the string P in this manner guarantees disjointness of breaks and repetitive regions. As in the proof of Lemma 5.19, a slightly different approach is needed if the algorithm encounters the end of P while growing a repetitive region. If this happens, the region is also extended to the left. This way, the algorithm either obtains a single repetitive region (which is not necessarily disjoint with the previously created ones, so it is returned on its own) or learns that the whole string P is approximately periodic.

Next, we fill in missing details of the implementation of the previous steps in the `PILLAR` model. To that end, first note that the `PILLAR` model includes a `Period` operation of checking if the period of a string S satisfies $\operatorname{per}(S) \leq |S|/2$; computing $\operatorname{per}(S)$ in case of a positive answer. Since our threshold $m/128k$ satisfies $\lfloor m/128k \rfloor \leq \lfloor m/8k \rfloor/2$, no specific work is required to obtain the period of an unprocessed fragment of $\lfloor m/8k \rfloor$ characters of P.

To compute a repetitive region starting at a fragment $P[j\,..\,j')$ with string period Q, we use a generator $\mathsf{G} = \texttt{EditGen}(P[j\,..\,m), Q)$ from Lemma 6.2: for subsequent values $\delta \geq 1$, we find the shortest prefix P'_δ of $P[j\,..\,m)$ such that $\delta_E(P'_\delta, Q^*) = \delta$, until no such prefix exists or $\delta \geq 8k/m \cdot |P'_\delta|$. This is possible because the $(\delta-1)$-st call to $\texttt{Next}(\mathsf{G})$ returns the length π of the longest prefix of $P[j\,..\,m)$ with $\delta_E(P[j\,..\,j+\pi), Q^*) < \delta$. If $\delta \geq 8k/m \cdot |P'_\delta|$, then we have identified a repetitive region P'_δ. Otherwise, we reach $\pi = m - j$ and retrieve π' such that $\delta_E(P[j\,..\,m), Q^*) = \delta_E(P[j\,..\,m), Q^\infty[0\,..\,\pi'))$ from the last call to $\texttt{Next}(\mathsf{G})$. In this case, we similarly use a generator $\mathsf{G}' = \texttt{EditGen}^R(P, \operatorname{rot}^{-\pi'}(Q))$ from Lemma 6.2: For subsequent values $\delta \geq 1$, we find the shortest suffix P''_δ of P such that $\delta_E(P''_\delta, {}^*\operatorname{rot}^{-\pi'}(Q)) = \delta$, until no such suffix exists or we have $|P''_\delta| \geq |P[j\,..\,m)|$ and $\delta \geq 8k/m \cdot |P''_\delta|$. Again, this is possible because the $(\delta-1)$-st call to $\texttt{Next}(\mathsf{G}')$ returns the length π of the longest suffix of P with $\delta_E(P[m-\pi\,..\,m), {}^*\operatorname{rot}^{-\pi'}(Q)) < \delta$. If we reach $\pi = m$, then we return Q as an approximate period of P; otherwise, we return the final suffix P''_δ as a long repetitive region. Consider Algorithm 6.2 for implementation details.

For the correctness, since our algorithm follows the proof of Lemma 5.19, we need to show only that our method of finding repetitive regions correctly implements the corresponding steps in Algorithm 5.1.

In a first step, we inductively prove that each considered prefix P'_δ of $P[j\,..\,m)$ satisfies $\delta_E(P'_\delta, {}^*Q^*) = \delta \le \lceil 8k/m \cdot |P'_\delta| \rceil$. The case of $\delta = 1$ is easy since $\delta_E(P'_1, {}^*Q^*) \le \delta_E(P'_1, Q^*) = 1$, since $\delta_E(P'_1, {}^*Q^*) = 0$ would imply $\delta_E(P'_1, Q^*) = 0$ because Q is a prefix of P'_1, and since $8k/m \cdot |P'_1| > 0$ due to $|P'_1| > 0$. Next, we prove that the claim holds for $\delta + 1$ assuming that it holds for δ. The inductive assumption guarantees $\delta_E(P'_\delta, {}^*Q^*) = \delta_E(P'_\delta, Q^*) = \delta$. Since the algorithm proceeded to the next step, we have $\delta < 8k/m \cdot |P'_\delta|$. In particular, $|P'_\delta| \ge (2\delta + 1) \cdot m/128k \ge (2\delta + 1)|Q|$, so we can apply Lemma 6.4 to P'_δ. If $\delta_E(P'_{\delta+1}, {}^*Q^*) \le \delta$, then Lemma 6.4 yields $\delta_E(P'_{\delta+1}, Q^*) = \delta_E(P'_{\delta+1}, {}^*Q^*) \le \delta$, which contradicts the definition of $P'_{\delta+1}$. Hence, $\delta_E(P'_{\delta+1}, {}^*Q^*) \ge \delta + 1$. Due to $\delta_E(P'_{\delta+1}, Q^*) = \delta + 1$, we have $\delta_E(P'_{\delta+1}, {}^*Q^*) = \delta + 1$. Moreover, $\lceil 8k/m \cdot |P'_{\delta+1}| \rceil \ge \lceil 8k/m \cdot |P'_\delta| \rceil > \delta$ guarantees $\lceil 8k/m \cdot |P'_{\delta+1}| \rceil \ge \delta + 1$, which completes the proof.

In particular, if we encounter a prefix P'_δ that satisfies $\delta \ge 8k/m \cdot |P'_\delta|$, then $\delta_E(P'_\delta, {}^*Q^*) = \lceil 8k/m \cdot |P'_\delta| \rceil$. If no such prefix P'_δ exists, then we have $\delta_E(R, {}^*Q^*) < 8k/m \cdot |R|$ for each non-empty prefix of $P[j\,..\,m)$ (because $R = P'_\delta$ is the shortest prefix R of $P[j\,..\,m)$ with $\delta_E(R, {}^*Q^*) = \delta$). Thus, Line 9 of Algorithm 5.1 is correct.

In the following, we assume that no such prefix R exists. In particular, we have $\delta_E(P[j\,..\,m), {}^*Q^*) < 8k/m \cdot |P[j\,..\,m)|$. Then, the last call to `Next(G)` resulted in $(m - j, \pi')$ with $\delta_E(P[j\,..\,m), Q^*) = \delta_E(P[j\,..\,m), Q^\infty[0\,..\,\pi'))$. Since P'_δ with $\delta = \delta_E(P[j\,..\,m), Q^*)$ satisfies $\delta_E(P'_\delta, {}^*Q^*) = \delta_E(P'_\delta, Q^*) = \delta$, we have

$$\begin{aligned}&\delta_E(P[j\,..\,m), {}^*Q^*)\\ &\quad = \delta_E(P[j\,..\,m], Q^*) = \delta_E(P[j\,..\,m), {}^*\mathrm{rot}^{-\pi'}(Q)) = \delta.\end{aligned}$$

We inductively prove that each considered suffix of P''_δ of P with $|P''_\delta| > j - m$ satisfies $\delta_E(P''_\delta, {}^*Q^*) = \delta \le \lceil 8k/m \cdot |P''_\delta| \rceil$. Let us prove that this claim is true for $\delta + 1$ assuming that is it true for δ (unless $\delta = 0$, when there is no assumption). If $\delta < \delta_E(P[j\,..\,m), {}^*Q^*)$, then $|P''_{\delta+1}| \le j - m$ and the claim is void, so we consider only $\delta \ge \delta_E(P[j\,..\,m), {}^*Q^*)$. If we have

$$\delta > \delta_E(P[j\,..\,m), {}^*Q^*) = \delta_E(P[j\,..\,m), {}^*\mathrm{rot}^{-\pi'}(Q)),$$

then $|P''_\delta| > j - m$ and the inductive assumption guarantees

$$\delta_E(R, {}^*Q^*) = \delta_E(R, {}^*\mathrm{rot}^{-\pi'}(Q)) = \delta$$

for $R = P''_\delta$. Otherwise, we have

$$\delta = \delta_E(P[j\,..\,m), {}^*Q^*) = \delta_E(P[j\,..\,m), {}^*\mathrm{rot}^{-\pi'}(Q)),$$

in which case $\delta_E(R, {}^*Q^*) = \delta_E(R, {}^*\mathrm{rot}^{-\pi'}(Q)) = \delta$ holds for $R = P[j\,..\,m)$. In either case, we also have $\delta < 8k/m \cdot |R|$, and hence $|R| \ge (2\delta + 1) \cdot m/128k \ge (2\delta + 1)|Q|$. Therefore, we can apply Lemma 6.4 to R. If $\delta_E(P''_{\delta+1}, {}^*Q^*) \le \delta$, then Lemma 6.4 yields

$$\delta_E(P''_{\delta+1}, {}^*\mathrm{rot}^{-\pi'}(Q)) = \delta_E(P''_{\delta+1}, {}^*Q^*) \le \delta,$$

which contradicts the definition of $P''_{\delta+1}$. Hence, $\delta_E(P''_{\delta+1}, {}^*Q^*) \ge \delta + 1$. Due to $\delta_E(P''_{\delta+1}, {}^*\mathrm{rot}^{-\pi'}(Q)) = \delta + 1$, we have $\delta_E(P''_{\delta+1}, {}^*Q^*) = \delta + 1$. Moreover, $\lceil 8k/m \cdot |P''_{\delta+1}|\rceil \ge \lceil 8k/m \cdot |R|\rceil > \delta$ guarantees $\lceil 8k/m \cdot |P''_{\delta+1}|\rceil \ge \delta + 1$, which completes the proof.

In particular, if we encounter a suffix P''_δ that satisfies $|P''_\delta| > m - j$ and $\delta \ge 8k/m \cdot |P''_\delta|$, then $\delta_E(P''_\delta, {}^*Q^*) = \lceil 8k/m \cdot |P''_\delta|\rceil$. On the other hand, if no such suffix P''_δ exists, then $\delta_E(R, {}^*Q^*) < 8k/m \cdot |R|$ holds for each suffix R of P of length $|R| > m - j$ (because $R = P'_\delta$ is the shortest suffix R of P with $\delta_E(R, {}^*Q^*) = \delta$ assuming $\delta > \delta_E(P[j\,..\,m), {}^*Q^*)$). Thus, Line 15 of Algorithm 5.1 is also implemented correctly.

For the running time analysis, observe that each iteration of the outer while loop processes at least $\lfloor m/8k \rfloor$ characters of P, so there are at most $O(k)$ iterations of the outer while loop. In each iteration, we perform one `Period` operation, a constant number of `Access` operations, and at most $8k/m \cdot (j' - j)$ calls to the generator `EditGen`.

Algorithm 6.3 Finding a witness $Q^\infty[x\,..\,y)$ for $\delta_E(S, {}^*Q^*) \le k$.

```
FindAWitness($k$, $Q$, $S$)
    // Compute ``correct'' rotation(s) of $Q$
    if $|Q| \le 3k+1$ then $J \leftarrow [0\,..\,|Q|)$;
    else
        multi-set $R \leftarrow \{\}$;
        for $i \leftarrow 0$ to $2k$ do
            $R \leftarrow R \cup$ Rotations($S[i|Q|\,..\,(i+1)|Q|)$, $Q$);
        $I \leftarrow \bigcup\{[p\,..\,p+k] : p \in \mathbb{Z}$ and $[p\,..\,p+k]$ contains at least $k+1$ elements of $R\}$;
        Let $J \subseteq [0\,..\,2|Q|)$ denote a shortest interval that satisfies $I \bmod |Q| \subseteq J \bmod |Q|$;
    // Compute the start position of the witness
    $Occ \leftarrow$ Verify($S$, $Q^\infty$, $k$, $J$);
    if $Occ = \emptyset$ then return $\perp$;
    Let $(x, k') \in Occ$ be an arbitrary element minimizing $k'$;
    // Compute the end position of the witness
    generator G $\leftarrow$ EditGen($S$, $\mathrm{rot}^{-x}(Q)$);
    for $i \leftarrow 1$ to $k'$ do Next(G);
    $(\lambda, \lambda') \leftarrow$ Next(G);
    return $Q^\infty[x\,..\,x+\lambda')$;
```

These calls use $O((8k/m\cdot(j'-j))^2)$ time in the PILLAR model, which is $O(k^2)$ in total across all iterations (since the function $x \mapsto x^2$ is convex). Similarly, we bound the running time of the calls to the generator EditGen^R: As we find at most $8k/m \cdot m = 8k$ errors, EditGen^R uses at most $O(k^2)$ time. Overall, Algorithm 6.2 thus uses $O(k^2)$ time in the PILLAR model. ◻

6.3 Computing Occurrences in the Periodic Case

We start this section with a subroutine to compute a *witness* that a string S has a small edit distance to a string Q^∞.

LEMMA 6.6 (`FindAWitness(`k`, `Q`, `S`)`). *Let k denote a positive integer, let S denote a string, and let Q denote a primitive string that satisfies $|S| \geq (2k+1)|Q|$ or $|Q| \leq 3k+1$.*

Then, we can be compute a witness $Q^\infty[x\,..\,y)$ *such that*

$$\delta_E(S, Q^\infty[x\,..\,y)) = \delta_E(S, {}^*Q^*) \leq k,$$

or report that $\delta_E(S, {}^*Q^*) > k$. The algorithm takes $O(k^2)$ time in the `PILLAR` *model.*

PROOF. For a set $A \subseteq \mathbb{Z}$ and an integer $p > 0$, we define $A \bmod p := \{a \bmod p \mid a \in A\}$. We first compute a (short) interval J such that $\mathrm{Occ}^E_k(S, Q^\infty) \bmod |Q| \subseteq J \bmod |Q|$. If $|Q| \leq 3k+1$, then we simply set $J = [0\,..\,|Q|)$. Otherwise, we proceed similarly as in the Hamming distance setting (Lemma 4.6), where we computed a majority string of the first $2k+1$ length-$|Q|$ subsequent fragments $S_1, \ldots, S_{2k}$ of S. However, now we need to accommodate for insertions and deletions of a k-error occurrence. Hence, we first compute an auxiliary set I defined as the union of intervals $[p\,..\,p+k]$ such that for at least $k+1$ fragments S_i of S, we have $Q = \mathrm{rot}^j(S_i)$ for $j \in [p\,..\,p+k]$. Finally, we set $J \subseteq [0\,..\,2|Q|)$ to be a shortest interval satisfying $I \bmod |Q| \subseteq J \bmod |Q|$. Here, $J \bmod |Q|$ can be interpreted as a shortest cyclic interval (modulo $|Q|$) containing $I \bmod |Q|$.

Having computed the set J, we use `Verify` from Lemma 6.3 to determine at which starting position x in J we have an occurrence with the fewest number of errors (or to report that the number of errors is greater than k everywhere). Finally, we use an `EditGen` from Lemma 6.2 to compute the ending position y of the occurrence of S as a prefix of $Q^\infty[x\,..\,)$. Consider Algorithm 6.3 for a pseudo-code implementation.

The correctness is based on the aforementioned characterization of J.

Claim 6.7. *The interval J satisfies*

$$\mathrm{Occ}_k^E(S, Q^\infty) \bmod |Q| \subseteq J \bmod |Q|.$$

Proof. The claim trivially holds if $|Q| \le 3k+1$, so we assume that $|Q| > 3k+1$. For every $i \in [\,0\,..\,2k\,]$, define $S_i := S[\,i\,|Q|\,..\,(i+1)\,|Q|\,)$. Consider an optimum alignment between S and its k-error occurrence $Q^\infty[\,x\,..\,y\,)$, and let $Q_i = Q^\infty[\,x_i\,..\,x_{i+1}\,)$ denote the fragment aligned to S_i. Consider the multi-set $R := \bigcup_i \texttt{Rotations}(S_i, Q)$. Next, consider the values $\delta_i := x_i - i|Q|$ for $i \in [\,0\,..\,2k\,]$. We have $\delta_0 = x$, and $\delta_{i+1} = \delta_i + |Q_i| - |S_i|$ for $i > 0$. Since

$$\sum_{i=0}^{2k} \big||Q_i| - |S_i|\big| \le \sum_{i=0}^{2k} \delta_E(Q_i, S_i) \le k,$$

all values δ_i belong to an interval of the form $[p\,..\,p+k]$ for some integer p. Moreover, note that $Q_i = S_i$ holds for at least $k+1$ fragments Q_i; these fragments satisfy $Q = \mathrm{rot}^{\delta_i}(S_i)$ and thus contribute δ_i to R. We conclude that there is an interval $[p\,..\,p+k]$ containing x and at least $k+1$ elements of R. Consequently, we have $x \in I$. By definition of J, this yields $x \bmod |Q| \in J \bmod |Q|$.

Now, let $Q^\infty[\,x\,..\,y\,)$ denote a witness that satisfies

$$\delta_E(S, {}^*Q^*) = \delta_E(S, Q^\infty[\,x\,..\,y\,)).$$

By Claim 6.7, there is a matching fragment

$$Q^\infty[\,x'\,..\,y'\,) = Q^\infty[\,x\,..\,y\,)$$

starting at $x' \in J$. Thus, we may assume without loss of generality that $x \in J$. As we verify all possible starting positions in J using `Verify` from Lemma 6.3, we correctly compute the starting position x of a witness occurrence of S in Q^∞. Further, as we use an `EditGen` from Lemma 6.2, we also compute the corresponding ending position correctly.

As for the running time, we prove the following characterization of J.

Claim 6.8. *The interval J satisfies $|J| \le 3k+1$.*

Proof. The claim trivially holds if $|Q| \le 3k+1$, so we assume that $|Q| > 3k+1$. Recall that the multiset R in Algorithm 6.3 is the union of $2k+1$ sets $\{j \in \mathbb{Z} : \mathrm{rot}^j(S_i) = Q\}$. As the string Q is primitive, R is the union of at most $2k+1$ infinite arithmetic progressions with difference $|Q|$. In particular, if $[\,p\,..\,p{+}k\,]$ and $[\,p'\,..\,p'{+}k\,]$ contain at least $k{+}1$ elements of R each, then $([\,p\,..\,p{+}k\,] \bmod |Q|) \cap ([\,p'\,..\,p'{+}k\,] \bmod |Q|) \ne \emptyset$, and thus $[\,p'\,..\,p'+k\,] \bmod |Q| \subseteq [\,p-k\,..\,p+2k\,] \bmod |Q|$. Since J is the union of such intervals $[\,p'\,..\,p'+k\,]$, we have $J \bmod |Q| \subseteq [\,p-k\,..\,p+2k\,] \bmod |Q|$. By definition of I, we conclude that $|I| \le |[\,p-k\,..\,p+2k\,]| = 3k+1$.

Now, observe that computing the multiset R[26] takes $O(k)$ time in the `PILLAR` model; computing the sets I and J can be done in $O(k \log\log k)$ time by sorting R (restricted to $[\,0\,..\,|Q|)$) and a subsequent cyclic scan over R. Further, by Claim 6.8, we call `Verify` on an interval of length $O(k)$; hence the call to `Verify` takes $O(k^2)$ time in the `PILLAR` model. Finally, as we query the `EditGen` for up to k errors, the last step of the algorithm takes $O(k^2)$ time in the `PILLAR` model as well. Hence in total, `FindAWitness` runs in $O(k^2)$ time in the `PILLAR` model, completing the proof.

26. We represent R as the union of infinite arithmetic progressions modulo $|Q|$.

Lemma 6.9 (`FindRelevantFragment(`P`,` T`,` k`,` d`,` Q`)`). *Let P denote a pattern of length m, let T denote a text of length n, and let $0 \le k \le m$ denote a threshold such that $n < {}^3\!/_2\, m + k$. Further, let $d \ge 2k$ denote a positive integer and let Q denote a primitive string that satisfies $|Q| \le m/8d$ and $\delta_E(P, {}^*Q^*) \le d$.*

Then, there is an algorithm that computes a fragment $T' = T[\,\ell\,..\,r\,)$ and an integer range I such that $\delta_E(T', {}^*Q^*) \le 3d$, $|\mathrm{Occ}^E_k(P,T)| = |\mathrm{Occ}^E_k(P,T')|$, $|I| \le 6d+1$, and $\mathrm{Occ}^E_k(P,T') \bmod |Q| \subseteq I \bmod |Q|$. The algorithm runs in $O(d^2)$ time in the `PILLAR` model.

Algorithm 6.4 A PILLAR algorithm computing a *relevant* fragment T' of T containing all k-error occurrences of P in T, and an interval I such that $\mathrm{Occ}_k^E(P, T') \bmod |Q| \subseteq I \bmod |Q|$.

```
FindRelevantFragment($P, T, k, d, Q$)
    $Q^\infty[x\,..\,y) \leftarrow$ FindAWitness($d, Q, P$);
    $Q' \leftarrow$ FindAWitness($\lfloor 3/2\, d \rfloor, Q, T[n-m+k\,..\,m-k)$);
    if $Q' = \perp$ then return $(\varepsilon, \emptyset)$;
    $Q^\infty[x'\,..\,y') \leftarrow Q'$;
    // Extend $Q'$ as much as possible to the right.
    generator G $\leftarrow$ EditGen($T[n-m+k\,..\,n)$, $\mathrm{rot}^{-x'}(Q)$);
    for $i \leftarrow 0$ to $\lfloor 3/2\, d \rfloor$ do $(\lambda, \_) \leftarrow$ Next(G);
    $r \leftarrow n-m+k+\lambda$;
    // Extend $Q'$ as much as possible to the left.
    generator G' $\leftarrow$ EditGen$^R$($T[0\,..\,m-k)$, $\mathrm{rot}^{-y'}(Q)$);
    for $i \leftarrow 0$ to $\lfloor 3/2\, d \rfloor$ do $(\lambda', \_) \leftarrow$ Next(G');
    $\ell \leftarrow m-k-\lambda'$;
    return $(T[\ell\,..\,r), [n-m+k-\ell+x-x'-3d\,..\,n-m+k-\ell+x-x'+3d])$;
```

Proof. We first call FindAWitness from Lemma 4.6 twice in order to find a fragment $Q^\infty[x\,..\,y)$ such that $\delta_E(P, Q^\infty[x\,..\,y)) = \delta_E(P, {}^*Q^*) \le d$ and a fragment $Q^\infty[x'\,..\,y')$ such that

$$\begin{aligned}\delta_E(T[n-m+k\,..\,m-k), Q^\infty[x'\,..\,y'))\\ = \delta_E(T[n-m+k\,..\,m-k), {}^*Q^*) \le 3/2\, d.\end{aligned}$$

If the fragment $Q^\infty[x'\,..\,y')$ does not exist, we return the empty string $T' = \varepsilon$ and the empty interval $I = \emptyset$. Otherwise, we proceed by computing the rightmost position r such that $\delta_E(T[n - m + k\,..\,r), \mathrm{rot}^{-x'}(Q)^*) \le 3/2\, d$ and the leftmost position ℓ such that $\delta_E(T[\ell\,..\,m-k), {}^*\mathrm{rot}^{-y'}(Q)) \le 3/2\, d$. That is, we "extend" the fragment found in the text as much as possible. Afterwards, we return the fragment $T' = T[\ell\,..\,r)$ and the interval

$$I = [\,n - m + k - \ell + x - x' - 3d\,..\,n - m + k - \ell + x - x' + 3d\,].$$

Consider Algorithm 6.4 for implementation details.

For the correctness, note that the due to the assumption on Q, the first call to `FindAWitness` is valid and indeed returns a witness $Q^\infty[x\,..\,y)$. Next, consider a k-error occurrence $T[p\,..\,q)$ of P. By triangle inequality (Lemma 1.3), we have

$$\delta_E(T[p\,..\,q), Q^\infty[x\,..\,y)) \le k + \delta_E(P, Q^\infty[x\,..\,y))) \le {}^3\!/_2\, d.$$

Due to $|T[p\,..\,q)| \ge m-k$, we have $q \ge m-k$ and $p \le n-m+k$, which yields

$$\delta_E(T[n-m+k\,..\,m-k), Q^\infty[x''\,..\,y'')) \le {}^3\!/_2\, d$$

for some integers x'', y'' with $x \le x'' \le y'' \le y$. Moreover, as in the proof of Lemma 5.2, we have $|T[n-m+k\,..\,m-k)| = 2(m-k) - n \ge (3d+1)|Q|$ or $|Q| = 1$, so the second call to `FindAWitness` is valid. Thus, if the call returns $\perp$, then $\mathrm{Occ}_k^E(P,T) = \emptyset$.

Now assume that the call returned a witness $Q^\infty[x'\,..\,y')$. Next, we apply Lemma 6.4 for $S = T[n-m+k\,..\,m-k)$. This is indeed possible because $|S| \ge (3d+1)|Q|$ or $|Q| = 1$. Due to

$$\delta_E(T[n{-}m{+}k\,..\,q), {}^*Q^*) \le \delta_E(T[n{-}m{+}k\,..\,q), Q^\infty[x''\,..\,y)) \le {}^3\!/_2\, d,$$

Lemma 6.4 yields

$$\delta_E(T[n{-}m{+}k\,..\,q), \mathrm{rot}^{-x'}(Q)^*) = \delta_E(T[n{-}m{+}k\,..\,q), {}^*Q^*) \le {}^3\!/_2\, d.$$

Hence, we have $q \le r$, as r is computed correctly using `EditGen` from Lemma 6.2. Symmetrically, due to

$$\delta_E(T[p\,..\,m+k), {}^*Q^*) \le \delta_E(T[p\,..\,m+k), Q^\infty[x\,..\,y'')) \le {}^3\!/_2\, d,$$

Lemma 6.4 yields

$$\delta_E(T[\,p\,..\,m-k\,), {}^*\mathrm{rot}^{-y'}(Q)) = \delta_E(T[\,p\,..\,m+k\,), {}^*Q^*) \leq {}^3\!/_2\, d.$$

Hence, we have $p \geq \ell$, as ℓ is computed correctly using $\texttt{EditGen}^R$ from Lemma 6.2. We conclude that $T[\,p\,..\,q\,)$ is contained in $T' = T[\,\ell\,..\,r\,)$. Since $T[\,p\,..\,q\,)$ was an arbitrary k-error occurrence of P in T, this implies $|\mathrm{Occ}^E_k(P, T')| = |\mathrm{Occ}^E_k(P, T)|$.

Now consider the fragment $T[\,p\,..\,m-k\,)$ whose prefix $T[\,p\,..\,m-k\,)$ satisfies

$$\delta_E(T[\,p\,..\,m-k\,), Q^\infty[\,x\,..\,y''\,)) \leq {}^3\!/_2\, d$$

and whose suffix $T[\,n-m+k\,..\,m-k\,)$ satisfies

$$\delta_E(T[\,n-m+k\,..\,m-k\,), Q^\infty[\,x'\,..\,y'\,)) \leq {}^3\!/_2\, d.$$

We apply Lemma 5.4 to $T[\,p\,..\,m-k\,)$; this is indeed possible because $|T[\,n-m+k\,..\,m-k\,)| \geq (3d+1)|Q|$ or $|Q| = 1$. Lemma 5.4 now implies $(n-m+k-p+x-x'+3d) \bmod |Q| \leq 6d$. Hence, we have

$$p \bmod |Q| \in [\,n-m+k+x-x'-3d\,..\,n-m+k+x-x'+3d\,] \bmod |Q|.$$

As $T[\,p\,..\,q\,)$ is an arbitrary k-error occurrence of P in T, we have

$$\mathrm{Occ}^E_k(P, T') \bmod |Q| \subseteq I \bmod |Q|.$$

Further, $|I| \leq 6d+1$ holds trivially by construction.

Consider the fragment T, whose prefix $T[\,\ell\,..\,m-k\,)$ satisfies

$$\delta_E(T[\,\ell\,..\,m-k\,), {}^*Q^*) \leq {}^3\!/_2\, d$$

and whose suffix $T[\,n-m+k\,..\,r\,)$ satisfies

$$\delta_E(T[\,n-m+k\,..\,r\,), {}^*Q^*) \leq {}^3\!/_2\, d.$$

Again, we apply Lemma 5.4 to T'; this is indeed possible because $|T[\,n-m+k\,..\,m-k\,)| \ge (3d+1)|Q|$ or $|Q| = 1$. Lemma 5.4 now implies $\delta_E(T', {}^*Q^*) \le 3d$, as claimed.

As for the running time in the `PILLAR` model, observe that the calls to `FindAWitness` use $O(d^2)$ time; the same is true for the usage of `EditGen` and `EditGen`^{R}. Thus, the algorithm takes $O(d^2)$ time in the `PILLAR` model.

Lemma 6.10 (`Locked(`S`,` Q`,` d`,` k`)`: Impl. of Lemma 5.11). *Let S denote a string, let Q denote a primitive string, let d denote a positive integer such that $\delta_E(S, {}^*Q^*) \le d$ and $|S| \ge (2d+1)|Q|$, and let k denote a non-negative integer.*

Then, there is an algorithm that computes disjoint locked fragments $L_1, \dots, L_\ell \preceq S$ such that L_1 is a k-locked prefix of S, L_ℓ is a suffix of S, and $\delta_E(L_i, {}^*Q^*) > 0$ for $1 < i < \ell$. Moreover, we have

$$\delta_E(S, {}^*Q^*) = \sum_{i=1}^{\ell} \delta_E(L_i, {}^*Q^*) \quad \textit{and} \quad \sum_{i=1}^{\ell} |L_i| \le (5|Q|+1)d + 2(k+1)|Q|.$$

The algorithm takes $O(d^2 + k)$ time in the `PILLAR` model.

Proof. We implement the proof of Lemma 5.11. We start with an overview of the algorithm; see also Algorithm 6.5 for implementation details.

First, we construct an optimal alignment between S and a substring of Q^∞. For this, we first use `FindAWitness` of Lemma 6.6 to obtain positions $x \le y$ such that $\delta_E(S, {}^*Q^*) = \delta_E(S, Q^\infty[\,x\,..\,y\,))$. Then, we apply a generator `EditGen` $(S, \mathrm{rot}^{-x}(Q))$ of Lemma 6.2 to construct an optimal alignment A between S and $Q^\infty[\,x\,..\,x+\pi'\,)$ for some integer $\pi' \ge 0$ (note that we cannot guarantee $y = x + \pi'$).

Then, based on the alignment A, we construct a decomposition $S = S_0^{(0)} \cdots S_{s^{(0)}}^{(0)}$ such that $S_i^{(0)}$ is aligned with

$$Q_i^{(0)} := Q^\infty[\,max(x, (|Q|-1)\lceil x/|Q| \rceil)\,..\,min(|Q|\lceil x/|Q| \rceil, x+\pi)\,)$$

Algorithm 6.5 Computing locked fragments in a string S.

```
Locked($S, Q, d, k$)
    $Q^\infty[x\,..\,y) \leftarrow$ FindAWitness($d, Q, S$);
    generator G $\leftarrow$ EditGen($S, \mathrm{rot}^{-x}(Q)$);
    do $(\pi, \pi') \leftarrow$ Next(G) while $\pi < |S|$;
    $A \leftarrow$ Alignment(G);
    $\ell_Q \leftarrow x$; $\ell_S \leftarrow 0$;
    $r_Q \leftarrow |Q|\lceil x/|Q|\rceil$; $r_S \leftarrow r_Q - \ell_Q$;
    $\Delta \leftarrow k + 1$;
    queue $F$;
    foreach $(s, q) \in A \cup (\pi, \pi')$ do
        if $s = \perp$ then $s \leftarrow q + x + r_S - r_Q - 1$;
        if $q = \perp$ then $q \leftarrow s - x + r_Q - r_S - 1$;
        if $x + q \geq r_Q$ then
            push($F$, $(S[\ell_S\,..\,r_S), \Delta)$);
            $\ell_Q \leftarrow |Q|\lfloor (x+q)/|Q| \rfloor$; $\ell_S \leftarrow r_S + \ell_Q - r_Q$;
            $r_Q \leftarrow \ell_Q + |Q|$; $r_S \leftarrow r_S + |Q|$;
            $\Delta \leftarrow 0$;
        $r_S \leftarrow r_Q - x + s - q$;
        if $(s, q) \neq (\pi, \pi')$ then $\Delta \leftarrow \Delta + 1$;
    push($F$, $(S[\ell_S\,..\,|S|), \Delta)$);
    stack $L$;
    while $F$ is not empty do
        $(S[\ell\,..\,r), \Delta) \leftarrow$ front($F$); pop($F$);
        while true do
            if top($L$) $= S[\ell'\,..\,r')$ and $r' = \ell$ then
                $\ell \leftarrow \ell'$;
                pop($L$);
            else if front($F$) $= (S[\ell'\,..\,r'), \Delta')$ and $\ell' = r$ then
                $r \leftarrow r'$;
                $\Delta \leftarrow \Delta + \Delta'$;
                pop($F$);
            else if $\Delta > 0$ then
                $\ell \leftarrow max(0, \ell - |Q|)$;
                $r \leftarrow min(|S|, r + |Q|)$;
                $\Delta \leftarrow \Delta - 1$;
            else
                push($L$, $S[\ell\,..\,r)$);
                break;
    return $L$;
```

in the decomposition A, and a sequence $\Delta_i^{(0)}$ such that we have $\Delta_i^{(0)} = \delta_E(S_i^{(0)}, Q_i^{(0)})$ for $i > 0$ and $\Delta_0^{(0)} = \delta_E(S_0^{(0)}, Q_0^{(0)}) + k + 1$. As this sequence might be long, we generate only *interesting* fragments $S_i^{(0)}$ and store them, along with the values $\Delta_i^{(0)}$, in a queue F in left-to-right order. (Recall that $S_i^{(t)}$ is interesting if $i = 0$, $i = s^{(t)}$, $S_i^{(t)} \neq Q$, or $\Delta_i^{(t)} > 0$.)

We construct interesting fragments $S_i^{(0)}$ as follows.[27] We maintain a fragment $^\infty[\,\ell_Q \,..\, r_Q\,)$, interpreted as $Q_i^{(0)}$ for increasing values of i, a fragment $S[\,\ell_S \,..\, r_S\,)$, interpreted as a candidate for $S_i^{(0)}$, and an integer Δ, interpreted as $\Delta_i^{(0)}$. These values are initialized to $Q^\infty[\,x \,..\, |Q|\lceil x/|Q|\rceil\,)$, $S[\,0 \,..\, Q|\lceil x/|Q|\rceil - x\,)$, and $k+1$, respectively.

27. One might consider this process to be *slightly* tedious.

Next, we process pairs (s, q) corresponding to subsequent errors in the alignment A. The interpretation of the j-th pair (s, q) is that $S[\,0 \,..\, s\,)$ is aligned with $Q^\infty[\,x \,..\, x + q\,)$ with j errors so that the j-th error is a substitution of $S[\,s\,]$ into $Q^\infty[\,x + q\,]$, and insertion of $Q^\infty[\,x + q\,]$, or a deletion of $S[\,s\,]$.

We perform the first step of processing (s, q) only if the fragment $Q^\infty[\,x \,..\, x + q\,)$ is not (yet) contained in $Q^\infty[\,\ell_Q \,..\, r_Q\,)$. If this is not the case, then we push $S[\,\ell_S \,..\, r_S\,)$ with budget Δ to the queue F of interesting fragments, and we update the maintained data: The fragment $Q^\infty[\,\ell_Q \,..\, r_Q\,)$ is set to be the fragment of Q^∞ matching Q and containing $Q^\infty[\,x + q\,]$; between the previous and the current value of $Q^\infty[\,\ell_Q \,..\, r_Q\,)$, there are zero or more copies of Q aligned in A without error. Hence, we skip the same number of copies of Q in S[28] and set $S[\,\ell_S \,..\, r_S\,)$ to be the subsequent fragment of length $|Q|$. Finally, the budget Δ is reset to 0.

28. These are the uninteresting fragments $S_i^{(0)}$.

In the second step, we update r_S according to the type of the currently processed error: We increment r_S in case of deletion of $S[\,s\,]$ and we decrement r_S in case of insertion of $Q^\infty[\,x + q\,]$. This way, we guarantee that $|S(\,s \,..\, r_S\,)| = |Q^\infty(\,x + q \,..\, r_Q\,)|$, and that A aligns $S[\,\ell_S \,..\, r_S\,)$ with $Q^\infty[\,\ell_Q \,..\, r_Q\,)$ provided that we have already processed all errors involving $Q^\infty[\,\ell_Q \,..\, r_Q\,)$. Additionally, we increase Δ to acknowledge the currently processed error between $S[\,\ell_S \,..\, r_S\,)$ and $Q^\infty[\,\ell_Q \,..\, r_Q\,)$.

In a similar way, we process $(s, q) = (|S|, \pi')$, interpreting it as extra substitution. This time, however, we do not increase Δ (because this is a not a real error). Finally, we push $S[\,\ell_S \,..\, |S|\,) = S^{(0)}_{s^{(0)}}$ with budget Δ to the queue F.

In a second phase of the algorithm, we transform the decomposition $S = S^{(0)}_0 \cdots S^{(0)}_{s^{(0)}}$ and the sequence $\Delta^{(0)}_0 \cdots \Delta^{(0)}_{s^{(0)}}$ using the four types of merge operations described in the proof of Lemma 5.6.

We maintain an invariant that a stack L contains already processed interesting fragments, all with budget equal to 0, in left-to-right order (so that $\texttt{top}(L)$ represents the rightmost one), while F contains fragments that have not been processed yet (and may have positive budgets) also in the left-to-right order (so that $\texttt{front}(F)$ represents the leftmost one). Additionally, the currently processed fragment $S[\,\ell \,..\, r\,)$ is guaranteed to be to the right of all fragments in L and to the left of all fragments in F. The fragments in L, the fragment $S[\,\ell \,..\, r\,)$, and the fragments in F form the sequence of all interesting fragments in the current decomposition $S = S^{(t)}_0 \cdots S^{(t)}_{s^{(t)}}$.

In each iteration of the main loop, we pop the front fragment $S[\,\ell \,..\, r\,)$ with budget Δ from the queue F and exhaustively perform merge operations involving it: We first try applying a type-1 merge with the fragment to the left (which must be $\texttt{top}(L)$). If this is not possible, we type applying a type-1 merge with the fragment to the right (which must be $\texttt{front}(F)$). If also this is not possible, then $S[\,\ell \,..\, r\,)$ is surrounded by uninteresting fragments. In this case, we perform a type-2, type-3, or 4 merge provided that $\Delta > 0$. Otherwise, we push $S[\,\ell \,..\, r\,)$ to L and proceed to the next iteration.

Finally, the algorithm returns the sequence of (locked) fragments represented in the stack L.

The correctness of the algorithm follows from Lemma 5.11; no deep insight is needed to prove that our implementation indeed follows the procedure described in the proof of Lemma 5.6 and extended in the proof of Lemma 5.11.

For the running time, the initial call to `FindAWitness` and applying the generator G each take $O(d^2)$ time in the `PILLAR` model. As the alignment A is of size $|A| \leq d$, the **for** loop in Line 10 takes $O(d)$ time and generates $O(d)$ interesting locked fragments with total budget $O(d + k)$. Each iteration of the **while** loop in Line 24 decreases the number of interesting locked fragments or their total budget, so there are at most $O(d+k)$ iterations in total. Overall the algorithm runs in $O(d^2+k)$ time in the PILLAR model.

LEMMA 6.11 (SynchedMatches(P, T, I, d, d', k, Q)). *Let P denote a pattern of length m, let $0 \le k \le m$ denote a threshold, and let T denote a text of length $n \le {}^3/_2\, m + k$. Further, let I denote an integer range and let Q denote a primitive string that satisfies $\delta_E(P, {}^*Q^*) \le d$ and $\delta_E(T, {}^*Q^*) \le d'$.*

There is an algorithm that computes the set $\mathrm{Occ}^E_k(P,T) \cap (I + |Q|\mathbb{Z})$ as $O(|I|d'(d+k))$ arithmetic progressions. The algorithm takes $O(kd'(d+k)(k+|I|+d+d'))$ time in the PILLAR model.

PROOF. The algorithm resembles the proof of Lemma 5.2(4). Consult Algorithm 6.6 for pseudo-code; in the interest of readability we use $\mathrm{Occ}^E_k(P,T)$ instead of $\mathrm{Occ}^E_k(P,T) \cap (I+|Q|\mathbb{Z})$ in the pseudo-code. Note that if $|I| > |Q|$ we can replace I with $[\,0\,..\,Q\,)$, since in this case $I + |Q|\mathbb{Z} = \mathbb{Z} = [\,0\,..\,Q\,) + |Q|\mathbb{Z}$. Hence, we can assume that $|I| \le |Q|$.

First, using Algorithm 6.5, we obtain $\mathcal{L}^P :=$ Locked(P, Q, d, k) and $\mathcal{L}^T :=$ Locked($T, Q, d', 0$). We have $\ell^P := |\mathcal{L}^P| \le \delta_E(P, {}^*Q^*) + 2 \le d + 2$ and $\ell^T = |\mathcal{L}^T| \le \delta_E(T, {}^*Q^*) \le d' + 2$.

Then, for each of the $O(dd')$ pairs of locked fragments $L^P_i = P[\,\ell\,..\,r\,) \in \mathcal{L}^P$ and $L^T_j = T[\,\ell'\,..\,r'\,) \in \mathcal{L}^T$ we (implicitly) mark the positions in the interval $[\,\ell' - r - k\,..\,r' - \ell + k\,)$. We also mark all positions in $[\,n-m-k\,..\,n-m+k\,]$. We decompose the set of marked positions M into $O(dd')$ maximal ranges $J \subseteq M$. For each such maximal range J, for each maximal range $J' \subseteq J \cap (I+|Q|\mathbb{Z})$, we call Verify($P, T, k, J'$) and add its output to $\mathrm{Occ}^E_k(P,T) \cap (I+|Q|\mathbb{Z})$. This guarantees that we correctly compute all elements of the set $\mathrm{Occ}^E_k(P,T) \cap (I+|Q|\mathbb{Z}) \cap M$.

The decomposition of M into maximal ranges yields a decomposition of $[\,0\,..\,n-m+k\,) \setminus M$ into $O(dd')$ maximal ranges. For each such maximal range J, we rely on the characterization of Claim 5.17 in order to compute $\mathrm{Occ}^E_k(P,T) \cap (I+|Q|\mathbb{Z}) \cap J$. Recall that for $p, p' \in J$ with $p \equiv p' \pmod{|Q|}$ we have $p \in \mathrm{Occ}^E_k(P,T)$ if and only if $p' \in \mathrm{Occ}^E_k(P,T)$. Hence, it suffices to restrict our attention to the intersection of the first (at most) $|Q|$ positions of J with $I + |Q|\mathbb{Z}$.

■ ALGORITHM **6.6** Computing k-error occurrences in the presence of locked regions in text and pattern.

```
SynchedMatches($P, T, I, k, d, d', Q$)
    $\mathcal{L}^P \leftarrow$ Locked($P, Q, d, k$);
    $\mathcal{L}^T \leftarrow$ Locked($T, Q, d', 0$);
    $M \leftarrow [\, n-m-k \,..\, n-m+k \,]$;
    foreach $P[\,\ell \,..\, r\,) \in \mathcal{L}^P$ do
        foreach $T[\,\ell' \,..\, r'\,) \in \mathcal{L}^T$ do
            $M \leftarrow M \cup (\,\ell' - r - k \,..\, r' - \ell + k\,)$;
    $M \leftarrow M \cap [\,0 \,..\, n-m+k\,)$;
    foreach maximal range $J \subseteq M$ do
        foreach maximal range $J' \subseteq J \cap (I + |Q|\mathbb{Z})$ do
            $\mathrm{Occ}^E_k(P,T) \leftarrow \mathrm{Occ}^E_k(P,T) \cup \{pos \mid (pos, k_{pos}) \in$ Verify($P, T, k, J'$)$\}$;

    foreach maximal range $[\,\ell \,..\, r\,) \subseteq [\,0 \,..\, n-m+k\,) \setminus M$ do
        $J \leftarrow [\,\ell \,..\, min(r, \ell + |Q|)\,)$;
        foreach maximal range $J' \subseteq J \cap (I + |Q|\mathbb{Z})$ do
            foreach $(pos, k_{pos}) \in$ Verify($P, T, k, J'$) do

                $\mathrm{Occ}^E_k(P,T) \leftarrow \mathrm{Occ}^E_k(P,T) \cup ((pos + |Q|\mathbb{Z}) \cap [\,\ell \,..\, r\,))$;
    return $\mathrm{Occ}^E_k(P,T)$;
```

This intersection consists of at most two intervals of total size at most $|I|$. We call `Verify` for each of them, and for each position returned by these `Verify` queries, we add an arithmetic progression to $\mathrm{Occ}_k^E(P,T) \cap (I + |Q|\mathbb{Z})$.

We now proceed to analyze the time complexity of the algorithm in the `PILLAR` model. The two calls to `Locked` require $O(d^2+k+d'^2)$ time in total in the `PILLAR` model due to Lemma 6.10. We then decompose M into maximal ranges, which can be implemented in $O(dd' loglog(dd'))$ time. The interval R of positions marked due to locked regions L_i^P and L_j^T is of size $|L_i^P| + |L_j^T| + 2k - 1$; the number of maximal ranges $R' \subseteq R \cap (I + |Q|\mathbb{Z})$ is at most $(|R| + 2|Q| - 2|I|)/|Q|$. Consequently, the total number of maximal ranges of size at most $|I|$ that we need to consider intervals does not exceed $2k + 1$ plus

$$\begin{aligned}
&\sum_{i=1}^{\ell^P}\sum_{j=1}^{\ell^T}\frac{|L_i^P| + |L_j^T| + 2|Q| - 2|I|}{|Q|}\\
&\quad\le \frac{\ell^T}{|Q|}\sum_{i=1}^{\ell^P}|L_i^P| + \frac{\ell^P}{|Q|}\sum_{i=1}^{\ell^T}|L_i^T| + \frac{2\ell^P\ell^T(|Q| - |I|)}{|Q|}\\
&\quad= O((d'(d|Q| + k|Q|) + dd'|Q| + dd'|Q|)/|Q|)\\
&\quad= O(d'(d+k)).
\end{aligned}$$

Each call to `Verify` in Line 11 of Algorithm 6.6 requires time $O(k(k+|J'|))$ by Lemma 6.3. By the above analysis, we make $O(d'(d+k))$ calls to `Verify`, each time for an interval of size at most $|I|$. Hence, we can upper bound the overall running time for this step by $O(d'(d+k)k(k+|I|))$. Finally, the total time required by `Verify` queries in Line 15 of Algorithm 6.6 is $O(dd'k(k+|I|))$ as we call `Verify` $O(dd')$ times, each time for an interval of size $O(|I|)$. Thus, the overall running time is

$$\begin{aligned}
&O(d'(d+k)k(k+|I|) + d^2 + d'^2 + dd' loglog(dd'))\\
&\quad= O(kd'(d+k)(k+|I|+d+d')).
\end{aligned}$$

The bounds obtained in the time complexity analysis also imply that our representation of $\mathrm{Occ}^E_k(P,T) \cap (I + |Q|\mathbb{Z})$ consists of $O(|I|d'(d+k))$ arithmetic progressions. ⌟

⌜ Lemma 6.12 (`PerMat(`P`, `T`, `k`, `d`, `Q`)`). *Let P denote a pattern of length m and let T denote a text of length n. Further, let $0 \le k \le m$ denote a threshold, let $d \ge 2k$ denote a positive integer, and let Q denote a primitive string that satisfies $|Q| \le m/8d$ and $\delta_E(P, {}^*Q^*) \le d$.*

We can compute the set $\mathrm{Occ}^E_k(P,T)$, using $O(n/m \cdot d^4)$ time in the `PILLAR` *model.*

Proof. We consider $\lfloor 2n/m \rfloor$ blocks $T_0, \dots, T_{\lfloor 2n/m \rfloor - 1}$ of T, each of length at most $3/_2\, m + k - 1$, where the i-th block starts at position $i \cdot m/2$, that is,

$$T_i := T[\, \lfloor i \cdot m/2 \rfloor \,..\, min\{n, \lfloor (i+3) \cdot m/2 \rfloor + k - 1\}\,).$$

Observe that each k-error occurrence of P in T is contained in at least one of the fragments T_i: Specifically, T_i covers all occurrences starting in $[\, \lfloor i \cdot m/2 \rfloor \,..\, \lfloor (i+1) \cdot m/2 \rfloor\,)$. For each block T_i, we call `FindRelevantFragment(`P`, `T_i`, `k`, `d`, `Q`)` from Lemma 6.9 to obtain a fragment $T'_i = T[\, \ell_i \,..\, r_i\,)$ containing all k-error occurrences of P in T_i and an integer range I_i. Lemma 6.9 guarantees that we have $\delta_E(T'_i, Q) \le 3d$ and $|I_i| \le 6d + 1$. We call `SynchedMatches(`P`, `T'_i`, `I_i`, `d`, `$3d$`, `k`, `Q`)` from Lemma 6.11 next. The output of the call to `SynchedMatches` consists of $O((6d+1)3d(d+k)) = O(d^3)$ arithmetic progressions. For each obtained arithmetic progression, we first add $\lfloor i \cdot m/2 \rfloor$ to all of its elements, and, if $i < \lfloor 2n/m \rfloor - 1$, we intersect the resulting arithmetic progression with $[\, \lfloor i \cdot m/2 \rfloor \,..\, \lfloor (i+1) \cdot m/2 \rfloor\,)$; finally, we add the obtained set to $\mathrm{Occ}^E_k(P,T)$. The intersection step guarantees that each k-error occurrence is accounted for by exactly one block.

For the correctness, note that by Lemma 6.9, for each i, we have

$$\mathrm{Occ}^E_k(P, T'_i) \bmod |Q| \subseteq I_i \bmod |Q|.$$

Thus, the call to `SynchedMatches` indeed computes all occurrences of P in T_i.

Each call to `FindRelevantFragment` requires $O(d^2)$ time, while each call to `SynchedMatches` requires time $O(3kd(d+k)(k+(6d+1)+d+3d)) = O(d^4)$. The claimed overall running time follows. ◼

6.4 Computing Occurrences in the Non-Periodic Case

As a next building block, we again discuss how to obtain all k-edits occurrences in the non-periodic case.

Lemma 6.13 (`BreakMatches(`P`, `T`, `$\{B_1,\dots,B_{2k}\}$`, `k`)`: Impl. of Lemma 5.21). *Let k denote a threshold and let P denote a pattern of length m having $2k$ disjoint breaks $B_1,\dots,B_{2k} \preccurlyeq P$ each satisfying $\operatorname{per}(B_i) \geq m/128k$. Further, let T denote a text of length $n \leq {}^3/_2\, m + k$.*

Then, we can compute the set $\mathrm{Occ}^E_k(P,T)$ using $O(k^3)$ time in the `PILLAR` model.

Proof. We proceed similarly to Lemma 4.10: Instead of marking positions, we now mark blocks of length k; in the end, we verify complete blocks at once using `Verify` from Lemma 6.3. Consider Algorithm 6.7 for pseudo-code.

For the correctness, note that we have placed the marks as in the proof of Lemma 5.21; in particular, by Claim 5.23, any block $[jk\mathinner{.\,.}(j+1)k)$ that contains any position $\pi \in \mathrm{Occ}^E_k(P,T)$ has at least k marks. As we verify each such block using `Verify` from Lemma 6.3, we report no false positives, and thus the algorithm is correct.

Next, we analyze the running time. As every break B_i has period $\operatorname{per}(B_i) > m/128k$, every call to `ExactMatches` uses $O(k)$ time in the `PILLAR` model by Lemma 2.6; thus, all calls to `ExactMatches` in total take $O(k^2)$ time in total. Next, by Claim 5.22, we place at most $O(k^2)$ marks in T, so the marking step uses $O(k^2)$ operations in total. Further, finding all positions in T with at least k marks can be done via a linear scan over the multi-set M of all marks after sorting M, which can be done in time $O(k^2 \log\log k)$. Finally, as there are at most $O(k^2/k) = O(k)$ blocks that we verify, and every call

Algorithm 6.7 A PILLAR model algorithm for Lemma 5.21.

```
BreakMatches($P, T, \{B_1 = P[b_1 \,.\,.\, b_1 + |B_1|), \dots, B_{2k} = P[b_{2k} \,.\,.\, b_{2k} + |B_{2k}|)\}, k$)
    multi-set $M \leftarrow \{\}$; $\mathrm{Occ}_k^E(P,T) \leftarrow \{\}$;
    for $i \leftarrow 1$ to $2k$ do
        foreach $\tau \in$ ExactMatches($B_i, T$) do
            $M \leftarrow M \cup \{\lfloor(\tau - b_i - k)/k\rfloor\}$;      // Mark block $\lfloor(\tau - b_i - k)/k\rfloor$ of $T$
            $M \leftarrow M \cup \{\lfloor(\tau - b_i)/k\rfloor\}$;          // Mark block $\lfloor(\tau - b_i)/k\rfloor$ of $T$
            $M \leftarrow M \cup \{\lfloor(\tau - b_i + k)/k\rfloor\}$;      // Mark block $\lfloor(\tau - b_i + k)/k\rfloor$ of $T$
            $M \leftarrow M \cup \{\lfloor(\tau - b_i + 2k)/k\rfloor\}$;     // Mark block $\lfloor(\tau - b_i + 2k)/k\rfloor$ of $T$
    sort $M$;
    foreach $\pi \in [0 \,.\,.\, n-m]$ that appears at least $k$ times in $M$ do
        $\mathrm{Occ}_k^E(P,T) \leftarrow \mathrm{Occ}_k^E(P,T) \cup \{pos \mid (pos, k_{pos}) \in$
         Verify($P, T, k, [\pi \cdot k \,.\,.\, (\pi+1) \cdot k)$)$\}$;
    return $\mathrm{Occ}_k^E(P,T)$;
```

to Verify takes time $O(k^2)$ in the PILLAR model, the verifications take $O(k^3)$ time in the PILLAR in total. Overall, Algorithm 6.7 thus takes $O(k^3)$ time in the PILLAR model. ◾

Lemma 6.14 (RepMat$(P, T, \{(R_1, Q_1) \dots, (R_r, Q_r)\}, k)$: Impl. of Lemma 5.24). *Let P denote a pattern of length m and let $k \le m$ denote a threshold. Further, let T denote a string of length $n \le {}^3\!/_2\, m + k$. Suppose that P contains disjoint repetitive regions $R_1, \dots, R_r$ of total length at least $\sum_{i=1}^{r} |R_i| \ge 3/8 \cdot m$ such that each region R_i satisfies $|R_i| \ge m/8k$ and has a primitive approximate period Q_i with $|Q_i| \le m/128k$ and $\delta_H(R_i, Q_i^*) = \lceil 8k/m \cdot |R_i|\rceil$.*

Then, we can compute the set $\mathrm{Occ}_k^E(P,T)$ using $O(k^4)$ time in the PILLAR model.

Proof. As in the proof of Lemma 5.24, set $m_R := \sum_{i=1}^{r} |R_i| \ge 3/8 \cdot m$ and define for every $1 \le i \le r$ the values $k_i := \lfloor 4 \cdot k/m \cdot |R_i|\rfloor$ and $d_i := \lceil 8 \cdot k/m \cdot |R_i|\rceil = |\mathrm{Mis}(R_i, Q_i^*)|$; write $R_i = P[r_i \,.\,.\, r_i + |R_i|)$.

Again, we proceed similarly to the Hamming distance setting (Lemma 4.11). However, instead of marking positions, we now mark blocks of length k; in the end, we then verify complete blocks at once using `Verify` from Lemma 6.3. Note that we need to ensure that we mark a block of T only at most once for each repetitive part R_i; we do so by first computing a set of all blocks to be marked due to R_i (thereby removing duplicates) and then merging the sets computed for every R_i into a multi-set. Consider Algorithm 6.8 for the complete algorithm visualized as pseudo-code.

For the correctness, first note that in every call to `PerMat` from Lemma 6.12, we have

$$16k/m \cdot |R_i| \geq d_i = \lceil 8k/m \cdot |R_i| \rceil = \delta_H(R_i, Q_i^*) \geq 2k_i,$$

hence $|Q_i| \leq m/128k \leq |R_i|/8d_i$; thus, we can indeed call `PerMat` in this case. Further, note that we have placed the marks as in the proof of Lemma 5.24; in particular, by Claim 5.26, any block $[jk \mathinner{.\,.} (j+1)k]$ that contains any position $\pi \in \mathrm{Occ}_k^E(P,T)$ has at least $m_R - m/4$ marks. As we verify every possible candidate using `Verify` from Lemma 6.3, we report no false positives, and thus the algorithm is correct.

For the running time in the `PILLAR` model, observe that during the marking step, for every repetitive region R_i we call `PerMat` once. In total, all calls to `PerMat` take

$$\sum_i O(n/|R_i| \cdot d_i^4) = \sum_i O(|R_i|/m \cdot k^4) = O(k^4)$$

time in the `PILLAR` model. Further, for every R_i, we place at most $O(|\lfloor \mathrm{Occ}_{k_i}^E(R_i,T)/k \rfloor|)$ (weighted) marks, which can be bounded by $O(|\lfloor \mathrm{Occ}_{k_i}^E(R_i,T)/k \rfloor|) = O(n/|R_i| \cdot d_i) = O(k)$ using Corollary 5.18. Thus, we place $|M| = O(k^2)$ (weighted) marks in total. Hence, the marking step in total takes $O(k^4)$ time in the `PILLAR` model.

As the multi-set M contains at most $O(k^2)$ (weighted) marks, we can sort M (by positions) in time $O(k^2 \log\log k)$; afterwards, we can find the elements with total weight at least $m_R - m/4$ via a

Algorithm 6.8 A PILLAR model algorithm for Lemma 5.24.

```
RepMat($P, T, \{(R_1 = P[r_1 \mathinner{.\,.} r_1 + |R_1|), Q_1) \dots, (R_r = P[r_r \mathinner{.\,.} r_r + |R_r|), Q_r)\}, k$)
    multi-set $M \leftarrow \{\}$; $\mathrm{Occ}^E_k(P,T) \leftarrow \{\}$;
    for $i \leftarrow 1$ to $r$ do
        set $M_i \leftarrow \{\}$;
        foreach $\tau \in$ PerMat($R_i, T, \lfloor 4 \cdot k/m \cdot |R_i| \rfloor, \lceil 8 \cdot k/m \cdot |R_i| \rceil, Q_i$) do
            $M_i \leftarrow M_i \cup \{(\lfloor(\tau - r_i - k)/k\rfloor, |R_i|)\}$;    // Place $|R_i|$ marks at bl.$\lfloor(\tau - r_i - k)/k\rfloor$
            $M_i \leftarrow M_i \cup \{(\lfloor(\tau - r_i)/k\rfloor, |R_i|)\}$;    // Place $|R_i|$ marks at block $\lfloor(\tau - r_i)/k\rfloor$
            $M_i \leftarrow M_i \cup \{(\lfloor(\tau - r_i + k)/k\rfloor, |R_i|)\}$;    // Place $|R_i|$ marks at bl.$\lfloor(\tau - r_i + k)/k\rfloor$
            $M_i \leftarrow M_i \cup \{(\lfloor(\tau - r_i + 2k)/k\rfloor, |R_i|)\}$;    // Place $|R_i|$ marks at bl.$\lfloor(\tau - r_i + 2k)/k\rfloor$
        $M \leftarrow M \cup M_i$;
    sort $M$ by positions;
    foreach $\pi \in [0 \mathinner{.\,.} n - m]$ appearing at least $\sum_{(\pi,v)\in M} v \ge \sum_{i=1}^{r} |R_i| - m/4$ times in $M$ do
        $\mathrm{Occ}^E_k(P,T) \leftarrow \mathrm{Occ}^E_k(P,T) \cup \{pos \mid (pos, k_{pos}) \in$ Verify($P, T, k, [\pi \cdot k \mathinner{.\,.} (\pi+1) \cdot k)$)$\}$;
    return $\mathrm{Occ}^E_k(P,T)$;
```

linear scan over M in time $O(k^2)$. As there are (by Claims 5.25 and 5.26) at most $O(k)$ blocks with at least $m_R - m/4$ marks, we call `Verify` at most $O(k)$ times. As we call verify always on a whole block of length k at once, each call to `Verify` takes $O(k^2)$ time in the `PILLAR` model. Hence, the verification step in total takes $O(k^3)$ time in the `PILLAR` model. In total, Algorithm 6.8 thus takes time $O(k^4)$ in the `PILLAR` model. ▟

6.5 A `PILLAR` Model Algorithm for Pattern Matching with Edits

Finally, we are ready to prove Theorem 6.1.

Theorem 6.1. *Given a pattern P of length m, a text T of length n, and a positive integer $k \le m$, we can compute (a representation of) the set* $\mathrm{Occ}_k^E(P,T)$ *using $O(n/m \cdot k^4)$ time in the `PILLAR` model.*

Proof. We proceed, as in Theorem 4.1, by separately considering each of the three possible outcomes of `Analyze` (P,k). Consider Algorithm 6.9 for pseudo-code.

If there is an approximate period Q of P we call `PerMat` (from Lemma 6.12). Else, for each of the $\lfloor 2n/m \rfloor$ blocks $T_0, \dots, T_{\lfloor 2n/m \rfloor - 1}$, where $T_i := T[\,\lfloor i \cdot m/2 \rfloor \,..\, min\{n, \lfloor (i+3)\cdot m/2 \rfloor + k - 1\}\,)$, we call `BreakMatches` (from Lemma 6.13) or `RepMat` (from Lemma 6.14), depending on the case we are in, and add the computed occurrences in $\mathrm{Occ}_k^E(P,T)$.

The correctness in the approximately periodic case follows from Lemma 6.12 and the fact that we can indeed call `PerMat` since, due to Lemma 6.5, string Q satisfies $\delta_E(P, {}^*Q^*) \le 8k$ and $|Q| \le m/128k \le m/(8 \cdot 8k)$. In the other cases, first observe that each length-$(m+k)$ fragment of T is contained in at least one of the fragments T_i and hence we do not lose any occurrences. Second, by Lemma 6.5 and due to $|T_i| \le {}^3/_2\, m + k$, the parameters in the calls to `BreakMatches` (from Lemma 6.13) and `RepMat` (from Lemma 6.14) each satisfy the requirements. Finally, the intersection step in Line 13 of Algorithm 6.9 guarantees that we account for each k-error occurrence exactly once.

ALGORITHM **6.9** Computing k-error occurrences in the PILLAR model.

```
EditOccurrences($P, T, k$)
    ( $B_1, \dots, B_{2k}$ or $(R_1, Q_1), \dots, (R_r, Q_r)$ or $Q$ ) ← Analyze($P, k$);
    $\mathrm{Occ}_k^E(P,T) \leftarrow \{\}$;
    if approximate period $Q$ exists then
        return PerMat($P, T, k, 8k, Q$);
    for $i \leftarrow 0$ to $\lfloor 2n/m \rfloor - 1$ do
        $T_i \leftarrow T[\,\lfloor i \cdot m/2 \rfloor \,..\, min\{n, \lfloor (i+3) \cdot m/2 \rfloor - 1 + k\}\,)$;
        if breaks $B_1, \dots, B_{2k}$ exist then
            $\mathrm{Occ}_k^E(P,T_i) \leftarrow$ BreakMatches($P, T_i, \{B_1, \dots, B_{2k}\}, k$);
        else if repetitive regions $(R_1, Q_1), \dots, (R_r, Q_r)$ exist then
            $\mathrm{Occ}_k^E(P,T_i) \leftarrow$ RepMat($P, T_i, \{(R_1, Q_1), \dots, (R_r, Q_r)\}, k$);
        if $i < \lfloor 2n/m \rfloor - 1$ then
            $V \leftarrow \{\ell + \lfloor i \cdot m/2 \rfloor \mid \ell \in \mathrm{Occ}_k^E(P,T_i)\} \cap [\,\lfloor i \cdot m/2 \rfloor \,..\, \lfloor (i+1) \cdot m/2 \rfloor\,)$;
        $\mathrm{Occ}_k^E(P,T) \leftarrow \mathrm{Occ}_k^E(P,T) \cup V$;
    return $\mathrm{Occ}_k^E(P,T)$;
```

For the running time in the PILLAR model, we have that the call to Analyze takes $O(k^2)$ time in the PILLAR model, the call to PerMat takes $O(n/m \cdot k^4)$ time in the PILLAR model, each call to BreakMatches takes $O(k^3)$ time in the PILLAR model, and each call to RepMat takes $O(k^4)$ time in the PILLAR model. Finally, as the output of our calls to BreakMatches and RepMat is of size $O(k^2)$ and is sorted, Lines 13 and 14 require $O(k^2)$ time. As there are at most $O(n/m)$ calls to BreakMatches and RepMat, we can bound the total time in the PILLAR model by $O(n/m \cdot k^4)$. ⌟

As in the Hamming distance case, we now immediately obtain the main results. In particular, combining Lemma 2.9 and Theorem 6.1, we obtain an algorithm for pattern matching with edits that is, again, not slower than the known algorithm by Cole and Hariharan [33].

Corollary 6.15. *Given a text T of length n, a pattern P of length m and a threshold k, we can compute $\mathrm{Occ}^E_k(P,T)$ in time $O(n + n/m \cdot k^4)$.*

Combining Theorems 2.13 and 6.1 as in the Hamming distance case, we obtain the main result for the fully-compressed setting.

Corollary 6.16. *Let $\mathcal{G}_T$ denote an SLP of size n generating a string T, let $\mathcal{G}_P$ denote an SLP of size m generating a string P, let k denote a threshold, and set $N := |T| + |P|$.*

Then, we can compute $|\mathrm{Occ}^E_k(P,T)|$ in time $O(m \log N + n\, k^4 \log^3 N)$.

The elements of $\mathrm{Occ}^E_k(P,T)$ can be reported within $|\mathrm{Occ}^E_k(P,T)|$ extra time.

Finally, combining Theorems 2.15 and 6.1, we also obtain the main result for the dynamic setting.

Corollary 6.17. *We can maintain a collection $\mathcal{X}$ of non-empty persistent strings of total length N under the operations* `makestring`(U), `concat`(U,V), `split`(U,i) *requiring time $O(\log N + |U|)$, $O(\log N)$ and $O(\log N)$, respectively, so that given two strings $P,T \in \mathcal{X}$ with $|P| = m$ and $|T| = n$ and a threshold k, we can compute the set $\mathrm{Occ}^E_k(P,T)$ in time $O(n/m \cdot k^4 \log^2 N)$.*[29]

29. All running time bounds hold w.h.p.

Conclusions and Open Questions

In the preceding chapters, we have seen tight characterizations of the solution structures of the pattern matching with mismatches or errors problems. Further, we discussed algorithms—abstracted to the PILLAR model—that exploited said structure.

As a first concluding remark, for both Theorems 3.1 and 5.1, we can easily strengthen the (almost) periodic case to allow for only $(1+\varepsilon)$ mismatches or errors to a periodic string:

Theorem. *Given a pattern P of length m, a text T of length n, a threshold $k \le m$, and a positive ε, at least one of the following holds for a function f:*

- *We have $|\mathrm{Occ}_k^H(P,T)| \le f(\varepsilon) \cdot 576 \cdot n/m \cdot k$.*
- *There is a primitive Q with $|Q| \le m/128k$ and $\delta_H(P, Q^*) < (1+\varepsilon)\,k$.*

Theorem. *Given a pattern P of length m, a text T of length n, a threshold $k \le m$, and a positive ε, at least one of the following holds for a function f:*

- *We have $|\mathrm{Occ}_k^E(P,T)| \le f(\varepsilon) \cdot 642045 \cdot n/m \cdot k$.*
- *There is a primitive Q with $|Q| \le m/128k$ and $\delta_E(P, {}^*Q^*) < (1+\varepsilon)\,k$.*

We did not try to optimize the constants in Theorems 3.1 and 5.1, especially the constant *642045* from Theorem 5.1 almost begs to be improved.

For the algorithm side, specifically for pattern matching with mismatches, it seems as if we fully exploited the structural insight from Theorem 3.1—at least in terms of possible improvements for the dependency on k in the running time of the PILLAR model algorithm.[30] Essentially, to improve the current running time of $O(k^2)$, not only would one need to avoid handling up to $O(k^2)$ marked positions at once—it is also not admissible to Verify $O(k)$ different starting positions using a Verify that takes $O(k)$ time per call. While such a feat does not sound totally impossible, achieving it does seem to be unrealistic. Perhaps it could hence be more time-efficient to look for matching (conditional) lower bounds instead.

30. Technically, this is not completely true: Recall the extra *loglog k* factor in the running time due to *sorting* $O(k^2)$ numbers. Removing said factor seems to be an interesting problem in itself.

In contrast, for pattern matching with errors, there is hope. Observe that we Verify $O(k)$ blocks in time $O(k^2)$ each—this yields a dependency of $O(k^3)$. As in [33], the critical case for the running time is the (almost) periodic case, where we lose a factor of $O(k)$ in the implementation of SynchedMatches. However, with the additional insights gained in 5, it is conceivable that there is a faster implementation of SynchedMatches—in total yielding a faster algorithm also for the standard setting where both text and pattern are uncompressed.

In any case, such improvements seemingly cannot go beyond $O(k^3)$—for reasons very similar to the Hamming distance case: In light of the (conditional) $O(n^2)$ lower bound of computing the edit distance between two strings [7], trying to improve Verify beyond $O(k^2)$ seems to be lunatic. Again, perhaps it is possible to obtain a (conditional) lower bound ruling out any improvements beyond $O(k^3)$—this would at least give a concrete reason for this seemingly impenetrable barrier.

As a second concluding remark, recall that the motivation for the PILLAR model was to give a simple interface that unifies many of the classically studied settings in stringology. We saw implementations for three major settings, but other settings are imaginable.

Finally, it would be interesting to see the algorithms from the previous chapters implemented into code. Especially for pattern matching with mismatches, the algorithm is reasonably simple to allow such a feat—It would be interesting to see if the improvements in the running time then also yield actual improvements in real-life. In particular in the fully-compressed setting this seems to be not too far-fetched: The possibly very good compression of SLPs does allow for some "slack" in possible algorithms—it is far from clear though if said "slack" is enough. For the edit distance case, the constant *642045* seems to be prohibitive. Hence, any implementation would probably have to be preceded by an optimization of said constant and probably a simplification of the algorithm as well.

Part II

Counting Patterns in Graphs

$\boxed{7}$ $\boxed{8}$ $\boxed{9}$ $\boxed{10}$ $\boxed{11}$

Introduction to Part II

In the second part of this thesis, we focus on *counting patterns in graphs*. Detecting, counting and enumerating patterns in graphs are among the most well-studied computational problems in theoretical computer science, with applications in many fields—in biology when studying networks of interactions between proteins [107, 54]; in statistical physics when studying how certain molecules can be arranged on a surface [113, 67, 68]. Even more applications include the study of neural and social networks [89] and database theory [56], to name but a few.

As an illustrative example, say you have a set of different *wines* and a set of different *cheeses*. You want to find matching pairs of *wine* and *cheese*, but without letting any *wine* or *cheese* go to waste. *Finding* a solution—a *perfect matching*—easily,[31] you are intrigued and start to *count* the possible solutions—just to quickly give up: As was seen by Valiant [117, 118], by *counting* perfect matchings in a bipartite graph, we can compute the so-called *permanent* of a matrix[32]—In particular, computing the permanent is much harder than computing the related *determinant* of a matrix. In fact, computing the permanent is #***P***-complete [117, 118], and thus harder than every problem in the polynomial-time hierarchy ***PH*** [115].[33] This surprising result motivated the subfield of *Counting Complexity Theory*—the study of problems where the task is to *count* the solutions and not just to find any.

31. Using for instance the algorithm of Edmonds [43].

32. In this case the bi-adjacency matrix of the bipartite graph.

33. Read: *very* hard.

Observe that this is in stark contrast to the first part of this thesis—When concerning *strings*, finding and counting (approximate) matches of a pattern string turned out to be equally easy and hence we could focus on *finding* patterns.

Finding and Counting Graph Homomorphisms

One possible way of formalizing the notion of "finding patterns in a graph" is to turn to *graph homomorphisms*.[34] For graph classes $\mathcal{H}$ and $\mathcal{G}$, in the decision problem $\textsc{Hom}(\mathcal{H} \to \mathcal{G})$, we are given graphs $H \in \mathcal{H}$ and $G \in \mathcal{G}$, and are to decide whether there is a mapping $h : V(H) \to V(G)$ such that for any edge $\{u, v\}$ in $E(H)$, the edge $\{h(u), h(v)\}$ exists in $E(G)$. Naturally, this (well-known) problem has been studied in the literature—In fact first results were already shown in the late 1970s and 1980s [50, 85, 3].

34. As we see later, essentially any notion of finding (or counting) "patterns" can be reduced to finding (or counting) graph homomorphisms. This includes in particular (*induced*) *subgraphs*.

Complexity Dichotomies for Finding Graph Homomorphisms

Naturally, finding graph homomorphisms is a hard problem—if both classes $\mathcal{H}$ and $\mathcal{G}$ contain all graphs (we write $\mathcal{H} = \mathcal{G} = \top$) the problem $\textsc{Hom}(\mathcal{H} \to \mathcal{G})$ is easily ***NP***-complete: Checking whether a graph H admits a homomorphism to the complete graph on 3 vertices $K_3 = \Delta$ is equivalent to checking whether H is 3-colorable, a classical ***NP***-hard problem (see for instance [49]).

Motivated by the hardness in the general case, special cases of the problem $\textsc{Hom}(\mathcal{H} \to \mathcal{G})$ were investigated. In the *language-restricted* version of $\textsc{Hom}(\mathcal{H} \to \mathcal{G})$, we assume that only the class $\mathcal{H} = \top$ is the set of all graphs and that the class $\mathcal{G}$ is somehow restricted.[35] Setting $\mathcal{G} = \{K_3\}$, we observe that our example from before is also an example of a language-restricted version of the homomorphism counting problem—The problem $\textsc{Hom}(\top \to \mathcal{G})$ is ***NP***-hard in general as well. However, if the class $\mathcal{G}$ contains only *bipartite* graphs, the problem $\textsc{Hom}(\top \to \mathcal{G})$ is solvable in polynomial time [59]. In fact, Hell and Nešetřil [59] also prove the following hardness result: If the class $\mathcal{G}$ contains a non-bipartite graph, the problem $\textsc{Hom}(\top \to \mathcal{G})$ is ***NP***-hard. Together, this yields a *complexity dichotomy*: for any problem $\textsc{Hom}(\top \to \mathcal{G})$, we obtain its complexity just by looking at the class $\mathcal{G}$.

35. For instance, think of the class of all bipartite graphs.

With the problem $\textsc{Hom}(\top \to \mathcal{G})$ essentially fully understood, the focus shifted to understanding "the other side", that is the case where the class $\mathcal{G}$ contains all graphs instead. This *structurally re-*

stricted version of the problem is especially interesting from the viewpoint of *parameterized complexity theory*: Given two graphs H, G to compute the number of homomorphisms from H to G, we think of the graph H—the pattern—as being much smaller than the host graph G. Now, on "practical instances"[36] an algorithm running in time $2^{O(|V(H)|)} \cdot |V(G)|$ might be more usable than the trivial brute-force algorithm running in time $O(|V(G)|^{|V(H)|})$ (which always exists).

36. For instance if the pattern has constant size.

Specifically, Grohe [55] proved that if every graph in the graph class $\mathcal{H}$ contains only graphs for which the so-called *treewidth* is small,[37] then $\textsc{Hom}(\mathcal{H} \to \top)$ is "fixed-parameter tractable"[38] (and in fact even polynomial-time solvable); otherwise it is "$\boldsymbol{W}[1]$-hard" (essentially a parameterized equivalent of $\boldsymbol{NP}$-hardness). We formalize these notions later; also consult [93, 47, 36] for an in-depth introduction to parameterized complexity theory.

37. Strictly speaking, the graphs in the class $\mathcal{H}$ need to be only *homomorphically equivalent* to graphs with small treewidth.

38. That is, solvable in time $f(|V(H)|) \cdot |V(G)|^{O(1)}$ for graphs $H \in \mathcal{H}, G \in \top$.

Complexity Dichotomies for Counting Graph Homomorphisms

In this part of the thesis, we focus on *counting* the number of homomorphisms between graphs. For two graph classes $\mathcal{H}$ and $\mathcal{G}$, we define the counting version of the homomorphism problem (denoted by $\#\textsc{Hom}(\mathcal{H} \to \mathcal{G})$) as follows: Given graphs $H \in \mathcal{H}$ and $G \in \mathcal{G}$, the task is to compute the number of (graph) homomorphisms from the graph H to the graph G. As in the decision realm, the language-restricted version $\#\textsc{Hom}(\top \to \mathcal{G})$ has been studied in the context of the counting constraint satisfaction problem: A dichotomy theorem of Dyer and Greenhill [42] implies that the problem $\#\textsc{Hom}(\top \to \mathcal{G})$ is solvable in polynomial time or $\#\boldsymbol{P}$-complete depending on an explicit criterion on $\mathcal{G}$.

The structurally restricted version of the graph homomorphism problem has been studied in the counting regime as well: A counting analogue of Grohe's dichotomy was established by Dalmau and Jonsson [37]. In fact the explicit criterion on $\mathcal{H}$ is almost the same: $\#\textsc{Hom}(\mathcal{H} \to \top)$ is solvable in polynomial time if and only if there is a constant bound on the treewidth of the graphs in the class $\mathcal{H}$; otherwise the problem $\#\textsc{Hom}(\mathcal{H} \to \top)$ is complete for the class $\#\boldsymbol{W}[1]$

(where the class $\#\mathbf{W}[1]$ is the counting equivalent of $\mathbf{W}[1]$). We take a closer look at the hardness part of the above dichotomy later in this part.

The Doubly Restricted Version of Counting Homomorphisms

The previous results yield a surprisingly clean picture of the complexity landscape of the problems of finding and counting graph homomorphisms for both, the language-restricted and the structurally restricted version. However, none of the previous results are applicable for the *doubly restricted* version: Instead of restricting only $\mathcal{H}$ or $\mathcal{G}$, we consider the problem $\#\textsc{Hom}(\mathcal{H} \to \mathcal{G})$ where *both* classes are fixed. This can be seen as a special case of both the structurally restricted version and the language-restricted version. In particular, the known dichotomies translate only for certain pairs of classes $\mathcal{H}, \mathcal{G}$, leaving a wide gap in the complexity landscape to be explored. In particular, it is imaginable that for real-world instances, *both* graphs H and G have a certain structure that can be exploited. In fact, we can show exactly that: If we consider so-called *Kneser graphs* for $\mathcal{H}$, then we can (reversibly) encode any problem in $\#\mathbf{W}[1]$ into a homomorphism problem. Formally, we obtain the following *universality result*.

Main Theorem 5 (Universality for $\mathbf{W}[1]$ and $\#\mathbf{W}[1]$). *For any problem P in $\mathbf{W}[1]$, there are graph classes $\mathcal{H}_P$ and $\mathcal{G}_P$ such that*

$$P \equiv^{fpt}_T \textsc{Hom}(\mathcal{H}_P \to \mathcal{G}_P).$$

and for any problem P' in $\#\mathbf{W}[1]$, there are classes $\mathcal{H}'_P$ and $\mathcal{G}'_P$ such that

$$P' \equiv^{fpt}_T \#\textsc{Hom}(\mathcal{H}'_P \to \mathcal{G}'_P),$$

where $\equiv^{fpt}_T$ denotes interreducibility (sometimes also called equivalence) with respect to parameterized Turing-reductions.
The classes $\mathcal{H}_P$ and $\mathcal{H}'_P$ are recursively enumerable and the classes $\mathcal{G}_P$ and $\mathcal{G}'_P$ are recursive.[39]

39. Not to worry, we recall the basic concepts used in this statement later in this part.

Observe that Main Theorem 5 makes a clear categorization of the problems $\textsc{Hom}(\mathcal{H} \to \mathcal{G})$ into "easy" (that is fixed-parameter tractable) and "hard" (that is $\boldsymbol{W[1]}$-hard or $\#\boldsymbol{W[1]}$-hard) cases unlikely: Think of the work of Ladner [76] on $\boldsymbol{NP}$-intermediate problems and its generalization to the parameterized setting by Downey and Fellows [41]. In particular, a general partition of the class $\boldsymbol{W[1]}$ in fixed-parameter tractable and $\boldsymbol{W[1]}$-complete problems is very unlikely—a similar reasoning applies to $\#\boldsymbol{W[1]}$.

Note that Main Theorem 5 and in particular its consequences for the classes $\#\boldsymbol{W[1]}$ and $\boldsymbol{W[1]}$ are independent from the "non-dichotomy" results of Bodirsky and Grohe [14] and Chen, Thurley, and Weyer [30], which rule out a $\boldsymbol{P}$ vs. $\boldsymbol{NP}/\#\boldsymbol{P}$ dichotomy for the structurally restricted versions: Bodirsky and Grohe [14] prove a $\boldsymbol{P}$ vs. $\boldsymbol{NP}$ non-dichotomy by a modification of the result of Ladner [76]; however, this has no direct implications from neither a parameterized complexity nor a counting complexity point of view. Independently, Chen et al. [30] gave a similar result also for the counting version and hence obtained a $\boldsymbol{P}$ vs. $\#\boldsymbol{P}$ non-dichotomy result; again, this has no implications for the parameterized setting.

From Homomorphisms to (Induced) Subgraphs

In a sense, the graph homomorphism problem is sufficiently understood. As promised, we now move to *induced subgraphs*.

Specifically, for a graph property Φ, we consider the problem $\#\textsc{IndSub}(\Phi)$, where we are given a graph G and an integer k, and we are to compute the number of all k-vertex induced subgraphs of G that satisfy Φ.[40] Observe the trivial $O(|V(G)|^k)$ brute-force algorithm for this problem.

40. Note that this is a slight abuse of notation: For two graph *classes* $\mathcal{H}$ and $\mathcal{G}$, in the similar, but different problem $\#\textsc{IndSub}(\mathcal{H} \to \mathcal{G})$ the task is, given $H \in \mathcal{H}$ and $G \in \mathcal{G}$ to compute how often H appears as an induced subgraph in G—we have to count a *single* pattern from $\mathcal{H}$. In contrast, in $\#\textsc{IndSub}(\Phi)$, we have to count *all* patterns of a fixed size.

Introduced by Jerrum and Meeks [62], $\#\textsc{IndSub}(\Phi)$ has seen a long line of research [63, 87, 64] that yielded $\#\boldsymbol{W[1]}$-hardness for many, very specific properties Φ.

1 The property Φ holds for a graph H if and only if H is connected.

2 The property Φ has *low edge-densities*; this is true for instance for all sparse properties such as planarity and made formal in Section 10.3.
3 The property Φ holds for a graph H if and only if the number of edges of H is even/odd.
4 The property Φ is closed under the addition of edges, and the minimal elements have large treewidth.

However, on a closer look, only Item 2 yields a (conditional) lower bound that comes remotely close to the trivial upper bound from the brute-force algorithm. This is mainly due to the fact that the other results use Ramsey's Theorem to obtain a clique (or independent set) of size k in a graph of size $R(k) = 2^{\Theta(k)}$ [95, 110, 44]. Using such a construction, the best lower bound may ever hope for is of order $f(k) \cdot |V(G)|^{o(\log k)}$ for any function f.

Alas, as Curticapean et al. [35] observed, the framework by Lovász [83] lifts the dichotomies from counting homomorphisms to counting (induced) subgraphs. The catch is, that the explicitness of the criterion for counting homomorphisms does not transfer: In particular, we can express #IndSub(Φ) as a finite linear combination of homomorphism numbers—we even get a monotonicity statement[41]—however, the coefficients of the linear combination are alternating sums and notoriously hard to analyze.[42] In fact, a series of work [100, 40] found topological and algebraic interpretations of said coefficients. We continue this line of work by giving an interpretation in terms of so-called f-vectors and h-vectors of a property Φ. Among other results, this interpretation allows us to resolve an open conjecture by Jerrum and Meeks.

41. That is, if computing any of the summands of the linear combination is hard, then so is the whole linear combination.

42. Later, we take a detailed look at both the framework of Lovász [83] as well as the coefficients of said linear combination.

⌜ Conjecture (Jerrum and Meeks [63, 64]). *Let Φ denote a graph property that depends only on the number of edges of a graph. If the property Φ is not trivial,[43] then #IndSub(Φ) is #$\mathbf{W}[1]$-complete.* ⌟

43. We say a property Φ is non-trivial if there are only finitely many k where Φ is true of false for all graphs with k vertices.

We prove their conjecture.

Main Theorem 6. *Let Φ denote a non-trivial computable graph property that depends only on the number of edges of a graph. #IndSub(Φ) is #$\mathbf{W}[1]$-complete and cannot be solved in time $f(k) \cdot |V(G)|^{o(k/\log k)}$ for any f, unless ETH fails.*

Further, we obtain #$W[1]$-hardness results and (almost) tight conditional lower bounds for large classes of properties Φ. As an example, we obtain hardness whenever Φ is monotone.

Main Theorem 7. *Let Φ denote a monotone graph property that is not trivial. Then #IndSub(Φ) is #$\mathbf{W}[1]$-complete and cannot be solved in time $f(k) \cdot |V(G)|^{o(k/\log^{1/2} k)}$ for any function f, unless ETH fails.*

Almost Everything You Need to Know About Graphs 7

Let us start with basic definitions and concepts used throughout this part. We assume that numbers are encoded in binary; we write $\mathbb{N} \subseteq \{0,1\}^*$.[44] For a positive integer n, we define $[\,n\,] := \{1,\dots,n\}$ and set $[\,0\,] := \{\}$. For a finite set S, we write $\#S$ and $|S|$ for the cardinality of S—we have $\#[\,n\,] = |[\,n\,]| = n$.

44. Whenever we use natural numbers and the symbol $\mathbb{N}$ in particular, we explicitly specify whether 0 is to be included or not.

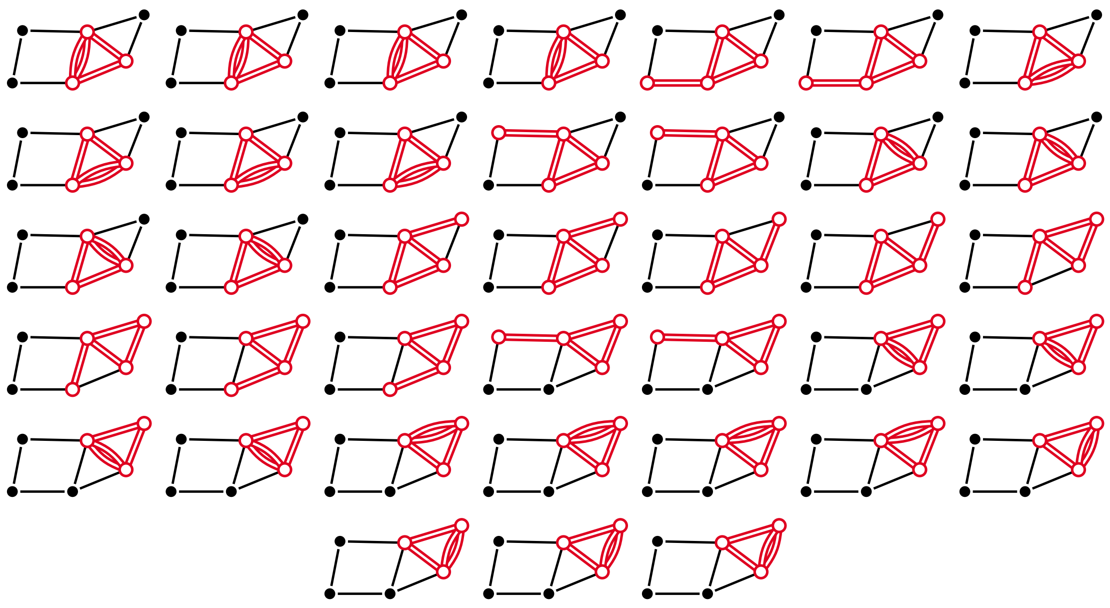
For two functions $f : A \to B$ and $g : B \to C$, we write $f \circ g$ for the function that maps $x \in A$ to $g(f(x)) \in C$; we call $f \circ g$ the *composition of f and g*. Further, we use the symbol $\star$ to denote *partial function applications*: For a function $f : A \times B \to C$ and an element $a \in A$, we write $f(a,\star)$ for the function that maps $b \in B$ to $f(a,b)$.[45] This notation naturally generalizes to other domains.

45. Equivalently, $f(a,\star) := f_{|\{a\}\times B}$.

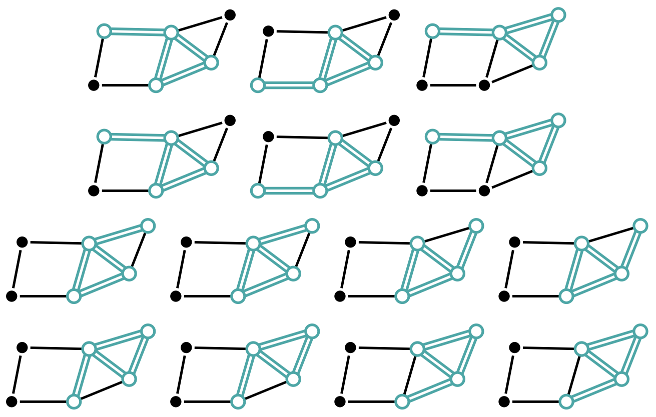
7.1 Graphs and Mappings between Graphs

In this part, the central objects of study are *graphs*. A graph $G = (V(G), E(G))$ is a pair consisting of a finite set of vertices $V(G)$ and a finite set of edges $E(G) \subseteq \{\{u,v\} \mid u,v \in V(G)\}$—The graphs that we consider are unlabeled, undirected, and do not contain parallel edges. The size of a graph G is the size of its vertex set $V(G)$.

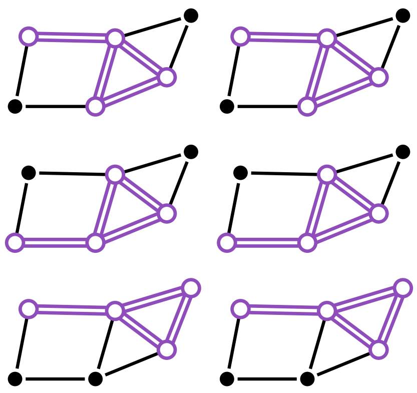
Of key relevance are mappings or *morphisms* between graphs—for two graphs H and G, a morphism $f : H \to G$ is a mapping from $V(H)$ to $V(G)$. A *homomorphism* is a morphism f that preserves edges, that is, for all $\{u,v\} \in E(H)$ we have $\{f(u),f(v)\} \in E(G)$. An *embedding* is an injective homomorphism. A *strong embedding* is an embedding f that additionally preserves non-edges, that is $\{f(u),f(v)\} \in E(G)$ if and only if $\{u,v\} \in E(H)$. Finally, an *isomorphism* is a surjective strong embedding. If there is an isomorphism between two graphs H and G, we say that H and G are isomorphic and write $H \cong G$.



(a) There are exactly 38 homomorphisms from the graph to the graph . Multiple edges of that are mapped to the same edge of are depicted as parallel edges. Note that while two homomorphisms may map vertices from to different vertices of , the image of the homomorphisms (and thus the corresponding subgraphs of) may coincide.



(b) Exactly 14 of the homomorphisms from to are also embeddings.



(c) Exactly six of the homomorphisms from to are also strong embeddings.

☐ FIGURE 7.1. Morphisms from to : There are 38 homomorphisms, 14 embeddings and six strong embeddings from to . In other words, we have $\#Hom(\rightarrow) = 38$, $\#Emb(\rightarrow) = 14$, and $\#StrEmb(\rightarrow) = 6$.

Further, we write $Hom(\, H \to G\,)$ to denote the set of all homomorphisms from H to G, we write $Emb(\, H \to G\,)$ for the set of all embeddings, and we write $StrEmb(\, H \to G\,)$ for the set of all strong embeddings. In particular, we write $\#Hom(\, H \to \star\,)$ for the function that maps a graph G to $\#Hom(\, H \to G\,)$. Consider Figure 7.1 for an example and its visualization. Finally, very rarely, we need *color-prescribed* homomorphisms: For graphs G and H and a homomorphism c in $Hom(\, G \to H\,)$, we write $cp\text{-}Hom(H \to G)$ for all homomorphisms h from H to G that map vertices from h to the vertex (color) class prescribed by c. Formally: $cp\text{-}Hom(H \to G) := \{h \in Hom(\, H \to G\,) \mid \forall v : c(h(v)) = v\}$.

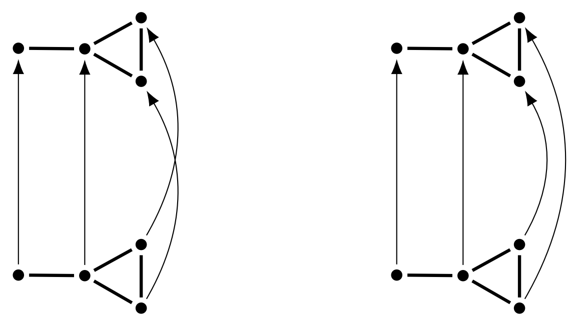
As a special case, we also consider morphisms from a graph G to itself, so-called *endomorphisms*. In particular, we are interested in endomorphisms that are also isomorphisms—we call them *automorphisms* and write $Aut(\, G\,)$ for the set of all automorphisms of G.

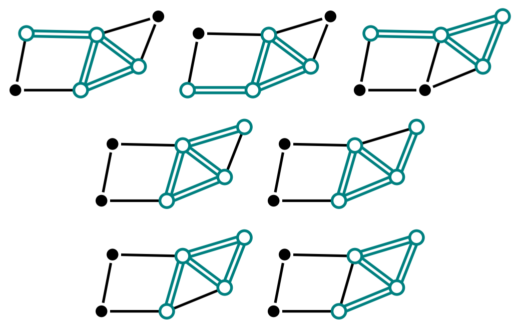
A *subgraph* of a graph G is a graph H that consists of a subset of the vertices in $V(G)$, and a subset of the edges in $E(G)$ that are incident on two vertices in $V(H)$—we have $V(H) \subseteq V(G)$ and $E(H) \subseteq \{\{u,v\} \in E(G) \mid u,v \in V(H)\}$. Note that any subset of $E(G)$ uniquely defines a subgraph of G. Hence, for a subset $F \subseteq E(G)$, we write $G\{\,F\,\}$ for the subgraph of G with the edge set F. We say a subgraph $G\{\,F\,\}$ is proper if $F \neq E(G)$. Further, we write $Sub(\, H \to G\,)$ to denote the set of all subgraphs of G that are isomorphic to H. Observe that each subgraph in $Sub(\, H \to G\,)$ corresponds to $\#Aut(\, H\,)$ embeddings from H to G—we have

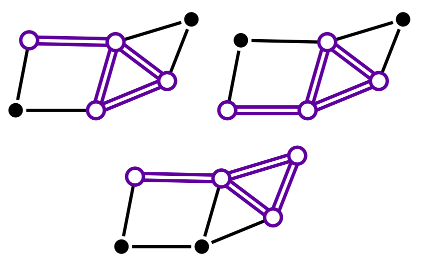
$$\#Sub(\, H \to G\,) = \#Emb(\, H \to G\,)/\#Aut(\, H\,). \tag{7.1}$$

Consult Figure 7.2b for an example and its visualization.

An *induced subgraph* of a graph G is a graph H that consists of a subset of the vertices in $V(G)$, and all of the edges in $E(G)$ that are incident on two vertices in $V(H)$—we have $V(H) \subseteq V(G)$ and $E(H) = \{\{u,v\} \in E(G) \mid u,v \in V(H)\}$. Note that any subset of $V(G)$ uniquely defines an induced subgraph of G. Hence, for a subset $W \subseteq V(G)$, we write $G[\,W\,]$ for the induced subgraph of G with the vertex set W. We say an induced subgraph $G[\,W\,]$ is


(a) The graph ··◁ has exactly two automorphisms.


(b) The graph ··◁ appears exactly seven times as a (not necessarily induced) subgraph of ⊏⧄. Note that two subgraphs are the same if and only if they consist of the same set of edges.


(c) The graph ··◁ appears exactly three times as an induced subgraph of the graph ⊏⧄. Note that two induced subgraphs are the same if and only if they consist of the same set of vertices.

☐ Figure 7.2. Automorphisms of ··◁ and counting ··◁ in ⊏⧄: The graph ··◁ appears seven times as a subgraph in ⊏⧄ and thrice as an induced subgraph. Observe that each (induced) subgraph corresponds to $\#Aut(··◁) = 2$ (strong) embeddings from ··◁ to ⊏⧄. In other words, we have $\#Sub(··◁ \to ⊏⧄) = \#Emb(··◁ \to ⊏⧄)/\#Aut(··◁) = 7$ and $\#IndSub(··◁ \to ⊏⧄) = \#StrEmb(··◁ \to ⊏⧄)/\#Aut(··◁) = 3$.

proper if $W \neq V(G)$. Further, we write $\mathit{IndSub}(\,H \to G\,)$ to denote the set of all induced subgraphs of G that are isomorphic to H. Observe that each induced subgraph in $\mathit{Sub}(\,H \to G\,)$ corresponds to $\#\mathit{Aut}(\,H\,)$ strong embeddings from H to G—we have

$$\#\mathit{IndSub}(\,H \to G\,) = \#\mathit{StrEmb}(\,H \to G\,)/\#\mathit{Aut}(\,H\,). \tag{7.2}$$

Consult Figure 7.2c for an example and its visualization.

Finally, we say a graph H is a *core* if and only if there is no homomorphism from H to a proper subgraph of H. In particular, every endomorphism of a core is an automorphism.

7.2 Common Graph Parameters

We discussed different mappings between graphs and different notions of subgraphs—however, we did not yet discuss properties of graphs themselves; that, we do in this section.

We start with global properties of graphs. To that end, we call edges of the form $\{u,u\}$ a self-loop.[46] Now, we say that a graph is *loop-free* if it does not contain a self-loop.

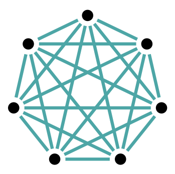
46. Note that we write the set $\{u\}$ as $\{u,u\}$ to emphasize it being an edge from u to u.

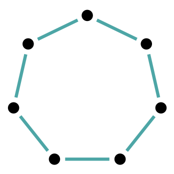
Next, if two vertices u,v in $V(G)$ form an edge $\{u,v\}$ in $E(G)$, we say that u and v are adjacent and we say that the edge $\{u,v\}$ is *incident* on u and v. If all vertices of a graph are pairwise adjacent, we say that the graph is a *complete graph* or a *clique*. We write K_n to denote the clique with n vertices. We write $\omega(G)$ for the size of the largest subgraph of G that is a clique. We write $K_{a,b}$ for the biclique with sizes a and b, that is, for the graph

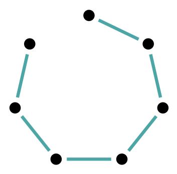
$$K_{a,b} := (A \cup B, \{(u,v) \mid u \in A, v \in B\}); \quad a = |A|, b = |B|.$$

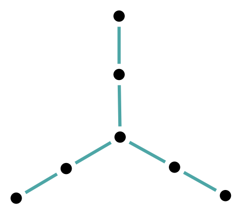
Similarly, we write $\boxplus_k$ for the square grid graph on $k \cdot k$ vertices. That is, we have

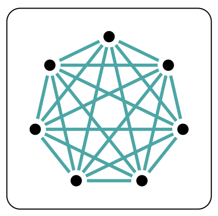
$$\boxplus_k := ([\,k\,] \times [\,k\,], \{\{(u,v),(x,y)\} \mid |u-x| + |v-y| = 1\}.$$

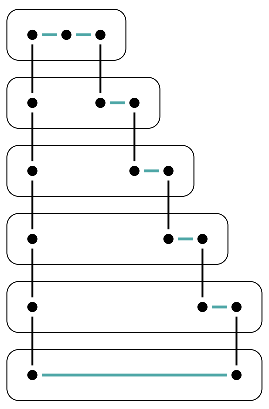

(a) The clique K_7.

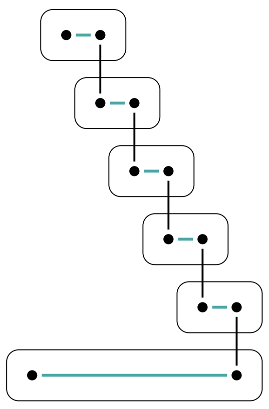

(b) The cycle C_7.

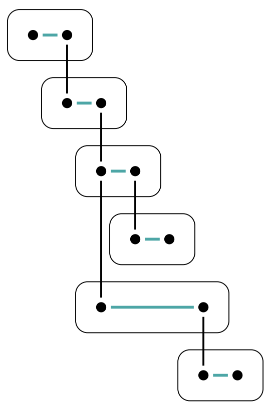

(c) The path P_7.


(d) A tree T with 7 vertices.


(e) All vertices need to be in a single bag, we have $tw(K_7) = 6$.
In general, we have $tw(K_n) = n - 1$.


(f) All but one edge get their own bag, the remaining edge is added to the top-most bag, we have $tw(C_7) = 2$.
In general: $tw(C_n) = 2$.


(g) Each edge gets its own bag, we have $tw(P_7) = 1$.
In general: $tw(P_n) = 1$.


(h) Each edge gets its own bag, we have $tw(T) = 1$.
In general, for any tree T we have $tw(T) = 1$.

☐ FIGURE 7.3. The complete graph K_7, the cycle C_7, the path P_7, and a tree with seven vertices, as well as tree decompositions with minimal width for each graph. Edges of graphs are depicted in green; if a vertex appears in multiple bags, the corresponding copies in the bags are connected with a black edge. Note that black edges also indicate which bags of the tree decomposition are connected.

The *degree* of a vertex v is the number of edges incident on v; denoted by $d(v)$.[47] We define the *average degree* of a graph G as

47. Note that a loop is counted twice.

$$d(G) := \frac{1}{|V(G)|} \cdot \sum_{v \in V(G)} d(v) = \frac{|E(G)|}{2|V(G)|}.$$

Intuitively, graphs with high average degree look like a clique. Formally, extremal graph theory gives us the following fact.

FACT 7.1 (TURÁN'S THEOREM, [SEE 83, SECTION 2.1]). *Any graph G with an average degree larger than $(1-1/r)/4 \cdot |V(G)|$ contains the clique K_{r+1} as a subgraph.*[48] ⏌

48. Note that equivalently, we could require G to have more than $(1 - 1/r) \cdot |V(G)|^2/2$ edges.

The *neighborhood* $N_G(v) = N_G^1(v)$ of a vertex is the set of all vertices in $V(G)$ that are adjacent to v. Similarly, we write $N_G^i(v)$ to denote the set of all vertices adjacent to any vertex in $N_G^{i-1}(v)$; we write $N_G^*(v) := N_G^1(v) \cup \dots \cup N_G^{|V(G)|}$ for the set of all vertices that are *reachable* from v. If all vertices are reachable from each other in G, we say that G is *connected*. We say a vertex is *isolated*, if it has no neighbors.

A *cycle* is a connected graph G that satisfies $|V(G)| = |E(G)|$ and every vertex in $V(G)$ has degree 2. We write C_n for the cycle with n vertices. Similarly, a *path* is a cycle with exactly one edge removed; we write P_n to denote the path with n vertices.[49]

49. While technically undefined, we also allow the path P_2 with two vertices and a single edge.

A *tree* is a connected graph T with $|V(T)| = |E(G)| + 1$. Next, we introduce a concept of "tree-likeness" of a graph. To that end, we define a tree decomposition of a graph G as follows: A *tree decomposition* of a graph G is a pair (T, b) of a tree T and a mapping b that maps every vertex t in $V(T)$ to a subset (a "bag") of vertices $b(t) \subseteq V(G)$, where

1 for each edge $\{u, v\}$ in $E(G)$ there is a bag that contains both u and v,
2 every vertex v in $V(G)$ appears in some bag,

3 for each vertex u in $V(G)$, the vertices of T whose bag contains u form a tree.

The *width* of a tree decomposition is the size of its largest bag *minus one*. The *treewidth* of a graph G is the minimal width of any tree decomposition of G. Consult Figure 7.3 for examples of most of the introduced graphs, as well as depictions of minimal tree decompositions of these graphs. We use the treewidth of a graph only in a black-box fashion; hence, consult the literature, for instance Cygan et al. [36, Chapter 7], for discussions as to why the depicted decompositions are indeed optimal.

One of the aforementioned black-box uses of the treewidth of a graph happens in the following well-known fact from extremal graph theory (and its applications later).

⌜ FACT 7.2 (FOLKLORE, [SEE FOR INSTANCE 24, COR. 1]). *Any graph G with an average degree of at least δ has a treewidth of at least $\delta/2$.*[50] ⌟

50. Compare Facts 7.1 and 7.2: Fact 7.1 implies a treewidth lower bound of at least r (see also Figure 7.3e). Hence, for small graphs ($n := |V(G)| < 32$), Fact 7.1 always yields a better bound, otherwise Fact 7.1 yields a better bound if r is chosen outside of $((n-q)/16, (n+q)/16)$; $q := (n^2 - 32n)^{1/2}$. For us, this is not relevant, though.

Another important parameter of a graph is the size of the smallest subgraph that is a cycle. We call this number the *girth* of a graph. In particular, we are interested in the *odd girth* of a graph, that is, the size of the smallest subgraph that is a cycle with an odd size; if no such odd cycle exists we set the (odd) girth to -1. Homomorphisms preserve the odd girth of a graph in the following sense.

⌜ LEMMA 7.3 (FOLKLORE). *For any two loop-free graphs H and G with $\#Hom(H \to G) > 0$, the odd girth of G is at most the odd girth of H.*

▭ PROOF. Observe that a loop-free graph H has an odd girth of at most γ if there is a homomorphism f from the cycle C_γ to H—we are done if f maps each vertex of C_γ to different vertex in H; otherwise, as H is loop-free, f cannot map adjacent vertices of C_γ to the same vertex in H. As γ is odd, H has thus some odd cycle of length at most $\gamma - 2$ as a subgraph.

Now, the concatenation of homomorphisms is again a homomorphism, so any homomorphism from a cycle C_γ to H yields a homomorphism from C_γ to G, yielding the claim. ⌟

Next, an *F-coloring* of a graph G is a graph homomorphism $c \in Hom(\, G \to F\,)$. We say that a graph G is *F-colorable* or allows a coloring into F if G has an (unspecified) F-coloring. If a graph G comes with an F-coloring $c \in Hom(\, G \to F\,)$, we say that G is *(F, c)-colored*. If the coloring is understood from the context, we also call G just F-colored. Further, we say that G is *k-colorable*, if G has a K_k-coloring. We call a 2-colorable graph *bipartite*; we say that the biclique $K_{a,b}$ a complete bipartite graph.

The *chromatic number* $\chi(G)$ of a graph G is the smallest k such that G is k-colorable. We say a graph G is *perfect*, if all induced subgraphs $G[\,F\,]$ of G satisfy $\chi(G[\,F\,]) = \omega(G[\,F\,])$. Again, homomorphisms preserve the chromatic of a graph in the following sense; note that the relation sign is flipped compared to Lemma 7.3.

⌜ LEMMA 7.4 (FOLKLORE). *For any two graphs H and G that satisfy $\#Hom(\, H \to G\,) > 0$, we have $\chi(H) \leq \chi(G)$.*

▭ PROOF. The concatenation of homomorphisms is again a homomorphism. Hence, using the homomorphism from H to G, any homomorphism from G to any K_k yields a homomorphism from H to K_k, yielding the claim. ⌟

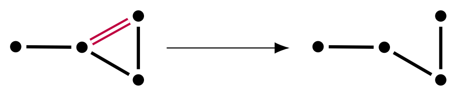
Note that Lemmas 7.3 and 7.4 together imply that there is no homomorphism between two graphs H and G if both the odd girth and the chromatic number are smaller in H than in G.

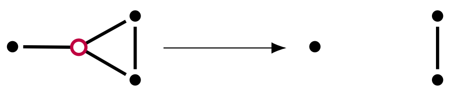
7.3 Operations on Graphs

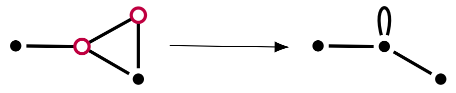
Throughout this part, on multiple occasions we *transform* graphs into different graphs. We discuss the operations used to accomplish those transformations next.

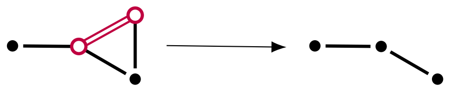
We start with the *deletion of edges* and *vertices*: We write $G \setminus \{u,v\} := G\{\,E(G) \setminus \{u,v\}\,\}$ for the deletion of the edge $\{u,v\}$ and $G \setminus u := G[\,V(G) \setminus \{u\}\,]$ for the deletion of the vertex u.

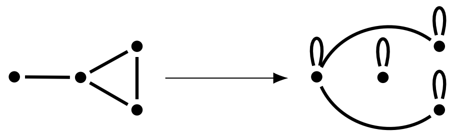
Similar, but different, is the *contraction* of two vertices u and v. Intuitively, the vertices u and v are merged into a single vertex uv,

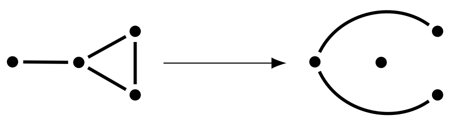

(a) The deletion of an edge.

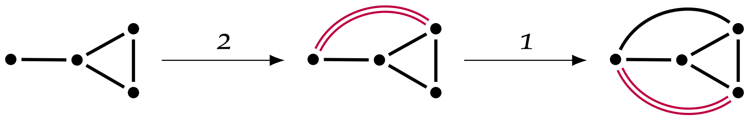

(b) The deletion of a vertex.


(c) The contraction of two vertices.

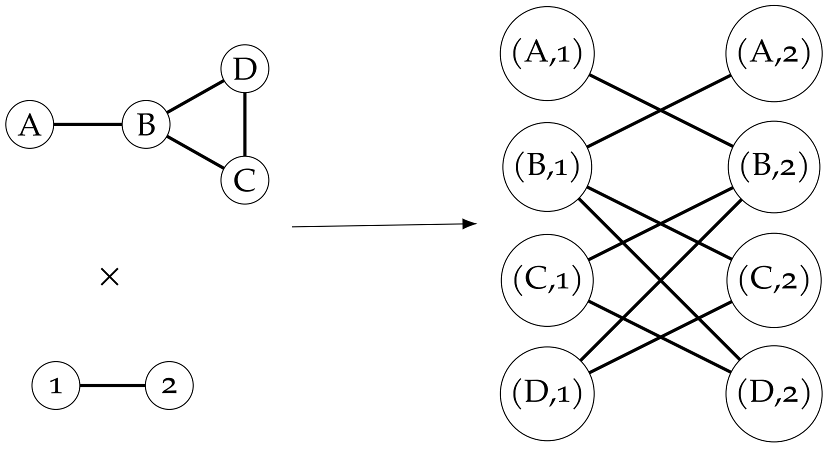

(d) The contraction of an edge.


(e) The looped complement.


(f) The complement.



(g) All possible edge extensions of ∙–◁. Written atop the arrows is the number of different edge extension that result in isomorphic graphs.



(h) The categorical product of two graphs. Note that the vertex labels are added only for illustrative purpose; the graphs are unlabeled.

□ FIGURE 7.4. The various operations on graphs illustrated.

any edges incident on u or v become incident on the new vertex uv, with duplicate edges removed. Formally, we set

$$\begin{aligned} V(G/^{\circ}\{u,v\}) &:= V(G) \setminus \{u,v\} \cup \{uv\} \\ E(G/^{\circ}\{u,v\}) &:= E(G[\,V(G) \setminus \{u,v\}\,]) \\ &\quad \cup \{\{t, uv\} \mid t \in (N_G(u) \cup N_G(v)) \setminus \{u,v\}\} \\ &\quad \cup \{\{uv, uv\} \mid \{u,v\} \in E(G)\}. \end{aligned}$$

Further, the contraction of an edge $\{u,v\}$ is the contraction of the vertices u and v, except that we delete the self-loop $\{uv,uv\}$: We write $G/\{u,v\} := G/^{\circ}\{u,v\} \setminus \{uv,uv\}$.[51] Consult Figures 7.4a to 7.4d for examples of the introduced operations.

51. Note that this operation is well-defined even if the edge $\{u,v\}$ did not exist in the first place. Hence, we could also define $G/\{u,v\} := (G \setminus \{u,v\})/^{\circ}\{u,v\}$.

Combining the previous operations, we say that a *minor* of a graph G is a graph that can be obtained via (repeated) deletions of vertices or edges from G or via contraction of (existing) edges of G.

Using the notion of a minor, we can understand graphs of high average degree even further.

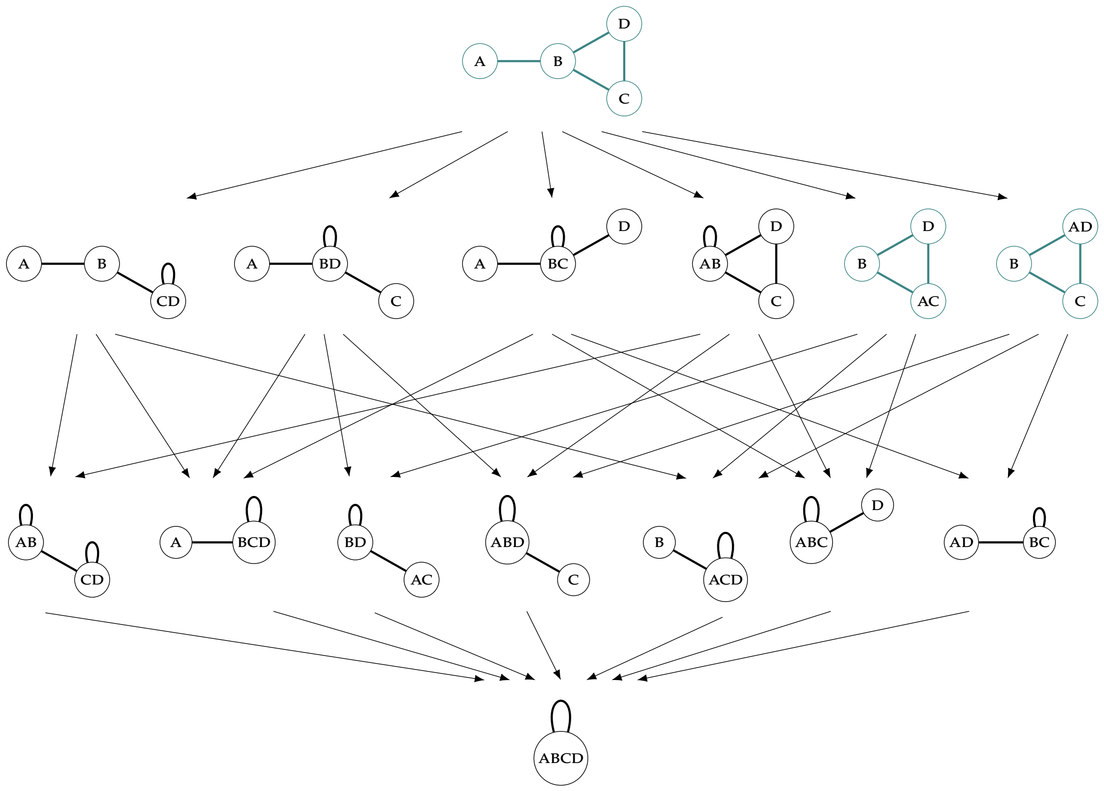
FACT 7.5 (KOSTOCHKA [75], THOMASON [114]). *Any graph with an average degree of at least $2.68\, r \log^{1/2} r$ has K_r as a minor.*[52]

52. The exact constant in Thomason [114] is $\alpha = 1 + \log(2\alpha)$; Compare Fact 7.5 with Facts 7.1 and 7.2. Fact 7.1 also yields a bound on the size of a clique minor; similar trade-offs as between Facts 7.1 and 7.2 apply. With Fact 7.2, the situation is more complicated: The treewidth of a graph is at least the treewidth of any of its minors, so Fact 7.5 yields a treewidth lower bound of $r - 1$. The other direction is *not known* to be true.

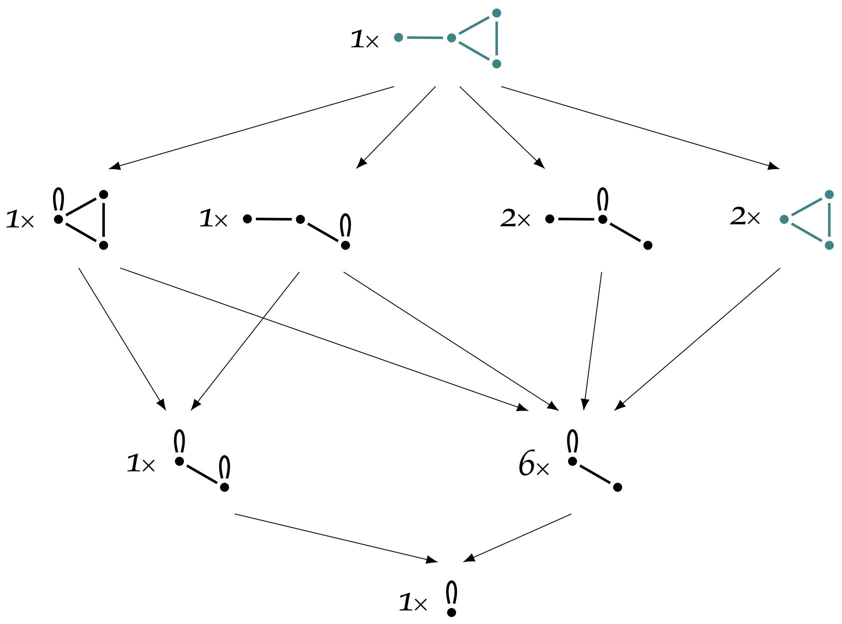
In a next step, we generalize the contraction of vertices to the contraction of sets of vertices. To that end, we say that for a set S, a *partition* ρ of S is a pairwise disjoint set of subsets of S (called *blocks* of ρ) whose union is again S. Now, for a partition ρ of $V(G)$, the *quotient graph* is the graph obtained from G by contracting all vertices in $V(G)$ that are contained in the same block of ρ. We write $G/^{\circ}\rho$. If we contract only vertices not connected by an edge we also write G/ρ for this operation. In contrast to notation of Curticapean et al. [35], the *spasms* of a graph G are all quotient graphs of G.[53] Consult Figure 7.5 for an example illustrating the spasms of ⋯◁.

53. In [35], the spasm contains only loop-free graphs, they search for patterns in loop-free graphs only.

Having discussed various operations to shrink a graph, we turn to operations that (potentially) enlarge graphs.



(a) The quotient graphs of •-◁. In each step, two (additional) vertices are contracted; vertex labels are added for illustrative purpose only. The loop-free spasm of •-◁ are colored in green.



(b) The isomorphism classes of quotient graphs of •-◁. In each step, two (additional) vertices are contracted. Isomorphic graphs are shown only once, next to the number of how often they appear. The loop-free spasms of •-◁ are colored in green.

□ FIGURE 7.5. The (isomorphism classes of) quotient graphs of •-◁.

We start with *edge extensions*: We write $G \cup S$ for the graph obtained from G by adding S to the set of edges:

$$V(G \cup S) := V(G),$$
$$E(G \cup S) := E(G) \cup S.$$

We write $\#Ext(H \to G)$ for the number of graphs isomorphic to G that can be obtained by adding edges to H. In other words

$$\#Ext(H \to G) := \#\{S \mid H \cup S \cong G\}.$$

Observe that we have $\#Ext(H \to G) = 0$ if H and G have a different number of vertices and that we have $\#Ext(H \to K_k) = 1$ for any graph H with k vertices. Consider Figure 7.4g for an illustration of an example.

Next, we define the *looped complement* of a graph G as the graph containing all edges that are not in $E(G)$; we write

$$\overline{G}^{\circ} := (V(G), \{\{u,v\} \notin E(G) \mid u,v \in V(G)\}).$$

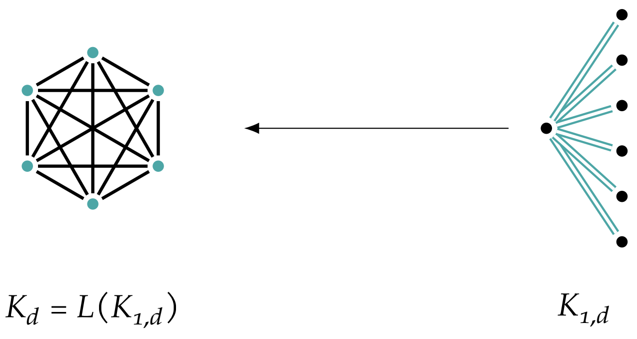
Removing the self-loops of $\overline{G}^{\circ}$, we obtain the *complement* of G; we write $\overline{G} := \overline{G}^{\circ} \setminus \{\{u,u\} \mid u \in V(G)\}$.[54] Consult Figures 7.4e to 7.4g for examples of the introduced operations. Finally, the following easy lemma reformulates our intuition of induced subgraphs (and strong embeddings) preserving both edges and non-edges.

54. We have $\overline{\overline{G}^{\circ}}^{\circ} = G$; also see that $\overline{\overline{G}}$ removes all self-loops from G.

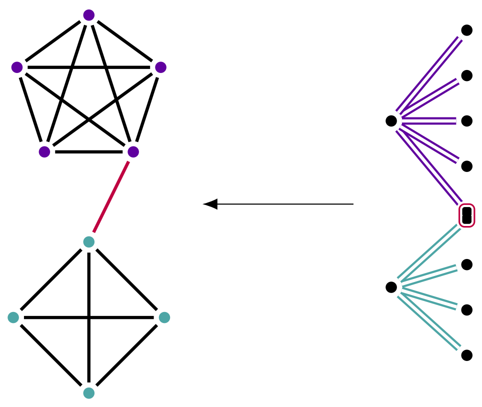
LEMMA 7.6 ([SEE 83, SECTION 5.2.3]). *For any graphs H, G, we have $\#IndSub(H \to G) = \#IndSub(\overline{H} \to \overline{G})$.*

PROOF. Fix a $h \in StrEmb(H \to G)$. For any vertices $u, v \in V(H)$, we have $\{u,v\} \in E(H) \Leftrightarrow \{h(u),h(v)\} \in E(G)$ and in particular $\{u,v\} \notin E(H) \Leftrightarrow \{h(u),h(v)\} \notin E(G)$. ◿

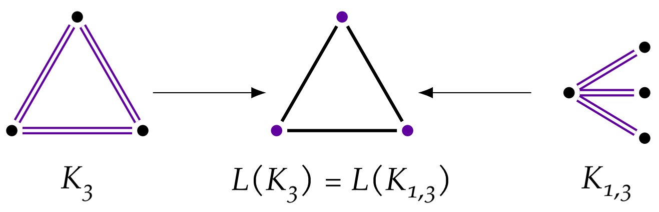
Another global operation on a graph is the *line graph* operation: For a graph G, we obtain its associated line graph $L(G)$ by taking the edges $E(G)$ as vertices of $L(G)$ and connecting two vertices



(a) A clique K_d of size d is the line graph of a star $K_{1,d}$ with d rays.



(b) Connecting two vertex disjoint cliques corresponds to merging vertices in the corresponding primal graphs. Note that the resulting primal graph stays bipartite.



(c) The claw $K_{1,3}$ and the triangle K_3 have the same line graph K_3.

Figure 7.6. Examples of line graphs.

in $V(L(G))$ if the corresponding edges in G are incident on exactly one common vertex. Consult Figure 7.6 for examples of line graphs.

Finally, the (*categorical*) *product* of two graphs G and H is the following graph:

$$\begin{aligned} V(G \times H) &:= \{(v_G, v_H) \mid v_G \in V(G), v_H \in V(H)\} \\ E(G \times H) &:= \{\{(u_G, u_H), (v_G, v_H)\} \\ &\qquad \mid \{u_G, v_G\} \in E(G) \text{ and } \{u_H, v_H\} \in E(H)\}. \end{aligned}$$

Consult Figure 7.4h for an example. It is well-known and easy to see that the function $\#Hom(\, H \to \star \,)$ is multiplicative with respect to $\times$.

⌜ LEMMA 7.7 (LOVÁSZ [83, EQUATION 5.30]). *For any three graphs F, G, H, we have $\#Hom(\, F \to G \times H \,) = \#Hom(\, F \to G \,) \cdot \#Hom(\, F \to H \,)$.*

▭ PROOF. Fix a homomorphism $h \in Hom(\, F \to G \times H \,)$ and write $(v_G, v_H) := h(v)$. By construction, the mappings $h_G := v \mapsto v_G$ and $h_H := v \mapsto v_H$ are homomorphisms from F to G and H, respectively.

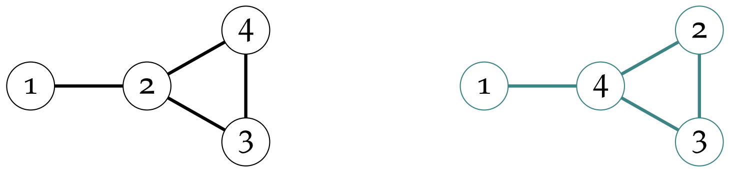
For the other direction, fix homomorphisms $h_G \in Hom(\, F \to G \,)$ and $h_H \in Hom(\, H \to G \,)$; again, by construction, the mapping $h := v \mapsto (h_G(v), h_H(v))$ is a homomorphism from F to $G \times H$. ⌟

Similarly, we can see that the product also preserves F-colorings.

⌜ LEMMA 7.8 (FOLKLORE). *For any three graphs F, G and H that satisfy $\#Hom(\, G \to F \,) + \#Hom(\, H \to F \,) > 0$, we have $\#Hom(\, G \times H \to F \,) > 0$.*

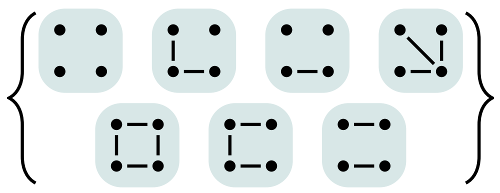
▭ PROOF. First consider the case when G is F-colorable and write $h \in Hom(\, G \to F \,)$ to denote an F-coloring of G. Observe that the mapping $(v_G, v_H) \mapsto h(v_G)$ is indeed an F-coloring of $G \times H$.

If H is F-colorable, the proof is symmetric. ⌟

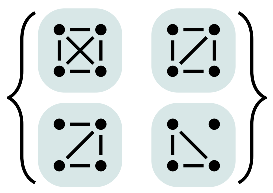


$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

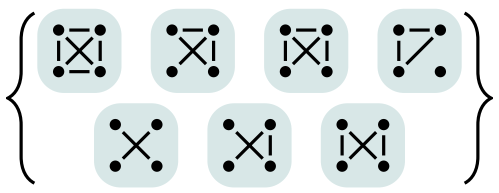
(a) Two labelings of ·-◁ and the corresponding adjacency matrices. The canonical labeling is highlighted in green.



(b) The graph class $\mathcal{G}_{\Phi,4}$.
The graphs are implicitly labeled.



(c) The graph class $\mathcal{G}_{\neg\Phi,4}$.
The graphs are implicitly labeled.



(d) The graph class $\mathcal{G}_{\overline{\Phi},4}$.
The graphs are implicitly labeled.

□ FIGURE 7.7. Two labelings of ·-◁; the classes $\mathcal{G}_{\Phi,4}$, $\mathcal{G}_{\neg\Phi,4}$ and $\mathcal{G}_{\overline{\Phi},4}$ for the graph property $\Phi(G) = 1 :\Leftrightarrow G$ is bipartite. Note that the only graph ·-◁ itself is an induced subgraph of ·-◁ with four vertices, so we have $\#\mathit{IndSub}(\Phi, 4 \to$ ·-◁$) = 0$, $\#\mathit{IndSub}(\neg\Phi, 4 \to$ ·-◁$) = 0$, and $\#\mathit{IndSub}(\overline{\Phi}, 4 \to$ ·-◁$) = 0$.

7.4 Graph Classes and Graph Properties

In this part, we seldom consider graphs in isolation—Most of the time, we are concerned with sets of graphs that share a common property. To make these notions formal, we first need to take a step back and discuss how we *encode* graphs.

First, for a graph G, a *labeling* is a bijection between $V(G)$ and $[\,|V(G)|\,]$. Next, the *adjacency matrix* of G for a labeling $\ell : V(G) \leftrightarrow [\,|V(G)|\,]$ is a symmetric matrix $A_{G,\ell} \in \{0,1\}^{|V(G)|\times|V(G)|}$, where

$$A_{G,\ell}[\,\ell(u),\ell(v)\,] = A_{G,\ell}[\,\ell(v),\ell(u)\,] := \begin{cases} 1, & \{u,v\} \in E(G) \\ 0, & \text{otherwise.} \end{cases}$$

Linearizing adjacency matrices by diagonal,[55] we can sort them lexicographically; now a *canonical labeling* of G is a labeling that corresponds to an adjacency matrix that is lexicographically a smallest matrix, which we call a canonical adjacency matrix A_G of G. We say a graph is *unlabeled*, if it is labeled with any canonical labeling. [56] Consult Figure 7.7a for an example.

55. That is as $A[\,1,1\,]$, $A[\,2,1\,]$, $A[\,1,2\,]$, ...

56. Observe that for a clique, all adjacency matrices look the same, so lexicographically smallest adjacency matrices may not be unique for a graph. Further, we do not want to *compute* canonical labelings, as that would most likely be infeasible. Hence, most of the time, we abuse notation and write G for the canonical adjacency matrix A_G.

Observe that two isomorphic graphs have the same canonical adjacency matrix—this is intentional, as we want graph properties to be *independent* of specific labelings. Specifically, a graph property Φ is a function from canonical adjacency matrices to $\{0,1\}$. We abuse notation and write $\Phi(G)$ for $\Phi(A_G)$. Further, we say that a graph G *satisfies* Φ if $\Phi(G) = 1$; we call the set of all graphs satisfying Φ the *graph class* $\mathcal{G}_\Phi$ of Φ. Again, we abuse notation and say that a graph G is in $\mathcal{G}_\Phi$ if $\Phi(G) = 1$.

For a positive integer k and a graph property Φ, we write $\mathcal{G}_{\Phi,k}$ for the class of all (canonical adjacency matrices of) graphs with k vertices that satisfy Φ. Further, given a graph G, a positive integer k, and a graph property Φ, we write $\mathit{IndSub}(\,\Phi,k \to G\,)$ for the set of all induced subgraphs with k vertices of G that satisfy Φ. We write $\#\mathit{IndSub}(\,\Phi,k \to \star\,)$ for the function that maps a graph G to $\#\mathit{IndSub}(\,\Phi,k \to G\,)$.

Given a graph property Φ, we define $\neg\Phi(H) = 1 :\Leftrightarrow \Phi(H) = 0$ as the *negation* of Φ. Further, we define $\overline{\Phi}(H) = 1 :\Leftrightarrow \Phi(\overline{H}) = 1$ as the *inverse* of Φ.[57] The negation and inverse of a graph class are defined accordingly. Consult Figures 7.7b to 7.7d for an example. We have the following (easy) equalities.

57. We explicitly do not use the word "complement" for graph properties to avoid possible confusions between $\neg\Phi$ and $\overline{\Phi}$.

Lemma 7.9. *For every graph property Φ, graph G, and positive integer k, we have*

$$\#\mathit{IndSub}(\,\neg\Phi, k \to G\,) = \binom{|V(G)|}{k} - \#\mathit{IndSub}(\,\Phi, k \to G\,),\ \textit{and}$$
$$\#\mathit{IndSub}(\,\overline{\Phi}, k \to G\,) = \#\mathit{IndSub}(\,\Phi, k \to \overline{G}\,).$$

Proof. The first equation is immediate. For the second equation, observe that

$$\begin{aligned}\#\mathit{IndSub}(\,\overline{\Phi}, k \to G\,) &= \sum_{H \in \overline{\Phi}_k} \#\mathit{IndSub}(\,H \to G\,)\\ &= \sum_{\overline{H} \in \Phi_k} \#\mathit{IndSub}(\,H \to G\,)\\ &= \sum_{\overline{H} \in \Phi_k} \#\mathit{IndSub}(\,\overline{H} \to \overline{G}\,)\\ &= \#\mathit{IndSub}(\,\Phi, k \to \overline{G}\,),\end{aligned}$$

where we use Lemma 7.6 in the third step. ⌟

We conclude by listing the most relevant types of graph classes used throughout this part.

- We write $\top$ for the graph class of all graphs, that is, for the graph class corresponding to the property that is always 1.
- A class is *non-trivial* if for infinitely many k, the class and its negation contain a graph with k vertices.
- A class is *monotone*, if it is closed under taking subgraphs. A prominent example is the class of all bipartite graphs (or in general all F-colorable graphs) [see 83, Section 4.1].

- A class is *hereditary*, if it is closed under taking induced subgraphs. Prominent example are the class of all line graphs and the class of all perfect graphs [see 83, Section 4.1].
- A class is *minor-closed*, if it is closed under taking minors. A prominent example is the class of all planar graphs [see 83, Section 4.1].

7.5 The Space of Graph Motif Parameters

In this section, we generalize the notion of a graph property—A *graph parameter* is a function from $\top$ to $\mathbb{R}$. In fact, we have already seen multiple functions on graphs, that can be recast as graph parameters: the treewidth $\mathrm{tw}(\star)$, the chromatic number $\chi(\star)$, and the size of the largest complete subgraph $\omega(\star)$. In particular, the functions $\#Hom(H \to \star)$, $\#Emb(H \to \star)$, $\#Sub(H \to \star)$, and so on, can be seen as graph parameters for any fixed graph H.

We now follow the general framework by Lovász [83], and the interpretation given by Curticapean et al. [35]. First observe that we can order the canonical adjacency matrices in $\top$ lexicographically by their linearization. Now, we can interpret $\#Hom$, $\#Sub$, $\#IndSub$, and so on, as infinite matrices with indices from $\top \times \top$ and entries in $(\mathbb{N} \cup \{0\})$, where the indices are ordered according to the aforementioned ordering on $\top$.

As in [35] we now define *graph motif parameters* as finite linear combinations of induced subgraph numbers. We represent such linear combinations as infinite vectors in $\mathbb{R}^{\top}$ with finite support.[58]

58. Matrix-matrix and matrix-vector multiplication generalize naturally.

Definition 7.10 (Curticapean et al. [35]). *A graph parameter $f : \top \to \mathbb{R}$ is a* graph motif parameter, *if there is a vector $\alpha \in \mathbb{R}^{\top}$ of finite support such that $f = \alpha \cdot \#IndSub$.*[59]

59. Note that we interpret both α and f as row vectors here.

As in [35], we define a scalar product on graph motif parameters via the natural scalar product on the corresponding vectors from $\mathbb{R}^{\top}$: For two finitely supported vectors $\alpha, \beta \in \mathbb{R}^{\top}$, we set

$$\langle \alpha, \beta \rangle := \sum_{F \in \top} \alpha_F \cdot \beta_F.$$

For two graph motif parameters $f = \alpha \cdot \#IndSub, g = \beta \cdot \#IndSub$, we set $\langle f, g \rangle := \langle \alpha, \beta \rangle \cdot \#IndSub$. Now, the set of all graph motif parameters together with scalar multiplication and point-wise addition form an infinite-dimensional vector space—the finitely supported row-span of the matrix *#IndSub*.

Relations between Graph Motif Parameters

For us, it is easier to work with homomorphisms than with induced subgraphs. In next two steps, we hence show that the row-spans of *#Hom*, *#Sub*, and *#IndSub* are in fact the same. For the basis changes, we now take a slightly different route than Curticapean et al. [35] which is closer to the route of Lovász [83]: In [35], the first step is to move from *#IndSub* to *#Sub* and then via *#Emb* to *#Hom*. For analyzing values in the resulting basis transformation matrices, it is more convenient to us to first move from *#IndSub* to *#StrEmb* and then again via *#Emb* to *#Hom*.[60]

Let us start with the change from *#IndSub* to *#StrEmb*. Using Equation (7.2) and setting $Aut := diag(\#Aut(F))$, we directly obtain $\#StrEmb = \#Aut \cdot \#IndSub$, and indeed also $\#IndSub = \#Aut^{-1} \cdot \#StrEmb$, as every graph has at least one automorphism (the trivial automorphism) and hence the diagonal matrix *#Aut* is invertible.[61]

As a next, we tackle the basis change from *#StrEmb* to *#Emb*. In other words, for a graph H, we want to express $\#StrEmb(H \to \star)$ as a linear combination of functions $\#Emb(F \to \star)$. To that end, consider an embedding $e \in Emb(H \to G)$ that is not also a strong embedding from H to G. This means that there is (at least) one non-edge $\{u, v\} \notin E(H)$ such that the edge $\{e(u), e(v)\}$ exists in G.

60. Note that thereby, we indirectly show that the expressions for the entries of the basis transformation matrices in [35] are equivalent to the corresponding terms that we obtain.

61. Note that by using Equation (7.1), we can also obtain $\#Emb = \#Aut \cdot \#Sub$.

In particular, e corresponds to a strong embedding from an edge extension of H. This immediately yields

$$\#Emb(H \to \star) = \sum_{R\,:\,\#Ext(H \to R)>0} \#Ext(H \to R) \cdot \#StrEmb(R \to \star), \quad (7.3)$$

or $\#Emb = \#Ext \cdot \#StrEmb$ in matrix notation.[62] Now observe that we have $\#Ext(G \to G) = 1$, and that the matrix $\#Ext$ is an upper triangular matrix—Hence, any finite principal submatrix[63] of $\#Ext$ is invertible and thus we obtain $\#StrEmb = \#Ext^{-1} \cdot \#Emb$. While this shows that $\#Emb$—and hence $\#Sub$—indeed spans the space of graph motif parameters, it is useful to us to actually compute $\#Ext^{-1}$. Using the Inclusion-Exclusion Principle we obtain

62. Note that the notation of edge extensions Ext that we use is not to be confused with the notion of extensions used in [35].

63. That is, any square submatrix with the same row and column indices.

$$\begin{aligned}&\#StrEmb(H \to \star)\\ &\quad = \sum_{R\,:\,\#Ext(H \to R)>0} (-1)^{\#E(R)-\#E(H)}\, \#Ext(H \to R) \cdot \#Emb(R \to \star).\end{aligned} \quad (7.4)$$

Consult again Figures 7.1b and 7.1c for a small example.

In a second step, following Borgs, Chayes, Lovász, Sós, and Vesztergombi [16], we now tackle the basis change from $\#Emb$ to $\#Hom$. In other words, for a graph H, we want to express the function $\#Emb(H \to \star)$ as a linear combination of $\#Hom(F \to \star)$. Consider a homomorphism h from a graph H to a graph G that is not injective, and write ρ for the partition of the vertices where two vertices are in the same block if they are mapped to the same vertex by h. Observe that the homomorphism $h/^\circ\rho : H/^\circ\rho \to G$ is injective.[64] Hence, we obtain

64. That is, the homomorphism that maps vertices corresponding to a block B of ρ to the vertex $h(v), v \in B$ (which is the same for every $v \in B$ by definition of ρ).

$$\begin{aligned}\#Hom(H \to \star) &= \sum_{\rho} \#Emb(H/^\circ\rho \to \star) && (7.5)\\ &= \sum_{\rho} \#Aut(H/^\circ\rho) \cdot \#Sub(H/^\circ\rho \to \star)\\ &= \sum_{F \in \top} \sum_{\substack{\rho \\ F \cong H/^\circ\rho}} \#Aut(F) \cdot \#Sub(F \to \star) && (7.6)\end{aligned}$$

Before we turn to reversing Equation (7.5), let us quickly discuss another interpretation of Equation (7.6). Observe that automorphisms of a quotient graph $H/{}^\circ\rho$ one-to-one correspond to *surjective* homomorphisms from H.[65] Write $\#Surj(\,H \to F\,)$ for the number of surjective homomorphisms from H to F. We obtain

65. For us, a surjective homomorphism is surjective both on the vertices *and on the edges.*

$$\#Surj(\,H \to F\,) = \sum_{\substack{\rho \\ F \cong H/{}^\circ\rho}} \#Aut(\,F\,). \tag{7.7}$$

Combining Equations (7.6) and (7.7) yields

$$\#Hom(\,H \to \star\,) = \sum_{F \in \top} \#Surj(\,H \to F\,) \cdot \#Sub(\,F \to \star\,), \tag{7.8}$$

or $\#Hom = \#Surj \cdot \#Sub$ in matrix form. In particular, observe that $\#Surj$ is an upper triangular matrix with non-zero main diagonal and is thus invertible—we have $\#Sub = \#Surj^{-1} \cdot \#Hom$. We can extend this argument further to obtain the following useful fact.

⌜ Lemma 7.11 ([35, Lemma 3.3]). *For any finite set of graphs P closed under surjective homomorphisms, the principal submatrix $\#Hom_P$ is invertible.*

▭ Proof. Observe that only graphs in P contribute in the sum of Equation (7.8). Further, $\#Surj_P$ is an upper triangular (square) matrix and $\#Sub$ is a lower triangular (square) matrix; hence, there product—$\#Hom_P$—is invertible. ⌟

Next, to understand the matrix $\#Surj^{-1}$—and thereby reverse Equation (7.5)—we turn to so-called *Möbius inversion*, a generalization of the Inclusion-Exclusion Principle. In particular, we need the following special case, which is due to Schützenberger and independently Frucht and Rota [see 98, page 359].

⌜ FACT 7.12 (FRUCHT, ROTA, SCHÜTZENBERGER [98, PAGE 359]). *For two partitions ρ, π of a set S, we write $\pi \geq \rho$ if every block of ρ is a subset of a block of π. Further, we write $\#\rho$ for the number of blocks of ρ and $\perp := \{\{s\} \mid s \in S\}$ for the discrete partition of S.*

For any functions f and g mapping partitions of S to the reals with[66]

$$f(\rho) = \sum_{\pi \geq \rho} g(\pi),$$

we have

$$g(\perp) = \sum_{\pi} (-1)^{\#S - \#\pi} \prod_{B \in \pi} (\#B - 1)!\, f(\pi).$$ ⌟

66. The function f is also called *upward zeta transformation* of g of the partition lattice. Consult for instance the textbook by Stanley [111] for an introduction to lattices and related concepts.

Using $H = H/^{\circ}\perp$, Fact 7.12 yields the reverse of Equation (7.5):

$$\#\mathit{Sub}(\, H \to \star\,) = \frac{1}{\#\mathit{Aut}(\, H\,)} \sum_{\rho} (-1)^{\#V(H) - \#\rho} \prod_{B \in \rho} (\#B - 1)! \cdot \#\mathit{Hom}(\, H/^{\circ}\rho \to \star\,). \tag{7.9}$$

As a special case of Equation (7.9), we are interested in the homomorphism coefficient $\mathsf{a}_{\mathit{Sub}(\, H \to \star\,)}(\, G\,)$ of a single graph $G = H/^{\circ}\rho$:[67]

67. There is no particular reason for the name a.

$$\mathsf{a}_{\mathit{Sub}(\, H \to \star\,)}(\, G\,) := \frac{(-1)^{\#V(H) - \#V(G)}}{\#\mathit{Aut}(\, H\,)} \cdot \sum_{\substack{\rho \\ H/^{\circ}\rho \cong G}} \prod_{B \in \rho} (\#B - 1)!. \tag{7.10}$$

Note that Equation (7.10) implicitly defines a basis transformation matrix from $\#\mathit{Sub}$ to $\#\mathit{Hom}$—$\#\mathit{Surj}^{-1}$. The crucial observation of Curticapean et al. [35] now is the following corollary.

Corollary 7.13. *We have* $\mathsf{a}_{Sub(\,H\to\star\,)}(\,G\,) \neq 0$ *if and only if*

$$\sum_{\substack{\rho\\ H/^{\circ}\rho\cong G}} 1 = \#Surj(\,H\to G\,)/\#Aut(\,G\,) \neq 0;$$

in other words, whenever G is in the spasm of H.

Finally, let us discuss the entries $\mathsf{a}_{IndSub(\,H\to\star\,)}(\,G\,)$ of the basis transformation matrix from *#IndSub* to *#Hom*. Combining Equations (7.4) and (7.10) with Equations (7.1) and (7.2), we obtain

$$\begin{aligned}
&\mathsf{a}_{IndSub(\,H\to\star\,)}(\,G\,)\\
&\quad= \sum_{R\,:\,\#Ext(\,H\to R\,)>0} \frac{(-1)^{\#E(R)-\#E(H)}\,\#Ext(\,H\to R\,)}{\#Aut(\,H\,)}\cdot \mathsf{a}_{Sub(\,R\to\star\,)}(\,G\,)\\
&\quad= \sum_{R\,:\,\#Ext(\,H\to R\,)>0} (-1)^{\#E(R)-\#E(H)}\,\#Ext(\,H\to R\,)\\
&\qquad\quad\cdot\frac{(-1)^{\#V(R)-\#V(G)}}{\#Aut(\,H\,)}\cdot\sum_{\substack{\rho\\ R/^{\circ}\rho\cong G}}\prod_{B\in\rho}(\#B-1)!.
\end{aligned}\tag{7.11}$$

In particular, we are interested to find out when $\mathsf{a}_{IndSub(\,H\to\star\,)}(\,G\,)$ is nonzero—A major fraction of this part is dedicated to (partially) solve just this “simple” task.

Let us conclude by discussing some easy, but useful results in that direction.

Lemma 7.14. *For any graphs G and H with* $|V(G)| < |V(H)|$, *we have* $\mathsf{a}_{IndSub(\,G\to\star\,)}(\,H\,) = 0$.

Proof. Observe in Equation (7.11) that only quotient graphs of extensions of G may contribute to $\mathsf{a}_{IndSub(\,G\to\star\,)}(\,H\,)$ and only if they are isomorphic to H. However, as both the (edge) extension and the quotient operation do not add new vertices, none of them can be isomorphic to G and hence $\mathsf{a}_{IndSub(\,G\to\star\,)}(\,H\,) = 0$.

Almost Everything You Need to Know About Parameterized Counting Problems

8

Recall that in [7], we defined the coefficient $\mathsf{a}_{IndSub(H\to\star)}(G)$ for graphs H, G and asked when it is nonzero. As a major part of this chapter, we discuss why this question is interesting for us—that is, we discuss how the question relates to showing (conditional) running time lower bounds for algorithms solving the problems of counting (induced) subgraphs in a graph. We start with a formal definition of the aforementioned problems and a gentle introduction to *Parameterized Complexity Theory*, the setting in which we consider these problems.

8.1 Parameterized (Promise) Problems

Let us start with recalling basic concepts from computational complexity theory. We say a function $f : A \to B$ is *computable*, if there is a Turing machine that on any (binary encoded) input $a \in A$ outputs $f(a) \in B$ and halts. Unless stated otherwise, the functions we consider are computable. We say a set $S \subseteq (\mathbb{N} \cup \{0\})$ is *computable, decidable*, or *recursive* if its indicator function $\mathbb{1}_S : (\mathbb{N} \cup \{0\}) \to \{0, 1\}$, $\mathbb{1}_S(s) := 1 \Leftrightarrow s \in S$ is computable.[68] We use the bijection between (isomorphism classes of) graphs $\top$ to $(\mathbb{N} \cup \{0\})$ from Section 7.5 and similarly say that a graph class is recursive, if its indicator function is computable. We say a set $S \subseteq (\mathbb{N} \cup \{0\})$ is *computably enumerable, semidecidable*, or *recursively enumerable* if it is the domain of a computable function.[69] We use the same notion for graph classes.

A *parameterization* κ is a polynomial-time computable function from the set $\{0, 1\}^*$ to $(\mathbb{N}\setminus\{0\})$.[70] A *parameterized counting problem* is a pair of a function $P : \{0, 1\}^* \to (\mathbb{N}\cup\{0\})$ and a parameterization κ. We call strings in $\{0, 1\}^*$ *instances* of (P, κ). We say a parameterized counting problem (P, κ) is *fixed-parameter tractable* (FPT) if there is

68. Equivalently, we can require that a Turing machine, on input $(\mathbb{N} \cup \{0\})$, accepts on any input $s \in S$ and rejects any other input.

69. Equivalently, we can require that a Turing machine accepts on any input $s \in S$ but it need not terminate on any other input.

70. We require parameterizations to be polynomial-time computable for technical reasons. See [47, Chapter 1.2] for a detailed discussion of this issue.

a computable function f such that we can solve any instance x of (P, κ) in time $f(\kappa(x)) \cdot |x|^{O(1)}$, where $|x|$ is size of the instance.

As an important example for a parameterized counting problem, consider #Clique—Given a graph G and a positive integer k, the task is to compute $\#Sub(K_k \to G) = \#IndSub(K_k \to G)$, the number of complete k-vertex subgraphs of G. We parameterize the problem #Clique by the pattern size k; we have $\kappa(G, k) := k$.

In this part, we study generalizations of #Clique. First, for a computable graph property Φ,[71] in #IndSub(Φ), we are given a graph G and a positive integer k, and the task is to compute the number $\#IndSub(\Phi, k \to G)$—the number of induced k-vertex subgraphs in G that satisfy Φ. We parameterize #IndSub(Φ) by k.

71. We use graph properties and not graph classes in the definition for historical reasons.

Observe that we can solve both #IndSub(Φ) and #Clique by brute-force in time $f(k) \cdot O(n^k)$ by iterating over all subsets of k vertices in G and counting which of the subsets induce a graph that satisfies Φ (or are a complete graph). Note that we can test membership in Φ in some time $f(k)$ for a computable f, as Φ is a computable graph property.[72]

72. For #Clique, the exponent k can be reduced using fast matrix multiplication.

Second, for graph classes $\mathcal{H}$ and $\mathcal{G}$, in #Hom($\mathcal{H} \to \mathcal{G}$), we are given graphs $H \in \mathcal{H}$ and $G \in \mathcal{G}$ and the task is to compute the number $\#Hom(H \to G)$—the number of homomorphisms from H to G. We parameterize by the size of H.

However, this definition of #Hom($\mathcal{H} \to \mathcal{G}$) is informal in the sense that it leaves out the specification of the output if the input is *invalid*, that is, if the graph H does not belong to the class $\mathcal{H}$ or the graph G does not belong to the class $\mathcal{G}$.

As a naive option to solve this issue, consider requiring the output to be 0 if the input is invalid. However, in this case we exclude a plethora of interesting cases from our studies, as even seemingly trivial instances might encode computationally infeasible (read ***NP***-hard) problems. Consider for example the class Δ of 3-colorable graphs. Following the naive option, the problem #Hom($\mathcal{H} \to \Delta$) becomes ***NP***-hard even if the class $\mathcal{H}$ contains only the graph K_1 consisting of a single vertex: #Hom($\{K_1\} \to \Delta$) encodes the 3-colorability problem: An instance (K_1, G) of the problem #Hom($\{K_1\} \to \Delta$) is mapped to 0 if and only if G is 3-colorable.

In particular, if $\#\textsc{Hom}(\{K_1\} \to \Delta)$ was FPT, this would yield an algorithm running in time $f(|K_1|) \cdot |V(G)|^{O(1)}$ which is polynomial in $|V(G)|$, and thus such an algorithm would imply $\boldsymbol{P} = \boldsymbol{NP}$. In sharp contrast, the number of homomorphisms from the graph K_1 to any graph G is just the number of vertices $|V(G)|$ and thus the hardness of the problem $\#\textsc{Hom}(\{K_1\} \to \Delta)$ stems only from enforcing invalid inputs to be mapped to zero.

Another option of solving the issue of invalid instances of the problem $\#\textsc{Hom}(\mathcal{H} \to \mathcal{G})$ reads as follows: If a given instance (H, G) consists of graphs $H \in \mathcal{H}$ and $G \in \mathcal{G}$, then we are supposed to compute the number of homomorphisms $\#Hom(\,H \to G\,)$ correctly. Otherwise, we may output *any* number. Formally, this requires us to model the problem $\#\textsc{Hom}(\mathcal{H} \to \mathcal{G})$ as a *promise problem*.[73]

[73] Goldreich [53, Chapter 2.4.1] states that "promise problems offer the most direct way of formulating natural computational problems." Indeed, some of the most striking results in complexity theory implicitly rely on promise problems. Examples are "gap problems" and "uniqueness promises"; we refer the reader to [53, Chapter 2.4.1.2] for a discussion.

DEFINITION 8.1. *A* parameterized promise counting (PPC) *problem is a triple (P, κ, Π) of a function $P : \{0, 1\}^* \to (\mathbb{N} \cup \{0\})$, a parameterization $\kappa : \{0, 1\}^* \to (\mathbb{N} \setminus \{0\})$, and a promise $\Pi \subseteq \{0, 1\}^*$.*

We say that a PPC problem (P, κ, Π) is computable in time t if there is a deterministic algorithm $\mathbb{A}$ that satisfies

1. On input $x \in \{0, 1\}^*$, the algorithm $\mathbb{A}$ runs in time $t(|x|)$.
2. On input $x \in \Pi$, the algorithm $\mathbb{A}$ outputs $P(x)$.

We call strings in $\{0, 1\}^*$ *instances* of (P, κ, Π) and we call all instances $x \in \Pi$ *valid*. We say a PPC problem (P, κ, Π) is FPT if there is a computable function f such that we can solve any instance x in time $f(\kappa(x)) \cdot |x|^{O(1)}$.

Observe that we recover the definition of parameterized counting problems and their fixed-parameter tractability if we set the promise Π to $\{0, 1\}^*$.

Further, observe that we obtain parameterized promise *decision* problems from Definition 8.1 by restricting the image of the function P to $\{0, 1\}$. In this case, Definition 8.1 coincides with the standard definition of (parameterized) promise problems (see for instance Definition 3.1 in the full version [11] of [10]).

Finally, we can give a formal definition of $\#\textsc{Hom}(\mathcal{H} \to \mathcal{G})$. For graph classes $\mathcal{H}$ and $\mathcal{G}$, in the PPC problem $\#\textsc{Hom}(\mathcal{H} \to \mathcal{G})$, we are given graphs $H \in \mathcal{H}$ and $G \in \mathcal{G}$, and the task is to compute the number of homomorphisms $\#\mathit{Hom}(\, H \to G\,)$. The parameter is $|V(H)|$ and the promise is the set of all (encodings of) pairs $(H, G) \in \mathcal{H} \times \mathcal{G}$.

For simpler and shorter proofs, we also define a colored variant $\#\textsc{cp-Hom}(\mathcal{H} \to \mathcal{G})$—Given $H \in \mathcal{H}$, and an (H, c)-colored $G \in \mathcal{G}$, the task is to compute the number of color-prescribed homomorphisms $\#\mathit{cp\text{-}Hom}(H \to G)$. Again, the parameter is $|V(H)|$ and the promise is the set of all (encodings of) pairs $(H, G) \in \mathcal{H} \times \mathcal{G}$.

In the same vein, we define the corresponding *decision problems* $\textsc{Hom}(\mathcal{H} \to \mathcal{G})$ and $\textsc{cp-Hom}(\mathcal{H} \to \mathcal{G})$—the difference is that we should output 1 if $\#\mathit{Hom}(\, H \to G\,)$ or $\#\mathit{cp\text{-}Hom}(H \to G)$ is positive.

Finally, observe that if membership of a graph in a class $\mathcal{G}$ can be tested in polynomial time and the class $\mathcal{H}$ is recursive, then there is no need to define the problem $\#\textsc{Hom}(\mathcal{H} \to \mathcal{G})$ as a promise problem. Instead, we can define the output to be 0 if a given pair (H, G) is not contained in $\mathcal{H} \times \mathcal{G}$; indeed $H \in \mathcal{H}$ can be verified in time $f(H)$ for some computable function f as, by assumption, $\mathcal{H}$ is recursive. In particular, the problems $\#\textsc{IndSub}(\Phi)$ and $\#\textsc{Clique}$) can be assumed to have no promises, as we do not restrict the graphs in which we search for the corresponding patterns.

8.2 Reductions and (Conditional) Hardness

A central concept in computational complexity theory is the notion of *reductions* between problems. For brevity, we discuss only reductions for PPC problems, the notions easily extend to decision problems or problems without promises.

Definition 8.2 (Parameterized Turing reductions). *For two* PPC *problems* (P, κ, Π) *and* (Q, λ, Y), *a* parameterized Turing reduction *from* (P, κ, Π) *to* (Q, λ, Y) *is a pair* $(\mathbb{A}^Q, (f, s))$ *of an algorithm* $\mathbb{A}^Q$ *that has oracle access to the function* Q *and a pair of computable functions* (f, s) *such that:*

1 *On input $x \in \{0,1\}^*$, the algorithm $\mathbb{A}^Q$ runs in time $f(\kappa(x)) \cdot |x|^{O(1)}$.*
2 *On input $x \in \Pi$, the algorithm $\mathbb{A}^Q$ computes the function $P(x)$.*
3 *On input $x \in \Pi$, the algorithm $\mathbb{A}^Q$ queries the oracle only on strings y with $y \in Y$ and $\lambda(y) \leq s(\kappa(x))$.*

We write $(P,\kappa,\Pi) \leq_T^{fpt} (Q,\lambda,Y)$ if such a reduction exists. If $\mathbb{A}^Q$ even runs in time $f(\kappa(x)) \cdot |x|$, we call the reduction linear[74] *and write $(P,\kappa,\Pi) \leq_T^{\ell\text{-}fpt} (Q,\lambda,Y)$.*[75] ⌟

74. Recall, that we assume that oracle queries take constant time.

75. Note that this also implies that the oracle is called on instances that are only linearly (in $|x|$) larger than $|x|$.

⌜ DEFINITION 8.3 (PARAMETERIZED (WEAKLY) PARSIMONIOUS REDUCTIONS). *For* PPC *problems (P,κ,Π) and (Q,λ,Y) a* parameterized weakly parsimonious reduction *from (P,κ,Π) to (Q,λ,Y) is a pair $(\mathbb{A},(f,g,s))$ of a deterministic algorithm $\mathbb{A}$ and a triple of computable functions (f,g,s) such that:*

1 *For all valid instances $x \in \Pi$, the algorithm $\mathbb{A}$ outputs a valid instance of (Q,λ,Y), that is $\mathbb{A}(x) \in Y$.*
2 *We can compute $P(x)$ by computing Q on the computed instance $\mathbb{A}(x)$ and the function $g(x)$; in particular, we have $P(x) = g(x) \cdot Q(\mathbb{A}(x))$.*
3 *The* PPC *problem (g,κ,Π) is fixed-parameter tractable.*
4 *On input $x \in \{0,1\}^*$, the algorithm $\mathbb{A}$ runs in time $f(\kappa(x)) \cdot |x|^{O(1)}$.*
5 *For all $x \in \{0,1\}^*$, the parameter of the instance $\mathbb{A}(x)$ is bounded by $s(\kappa(x))$, that is $\lambda(\mathbb{A}(x)) \leq s(\kappa(x))$.*

We write $(P,\kappa,\Pi) \leq^{w\text{-}fpt} (Q,\lambda,Y)$ if such a reduction exists. If g is the identity function on Π, then we call the reduction parsimonious *and we write $(P,\kappa,\Pi) \leq^{fpt} (Q,\lambda,Y)$. If $\mathbb{A}$ even runs in time $f(\kappa(x)) \cdot |x|$, we call the reduction* linear *and write $(P,\kappa,\Pi) \leq^{w\ell\text{-}fpt} (Q,\lambda,Y)$ and $(P,\kappa,\Pi) \leq^{\ell\text{-}fpt} (Q,\lambda,Y)$.*[76] ⌟

76. Note that this also implies that $\mathbb{A}$ "blows up" an instance x only linearly (in $|x|$).

We can easily see how the defined notions of reductions relate to each other.

◰ LEMMA 8.4. *For PPC problems (P, κ, Π) and (Q, λ, Υ), we have*

$$\begin{aligned}(P, \kappa, \Pi) \leq^{fpt} (Q, \lambda, \Upsilon) &\implies (P, \kappa, \Pi) \leq^{w\text{-}fpt} (Q, \lambda, \Upsilon)\\ &\implies (P, \kappa, \Pi) \leq^{fpt}_{T} (Q, \lambda, \Upsilon).\end{aligned}$$

Linearity transfers similarly.

▭ PROOF. For the first implication, observe that by definition parsimonious reductions are a special case of weakly parsimonious reductions. For the second implication, observe that we can interpret a parameterized weakly parsimonious reduction as a parameterized Turing reduction with a single oracle call. ◲

Further, we can easily see that all three notions of reducibility $\leq^{fpt}$, $\leq^{w\text{-}fpt}$, and $\leq^{fpt}_{T}$ (and their linear sub-variants) pass the sanity check of being transitive. We skip the proof, as it is not very instructive.

Instead, let us discuss a first example of a reduction—namely from #CLIQUE to #CONNECTEDCLIQUE, that is, the problem of having to compute $\#IndSub(\, K_k \to G\,)$ for graphs G that are connected.

◰ LEMMA 8.5 (FOLKLORE). *#CLIQUE $\leq^{\ell\text{-}fpt}_{T}$ #CONNECTEDCLIQUE and #CLIQUE $\leq^{w\ell\text{-}fpt}$ #CONNECTEDCLIQUE.*[77]

77. Note that by Lemma 8.4, the statement is redundant—intentionally—to emphasize that we discuss two different reductions to give examples for both types of reductions.

▭ PROOF. Suppose we are given a disconnected graph G_D as an input, but we have only an algorithm $\mathbb{A}$ for #CONNECTEDCLIQUE at hand. Let us discuss two different approaches to nevertheless compute the number of k-cliques in G_D:

▭ Compute the connected components of G_D and call $\mathbb{A}$ on each component (and input k) separately. The number of k-cliques in G_D is then the sum of the numbers computed for each component. Computing the connected components takes linear time and there are at most linearly many connected components, and hence we have #CLIQUE $\leq^{\ell\text{-}fpt}_{T}$ #CONNECTEDCLIQUE.

- Instead of calling $\mathbb{A}$ multiple times, we desire to call $\mathbb{A}$ only once. To that end, construct a new graph G from G_D, such that both G and G_D have the same number of k-cliques.
  For $k = 1$, that is, if the task is to count the vertices of G_D, this can be achieved by using a path graph with $|V(G_D)|$ vertices, that is $G := P_{|V(G_D)|}$. Similarly for $k = 2$, that is, if the task is to compute the number of edges of G_D, setting $G := P_{|E(G)|+1}$ suffices.
  Now for the general case, again compute the connected components $D_1, D_2, \dots$ of G_D. However, instead of calling $\mathbb{A}$ on each component separately, connect the connected components in a path-like fashion, that is, add an edge between (any vertex of) D_1 and (any vertex of) D_2, an edge between (any vertex of) D_2 and (any vertex of) C_3, and so on. Call the now connected graph G and call $\mathbb{A}$ on (G, k). As connecting two previously disconnected vertices cannot create a new k-clique for $k > 2$, the graphs G and G_D have the same number of k-cliques. Observe that the construction still runs in linear time, and hence we have #CLIQUE $\leq^{\ell\text{-}fpt}$ #CONNECTEDCLIQUE. ⌟

Finally, let us discuss a useful reduction from #CP-HOM$(\mathcal{H} \to \mathcal{G})$ to #HOM$(\mathcal{H} \to \mathcal{G})$—Hence to obtain hardness, we can reduce to the more structured problem #CP-HOM$(\mathcal{H} \to \mathcal{G})$, which simplifies some of the proofs to come.

LEMMA 8.6 ([99, 38]). *For any graph classes $\mathcal{H}$ and $\mathcal{G}$, we have #CP-HOM$(\mathcal{H} \to \mathcal{G}) \leq_T^{\ell\text{-}fpt}$ #HOM$(\mathcal{H} \to \mathcal{G})$.*

If, additionally, the class $\mathcal{H}$ contains only cores, then the reduction is weakly parsimonious.

If in a given instance (H, G) to #CP-HOM$(\mathcal{H} \to \mathcal{G})$, the graph G is H-colorable, then in all constructed instances (I, F) to #HOM$(\mathcal{H} \to \mathcal{G})$, the graph F is I-colorable.[78]

78. In fact, we can even ensure $I = H$.

PROOF. Fix graphs $H \in \mathcal{H}$ and $G \in \mathcal{G}$ and let $c \in Hom(G \to H)$ denote the H-coloring of G. Recall that a color-prescribed homomorphism $h \in cp\text{-}Hom(H \to G)$ is a homomorphism that additionally satisfies $c(h(v)) = v$ for all v in $V(H)$. Hence, if c is not surjective, then there is no color-prescribed homomorphism from H to G, thus we may assume that c is surjective.

Now, fix $h \in \textit{cp-Hom}(H \to G)$. As c is surjective, we have $c(h(V(H))) = V(H)$. In particular, as both h and c are homomorphisms, $h \circ c$ is an automorphism of H. In fact, each automorphism a of H can be decomposed into c and a homomorphism g in $\textit{Hom}(H \to G)$ such that $a = g \circ c$ and $c(g(V(H))) = V(H)$. Let us call homomorphisms g with $g \circ c \in \textit{Aut}(H)$ *colorful* and write

$$\textit{cf-Hom}(H \to G) := \{g \in \textit{Hom}(H \to G) \mid g \circ c \in \textit{Aut}(H)\}.$$

Claim 8.7. $\#\textit{cf-Hom}(H \to G) = \#\textit{Aut}(H) \cdot \#\textit{cp-Hom}(H \to G)$.

Proof. First observe that we have $h \in \textit{cf-Hom}(H \to G)$ and also $a \circ h \in \textit{cf-Hom}(H \to G)$ for any $a \in \textit{Aut}(H)$, as concatenation of automorphisms $(a \circ (h \circ c))$ is an automorphism again and concatenation is associative $(a \circ (h \circ c) = (a \circ h) \circ c)$.[79] Hence,

79. Observe that we have $a \circ b \circ h = c \circ h$ for $a, b, c \in \textit{Aut}(H)$ for the same reason.

$$\#\textit{cf-Hom}(H \to G) \leq \#\textit{Aut}(H) \cdot \#\textit{cp-Hom}(H \to G).$$

For the other direction, observe that distinct color-prescribed homomorphisms have different images and that an automorphism does not change the image. ⌟

For a set $W \subseteq V(H)$, write $G|_W := G[\{v \in V(G) \mid c(v) \in W\}]$ for the induced subgraph that contains only the colors in W.

Claim 8.8.
$$\#\textit{cf-Hom}(H \to G) = \sum_{W \subseteq V(H)} (-1)^{\#V(H) - \#W} \cdot \#\textit{Hom}(H \to G|_W).$$

Proof. Observe that a homomorphism that is not colorful is missing (perhaps multiple) colors from $V(H)$ in its image. Now the formula follows by the Inclusion-Exclusion Principle. ⌟

The combination of Claims 8.7 and 8.8 yields the desired parameterized Turing reduction, which is even linear as all instances can be easily obtained with a linear pass over G and $f(|V(H)|)$ extra time for a computable f.

For the weakly parsimonious reduction, Claim 8.7 is already weakly parsimonious for any graph class $\mathcal{H}$. For Claim 8.8, recall that a core H has no endomorphisms to a proper subgraph of H. However, for a non-empty set W, any homomorphism $g \in \mathit{Hom}(\, H \to G|_W \,)$ would yield an endomorphism $g \circ c$ to a proper subgraph of H—Hence for a core H, we have $\#\mathit{cf\text{-}Hom}(H \to G) = \#\mathit{Hom}(\, H \to G\,)$.

Finally, observe that if the graph G is H-colorable, then so are all its subgraphs, and hence in particular the graphs $G|_W$ for all sets K; completing the proof. ⌟

As a small addendum, observe that the we can obtain a reduction in the other direction almost trivially.

⌜ LEMMA 8.9 ([99, LEMMA 2.53]). *For any graph classes $\mathcal{H}$ and $\mathcal{G}$, we have* $\#\text{HOM}(\mathcal{H} \to \mathcal{G}) \leq^{\ell\text{-}fpt} \#\text{CP-HOM}(\mathcal{H} \to \mathcal{G})$.

▭ PROOF. Given an instance (H, G) to $\#\text{HOM}(\mathcal{H} \to \mathcal{G})$, consider the graph $H \times G$. By Lemma 7.8, $H \times G$ is H-colorable; indeed, the identity mapping on H lifts to an H-coloring c of $H \times G$.

First, fix a $h \in \mathit{cp\text{-}Hom}(H \to G)$ and observe that the mapping $f(v) := (v, h(g))$ is indeed in $\mathit{cp\text{-}Hom}(H \to H \times G)$. Next, fix a $f \in \mathit{cp\text{-}Hom}(H \to H \times G)$ and observe that the mapping $h(v) := u$, where $(v, u) = f(v)$, is indeed in $\mathit{Hom}(\, H \to G\,)$. ⌟

Hardness and Conjectured Hardness

In this part, we use reductions mainly to show that we cannot solve problems faster than some running time bound—In particular, we show that if we could do so, we would violate widely believed conjectures from parameterized and fine-grained complexity theory.

Our main working conjecture is the so-called Exponential-Time Hypothesis (ETH) which was introduced by Impagliazzo and Paturi [61]. Recall that in the Satisfiability (Sat) problem, we are given a boolean formula ϕ over n variables, and the task is to decide whether there is an assignment of truth values to the variables that satisfies ϕ. Recall further that in the variant 3-SAT, the formula ϕ

has the form $(\ell_{11} \vee \ell_{12} \vee \ell_{13}) \wedge (\ell_{21} \vee \ell_{22} \vee \ell_{23}) \wedge \cdots$, where the ℓ_{ij} are either a variable or a negated variable. ETH reads as follows.

CONJECTURE 8.10 (ETH). *No algorithm solving 3-SAT runs in time $\exp(o(n))$, where n is the number of variables of the input formula.*

Assuming ETH, one can show that the problem CLIQUE—and thus #CLIQUE—is not fixed-parameter tractable.[80]

80. Note that in *Fine-Grained Complexity Theory*, it is sometimes [1, 18] conjectured that CLIQUE has no algorithm running in time $|V(G)|^{2\omega k/3-\varepsilon}$ for any $\varepsilon > 0$, where $\omega < 2.373$ is the so-called matrix multiplication exponent. For algorithms not using matrix multiplication or other "non-combinatorial" techniques, a similar lower bound of $|V(G)|^{k-\varepsilon}$ is conjectured.

FACT 8.11 (CHEN, CHOR, FELLOWS, HUANG, JUEDES, KANJ, AND XIA [28], CHEN, HUANG, KANJ, AND XIA [29]). *No algorithm can solve CLIQUE in time $f(|V(H)|) \cdot |V(G)|^{o(k)}$ for any f, unless ETH fails.*

Observe that non-tractability extends to other problems via parameterized reductions, for instance to the aforementioned variant #CONNECTEDCLIQUE. Naturally, the same is true for concrete running time lower bounds.[81]

81. Note that with running times, we need to be more careful if we are interested in the concrete polynomial in the part not depending on the parameter. However, in this part, we are not interested in this, so-called *fine-grained complexity* of problems.

LEMMA 8.12. *Let (P, κ, Π) and (Q, λ, Y) denote PPC problems and assume that (P, κ, Π) reduces to (Q, λ, Y) with respect to any of $\leq^{fpt}$, $\leq^{w\text{-}fpt}$, or $\leq_T^{fpt}$ If (Q, λ, Y) is FPT, then so is (P, κ, Π).*

If the reduction is even linear, then any $g(\lambda(x)) \cdot |x|^c$-time algorithm for (Q, λ, Y) yields an $f(\kappa(x)) \cdot |x|^c$-time algorithm for (P, κ, Π), where c is a constant and f, g are computable.

PROOF. Follows from Definitions 8.2 and 8.3.

Using the problem #CLIQUE, we can define an important complexity class in parameterized complexity theory, called #**W**[1].

DEFINITION 8.13 ([46, 86]). *The class #**W**[1] contains all parameterized counting problems without promises that can be reduced to the problem #CLIQUE by parameterized parsimonious reductions.*

*The class **W**[1] is defined similarly for decision problems.*

Having defined a complexity class and reductions between its problems, it is natural to also define hardness and completeness.

DEFINITION 8.14. *A PPC problem (P, κ, Π) is* #$\mathbf{W}[1]$-hard *under parameterized parsimonious reductions if* #CLIQUE $\leq^{fpt}$ (P, κ, Π)*. Hardness under* $\leq^{w\text{-}fpt}$ *and* $\leq^{fpt}_{T}$ *is defined likewise. A parameterized counting problem (without promises) is* #$\mathbf{W}[1]$-complete *if it is contained in* #$\mathbf{W}[1]$ *and* #$\mathbf{W}[1]$*-hard.*[82]

82. The class is called #$\mathbf{W}[1]$, as it is part of the so-called W-hierarchy of complexity classes—As an equivalent definition, one can define the classes of the W-hierarchy via the so-called *weft* of a circuit. Consult [46] for more details.

Assuming ETH, any #$\mathbf{W}[1]$-hard problem is not FPT, as it is at least as hard as #CLIQUE. Any #$\mathbf{W}[1]$-complete problem is *interreducible* (denoted by $\equiv$) to #CLIQUE.[83]

83. That is, there is a reduction from and to #CLIQUE.

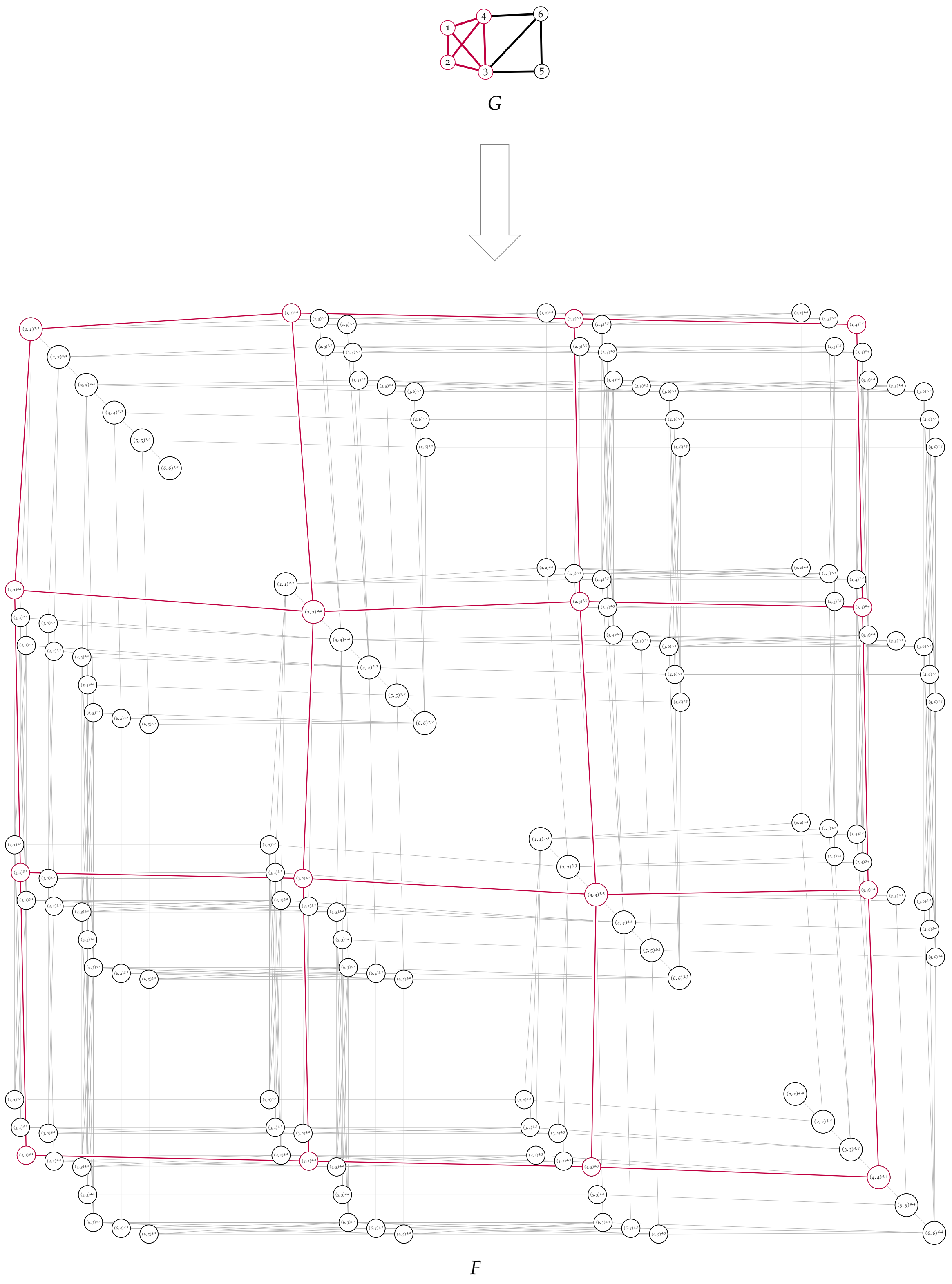
Regarding the specific problems we study, Jerrum and Meeks [62] have shown that for any computable property Φ, the problem #INDSUB(Φ) reduces to #CLIQUE using parameterized Turing reductions. Thus we can focus on the "hardness part" of the #$\mathbf{W}[1]$-completeness results for #INDSUB(Φ).

For the homomorphism side, Dalmau and Jonsson [37] proved a classification of #HOM($\mathcal{H} \to \top$) into polynomial-time and #$\mathbf{W}[1]$-complete cases based on the treewidth of the graphs in the class $\mathcal{H}$. Further, Marx [84] (implicitly) gave a stronger conditional lower bound for the hard cases.

LEMMA 8.15 ([37, 84]). *For any $\mathcal{H}$ semidecidable graph class,*

1 *If the treewidth of $\mathcal{H}$ is bounded then* #HOM($\mathcal{H} \to \top$) *is solvable in polynomial time.*
2 *Otherwise,* #HOM($\mathcal{H} \to \top$) *is* #$\mathbf{W}[1]$*-hard under parameterized Turing reductions and* #HOM($\mathcal{H} \to \top$) *has no algorithm running in time* $f(|V(H)|) \cdot |V(G)|^{o(tw(H)/\log(tw(H)))}$ *for any f, unless* ETH *fails.*

As we need to use more of the technical details of the proof of Lemma 8.15, we discuss both the original #$\mathbf{W}[1]$-hardness, as well as a proof for the ETH-based lower bound in more detail shortly. To that end, write $\boxplus$ for the graph class of all $k \times k$ square grids $\boxplus_k$ for $k \in \mathbb{N} \setminus \{0\}$. For the #$\mathbf{W}[1]$-hardness result for #PARTSUB($\mathcal{H} \to \top$), we first consider the intermediate problem #CP-HOM($\boxplus \to \top$).



□ Figure 8.1. The "grid-tiling" construction from Lemma 8.16.

⌜ LEMMA 8.16. #CONNECTEDCLIQUE $\leq^{w\text{-}fpt}$ #CP-HOM$(\boxplus \to \top)$.
Further, in the constructed instance $(\boxplus_k, F)$ of #CP-HOM$(\boxplus \to \top)$, the graph F is connected and $\boxplus_k$-colorable.

▭ PROOF. Given an instance (G,k) of #CONNECTEDCLIQUE, we define the following graph F.[84] (See also Figure 8.1.)

84. We order the vertices arbitrarily.

$$\begin{aligned} V(F) := & \bigcup_{i\in[k]} \{(v,v)^{i,i} \mid v \in V(G)\} \\ & \cup \bigcup_{i<j\in[k]} \{(u,v)^{i,j} \mid u \leq v, \{u,v\} \in E(G)\} \\ & \cup \bigcup_{i>j\in[k]} \{(u,v)^{i,j} \mid u \geq v, \{u,v\} \in E(G)\} \\ E(F) := & \{((u,v)^{i,j},(x,y)^{a,b}) \mid (u = x \wedge i = a) \text{ or } (v = y \wedge j = b)\} \\ & \cup \{((v,v)^{i,i},(v+1,v+1)^{i,i})\}. \end{aligned}$$

First, observe that F has a natural $\boxplus_k$-coloring that sends every vertex $v^{i,j}$ of F to the vertex (i,j) in $\boxplus_k$. Next, observe that the edges between vertices $u^{i,i}$ and $v^{i,i}$ make the graph F connected; as a color-prescribed homomorphism has to pick exactly one of these vertices, edges connecting these vertices serve no other purpose.

Next, recall from Equation (7.1) that we have $\#Sub(K_k \to G) = \#Emb(K_k \to G)/\#Aut(K_k)$; we easily see that $\#Aut(K_k) = k!$.

Fix a $g \in Emb(K_k \to G)$ and fix any labeling of K_k; we write $[k]$ for $V(K_k)$. Consider the mapping $h : V(\boxplus_k) \to F$, where $h((i.j)) := (g(i),g(j))^{i,j}$. Observe that the vertex $(g(i),g(j))^{i,j}$ always exists in F, as K_k contains the edge $\{i,j\}$ and g is an embedding. Further, for two adjacent vertices (i,j) and (x,y) of $\boxplus_k$, the vertices $(g(i),g(j))^{i,j}$ and $(g(x),g(y))^{x,y}$ are connected—again, as g is an embedding. Finally, g is trivially color-prescribed. This yields the lower bound $\#Emb(K_k \to G) \leq \#cp\text{-}Hom(\boxplus_k \to F)$.

For the other direction, fix a $f \in cp\text{-}Hom(\boxplus_k \to F)$. As f is color-prescribed, for every $(i,j) \in [k]\times[k]$, there is a single vertex $v^{i,j}$ in F with $v^{i,j} := f((i,j))$. Now, consider the mapping $e : V(K_k) \to G$ with $e(i) := v^{i,i}$. Observe that for two neighboring vertices (i,j) and (x,y) of $\boxplus_k$, the corresponding vertices $v^{i,j}$ and $v^{x,y}$ are neighbors as well, as f is a homomorphism. Hence, for any pair $i < j$, the vertices $v^{i,i}, v^{i,i+1}, \dots, v^{i,j}, v^{i+1,j}, \dots, v^{j,j}$ form a path. By construction of F, this in turn means that there is an edge $\{i,j\}$ in G—Hence, e is an embedding and thus $\#Emb(K_k \to G) \geq \#cp\text{-}Hom(\boxplus_k \to F)$.

Combining both inequalities, we obtain

$$\#Sub(K_k \to G) = \#Emb(K_k \to G)/k! = \#cp\text{-}Hom(\boxplus_k \to F)/k!,$$

which yields the desired reduction. ⌟

In the next step, we use the celebrated Excluded-Grid Theorem by Robertson and Seymour [96]—For space constraints, we do not discuss its proof here.

⌜ FACT 8.17 (EXCLUDED-GRID THEOREM, [96]). *Let $\mathcal{H}$ denote a recursively enumerable graph class of unbounded treewidth.*[85] *For any non-negative integer k, there is a $H_k \in \mathcal{H}$ with the grid $\boxplus_k$ as a minor.* ⌟

85. That is, there is no global constant t such that every graph in $\mathcal{H}$ has a treewidth of at most t.

Using Fact 8.17, we need to convince ourselves only that the hardness from Lemma 8.16 is monotone with respect to taking minors; that, we do next.

⌜ LEMMA 8.18 ([38, LEMMA 8]). *Let H denote a graph and let M denote a minor of H. Further, let F denote an M-colorable graph.*
Then, there is an H-colorable graph G of size $|V(G)| \leq |V(H)| \cdot |V(F)|$ that satisfies $\#cp\text{-}Hom(M \to F) = \#cp\text{-}Hom(H \to G)$.

▭ PROOF. Write c to denote an H-coloring of G.

If $M = H$, there is nothing to show. Otherwise, we can obtain M from H via a set of vertex deletions, edge deletions, and edge contractions. We show the claim for all three operations separately, the claim then follows inductively.

VERTEX DELETIONS. Assume that we have $M = H \setminus v$. Now, we obtain G from F by adding a new isolated vertex u: We set $G := (V(F) \cup \{u\}, E(F))$. Further, we extend c to G via $c(u) := v$. As any homomorphism $h \in \textit{cp-Hom}(H \to G)$ has to map v to u, we immediately obtain $\#\textit{cp-Hom}(M \to F) = \#\textit{cp-Hom}(H \to G)$.

Edge deletions. Assume that we have $M = H \setminus \{u, v\}$. Now, we obtain G from F by adding all edges between vertices x and y with $c(x) = u$ and $c(y) = v$: We set $G := (V(F), E(F) \cup \{\{x, y\} \mid c(x) = u, c(y) = v\})$. Observe that any $h \in \textit{cp-Hom}(M \to F)$ is also a color-prescribed homomorphism from H to G—clearly, any edge $\{h(u), h(v)\}$ exists in G; the other direction is immediate. Hence, $\#\textit{cp-Hom}(M \to F) = \#\textit{cp-Hom}(H \to G)$. For the size bound, we have $|V(G)| = |V(F)|$.

Edge contractions. Assume that we have $M = H/\{u, v\}$. Now, we obtain G from F by duplicating every vertex $x \in V(F), c(x) = uv$ and connecting both the original vertex and its copy: We set

$$\begin{aligned} V(G) &:= V(F) \cup \{d_x \mid x \in V(F), c(x) = uv\}, \\ E(G) &:= E(F) \cup \{\{x, d_x\} \mid d_x \in V(G)\} \\ &\qquad\qquad \cup \{\{a, d_x\} \mid \{a, x\} \in E(F)\}. \end{aligned}$$

Any $h \in \textit{cp-Hom}(M \to F)$ now one-to-one corresponds with a $g \in \textit{cp-Hom}(H \to G)$ with $g(u) := h(uv)$, $g(v) := d_{h(uv)}$, and $g(a) := h(a)$ otherwise. Hence, $\#\textit{cp-Hom}(M \to F) = \#\textit{cp-Hom}(H \to G)$. For the size bound, we have $|V(G)| \leq 2 \cdot |V(F)|$, as we may duplicate every vertex of F at most once.

For the size bound, write G_i for the graph obtained after i operations. Observe that in every vertex deletion, the constructed graph G_i satisfies $|V(G_i)| = |V(G_{i-1})| + 1$; observe that in every edge deletion, the constructed graph G_i satisfies $|V(G_i)| = |V(G_{i-1})|$; and observe that in every edge contraction, the constructed graph G_i satisfies $|V(G_i)| = |V(G_{i-1})| + |V(G_0)|$, where $G_0 = F$ denotes the graph originally given. As each edge contraction and vertex deletion operation reduce the size of H by one, we obtain $|V(G)| \leq |V(H)| \cdot |V(F)|$. ⌟

Finally, the #$\boldsymbol{W}[1]$-hardness part of Lemma 8.15 follows from Lemmas 8.5, 8.6, 8.16 and 8.18 and Fact 8.17.

Lemma 8.19 (Lemma 8.15, #$\boldsymbol{W}[1]$-hardness). *Let $\mathcal{H}$ denote a recursively enumerable graph class of unbounded treewidth.*

1 *We have* #Clique $\leq_T^{fpt}$ #Hom$(\mathcal{H} \to \top)$. *Further, in every oracle query (H, G), the graph G is H-colorable.*
2 *If, additionally, the class $\mathcal{H}$ contains only cores, then the reduction is weakly parsimonious. If the cores are also connected, then so are the graphs of the oracle query.*

Proof. Our goal is to count k-cliques by using an oracle for the problem #Hom$(\mathcal{H} \to \top)$.

First, we use the reduction #Clique $\leq^{w\ell\text{-}fpt}$ #ConnectedClique from Lemma 8.5. Next, we obtain #Clique $\leq^{w\ell\text{-}fpt}$ #Hom$(\boxplus \to \top)$ by adding Lemma 8.16. Note that in the constructed query $(\boxplus_k, F)$ to #Hom$(\boxplus \to \top)$, the graph F is connected and $\boxplus_k$-colorable.

By Fact 8.17, for any $\boxplus_k$ there is a graph H_k in $\mathcal{H}$ that has $\boxplus_k$ as a minor; further, we can compute this graph H_k as $\mathcal{H}$ is recursively enumerable. By Lemma 8.18, we can construct a graph G that is H_k-colorable and satisfies *#cp-Hom*$(\boxplus_k \to F)$ = *#cp-Hom*$(H_k \to G)$. Note that H_k may be disconnected only if $\mathcal{H}$ contains graphs that are disconnected. Further, if H_k is connected, we may also assume that G is connected as well.[86]

86. For any disconnected components of G, we can use the *edge deletions* step from Lemma 8.18 to make them connected.

Using Lemma 8.6, we transform the instance (H_k, G) into (multiple) instances (H, I) of #Hom$(\mathcal{H} \to \top)$—which we compute using our oracle to recover the solution to our original problem.

Finally, observe that the last step becomes weakly parsimonious if $\mathcal{H}$ contains only cores; and thus the same is true for the overall reduction. ∎

Note that the proofs of Lemma 8.5 and Item 2 of Lemma 8.19 directly yield the following result for the decision versions under parameterized many-one reductions [47, Definition 2.1].

COROLLARY 8.20. *Let $\mathcal{H}$ denote a recursively enumerable class of cores of unbounded treewidth. Then, there is a parameterized many-one reduction from* CLIQUE *to* HOM$(\mathcal{H} \to \top)$ *such that in every instance (H, G) to* HOM$(\mathcal{H} \to \top)$ *the graph G is connected and H-colorable.*

It is useful to us to apply the transformations between homomorphisms and color-prescribed homomorphisms to Lemma 8.18, thereby obtaining a similar reduction between the corresponding uncolored problems.

LEMMA 8.21. *Let $\mathcal{H}$ and $\mathcal{M}$ denote semidecidable graph classes, where for each graph $M \in \mathcal{M}$ there is a graph $H \in \mathcal{H}$ that has M as a minor. Then,* #HOM$(M \to \top) \leq_T^{\ell\text{-}fpt}$ #HOM$(H \to \top)$.

PROOF. Combine Lemma 8.9 with Lemma 8.18 and Lemma 8.6. Transitivity then yields the claim.

For the ETH-based lower bound of Lemma 8.15, we use a known hardness result for *Partitioned Subgraph Isomorphism*. For a graph class $\mathcal{H}$, in PARTSUB$(\mathcal{H} \to \top)$, we are given a graphs $H \in \mathcal{H}, G \in \top$, and a mapping $c : V(G) \to V(H)$; the task is to *decide* if there is an embedding φ from H to G with $c(\varphi(v)) = v$ for all v in $V(H)$.

FACT 8.22 (MARX [84, COROLLARY 6.2]). *Let $\mathcal{H}$ denote a recursively enumerable graph class of unbounded treewidth. No algorithm can solve* PARTSUB$(\mathcal{H} \to \top)$ *in time $f(|V(H)|) \cdot |V(G)|^{o(tw(H)/log(tw(H)))}$ for any computable f, unless* ETH *fails.*

LEMMA 8.23. *Let $\mathcal{H}$ denote a semidecidable graph class of unbounded treewidth. We have* PARTSUB$(\mathcal{H} \to \top) \leq^{w\ell\text{-}fpt}$ #HOM$(\mathcal{H} \to \top)$.

PROOF. Now, suppose we are given a graph $H \in \mathcal{H}$, an arbitrary graph G, and a coloring $c : V(G) \to V(H)$. We wish to count embedding φ from H to G that satisfies $c(\varphi(v)) = v$ for each vertex v in $V(H)$. First, observe that we can drop the requirement of φ being injective, as every homomorphism that preserves the coloring is injective. Further, observe that without loss of generality, we can assume that c is a homomorphism from G to H: Every edge $\{u,v\}$ in $E(G)$ with $\{c(u),c(v)\} \notin E(H)$ is irrelevant for finding a homomorphism φ from H to G that preserves the coloring c. Hence, we can delete all of those edges from G.

Thus, PartitionedSub$(\mathcal{H})$ is equivalent to cp-Hom$(\mathcal{H} \to \top)$ and at least as hard as #cp-Hom$(\mathcal{H} \to \top)$; by Lemma 8.6, the claimed lower bound holds for #Hom$(\mathcal{H} \to \mathcal{G})$ as well. ◿

8.3 The Complexity of Graph Motif Parameters

Having both seen that counting (induced) subgraphs in a graph is nothing more but computing linear combinations of homomorphism numbers (Section 7.5), and that we can classify the problem of computing homomorphism numbers into easy and hard cases based on the treewidth of the pattern (Lemma 8.15), it is natural to ask if we can obtain a similar dichotomy result for linear combinations of homomorphism numbers—and as a consequence for counting (induced) subgraphs. Note that *the other direction*, is immediate: If computing a finite linear combination is hard, then certainly it cannot be easier to compute all summands of said linear combination separately. In fact, it is somewhat surprising, that for homomorphism numbers, indeed reductions in *both* directions are possible.[87]This result by Curticapean et al. [35] (and independently Chen and Mengel [27]) is called *Complexity Monotonicity of Graph Motif Parameters* or *Complexity Monotonicity* for short. For completeness, we discuss a proof based on [35] next. For technical reasons, we need to be slightly more general—however, this does not affect the proof.

87. Consider the textbook example of counting satisfying assignments to a Sat formula and adding the number of non-satisfying assignments to it—Both problems separately are hard, computing their sum is trivial.

◸ Lemma 8.24 ([35, Lemma 3.6]). *Let $\mathcal{G}$ denote a graph class that is closed under categorical product with any graph from the right.*[88,89] *Further, for a finitely supported vector $\alpha \in \mathbb{R}^{\mathcal{G}}$ let $f = \alpha \cdot$ #Hom denote a graph motif parameter and write S for a graph in the support of α. Finally, let G denote a graph in $\mathcal{G}$.*

There is a deterministic algorithm $\mathbb{A}$ with oracle access to the function $(\alpha\cdot$#Hom$)(\star)$, that computes the number #Hom$(\,S \to G\,)$ in time $g(\alpha)\cdot$

88. That is, for any graph $G \in \mathcal{G}$, we also have $G \times H \in \mathcal{G}$ for any graph $H \in \top$.

89. Observe that this requirement is trivially met by the graph class $\top$.

$poly(|V(G)|)$, where g is a computable function depending only on α. Further, $\mathbb{A}$ queries the oracle at most $g(\alpha)$ times and each queried graph has a size of at most $max_{H \in supp(\alpha)} |V(H)||V(G)|$ vertices.

▭ PROOF. Let $\alpha \cdot \#Hom$ denote a graph motif parameter and let S denote a graph in the support of α. Further, let $G \in \mathcal{G}$ denote a graph. Recall from Lemma 7.7, that for any graph F, we have

$$\#Hom(\, S \to G \times F\,) = \#Hom(\, S \to G\,) \cdot \#Hom(\, S \to F\,). \qquad (8.1)$$

Expanding the $\alpha \cdot \#Hom$ and applying Equation (8.1) yields

$$(\alpha \cdot \#Hom)(G \times F) = \sum_{H} \#Hom(\, H \to G\,) \cdot \#Hom(\, H \to F\,). \quad (8.2)$$

Observe that plugging in different graphs for F into Equation (8.2), we obtain a system of linear equations. Observe further that we can query the oracle for the values $(\alpha \cdot \#Hom)(G \times F)$; and, as long as $|V(F)| \leq h(|V(H)|)$ for a computable h, we can also compute the values $\#Hom(\, H \to F\,)$ in some time $g(\alpha)$ for a computable function g. It remains to find a set P of graphs such that the corresponding principal submatrix $\#Hom_P$ is invertible. Now recall from Lemma 7.11, that this is true if P is closed surjective homomorphisms. Finally, observe that by Equation (7.7), we can obtain the surjective closure of a set of graphs by adding all spasms of graphs in the set. Therefore, for

$$P := \bigcup_{H \in \alpha} spasms(H),$$

the matrix $\#Hom_P$ is invertible.

The algorithm $\mathbb{A}$ now looks as follows: For every $F \in P$, we call the oracle for the value $(\alpha \cdot \#Hom)(G \times F)$ and we compute in time $g(\alpha)$ the matrix $\#Hom_P$, which we invert in time $g(\alpha)$. As $S \in P$, we thereby obtain the value $\#Hom(\, H \to S\,)$, as desired.

Finally, observe that in all queries the size of $(G \times F)$ is at most $|V(G)| \cdot max_{H \in P}|V(H)|$—as the quotient operation cannot increase the number of vertices, this yields the desired size bound. ⌟

Finally, let us combine Equation (7.11) and Lemma 8.24—note that by Lemma 7.14, for any fixed graph H, there are only finitely many F with $\mathsf{a}_{IndSub(H \to \star)}(F) \neq 0$, that is, the linear combination

$$\#IndSub(H \to \star) = \sum_{\substack{F \\ |V(F)| \leq |V(H)|}} \mathsf{a}_{IndSub(H \to \star)}(F) \#Hom(F \to \star)$$

corresponds to a graph motif parameter. As there are only finitely many graphs of size k, the same is true for the linear combination

$$\begin{aligned} \#IndSub(\Phi, k \to \star) &= \sum_{H \in \mathcal{G}_{\varphi,k}} \#IndSub(H \to \star) \\ &= \sum_{H \in \mathcal{G}_{\varphi,k}} \sum_{\substack{F \\ |V(F)| \leq |V(H)|}} \mathsf{a}_{IndSub(H \to \star)}(F) \#Hom(F \to \star) \\ &=: \sum_{\substack{F \\ |V(F)| \leq k}} \mathsf{a}_{IndSub(\Phi, k \to \star)}(F) \#Hom(F \to \star) \qquad (8.3) \end{aligned}$$

Adding Lemmas 8.15 and 8.24 directly yields the following result.

⌜ Corollary 8.25 ([35]). *Let Φ denote a computable graph property. The problem #IndSub(Φ) is either* FPT *or* #**W**[1]*-complete.* ⌟

Note that Corollary 8.25 yields only an *implicit* dichotomy—to obtain explicit hardness of #IndSub(Φ) for specific graph properties Φ, we need to understand which $\mathsf{a}_{IndSub(H \to \star)}(G)$ in Equation (8.3)—in particular, we want to identify graphs H with high treewidth with nonzero coefficients. We tackle this task in detail in 10.

Counting Graph Motif Parameters in F-colorable or Kőnig Graphs

9

Recall that back in Section 8.2, we exported more technical details from the proof of Lemma 8.15. In this chapter, we see first applications of said technical details.

9.1 Counting Homomorphisms and Subgraphs in F-Colorable Graphs

Let $\mathcal{H}$ denote a recursively enumerable class of graphs. Further, given a fixed graph F, let $\mathcal{C}_F$ denote the class of all graphs G that admit a homomorphism to F, that is, the class of F-colorable graphs. In this section, we generalize Lemma 8.15 to $\#\textsc{Hom}(\mathcal{H} \to \mathcal{C}_F)$; that is, counting the number of homomorphisms from a graph $H \in \mathcal{H}$ to a graph $G \in \mathcal{C}_F$. Note that the notion of $\mathcal{C}_F$ captures and generalizes the important special cases of the class of all bipartite graphs (when F is a single edge) or, more general, the class of all k-colorable graphs for any fixed number k (when F is the complete graph on k vertices).

Theorem 9.1. *Let F denote a graph, and let $\mathcal{H}$ denote a recursively enumerable class of graphs.*

1. *If the treewidth of the class $\mathcal{H} \cap \mathcal{C}_F$ is bounded then the* ppc *problem $\#\textsc{Hom}(\mathcal{H} \to \mathcal{C}_F)$ is polynomial-time solvable.*
2. *Otherwise, $\#\textsc{Hom}(\mathcal{H} \to \mathcal{C}_F)$ is $\#\mathbf{W[1]}$-hard.*

We prove Theorem 9.1 in two steps. First, in Lemma 9.3, we extend the known polynomial-time algorithm for $\#\textsc{Hom}(\mathcal{H} \to \top)$ for bounded-treewidth graph classes $\mathcal{H}$ to graph classes $\mathcal{H}_{easy}$ that do not contain F-colorable graphs of arbitrarily large treewidth. After that, we prove $\#W[1]$-hardness for all other graph classes.

Polynomial-Time Algorithm for the Tractable Cases

Let $\mathcal{H}_{easy}$ denote graph class that contains only graphs that are not F-colorable or graphs with a treewidth of at most c, where $c := c(\mathcal{H}_{easy})$ is a constant depending only on $\mathcal{H}_{easy}$. We obtain a polynomial-time algorithm for $\#Hom(\mathcal{H}_{easy} \to \mathcal{C}_F)$ as follows.

Given graphs $H \in \mathcal{H}_{easy}$ and $G \in \mathcal{C}_F$, we check, using the algorithm of Bodlaender [15], whether H has a treewidth $tw(H)$ of at most $c(\mathcal{H}_{easy})$. Next, if $tw(H) \leq c(\mathcal{H}_{easy})$, we use the standard dynamic programming algorithm of Díaz, Serna, and Thilikos [39] to compute $\#Hom(H \to G)$. Otherwise, that is if $tw(H) > c(\mathcal{H}_{easy})$, we output 0, as H is not F-colorable by definition of $\mathcal{H}_{easy}$. This last step is justified by the following observation.

Lemma 9.2. *For any graphs F, G, and H, if there is no homomorphism from H to F, but a homomorphism from G to F, then there is no homomorphism from H to G.*

Proof. Fix $g \in Hom(G \to F)$ and suppose there was an $h \in Hom(H \to G)$. As the concatenation of two homomorphisms is again a homomorphism, in particular $h \circ g : H \to F$ is again a homomorphism, which is a contradiction to the assumption that there is no homomorphism from H to F. ⌟

In total, we obtain the following algorithm.

Lemma 9.3 (Theorem 9.1, Item 1). *For any graph classes $\mathcal{H}_{easy}$ and $\mathcal{C}_F$, there is a polynomial-time algorithm for $\#Hom(\mathcal{H}_{easy} \to \mathcal{C}_F)$.*

Proof. The correctness follows directly from the definition of $\mathcal{H}_{easy}$ and Lemma 9.2.

For the running time, set $c := c(\mathcal{H}_{easy})$, $k := |V(H)|$, and $n := |V(G)|$. Using the algorithm by Bodlaender [15], we check if the given graph has a treewidth of at most c in time $c^{O(c^3)} \cdot k$. Next, using the algorithm by Díaz et al. [39], we can compute the number of homomorphisms from a graph with treewidth at most c in time $poly(k,c) \cdot n^{c+O(1)}$. Hence in total, our algorithm has a running time of $poly(k,c) \cdot n^{c+O(1)}$, which is polynomial. ⌟

#W[1]-Hardness for the Intractable Cases

Next, we discuss the #**W[1]**-hardness of $\#Hom(\,H \to \mathcal{C}_F\,)$ whenever the treewidth of $\mathcal{H} \cap \mathcal{C}_F$ is unbounded. However, as we have seen in Lemma 8.19, the existing hardness proof already shows the desired stronger result.

⌜ LEMMA 9.4 (THEOREM 9.1, ITEM 2). *Let $\mathcal{H}$ denote a recursively enumerable class of graphs such that the treewidth of $\mathcal{H} \cap \mathcal{C}_F$ is unbounded. Then, we have* #CLIQUE $\leq_T^{fpt}$ #HOM$(\mathcal{H} \to \mathcal{C}_F)$.

▭ PROOF. We use Item 1 of Lemma 8.19 for the class $\mathcal{H} \cap \mathcal{C}_F$: As in every oracle query (H, G), the graph G is H-colorable by a coloring c and every graph $H \in \mathcal{H} \cap \mathcal{C}_F$ is F-colorable by a coloring f, we obtain that the composition $c \circ f$ is an F-coloring of G. ⌟

Now, Theorem 9.1 holds by Lemmas 9.3 and 9.4.

Counting Subgraphs in F-colorable Graphs

In a next step, we discuss an easy generalization of the Complexity Monotonicity Framework to F-colorable graphs. Thereby, we show that the previous classification for counting homomorphisms to F-colorable graphs yields a complete classification for the associated subgraph counting problem—For to graph classes $\mathcal{H}$ and $\mathcal{G}$, in the PPC problem #SUB$(\mathcal{H} \to \mathcal{G})$ we are given graphs $H \in \mathcal{H}$ and $G \in \mathcal{G}$, and the task is to compute $\#Sub(\,H \to G\,)$. The parameter is $|V(H)|$; the promise is the set $\mathcal{H} \times \mathcal{G}$.

As an example, consider computing the number of k-matchings in a bipartite graph[90]—a well-known hard problem. In fact, the (#**W[1]**)-hardness criterion of #SUB$(\mathcal{H} \to \top)$ is indeed the similarity to counting matchings:

90. Here a k-matching is a set of k edges that are pairwise disjoint

FACT 9.5 (CURTICAPEAN AND MARX [34]). *Let $\mathcal{H}$ denote a recursively enumerable class of graphs.*

1 *If the matching number of the class $\mathcal{H}$ is bounded then the problem* #SUB$(\mathcal{H} \to \top)$ *is polynomial-time solvable.*
2 *Otherwise, the problem* #SUB$(\mathcal{H} \to \top)$ *is* #**W**[1]*-hard.* ⌟

Here, the *matching number* of a graph is the size of its largest matching and a graph class $\mathcal{H}$ has bounded matching number if there is an overall constant c such that the matching number of each graph $H \in \mathcal{H}$ is bounded by c.

Combining Equation (7.9) with the Complexity Monotonicity Framework and in particular Lemma 8.24, Curticapean et al. [35] strongly generalized Fact 9.5 with a much simpler proof. We can easily see that Lemma 8.24 generalizes to F-colorable graphs.

LEMMA 9.6. *Let F denote a fixed graph and write C_F for the graph class of all F-colorable graphs. Lemma 8.24 holds for C_F.*

PROOF. We need to show that F-colorable graphs are closed under categorical product—indeed they are, see Lemma 7.8. ⌟

Using Lemma 9.6, Fact 9.5 generalizes to F-colorable graphs:

THEOREM 9.7. *Let F denote a fixed graph and let $\mathcal{H}$ denote a recursively enumerable class of graphs.*

1 *If the matching number of the class $\mathcal{H} \cap C_F$ is bounded then the problem* #SUB$(\mathcal{H} \to C_F)$ *is polynomial-time solvable.*
2 *Otherwise, the problem* #SUB$(\mathcal{H} \to C_F)$ *is* #**W**[1]*-hard.*

PROOF. Proving Item 1 is easy: Let c denote the constant upper bound on the matching number of graphs in $\mathcal{H} \cap C_F$. Given $H \in \mathcal{H}$ and $G \in C_F$, we obtain the matching number of H in polynomial time using the Blossom-Algorithm [43]. If the result is greater than c, the promise yields $H \notin C_F$, in which case we can output 0 as H would be F-colorable if it was isomorphic to a subgraph of $G \in \mathcal{C}_F$. Otherwise, we use the algorithm from Item 1 of Fact 9.5.

For $\#\mathbf{W}[1]$-hardness, we construct a reduction from the problem $\#\text{Hom}(\mathit{spasms}(\mathcal{H}) \cap \mathcal{C}_F \to \mathcal{C}_F)$, where $\mathit{spasms}(\mathcal{H})$ is the set of all spasms of graphs in $\mathcal{H}$. We start with the following observation.

Claim 9.8. *The class $\mathit{spasms}(\mathcal{H}) \cap \mathcal{C}_F$ has unbounded treewidth if the class $\mathcal{H} \cap \mathcal{C}_F$ has unbounded matching number.*

Proof. Let $b \in \mathbb{N} \setminus \{0\}$ denote a positive integer. We show that there is a graph of treewidth at least b in the class $\mathit{spasms}(\mathcal{H}) \cap \mathcal{C}_F$. By assumption, there is a graph $H \in \mathcal{H} \cap \mathcal{C}_F$ such that H contains a matching M of size at least $|E(F)| \cdot b^2$; recall that $|E(F)|$ is a constant as the graph F is fixed.

Let c denote an F-coloring of H. Abusing notation, for an edge $\{u,v\}$ we write $c(\{u,v\}) := \{c(u),c(v)\}$ and say that c maps the edge $\{u,v\}$ to $c(\{u,v\})$. Now, as $|M| \geq |E(F)| \cdot b^2$, the coloring c maps at least b^2 edges in M to a common edge e in H. Let us call these edges $\{u_1,v_1\},\dots,\{u_{b^2},v_{b^2}\}$—we have $c(\{u_i,v_i\}) = e$ for all $i \in \{1,\dots,b^2\}$.

As M is a matching, we have that the vertices $U := \{u_1,\dots,u_{b^2}\}$ and $V := \{v_1,\dots,v_{b^2}\}$ form independent sets.[91] Hence, we can contract vertices in U and V such that we obtain a complete bipartite graph $K_{b,b}$ with b vertices on each side—Write ρ for the induced partition of $V(H)$. Observe that H/ρ is still F-colorable as we contracted only (independent) vertices that have the same color. Thus, we have that $H/\rho \in \mathit{spasms}(\mathcal{H}) \cap \mathcal{C}_F$. As the treewidth of $K_{b,b}$ is b and the treewidth of a graph cannot increase by taking subgraphs, we obtain that H/ρ has treewidth at least b. ⌟

91. That is, they are pairwise not connected by an edge.

By Claim 9.8 and Lemma 8.19, a reduction from the problem $\#\text{Hom}(\mathit{spasms}(\mathcal{H}) \cap \mathcal{C}_F \to \mathcal{C}_F)$ yields the desired $\#\mathbf{W}[1]$-hardness for $\#\text{Sub}(\mathcal{H} \to \mathcal{C}_F)$.

Now, for the reduction itself, we use Equation (7.9) and in particular Corollary 7.13: Given a graph $S \in \mathit{spasms}(\mathcal{H}) \cap \mathcal{C}_F$ we can compute[92] a graph $H \in \mathcal{H}$ with $F \in \mathit{spasms}(H)$ and

92. This takes time depending only on S.

$$\#Sub(\, H \to \star \,) = \sum_{R \in spasms(H)} \mathsf{a}_{Sub(\, H \to \star\,)}(\, R\,) \cdot \#Hom(\, R \to \star\,). \quad (9.1)$$

Next, observe that for F-colorable graphs G, in Equation (9.1) we have $\#Hom(\, R \to G\,) = 0$ for any graph R that is not F-colorable.

Hence, using an oracle query (H, G) for $\#\textsc{Sub}(\mathcal{H} \to \mathcal{C}_F)$ for any F-colorable graph G, we can compute the left-hand side of Equation (9.1); finally, using Lemma 9.6, we can also extract the specific summand $\#Hom(\, S \to G\,)$ from the right-hand side to solve the original instance to $\#\textsc{Hom}(spasms(\mathcal{H}) \cap \mathcal{C}_F \to \mathcal{C}_F)$. ⌟

9.2 Counting Homomorphisms to Kőnig graphs

We write $\mathcal{L}$ for the class of all line graphs. By Kőnig's Theorem, a line graph of a bipartite graph is also a perfect graph (see for instance [31]). To simplify notation, we hence call a line graph of a bipartite graph a *Kőnig graph*. We write ♕ to denote the class of all Kőnig graphs[93]. In this section, we analyze the complexity of the problem $\#\textsc{Hom}(\mathcal{H} \to$ ♕$)$ of counting homomorphisms from a graph from an arbitrary graph class $\mathcal{H}$ to a Kőnig graph.

93. The symbol ♕ is used since "König" is the German word for "King".

Checking Existence of Homomorphisms to Kőnig Graphs is FPT

We start by discussing the decision version, that is, the problem $\textsc{Hom}(\top \to$ ♕$)$. It turns out that if we are interested only in the existence, and not the number, of homomorphisms, then the problem becomes fixed-parameter tractable.

⌜ Theorem 9.9. *The decision problem $\textsc{Hom}(\top \to \mathcal{L})$ is FPT. Thus, also $\textsc{Hom}(\top \to$ ♕$)$ is FPT. In particular, given a graph H and a line graph L, it is possible to decide $\#Hom(\, H \to L\,) > 0$ in time $f(|V(H)|) \cdot O(|V(L)|^2)$, for some computable function f independent of H and L.*

▭ Proof. For the algorithm, we use the so-called clique partition of line graphs [58, Chapter 8]—we can partition the graph $E(L)$ into cliques such that every vertex of L is contained in at most two cliques. Here, every clique corresponds to a vertex of a primal graph of G such that $L(G) = L$. In particular, we can easily see that the size of the largest clique in the partition is exactly the maximum degree of G. Hence, we first compute a primal graph G of L, Next, we compute the maximum degree d of G.

Now set $k := |V(H)|$ and let $H_1, \dots, H_\ell$ denote the connected components of H. For every connected component H_i, we proceed as follows. If $d \geq k$ then there is a homomorphism from H_i to L, as we can embed H_i into a clique of size d. Otherwise, the properties of the clique partition yield that the degree of L is bounded by $2k$: Every vertex of L is contained in at most two cliques and every clique is of size at most $d < k$. Hence, we can perform a standard bounded search-tree algorithm: We guess the image $v \in V(L)$ of a vertex $h \in V(H_i)$. As the graph H_i is connected and $|V(H_i)| \leq k$, every homomorphism from H_i to L that maps h to v must also map every further vertex of H_i to a vertex in the k-neighborhood of v; which we can explore via brute-force to search for a homomorphism. Finally, we output 1 if we found a homomorphism from every connected component H_i and 0 otherwise.

For the running time, we can compute the primal graph G of L in time $O(|V(L)|^2)$ [80]; we can find the maximum degree d of G easily in $O(|V(G)|) = O(|V(L)|^2)$. In our bounded search tree approach, observe that as the maximum degree of L is at most $2k$, the size of the graph induced by the k-neighborhood of a vertex v is bounded by $(2k)^k$. Hence, the brute-force search for a homomorphism runs in time depending only on k.

The total running time is bounded by

$$O(|V(L)|^2) + f(|V(H)|) \cdot O(|V(L)|) \leq f(|V(H)|) \cdot O(|V(L)|^2);$$

this completes the proof. ⌟

Implicit Criterion for Hardness of Counting Homomorphisms to Kőnig Graphs

The FPT algorithm for the decision version from Theorem 9.9 motivates the study of the counting version: The most interesting hardness results in counting complexity theory are for problems with a tractable decision version [117]. To understand the complexity of $\#\textsc{Hom}(\mathcal{H} \to ♔)$, we first turn to the Complexity Monotonicity Framework again. This yields the following exhaustive, but *implicit* complexity classification.

THEOREM 9.10. *Let $\mathcal{H}$ denote a recursively enumerable class of graphs. Then the problem $\#\textsc{Hom}(\mathcal{H} \to ♔)$ is either fixed-parameter tractable or $\#\mathbf{W}[1]$-hard under parameterized Turing-reductions.*

Note that Kőnig graphs are a subset of the perfect graphs [31], as well as a subset of the line graphs (of arbitrary graphs). As line graphs are also claw-free graphs [8], Kőnig graphs are also a subset of the claw-free graphs. Hence, the hardness result for the problem $\#\textsc{Hom}(\mathcal{H} \to ♔)$ extends to perfect graphs, line graphs, and claw-free graphs as well.

To apply the Complexity Monotonicity Framework, we need a theorem by Whitney [120] that allows us to "reverse" the line graph operation.

FACT 9.11 (WHITNEY [120]). *Let H denote a connected line graph that is not isomorphic to the triangle. Then there is a unique graph F (up to isolated vertices) such that $L(F) = H$.*[94]

94. Recall from Figure 7.6c that both the triangle and the claw have the triangle as their line graph.

As the triangle is not bipartite, Fact 9.11 allows us to define the *inverse line graph operation* for line graphs of bipartite graphs G. Formally, write $\mathcal{C}_{-}$ for the class of bipartite graphs without isolated vertices, and set $L^{-1} : ♔ \to \mathcal{C}_{-}$, $L^{-1}(L(G)) := G$. In particular, we have that $L^{-1}(K_3) = K_{1,3}$.

As a useful property of L^{-1}, we observe that L^{-1} is *almost* compatible with (induced) subgraph numbers.

LEMMA 9.12. *For any line graph H and any Kőnig graph G, we have*

$$\#\mathit{IndSub}(\,H \to G\,) = \begin{cases} \#\mathit{Sub}(\,L^{-1}(H) \to L^{-1}(G)\,), & H \in ♔; \\ 0, & \text{else.} \end{cases} \qquad (9.2)$$

PROOF. Assume first that the graph $H = L(F)$ is a Kőnig graph.

Consider $S \subseteq V(G) = E(L^{-1}(G))$ such that $G[\,S\,] \cong H$ and set $D := L^{-1}(G)\{\,S\,\}$. By construction, we have $L(D) \cong H$. Thus, the graphs F and D are isomorphic by Fact 9.11 and the fact that neither F nor D is the triangle.[95]

Now let D denote a subgraph of $L^{-1}(G)$ that is isomorphic to F. As the graph F does not contain isolated vertices, D is determined by its set of edges. We set $S = E(D)$ and consider the induced subgraph $G[\,S\,]$. By construction, $G[\,S\,]$ is isomorphic to $L(D)$ which in turn is isomorphic to $L(F)$ (as the graphs D and F are isomorphic). This shows correctness for $H = L(F) \in ♔$.

If the graph H is not a Kőnig graph, then H is not a triangle. Thus, we have that $H = L(F)$ for some non-bipartite graph F which is uniquely determined (up to isolated vertices) by Fact 9.11. In this case, the same argument as before shows that any induced subgraph of G that is isomorphic to H yields a subgraph of $L^{-1}(G)$ that is isomorphic to F. As $L^{-1}(G)$ is bipartite and F is not, such an induced subgraph cannot exist; note that bipartite graphs are closed under taking subgraphs.

[95] Note that a bipartite graph cannot contain a subgraph isomorphic to a triangle.

In a next step, we convince ourselves that counting homomorphisms in Kőnig graphs is indeed a graph motif parameter.

LEMMA 9.13. *For any graph H and any Kőnig graph $G \in ♔$, we have*

$$\#\mathit{Hom}(\,H \to G\,) = (\alpha \cdot \#\mathit{Hom})(L^{-1}(G)),$$

for a finitely supported $\alpha \in \mathbb{Q}^{\top}$.

Proof. Fix a graph H and a Kőnig graph $G = L(F)$ and consider $\#\mathit{Hom}(\, H \to G\,)$. Combining Equations (7.3) and (7.5), we obtain

$$\begin{aligned}
\#\mathit{Hom}(\, H \to G\,) &= \sum_{\rho} \#\mathit{Emb}(\, H/^{\circ}\rho \to G\,) \\
&= \sum_{\rho} \sum_{\substack{R \\ \#\mathit{Ext}(\, H/^{\circ}\rho \to R\,)>0}} \#\mathit{Ext}(\, H/^{\circ}\rho \to R\,) \cdot \#\mathit{IndSub}(\, R \to G\,) \\
&=: \sum_{\substack{R \\ |V(R)| \le |V(H)|}} \mathsf{a}^{-1}_{\mathit{IndSub}(\, H \to \star\,)}(\, R\,) \cdot \#\mathit{IndSub}(\, R \to G\,)
\end{aligned}$$

Now, we use Lemma 9.12 to replace G with $L^{-1}(G)$.

$$\#\mathit{Hom}(\, H \to G\,) = \sum_{\substack{R \in \text{♔} \\ |V(R)| \le |V(H)|}} \mathsf{a}^{-1}_{\mathit{IndSub}(\, H \to \star\,)}(\, R\,) \cdot \#\mathit{Sub}(\, R \to L^{-1}(G)\,)$$

Finally, we use Equation (9.1) to return to homomorphisms.

$$\begin{aligned}
\#\mathit{Hom}(\, H \to G\,) &= \sum_{\substack{R \in \text{♔} \\ |V(R)| \le |V(H)|}} \mathsf{a}^{-1}_{\mathit{IndSub}(\, H \to \star\,)}(\, R\,) \\
&\cdot \left(\sum_{T \in \mathit{spasms}(R)} \mathsf{a}_{\mathit{Sub}(\, R \to \star\,)}(\, T\,) \cdot \#\mathit{Hom}(\, T \to L^{-1}(G)\,) \right) \qquad (9.3)
\end{aligned}$$

We skip the pain of expanding Equation (9.3), just observe that the right-hand side is indeed a graph motif parameter.

Using Lemma 9.13, we obtain a proof for Theorem 9.10.

Theorem 9.10. *Let $\mathcal{H}$ denote a recursively enumerable class of graphs. Then the problem $\#\text{Hom}(\mathcal{H} \to \text{♔})$ is either fixed-parameter tractable or $\#\mathbf{W}[\mathbf{1}]$-hard under parameterized Turing-reductions.*

Proof. Consider the graph class $\mathcal{H}_S$ of all bipartite graphs in any support Q_H for $H \in \mathcal{H}$.

$$\mathcal{H}_S := \bigcup_{H \in \mathcal{H}} (Q_H) \cap \mathcal{C}_{-}.$$

We proceed to show $\#\textsc{Hom}(\mathcal{H} \to ♔) \equiv_T^{fpt} \#\textsc{Hom}(\mathcal{H}_S \to \mathcal{C}_-)$. Note that this implies Theorem 9.10 by the classification of F-colorable graphs (Theorem 9.1) for $F = P_2$.

For the direction $\#\textsc{Hom}(\mathcal{H} \to ♔) \leq_T^{fpt} \#\textsc{Hom}(\mathcal{H}_S \to \mathcal{C}_-)$, assume that graphs $H \in \mathcal{H}$ and $G \in ♔$ are given. By Lemma 9.14, we can compute (in time depending only on H) a vector $\alpha_H \in \mathbb{Q}^\top$ with

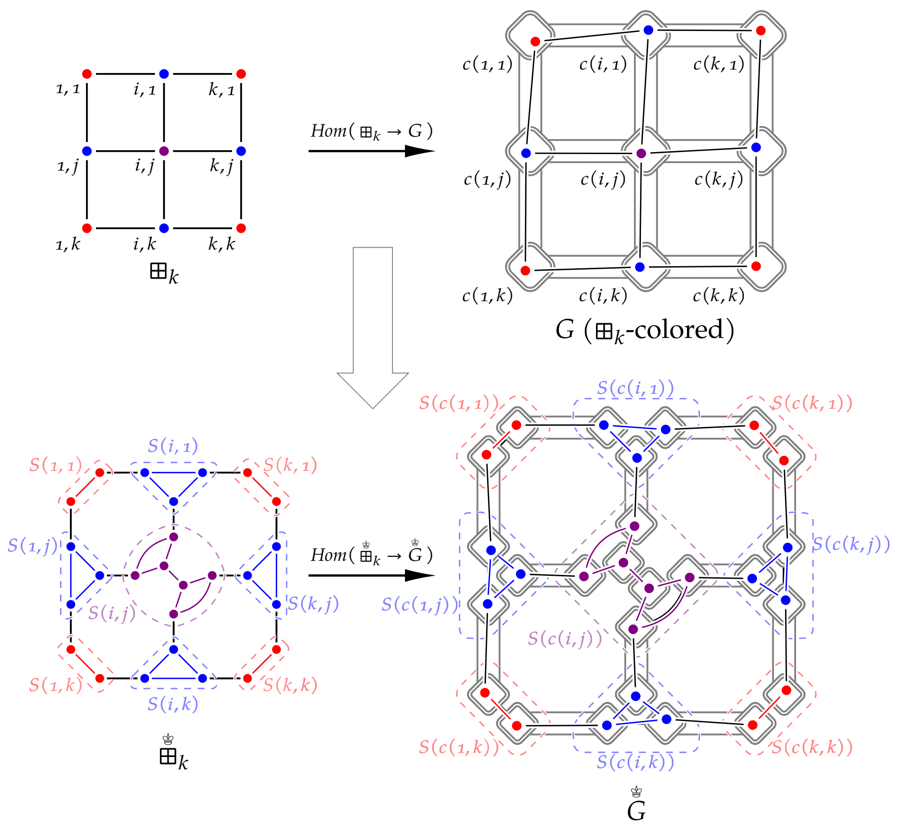
$$\#Hom(\,H \to G\,) = (\alpha_H \cdot \#Hom)(L^{-1}(G)). \tag{9.4}$$

As G is a Kőnig graph, we have that $L^{-1}(G)$ is bipartite. Hence, by Lemma 9.2, there is no homomorphism from a graph D to $L^{-1}(G)$ whenever D is not bipartite. Observe that we can verify whether a graph $D \in supp(\alpha)$ is bipartite in time depending only on α and thus $|V(H)|$. All other terms $\#Hom(\,D \to G\,)$ with $D \in \mathcal{C}_-$ can be obtained by querying the oracle for the problem $\#\textsc{Hom}(\mathcal{H}_S \to \mathcal{C}_-)$. Finally, we compute and the linear combination in Equation (9.4). This completes the first reduction.

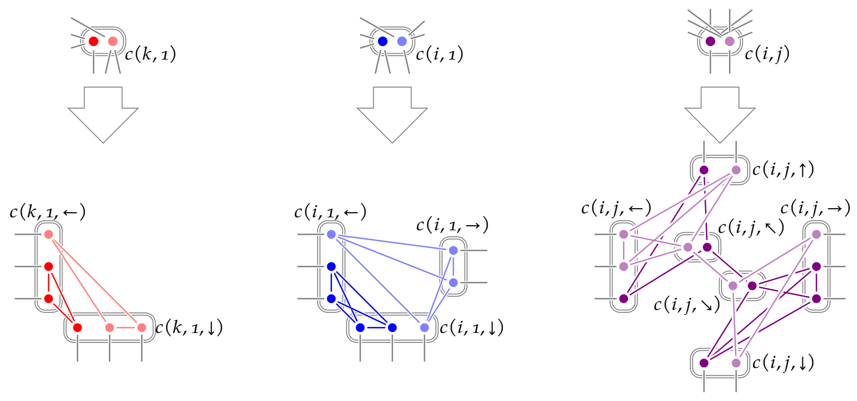
For the other direction, $\#\textsc{Hom}(\mathcal{H}_S \to \mathcal{C}_-) \leq_T^{fpt} \#\textsc{Hom}(\mathcal{H} \to ♔)$, assume that graphs $D \in \mathcal{H}_S$ and $G \in \mathcal{C}_-$ are given. By definition of the class $\mathcal{H}_S$, we have that the graph D is a bipartite graph in Q_H for some $H \in \mathcal{H}$. As $\mathcal{H}$ is recursively enumerable and the mapping $H \mapsto \alpha_H$ is computable, we can compute α_H in time depending only on $|V(F)|$ and α_H. By Lemma 9.6, we can also extract any summand of α_H using the oracle for $\#\textsc{Hom}(\mathcal{H} \to ♔)$ This completes the second reduction, and hence the proof. ⌟

An Explicit Criterion for Hardness of Counting Homomorphisms to Kőnig Graphs

Dissatisfied with the *implicit* nature of Theorem 9.10, we wish to find *explicit* criteria on $\mathcal{H}$ that make $\#\textsc{Hom}(H \to ♔)$ hard. Hence, we construct an explicit reduction from #Clique next, thereby proving the following hardness result.



(a) Corner vertices (red), border vertices (blue), and interior vertices (purple) get replaced by corresponding gadgets; the resulting graph $\overset{\text{♔}}{G}$ is a Kőnig graph. We use $c(\star)$ to denote the set of all vertices colored with $\star$ and $S(\star)$ to denote the gadget corresponding to $\star$; the numbers i and j denote intermediate columns and rows.



(b) The gadgets of Lemma 9.14 in detail.

FIGURE 9.1. General overview of the reduction from Lemma 9.14.

LEMMA 9.14. *Let $\mathcal{H}$ denote a recursively enumerable class of graphs. If $\mathcal{H}$ has unbounded treewidth and is closed under taking minors, then the problem* #HOM$(\mathcal{H} \to ♔)$ *is* #**W**[**1**]*-hard.*

PROOF. We prove #CP-HOM$(\boxplus \to \top) \leq_T^{fpt}$ #HOM$(\mathcal{H} \to ♔)$. The result then follows from the reduction Lemma 8.16. For the reduction, we use a gadget construction that transforms an arbitrary (grid-colored) graph G into a Kőnig graph such that the number of homomorphisms remains stable; see Figure 9.1a for an overview of the construction.

Let F denote a $\boxplus_k$-colored graph. We next discuss how to construct a Kőnig graph $\overset{♔}{F}$ from F. Let c denote the coloring of F. We partition the vertices of F into three disjoint sets (again, consider Figure 9.1a):

1 A vertex v is called a *corner vertex* if its coloring $c(v)$ is one of the values $(1,1)$, $(1,k)$, $(k,1)$, and (k,k).
2 A vertex v is called a *border vertex* if its coloring $c(v)$ satisfies

$$c(v) \in \{(i,j) \in [k]^2 \mid i \in \{1,k\} \vee j \in \{1,k\}\} \setminus \{1,k\} \times \{1,k\}.$$

3 All remaining vertices are called *interior vertices*.

We construct a gadget (graph) $S(v)$ for each vertex $v \in V(F)$, depending on the type of v.

1 The vertex v is a corner vertex. Assume that $c(v) = (1,1)$; the other cases are symmetric. Now let $N_\rightarrow$ denote the set of neighbors of v that are colored by c with $(1,2)$ and let $N_\downarrow$ denote the set of all neighbors of v that are colored by c with $(2,1)$. Note that that those are all neighbors of v as F is $\boxplus_k$-colored. For each vertex $u \in N_\rightarrow$, we add a vertex $v^u_\rightarrow$ and color it with $(1,1,\rightarrow)$. For each vertex $u \in N_\downarrow$, we add a vertex $v^u_\downarrow$ and color it with $(1,1,\downarrow)$. We obtain $S(v)$ by making all of the previous vertices adjacent to each other.
2 The vertex v is a border vertex. Assume that $c(v) = (1,j)$; the other cases are symmetric. Now let $N_\rightarrow$ denote the set of neighbors of v that are colored by c with $(1,j+1)$, let $N_\leftarrow$ denote the

set of neighbors of v that are colored by c with $(1, j - 1)$, and let $N_{\downarrow}$ denote the set of all neighbors of v that are colored by c with $(2, j)$. For each vertex $u \in N_{\rightarrow}$, we add a vertex $v^{u}_{\rightarrow}$ and color it with $(1, i, \rightarrow)$. We proceed similarly with $N_{\leftarrow}$ and $N_{\downarrow}$. We obtain $S(v)$ by making all of the previous vertices adjacent.

3 The vertex v is an interior vertex. Let v have color $c(v) = (i, j)$ and let $N_{\rightarrow}, N_{\leftarrow}, N_{\uparrow}$ and $N_{\downarrow}$ denote the sets of neighbors of v that are colored by c with $(i, j + 1)$, $(i, j - 1)$, $(i - 1, j)$ and $(i + 1, j)$, respectively. We add a new vertex $v^{u}_{\rightarrow}$ for each vertex $u \in N_{\rightarrow}$ and color it with $(i, j, \rightarrow)$; we proceed similarly with the sets $N_{\uparrow}, N_{\leftarrow}$, and $N_{\downarrow}$. Next, we add two new vertices $v^{\nwarrow}$ and $v^{\searrow}$, color them $(i, j, \nwarrow)$ and $(i, j, \searrow)$, and connect them by an edge. Then we create two cliques: The first clique contains the vertex $v^{\star}$ and all vertices that we colored with $(i, j, \leftarrow)$ or with $(i, j, \uparrow)$. The second clique contains the vertex $v^{\searrow}$ and all vertices that we colored with $(i, j, \rightarrow)$ or with $(i, j, \downarrow)$. The resulting graph is $S(v)$.

We obtain the graph $\overset{♔}{F}$ by connecting the gadgets as follows: Let $\{v, w\} \in E(F)$ denote an edge of F and assume that the vertex v has color $c(v) = (i, j)$ and the vertex w has color $c(w) = (i, j + 1)$; the remaining cases are processed similarly. By construction, the graph $S(v)$ contains a vertex $v^{w}_{\rightarrow}$ and the graph $S(w)$ contains a vertex $w^{v}_{\leftarrow}$. We connect those two vertices with an edge.

We first observe that this construction yields a planar graph if it is applied to the grid itself (Again, consider Figure 9.1a). Further, when applied to the graph F, we indeed obtain a Kőnig graph:

CLAIM 9.15. *If F is $\boxplus_k$-colored, then $\overset{♔}{F}$ is a Kőnig graph.*

PROOF. We construct a bipartite graph B such that the line graph $L(B)$ of B is the graph $\overset{♔}{F}$. To this end, observe that the gadgets of corner and border vertices are cliques, and the gadgets of interior vertices are two cliques that are connected by a single edge. Hence, the entire graph $\overset{♔}{F}$ is obtained by connecting vertex disjoint cliques with edges that are pairwise disjoint.

Now recall that a clique K_d is the line graph of the star $K_{1,d}$; consider also Figure 7.6a again for a visualization. For $K_{1,d}$, we call the single vertex of degree d the *center* of $K_{1,d}$; we call the other d vertices the *rays* of $K_{1,d}$.

Now, adding an edge between two vertex disjoint cliques corresponds to merging the right vertices of the corresponding rays of the primal graphs (Recall Figure 7.6b for a visualization). Consequently, we can construct a graph B whose line graph is $\overset{\text{♔}}{F}$ by merging right vertices of the rays corresponding to the edges that connect the cliques of the gadgets.

Finally, it is easy to see that B is bipartite: A 2-coloring is given by the function that maps the centers to 1 and the rays to 2. ◿

Now, consider the graphs $\overset{\text{♔}}{\boxplus}_k$ and $\overset{\text{♔}}{G}$ and recall that we colored the vertices of $\overset{\text{♔}}{G}$ (and $\overset{\text{♔}}{\boxplus}_k$) with triples $(i, j, \star) \in V(\overset{\text{♔}}{\boxplus}_k)$, where $\star$ is one of the symbols $\rightarrow, \leftarrow, \uparrow, \downarrow, \nwarrow$, and $\searrow$. Call this mapping $\overset{\text{♔}}{c}$ and observe that $\overset{\text{♔}}{c}$ is in fact not a $\overset{\text{♔}}{\boxplus}_k$-coloring of $\overset{\text{♔}}{G}$, as two vertices of G of the same color are adjacent in the gadget construction.

Hence, define a *weakly color-prescribed* homomorphism as a $h \in$ *Hom*$(\overset{\text{♔}}{\boxplus}_k \to \overset{\text{♔}}{G})$ such that the mapping $h \circ \overset{\text{♔}}{c}$ is the identity. We abuse notation and write *cp-Hom*$(\overset{\text{♔}}{\boxplus}_k \to \overset{\text{♔}}{G})$ for the set of all weakly color-prescribed homomorphisms.

CLAIM 9.16. *We have #cp-Hom*$(\overset{\text{♔}}{\boxplus}_k \to \overset{\text{♔}}{G})$ = *#cp-Hom*$(\boxplus_k \to G)$.

PROOF. Let $h \in$ *cp-Hom*$(\boxplus_k \to G)$ denote a color-prescribed homomorphism. We define the mapping $\overset{\text{♔}}{h} \in$ *cp-Hom*$(\overset{\text{♔}}{\boxplus}_k \to \overset{\text{♔}}{G})$ as follows: For every $i, j \in [k]$ we set

$$\begin{aligned}
\overset{\text{♔}}{h}(i,j,\rightarrow) &:= h(i,j)_{\rightarrow}^{h(i,j+1)}; & \overset{\text{♔}}{h}(i,j,\leftarrow) &:= h(i,j)_{\leftarrow}^{h(i,j-1)}\\
\overset{\text{♔}}{h}(i,j,\downarrow) &:= h(i,j)_{\downarrow}^{h(i+1,j)}; & \overset{\text{♔}}{h}(i,j,\uparrow) &:= h(i,j)_{\uparrow}^{h(i-1,j)}\\
\overset{\text{♔}}{h}(i,j,\searrow) &:= h(i,j)^{\searrow}; & \overset{\text{♔}}{h}(i,j,\nwarrow) &:= h(i,j)^{\nwarrow}
\end{aligned}$$

The construction of $\overset{\text{♔}}{G}$ immediately yields that $\overset{\text{♔}}{h}$ is a (weakly color-prescribed) homomorphism if h is color-prescribed. Further, the mapping $h \mapsto \overset{\text{♔}}{h}$ is a bijection. ◿

Next, observe that the proof of Lemma 8.6 generalizes to weakly color-prescribed homomorphisms: We need only that every automorphism of $\overset{\text{♔}}{\boxplus}_k$ can be decomposed into a homomorphism and $\overset{\text{♔}}{c}$, which is the case as $\overset{\text{♔}}{c}$ is not a homomorphism only because of edges between vertices of the same color.

Hence, we can compute $\#\mathit{cp\text{-}Hom}(\overset{\text{♔}}{\boxplus}_k \to \overset{\text{♔}}{G})$ with an oracle for $\#\mathit{Hom}(\overset{\text{♔}}{\boxplus}_k \to \star)$. Recall from the proof of Claim 8.8 that we query the oracle on graphs obtained from $\overset{\text{♔}}{G}$ by deleting vertices.

Next, observe that Kőnig graphs are closed under the removal of vertices: Deleting a vertex in a Kőnig graph is equivalent to deleting an edge in primal bipartite graph and bipartiteness is closed under the removal of edges. Thus, the oracle needs to support querying only for Kőnig graphs.

Now recall that the graph $\overset{\text{♔}}{\boxplus}_k$ is planar. By Fact 8.17, every class $\mathcal{H}$ of unbounded treewidth contains arbitrary large grids as minors. Further, every planar graph is the minor of some grid [97]. As the class $\mathcal{H}$ is minor-closed, we hence obtain that $\overset{\text{♔}}{\boxplus}_k$ is contained in $\mathcal{H}$ for every $k \in \mathbb{N} \setminus \{0\}$. Therefore, an oracle for the problem $\#\textsc{Hom}(\mathcal{H} \to ♔)$ suffices, completing the proof.

As an application of Lemma 9.14, we obtain that for minor-closed graph classes, the hardness of Lemma 8.15 can be restricted to the case when tho host graphs are Kőnig graphs.

Theorem 9.17. *Let $\mathcal{C}$ denote one of the classes of line-graphs, claw-free graphs or perfect graphs, or a non-empty union thereof. Further, let $\mathcal{H}$ denote a recursively enumerable class of graphs.*

1 *If the treewidth of $\mathcal{H}$ is bounded, then $\#\textsc{Hom}(\mathcal{H} \to \mathcal{C})$ is solvable in polynomial time.*
2 *Else, if $\mathcal{H}$ is additionally minor-closed, $\#\textsc{Hom}(\mathcal{H} \to \mathcal{C})$ is $\#\mathbf{W[1]}$-hard.*

Proof. The reduction $\#\textsc{Hom}(\mathcal{H} \to \mathcal{C}) \leq^{fpt}_{T} \#\textsc{Hom}(\mathcal{H} \to \top)$ is immediate. In fact, the reduction is the identity and preserves not only fixed-parameter tractability, but polynomial-time tractability as well. By Lemma 8.15, #Hom($\mathcal{H} \to \top$) is solvable in polynomial time if the class $\mathcal{H}$ has bounded treewidth.

If the class $\mathcal{H}$ has unbounded treewidth and is minor-closed, #**W[1]**-hardness follows from Lemma 9.14, as Kőnig graphs are claw-free [8], perfect [31] and, of course, line graphs. ◿

Theorem 9.17 raises the question of *"for which restrictions of* $\top$ *can we obtain similar hardness results?"* In 11 we show that an answer to that question is *"definitely not for every graph class"*—We construct a graph class, for which a dichotomy similar to Lemma 8.15 is *very, very unlikely*. While this answer may satisfy some, we leave the original question unanswered for everyone else.

Homomorphism Vectors of Graph Properties

10

In this chapter, we explore the hardness of the problem #IndSubΦ. In particular, we are interested in explicit criteria on Φ that guarantee hardness. As a main result, we show that it suffices that Φ is subgraph-closed to obtain almost-tight conditional lower bounds. All results are based on a better understanding of when the coefficient $\mathsf{a}_{IndSub(\Phi,k\to\star)}(F)$ is nonzero for high-treewidth graphs F.

10.1 f-Vectors and h-Vectors

In this chapter, we generalize two topological invariants of simplicial complexes to graph properties: The f-vector and the h-vector.[96]

96. We do not need simplicial graph complexes in this work. Hence we skip their definition. Consult Billera and Björner [13] to learn more about simplicial (graph) complexes.

Definition 10.1. *Let Φ denote a graph property, let k denote a positive integer and set $d = \binom{k}{2}$. The* f-vector $f^{\Phi,k} = (f_i^{\Phi,k})_{i=0}^{d}$ *of Φ and k contains the number of edge-subsets of size i of K_k such that the induced graph satisfies Φ;*[97]

97. In some parts of the literature, the f-vector comes with an index shift of -1 due to the topological interpretation of simplicial complexes.

$$f_i^{\Phi,k} := \#\{A \in \binom{E(K_k)}{i} \mid \Phi(K_k\{A\}) = 1\}, \quad i \in \{0,\dots,d\}.$$

The h-vector $h^{\Phi,k} = (h_\ell^{\Phi,k})_{\ell=0}^{d}$ *is defined by*

$$h_\ell^{\Phi,k} := \sum_{i=0}^{\ell} (-1)^{\ell-i} \cdot \binom{d-i}{\ell-i} \cdot f_i^{\Phi,k}, \quad \ell \in \{0,\dots,d\}.$$

We show that if suitable entries $h_\ell^{\Phi,k}$ are nonzero, there are a corresponding, nonzero numbers $\mathsf{a}_{IndSub(\Phi,k\to\star)}(F)$ for graphs F of high treewidth—and thus hardness for #IndSub(Φ). In some sense, we can view the result of Roth and Schmitt [100] as a very restricted special case as it shows that the non-vanishing of the reduced Euler characteristic of the complex (which is equal to the entry $h_d^{\Phi,k}$) implies non-vanishing of $\mathsf{a}_{IndSub(\Phi,k\to\star)}(K_k)$.

For many graph properties it is easy to obtain information about the f-vector (for instance that $f_\ell^{\Phi,k} = 0$ for sufficiently large ℓ with respect to k). We show that the f and h-vectors of a graph property are related by the so-called the f-polynomial.[98]

98. The f-polynomial is again a generalization of the same polynomial for simplicial complexes.

Definition 10.2. *Let Φ denote a graph property, let k denote a positive integer, and set $d = \binom{k}{2}$. The* f-polynomial *of Φ and k is the following univariate polynomial of degree at most d:*

$$\mathsf{f}_{\Phi,k}(x) := \sum_{i=0}^{d} f_i^{\Phi,k} \cdot x^{d-i}.$$

We are especially interested in the values of the *derivatives* of the f-polynomial at 0 and -1—As we see in the proof of Lemma 10.9, these values yield (up to combinatorial factors) the f-vector and the h-vector. Intuitively, we apply Hermite-Birkhoff interpolation (which we introduce below) on $\mathsf{f}_{\Phi,k}$ and its derivatives to prove that specific entries of $h^{\Phi,k}$ cannot vanish in case a sufficient number of entries of $f^{\Phi,k}$ do, unless Φ is false on all k-vertex graphs.

Hermite-Birkhoff Interpolation and Pólya's Theorem

In the classical interpolation problem, we seek to recover a (univariate) polynomial f from evaluations of f at (pairwise) different points. It is well known that we need $d + 1$ evaluations to *uniquely* recover a polynomial of degree d.

We need a slightly different kind of interpolation: Instead of evaluations of f at different points, we have access only to evaluations of f *and its derivatives* at a very limited number m of distinct points each—but we still wish to uniquely recover f. This problem is called *Hermite-Birkhoff interpolation*. For our purposes, it turns out that we need to evaluate polynomials only at at most two positions, namely 0 and -1. Fortunately, this case was fully solved by Pólya [94].

To be more formal, we follow the notation of Schoenberg [106].

FACT 10.3 (PÓLYA [94], SCHOENBERG [106]). *Let* $(\varepsilon_{ij}) \in \{0,1\}^{2\times d+1}$ *denote a matrix with* $\sum_{i,j} \varepsilon_{ij} = d+1$ *and set* $x_1 := -1$ *and* $x_2 := 0$. *Suppose that for all* $j \in \{0, \dots, d-1\}$, *we have*

$$\sum_{\ell=0}^{j} \varepsilon_{1\ell} + \varepsilon_{2\ell} \geq j+1.$$

Then, and only then, the system of $d+1$ *differential equations*

$$\mathtt{f}^{(j)}(x_i) = 0 \quad (\textit{for every } \varepsilon_{ij} = 1)$$

has $\mathtt{f} = \mathtt{0}$ *as a unique solution with degree at most* d.

10.2 CHARACTERIZING HOMOMORPHISM COEFFICIENTS VIA *f*-VECTORS AND *h*-VECTORS

In this section we prove the main technical result of this chapter:

THEOREM 10.4. *Write* Φ *for a computable graph property,* k *for a positive integer, and* w *for the Hamming weight*[99] *of* $f^{\Phi,k}$. *Suppose that* Φ *is not false on all* k*-vertex graphs. Then there is a graph* K *on* k *vertices and at least* $\binom{k}{2} - w + 1$ *edges with* $\mathsf{a}_{\mathit{IndSub}(\Phi,k\to\star)}(K) \neq 0$.

99. That is, the number of nonzero entries.

First, recall from Equation (8.3) that for any computable graph property Φ and positive integer k, there is a unique computable function $\mathsf{a} : \top \to \mathbb{Q}$ (with finite support) satisfying

$$\#\mathit{IndSub}(\Phi, k \to \star) = \sum_{\substack{H \\ |V(H)|\leq k}} \mathsf{a}_{\mathit{IndSub}(\Phi,k\to\star)}(H) \cdot \#\mathit{Hom}(H \to \star)$$

Now, for the remainder of the section, fix a (computable) graph property Φ and a positive integer k (and thus the function a). This allows us to simplify the notation for the f and h-vectors, as well as for the f-polynomial: We write $f := f^{\Phi,k}$, $h := h^{\Phi,k}$, and $\mathtt{f} := \mathtt{f}_{\Phi,k}$. Further, set $d := \binom{k}{2}$ and write $\mathcal{H}_i$ for the set of all graphs on k vertices and with i edges.

Next, we define the vector $\hbar_i$ as

$$\hbar_i := \sum_{K \in \mathcal{H}_i} \mathsf{a}_{IndSub(\Phi,k\to\star)}(K), \quad i \in \{0,\dots,d\},$$

that is, the i-th entry of $\hbar$ is the sum of the coefficients of graphs with k vertices and i edges in Equation (8.3). Now we are ready to establish the aforementioned connection between the coefficients of Equation (8.3) and the h-vector of the property Φ.

LEMMA 10.5. *We have $k! \cdot \hbar = h$.*[100]

100. Note that as a consequence, the h-vector of a simplicial graph complex is determined by the coefficients of its associated linear combination of homomorphisms.

PROOF. We start with investigating Equation (8.3) for graphs with exactly k vertices.

CLAIM 10.6 ([100]). *Let K denote a graph with k vertices.*

$$\mathsf{a}_{IndSub(\Phi,k\to\star)}(K) = \sum_{H \in \mathcal{G}_{\Phi,k}} \frac{(-1)^{\#E(K)-\#E(H)}}{\#Aut(H)} \cdot \#Ext(H \to K).$$

PROOF. Fix a graph K with k vertices. First observe that for K and a graph H with k vertices, Equation (7.11) simplifies to

$$\mathsf{a}_{IndSub(H\to\star)}(K) = \frac{(-1)^{\#E(K)-\#E(H)}}{\#Aut(H)} \cdot \#Ext(H \to K), \tag{10.1}$$

as all extensions R of H have k vertices and thus only the discrete partition $\perp$ may result in a quotient graph $R/^{\circ}\perp$ isomorphic to K. In particular, only for the (possible) extension K of H we may end up with a graph isomorphic to H—hence, we obtain Equation (10.1). Summing over all (k-vertex) graphs in $\mathcal{G}_{\Phi,k}$ yields the claim. ⌟

Next, we investigate the term $\sum_{K\in\mathcal{H}_\ell} \#Ext(H \to K)$.

CLAIM 10.7. *Let $\ell \in \{0,\dots,d\}$ denote an integer and let H denote a graph with k vertices and at most ℓ edges. Then, we have*

$$\sum_{K\in\mathcal{H}_\ell} \#Ext(H \to K) = \binom{d-\#E(H)}{\ell-\#E(H)}.$$

Proof. Any extension from the graph H to a graph with ℓ edges has to add $\ell - \#E(H)$ edges to H; there are exactly $d - \#E(H)$ possible choices for these $\ell - \#E(H)$ edges. Hence the claim follows from basic combinatorics.

Now, fix an $\ell \in \{0, \dots, d\}$; we proceed to show that $k! \cdot \hbar_\ell = h_\ell$, which yields the lemma. To that end, by definition of $\hbar$, we have

$$\begin{aligned} k! \cdot \hbar_\ell &= k! \cdot \sum_{K \in \mathcal{H}_\ell} \mathsf{a}_{IndSub(\Phi, k \to \star)}(K) \\ &= k! \cdot \sum_{K \in \mathcal{H}_\ell} \sum_{H \in \Phi_k} \#Aut(H)^{-1} \cdot (-1)^{\ell - \#E(H)} \cdot \#Ext(H \to K) \\ &= \sum_{H \in \Phi_k} k! \cdot \#Aut(H)^{-1} \cdot (-1)^{\ell - \#E(H)} \sum_{K \in \mathcal{H}_\ell} \#Ext(H \to K), \end{aligned}$$

where the second step holds due to Claim 10.6. Now observe that $\#Ext(H \to K) = 0$ if H has more edges than K. Thus, we have

$$\begin{aligned} k! \cdot \hbar_\ell &= \sum_{\substack{H \in \Phi_k \\ \#E(H) \le \ell}} k! \cdot \#Aut(H)^{-1} \cdot (-1)^{\ell - \#E(H)} \sum_{K \in \mathcal{H}_\ell} \#Ext(H \to K) \\ &= \sum_{\substack{H \in \Phi_k \\ \#E(H) \le \ell}} k! \cdot \#Aut(H)^{-1} \cdot (-1)^{\ell - \#E(H)} \cdot \binom{d - \#E(H)}{\ell - \#E(H)}, \end{aligned}$$

where the last equality holds by Claim 10.7.

Claim 10.8. *For any k-vertex graph H, we have $k! \cdot \#Aut(H)^{-1} = \#Sub(H \to K_k)$.*[101]

101. Note that H may have isolated vertices.

Proof. By Equation (7.1), showing $k! = \#Emb(H \to K_k)$ suffices. Observe that any permutation of $V(H)$ yields a valid embedding and that no (injective) embedding may map two vertices of H to the same vertex of K_k.

Finally, we observe

$$\begin{aligned} k! \cdot \hbar_\ell &= \sum_{\substack{H \in \Phi_k \\ \#E(H) \le \ell}} \#Sub(H \to K_k) \cdot (-1)^{\ell - \#E(H)} \cdot \binom{d - \#E(H)}{\ell - \#E(H)} \\ &= \sum_{i=0}^{\ell} \sum_{\substack{H \in \Phi_k \\ \#E(H) = i}} \#\{A \subseteq E(K_k) \mid K_k\{A\} \cong H\} \cdot (-1)^{\ell - i} \cdot \binom{d-i}{\ell - i} \\ &= \sum_{i=0}^{\ell} \#\{A \in \tbinom{E(K_k)}{i} \mid \Phi(K_k\{A\}) = 1\} \cdot (-1)^{\ell - i} \cdot \binom{d-i}{\ell-i} \\ &= \sum_{i=0}^{\ell} f_i \cdot (-1)^{\ell-i} \cdot \binom{d-i}{\ell-i} = h_\ell, \end{aligned}$$

completing the proof. ⌟

In the next step, we use Pólya's Theorem to prove that the Hamming weight of the f-vector determines an index β of the h-vector such that at least one entry of h with index at least β is non-zero. By Lemma 10.5 the same then follows for $\hbar$.

⌜ LEMMA 10.9. *Let w denote the Hamming weight of f and set $\beta = d - w$. If Φ is not false on all k-vertex graphs then at least one of the values $h_d, \dots, h_{\beta+1}$ is non-zero.*

PROOF. Recall $\mathtt{f}(x) := \sum_{i=0}^{d} f_i \cdot x^{d-i}$ and observe

$$\mathtt{f}^{(j)}(x) = \sum_{i=0}^{d-j} f_i \cdot \frac{(d-i)!}{(d-i-j)!} \cdot x^{d-j-i}.$$

Evaluating at $x = 0$, we obtain $\mathtt{f}^{(j)}(0) = f_{d-j} \cdot j!$. By assumption, we thus have $\mathtt{f}^{(j)}(0) = 0$ for $\beta + 1$ many indices j.

Further, we see that

$$\begin{aligned} \mathsf{f}^{(j)}(-1) &= \sum_{i=0}^{d-j} f_i \cdot \frac{(d-i)!}{(d-i-j)!} \cdot (-1)^{d-j-i} \\ &= j! \cdot \sum_{i=0}^{d-j} f_i \cdot \binom{d-i}{j} \cdot (-1)^{d-j-i} \\ &= j! \cdot \sum_{i=0}^{d-j} f_i \cdot \binom{d-i}{(d-j)-i} \cdot (-1)^{d-j-i} = j! \cdot h_{d-j}. \end{aligned}$$

Now assume for the sake of contradiction that each of the values $h_d, \dots, h_{\beta+1}$ is zero. Thus, $\mathsf{f}^{(j)}(-1) = 0$ for $j = 0, \dots, w-1$. Interpreting those evaluations of the derivatives of the *f*-polynomial as an instance of Hermite-Birkhoff interpolation, the corresponding matrix (ε_{ij}) looks as follows:[102]

102. Recall that an entry 1 in the matrix (ε_{ij}) represents an evaluation $\mathsf{f}^{(j)}(-1) = 0$ in the first row and an evaluation $\mathsf{f}^{(j)}(0) = 0$ in the second row.

$$\begin{array}{c} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 & 0 & \dots & 0 \\ \varepsilon_{20} & \varepsilon_{21} & \varepsilon_{22} & \dots & \varepsilon_{2(w-1)} & \varepsilon_{2w} & \dots & \varepsilon_{2d} \end{pmatrix} \\ \begin{matrix} 0 & 1 & 2 & \dots & w-1 & w & \dots & d \end{matrix} \end{array}$$

In particular, at least $\beta + 1 = d + 1 - w$ of the values ε_{2j} are 1; As $\beta + 1$ and w sum up to $d + 1$, we can easily verify that the conditions of Pólya's Theorem (Fact 10.3) are satisfied: Let us modify (ε_{ij}) by arbitrarily choosing *precisely* $\beta+1$ of the $\varepsilon_{2,j}$ that are 1 and set the others to 0, and call the resulting matrix E. Now, for all $j \in \{0, \dots, d-1\}$, we have $\sum_{\ell=0}^{j} \varepsilon_{1\ell} + \varepsilon_{2\ell} \geq j + 1$ and the first and second row of E sum up to exactly $d + 1$. Hence, the only polynomial of degree at most d that satisfies the corresponding instance of Hermite-Birkhoff interpolation is the zero polynomial. As we obtained E from (ε_{ij}) just by ignoring some vanishing conditions, the same conclusion is true for ε_{ij} and thus $\mathsf{f} = 0$ is the unique solution. This, however, contradicts the fact that the property Φ is not false on all k-vertex graphs, completing the proof. ◿

Combining Lemmas 10.5 and 10.9 yields our main technical result, which we restate here for convenience.

Theorem 10.4. *Write Φ for a computable graph property, k for a positive integer, and w for the Hamming weight[103] of $f^{\Phi,k}$. Suppose that Φ is not false on all k-vertex graphs. Then there is a graph K on k vertices and at least $\binom{k}{2} - w + 1$ edges with $\mathsf{a}_{IndSub(\Phi,k\to\star)}(K) \neq 0$.*

103. That is, the number of nonzero entries.

Proof. Set $d = \binom{k}{2}$ and $\beta = d - w$. By Lemma 10.9 at least one of $h_d^{\Phi,k}, \dots, h_{\beta+1}^{\Phi,k}$ is nonzero and thus, by Lemma 10.5, at least one of the values $\hbar_d, \dots, \hbar_{\beta+1}$ is nonzero as well. Next, observe that $\hbar_i = \sum_{K\in\mathcal{H}_i} \mathsf{a}_{IndSub(\Phi,k\to\star)}(K)$ for all $i \in \{0, \dots, d\}$, where $\mathcal{H}_i$ is the set of all graphs on k vertices and i edges. In particular, $\hbar_i \neq 0$ implies that $\mathsf{a}_{IndSub(\Phi,k\to\star)}(K)$ for at least one $K \in \mathcal{H}_i$. ∎

10.3 A Classification of #IndSub(Φ) by the Hamming Weight of the f-Vectors

In this section, we derive a general hardness result for #IndSub(Φ) based on the Hamming weight of the f-vector. In a sense, we "blackbox" Theorem 10.4; using the resulting classification, we establish first hardness results and almost tight conditional lower bounds for a variety of families of graph properties.

However, note that taking a closer look at the number of edges of the graphs with non-vanishing coefficients (as provided by Theorem 10.4) often yields improved, sometimes even matching conditional lower bounds; we defer the treatment of the refined analysis to Section 10.4.

Write $\mathcal{K}_\Phi$ for the set of all k such that $\mathcal{G}_{\Phi,k}$ is non-empty.

Theorem 10.10. *Let Φ denote a computable graph property and suppose that the set $\mathcal{K}_\Phi$ is infinite.[104] Let $\beta : \mathcal{K}_\Phi \to \mathbb{Z}_{\geq 0}$ denote the function that maps k to $\binom{k}{2} - hw(f^{\Phi,k})$. If $\beta(k) \in \omega(k)$ then #IndSub(Φ) is #**W[1]**-complete. Further, no algorithm can solve #IndSub(Φ) in time $g(k) \cdot |V(G)|^{o((\beta(k)/k)/\log(\beta(k)/k))}$ for any function g, unless ETH fails. The same is true for #IndSub($\overline{\Phi}$) and #IndSub($\neg\Phi$).*

104. Note that this condition is necessary for hardness: Otherwise there is a constant c such that we can output 0 whenever $k \geq c$ and solve the problem by brute-force if $k < c$, yielding an algorithm with a polynomial running time.

The $(log(\beta(k)/k))^{-1}$-term in the exponent is related to the question of whether it is possible to "beat treewidth" [84]. In particular, if the factor of $(log(tw(H)))^{-1}$ in Lemma 8.15 can be dropped, then all further results in this section can be strengthened to yield tight conditional lower bounds under ETH.

Proof. By Theorem 10.4, for each $k \in \mathcal{K}_\Phi$ there is a graph H_k with k vertices and at least $\beta(k)$ edges with $\mathsf{a}_{IndSub(\Phi,k\to\star)}(H_k) \neq 0$, The average degree of H_k satisfies

$$d(H_k) = \frac{1}{k} \cdot \sum_{v\in V(H_k)} deg(v) = \frac{2|E(H_k)|}{k} \geq \frac{2\beta(k)}{k}.$$

By Fact 7.2, we thus obtain that $tw(H_k) \geq \beta(k)/k$, which is unbounded as $\beta(k) \in \omega(k)$ by assumption.

Now, let $\mathcal{H}$ denote the set of all graphs H_k for $k \in \mathcal{K}_\Phi$. By Lemma 8.15, we obtain that #HOM$(\mathcal{H} \to \top)$ is #$W[1]$-complete and cannot be solved in time $g(k) \cdot |V(G)|^{o((\beta(k)/k)/log(\beta(k)/k))}$ for any function g, unless ETH fails. Further, by Lemma 8.24, the same is true for #INDSUB(Φ) as well. Finally, we use Lemma 7.9 to obtain the same result for #INDSUB$(\overline{\Phi})$ and #INDSUB$(\neg\Phi)$. ∎

Low Edge-Densities and Sparse Graph Properties

As a first application of Theorem 10.10, we consider properties Φ that satisfy

$$hw(f^{\Phi,k}) \in o(k^2).$$

We say that such a property Φ has a *low edge-density*. Properties with low edge-density subsume, for example, exclusion of a set of fixed minors such as planarity. They have been studied by Jerrum and Meeks [63], where they show that #INDSUB(Φ) is #$W[1]$-complete for these properties. However, their proof uses Ramsey's Theorem and thus establishes only an implicit conditional lower bound of $g(k) \cdot |V(G)|^{o(log\, k)}$. In contrast, we achieve the following, almost tight lower bound:

Theorem 10.11. *Let Φ denote a computable graph property with low edge-densities. Suppose that the set $\mathcal{K}_\Phi$ is infinite. Then* #IndSub(Φ) *is* #$\mathbf{W[1]}$*-complete and cannot be solved in time $g(k) \cdot |V(G)|^{o(k/log(k))}$ for any function g, unless* ETH *fails. The same is true for the problems* #IndSub$(\overline{\Phi})$ *and* #IndSub$(\neg\Phi)$.

Proof. If Φ has low edge-densities, then we have $\beta(k) = \binom{k}{2} - hw(f^{\Phi,k}) \in \Theta(k^2)$. Thus

$$o\left(\frac{\beta(k)/k}{log(\beta(k)/k)}\right) = o\left(k/log(k)\right).$$

The claim hence follows by Theorem 10.10.

The previous result applies, in particular, to sparse properties. In Section 10.4 we show, that a refined analysis based on Fact 7.1 as well as Theorem 10.4 establishes a tight conditional lower bound for sparse properties Φ that also satisfy a density condition on $\mathcal{K}_\Phi$.

Graph Properties Depending Only on the Number of Edges

Jerrum and Meeks [63, 64] conjectured that #IndSub(Φ) is #$\mathbf{W[1]}$-complete whenever a non-trivial Φ depends only on the number of edges of a graph, that is,

$$\forall H_1, H_2 : |E(H_1)| = |E(H_2)| \Rightarrow \Phi(H_1) = \Phi(H_2).$$

We answer this question affirmatively, even for properties that can depend both on the number of edges and vertices of the graph, and additionally provide an almost tight conditional lower bound:

Theorem 10.12. *Let Φ denote a computable graph property that depends only on the number of edges and the number of vertices of a graph. For trivial Φ, the problem* #IndSub(Φ) *is* FPT. *Otherwise,* #IndSub(Φ) *is* #$\mathbf{W[1]}$*-complete and cannot be solved in time $g(k) \cdot |V(G)|^{o(k/\log k)}$ for any function g, unless* ETH *fails.*[105]

105. Note that Theorem 10.12 is also true for #IndSub$(\overline{\Phi})$ and #IndSub$(\neg\Phi)$, as $\neg\Phi$ and $\overline{\Phi}$ depend only on the number of edges and vertices of a graph if and only if Φ does.

PROOF. First, assume that $\mathcal{G}_{\Phi,k}$ is trivial. Then, there is a constant c such that for every $k > c$, the class Φ, k is either empty or contains all graphs with k vertices. Hence, given as input a graph G and an integer k, we check whether $k \leq c$. If this is the case, we solve the problem by brute-force. Otherwise, we check whether $\mathcal{G}_{\Phi,k}$ is empty.[106] If $\mathcal{G}_{\Phi,k}$ is empty, we output 0; otherwise we output $\binom{n}{k}$. It is immediate that this algorithm yields fixed-parameter tractability.

106. This step is the reason why we get only fixed-parameter tractability and not necessarily polynomial-time tractability.

Now assume that $\mathcal{G}_{\Phi,k}$ is non-trivial. Since for $\mathcal{G}_{\Phi,k}$ we fix the number of vertices to be k, by assumption Φ_k depends only on the number of edges of a graph. Thus, we have

$$hw(f^{\neg\Phi,k}) = \binom{k}{2} - hw(f^{\Phi,k}). \tag{10.2}$$

Hence, set

$$\mathcal{G}_{\Psi,k} := \begin{cases} \mathcal{G}_{\Phi,k} & \text{if } hw(f^{\Phi,k}) \leq \frac{1}{2}\binom{k}{2} \\ \mathcal{G}_{\neg\Phi,k} & \text{otherwise,} \end{cases}$$

and $\Psi := \bigcup \Psi_k$. We observe that, by assumption, $\mathcal{K}_\Psi$ is infinite, and by Lemma 7.9 #INDSUB(Φ) and #INDSUB(Ψ) are equivalent. By definition and by Equation (10.2), we see that $hw(f^{\Psi,k}) \leq \binom{k}{2}/2$ and therefore $\beta(k) = \binom{k}{2} - hw(f^{\Psi,k}) \in \Theta(k^2)$. Thus, we have

$$o\left(\frac{\beta(k)/k}{\log(\beta(k)/k)}\right) = o\left(k/\log k\right).$$

The claim hence follows by Theorem 10.10. ⌟

Monotone Graph Properties

Recall that a property Φ is called monotone if it is closed under taking subgraphs. The decision version of #INDSUB(Φ), that is, deciding whether there is an induced subgraph of size k that satisfies Φ is known to be $W[1]$-complete if Φ is monotone. Further, Φ is non-trivial and $\mathcal{K}_\Phi$ is infinite (this follows implicitly by a result of Khot and Raman [71]). However, as the reduction of Khot and Raman is *not* parsimonious, the reduction does not yield $\#\mathbf{W}[1]$-completeness of the counting version. More importantly, the proof of Khot and Raman uses Ramsey's Theorem and thus implies only a conditional lower bound of $g(k) \cdot |V(G)|^{o(\log k)}$. Using our main result, we achieve a much stronger and almost tight lower bound under ETH.

THEOREM 10.13. *Write Φ for a computable, monotone, and non-trivial graph property and suppose that $\mathcal{K}_\Phi$ is infinite. Then, $\#\textsc{IndSub}(\Phi)$ is $\#\mathbf{W}[1]$-complete and cannot be solved in time $g(k) \cdot |V(G)|^{o(k/\log k)}$ for any function g, unless ETH fails. The same is true for the problems $\#\textsc{IndSub}(\overline{\Phi})$ and $\#\textsc{IndSub}(\neg\Phi)$.*

PROOF. As Φ is non-trivial, there is a graph F such that $\Phi(H)$ is false for every H that contains F as a (not necessarily induced) subgraph. Set $r = |V(F)|$ and fix $k \in \mathcal{K}(\Phi)$. By Fact 7.1 we have

that every graph H on k vertices with more than $(1 - \frac{1}{r}) \cdot \frac{k^2}{2}$ edges contains the clique K_{r+1} and thus F as a subgraph. Hence, Φ is false on every graph with k vertices and more than $(1 - 1/r) \cdot k^2/2$ edges. Therefore, we have

$$\beta(k) = \binom{k}{2} - hw(f^{\Phi,k}) \geq \binom{k}{2} - \left(1 - \frac{1}{r}\right) \cdot \frac{k^2}{2} = \frac{k^2}{2r} - \frac{k}{2} \in \Omega(k^2).$$

Thus, $\beta(k) \in \Theta(k^2)$ and we conclude that

$$o\left(\frac{\beta(k)/k}{\log(\beta(k)/k)}\right) = o\left(k/\log k\right).$$

The claim hence follows by Theorem 10.10.

10.4 REFINED LOWER BOUNDS AND CLIQUE-MINORS

Recall that the lower bounds of the previous section become tight if it is impossible to "beat treewidth", that is, if the $(\log(k))^{-1}$ factor in the exponent of Lemma 8.15 can be dropped. In this section, we show that the lower bounds of the previous section can also be refined—and in case of sparse properties even be made tight—without the assuming that you cannot "beat treewidth". We obtain these results by using Turán's Theorem (Fact 7.1) and the Kostochka-Thomason-Theorem (Fact 7.5).

Intuitively, we combine Fact 7.5 with Theorem 10.4 to obtain graphs with large clique-minors in the graph motif parameter associated with a graph property Φ as given by Equation (8.3). As graphs with large clique-minors have large treewidth, we then obtain (almost) tight bounds as a consequence of Fact 8.22.

We say that a subset $\mathcal{K}$ of the natural numbers is *dense* if there is a constant $\ell \geq 1$ such that for all $n \in \mathbb{N} \setminus \{0\}$ there is a $k \in \mathcal{K}$ that satisfies $n \leq k \leq \ell n$. Now recall that $\mathcal{K}_\Phi$ is the set of all k such that $\mathcal{G}_{\Phi,k}$ is not empty. The lower bounds for #IndSub(Φ) in this section require the set $\mathcal{K}_\Phi$ to be dense. We need this technicality to exclude certain artificial properties.[107] Fortunately, monotone properties are natural in the sense that they are always dense.

107. Consider for instance $\Phi(H) = 1$ if and only if H is an independent set and $|V(H)| = 2 \uparrow m$ for some $m \in \mathbb{N}$, where $2 \uparrow m$ is the m-fold exponential tower of 2. While this property satisfies the conditions of Theorem 10.10, we cannot construct a *tight* reduction to #IndSub(Φ) as the only non-trivial oracle queries satisfy $k = 2 \uparrow m$ for some $m \in \mathbb{N}$.

Lemma 10.14. *Let Φ denote a non-trivial monotone graph property. If $\mathcal{K}_\Phi$ is infinite, $\mathcal{K}_\Phi$ is the set of all positive integers and thus dense.*

Proof. Fix a $n \in \mathbb{N} \setminus \{0\}$. As $\mathcal{K}_\Phi$ is infinite, there is a $k \geq n$ in $\mathcal{K}_\Phi$. Thus there is a graph $H \in \mathcal{G}_{\Phi,k}$. Now delete $k - n$ arbitrary vertices of H and call the resulting graph R. As Φ is monotone, we have that $R \in \mathcal{G}_{\Phi,n}$ and hence $n \in \mathcal{K}_\Phi$ as well.

The hardness result is the basis for the lower bounds in this section. We reduce directly from Clique, hardness then follows from Fact 8.11.[108]

108. If you happen to be a believer in the Clique conjectures of *Fine-Grained Complexity Theory*, the reduction also yields tightness up

Lemma 10.15. *Let $r \geq 1$ denote a constant, let $\mathcal{H}$ denote a recursively enumerable graph class. Call a $k \in \mathbb{N} \setminus \{0\}$ good if there is a k-vertex graph $H \in \mathcal{H}$ with at least $k^2/2r - k/2$ edges and write $\mathcal{K}_\mathcal{H}$ for the set of all good k. If $\mathcal{K}_\mathcal{H}$ is dense, no algorithm can solve #Hom($\mathcal{H} \to \top$) in time $f(|V(H)|) \cdot |V(G)|^{o(|V(H)|/\log^{1/2}(|V(H)|))}$ for any f, unless ETH fails.*

Proof. As $\mathcal{K}_\mathcal{H}$ is dense, there is a constant $\ell \geq 1$ such that for all positive integers m, there is a $k \in \mathcal{K}_\mathcal{H}$ with $m \leq k \leq \ell m$.

Toward a reduction from Clique to $\#Hom(\mathcal{H} \to \top)$, let (G, m) denote an instance of Clique. Note that m is not necessarily contained in $\mathcal{K}_{\mathcal{H}}$, so in a first step, we compute a good integer $m \leq k \leq \ell m$ and a corresponding k-vertex graph $H_k \in \mathcal{H}$ with at least $k^2/2r - k/2$ edges. Using Fact 7.5 on H_k, we obtain that H_k has K_t as a minor for a $t = \Theta(k/log^{1/2}(k))$. Now, we transform G into a graph F_t such that F_t has K_t as a subgraph if and only if G has K_m as a subgraph. Using Lemma 8.21, we then transform F_t into a graph G_t such that $\#Hom(K_t \to F_t) = \#Hom(H_k \to G_t)$.[109] Finally, using the oracle for $\#Hom(\mathcal{H} \to \top)$, we check if $\#Hom(K_t \to F_t)$—the number of t cliques in F_t and thus the number of m cliques of the original graph G—is positive.

109. This is a simplification: Recall that the reduction in fact produces multiple graphs $G_t|_W$ and that the number $\#Hom(K_t \to F_t)$ is then recovered from a linear combination of homomorphism numbers of the constructed graphs.

To fill the above outline with technical details, assume that there is an algorithm $\mathbb{A}$ that solves $\#\text{Hom}(\Phi \to \top)$ in time $f(|V(H)|) \cdot |V(G)|^{o(|V(H)|/log^{1/2}|V(H)|)}$ for some function f. We use $\mathbb{A}$ to solve Clique in time $p(m) \cdot |V(G)|^{o(m)}$ for some function p.

Write (G, m) for the given instance of #Clique. First, we pad G to a graph G_k to transform (G, m) into a #Clique instance (G_k, k), where $k \in \mathcal{K}_{\mathcal{H}}$ and $m \leq k \leq \ell m$.[110] To that end, add $k - m$ times a new vertex to G and connect it to all other vertices in G (including previously created new vertices); call the resulting graph G_k. Observe that $\#Sub(K_m \to G) = \#Sub(K_k \to G_k)$ and that this step takes $O(n^{m-k})$ time, but also increases the parameter accordingly.

110. We can find such a k as $\mathcal{K}(\mathcal{H})$ is recursively enumerable.

Next, let $c \approx 2.68$ denote the constant from Fact 7.5 and set $r^{++} := r + 1$. Further, set

$$h(k) := cr^{++} \cdot log^{1/2}(k/cr^{++})$$

and $t := h/h(k)$. Next, we transform the instance (G_k, k) into an equivalent instance (F_t, t). To that end, set

$$V(F_t) := \{U \mid G_k[U] = K_{h(k)}\},$$
$$E(F_t) := \{\{U, W\} \mid G_k[U \cup W] = K_{2h(k)}\}.$$

Observe that F_t has $O(n^{h(k)})$ vertices and can be constructed in time $g(m) \cdot O(|V(G)|^{h(k)})$ for some computable function g. Further, observe that $\#Sub(K_t \to F_t) = \#Sub(K_k \to G_k)$.[111]

Next, we search for a graph $H_k \in \mathcal{H}$ with k vertices and at least $k^2/2r - k/2$ edges. As $\mathcal{H}$ is recursively enumerable, we can find H_k in time $b(k)$ for a computable function b. Now we see that

$$d(H) = \frac{1}{k} \cdot \sum_{v \in V(H)} deg(v) = \frac{2|E(H)|}{k} \geq \frac{k}{r} - 1.$$

For k large enough,[112] we have

$$\begin{aligned} d(H) &\geq \frac{k}{r} - 1 \geq \frac{k}{r^{++}} = c \cdot \frac{k}{cr^{++} \cdot \sqrt{log(k/cr^{++})}} \cdot \sqrt{log(k/cr^{++})} \\ &\geq c \cdot \frac{k}{cr^{++} \cdot \sqrt{log(k/cr^{++})}} \cdot \sqrt{log(k/cr^{++}) - log\sqrt{log(k/cr^{++})}} \\ &= ct\sqrt{log(t)}. \end{aligned}$$

Fact 7.5 thus implies that K_t is a minor of H. Using the reduction from Lemma 8.21, we can use the algorithm $\mathbb{A}$ to compute the number of homomorphisms from K_t to F_t. By assumption on $\mathbb{A}$, this takes time at most

$$\begin{aligned} &f(|V(H)|) \cdot |V(F_t)|^{o(|V(H)|/log^{1/2}|V(H)|)} \\ &\quad = f(k) \cdot (n^{h(k)})^{o(k/log^{1/2}(k))} = f(k) \cdot n^{o(k)}, \end{aligned}$$

where the last equation holds as $h(k) \in \Theta(log^{1/2}(k))$—recall that r^{++} and c are constants.

Now, it is easy to see that F_k contains a t-clique—and hence G an m-clique—if and only if the number of homomorphisms from

111. Formally, we have to round $h(k)$ and later $t = k/h(k)$. For the sake of readability, we assume that all logs and fractions yield integers. Note that this might require us to count $\lceil t \rceil$-cliques at the end of the proof, while the oracle can determine only the existence of a $\lfloor t \rfloor$-clique. Nevertheless, we can easily decide whether there is a $\lceil t \rceil$-clique by checking for each vertex whether its neighborhood contains a $\lfloor t$-clique.

112. If k is not large enough for the inequalities to hold, then k and thus m are bounded by a constant, and we can compute the number of m-cliques in G by brute-force.

K_t to F_k is at least 1. As $k \in \Theta(m)$ (recall that ℓ is a constant), the overall running time is bounded by

$$g(m) \cdot O(|V(G)|^{h(k)}) + b(k) + f(k) \cdot |V(G)|^{o(k)} \leq p(m) \cdot |V(G)|^{o(m)}$$

for $p(m) := g(m) + b(\ell m) + f(\ell m)$. This yields the desired contradiction and concludes the proof. ◪

Next, we conclude the refined lower bounds for #IndSub(Φ).

Theorem 10.16. *Let Φ denote a non-trivial, semidecidable and monotone graph property such that $\mathcal{K}_\Phi$ is infinite. No algorithm can solve #IndSub(Φ) in time $g(k) \cdot |V(G)|^{o(k/\log^{1/2}(k))}$ for any function g, unless ETH fails. The same is true for #IndSub($\overline{\Phi}$) and #IndSub($\neg\Phi$).*

Proof. We begin just as in the proof of Theorem 10.13: As Φ is non-trivial, there is a graph F such that $\Phi(H)$ is false for every H that contains F as a (not necessarily induced) subgraph. Set $r := |V(F)|$ and fix $k \in \mathcal{K}_\Phi$. By Fact 7.1 we have that every graph H on k vertices with more than $(1 - 1/r) \cdot k^2/2$ edges contains the clique K_{r+1} and thus F as a subgraph. Hence, Φ is false on every graph with k vertices and more than $(1 - 1/r) \cdot k^2/2$ edges. We now use Theorem 10.4 and obtain a graph H_k on k vertices and at least

$$\binom{k}{2} - hw(f^{\Phi,k}) + 1 \geq \binom{k}{2} - \left(1 - \frac{1}{r}\right) \cdot \frac{k^2}{2} + 1 \geq \frac{k^2}{2r} - \frac{k}{2}$$

edges such that $\mathsf{a}_{IndSub(\Phi,k\to\star)}(H_k) \neq 0$. Hence, Lemma 8.24 yields a tight reduction from the problem #Hom($\mathcal{H} \to \top$) where $\mathcal{H} := \{H_k \mid k \in \mathcal{K}_\Phi\}$. By the previous observation, the graph H_k has k vertices and at least $k^2/2r - k/2$ many edges, and by Lemma 10.14 the set of k such that $H_k \in \mathcal{H}$ is dense. Thus we can use Lemma 10.15, which concludes the proof—note that the results for #IndSub($\overline{\Phi}$) and #IndSub($\neg\Phi$) follow by Lemma 7.9. ◪

We continue with the refined lower bound for properties that depend only on the number of edges of a graph; in this case, we have to assume density.

THEOREM 10.17. *Let Φ denote a computable graph property that depends only on the number of edges of a graph. If the set of k for which Φ_k is non-trivial is dense, then $\#\textsc{IndSub}(\Phi)$ cannot be solved in time $g(k) \cdot |V(G)|^{o(k/\log^{1/2}(k))}$ for any function g, unless ETH fails.*

PROOF. We use the same set-up as in the proof of Theorem 10.12. In particular, we obtain Ψ such that $\#\textsc{IndSub}(\Phi)$ and $\#\textsc{IndSub}(\Psi)$ are equivalent, $\mathcal{K}_\Psi$ is dense, and $hw(f^{\Psi,k}) \leq \binom{k}{2}/2$ for all $k \in \mathcal{H}_\Psi$). We use Theorem 10.4 to obtain a graph H_k on k vertices and at least

$$\binom{k}{2} - hw(f^{\Phi,k}) + 1 \geq \frac{k^2}{4} - \frac{k}{2}$$

edges such that $\mathsf{a}_{IndSub(\Phi,k\to\star)}(H_k) \neq 0$. Combining Lemmas 10.15 and 8.24 (with $r = 2$) yields the claim. ∎

As a final result in this section, we establish a tight conditional lower bound for sparse properties; recall that we call a property Φ *sparse* if there is a constant s depending only on Φ such that Φ is false on every graph with k vertices and more than sk edges. Instead of relying in the Kostochka-Thomason-Theorem, it suffices to use Turán's Theorem, however.

THEOREM 10.18. *Let Φ denote a computable sparse graph property such that $\mathcal{K}(\Phi)$ is dense. Then $\#\textsc{IndSub}(\Phi)$ cannot be solved in time $g(k) \cdot |V(G)|^{o(k)}$ for any function g, unless ETH fails. The same is true for $\#\textsc{IndSub}(\overline{\Phi})$ and $\#\textsc{IndSub}(\neg\Phi)$.*

PROOF. Let s denote the constant from the definition of sparsity, and fix $k \in \mathcal{K}_\Phi$. Using Theorem 10.4, we obtain a k-vertex graph H_k

with at least $\binom{k}{2} - hw(f^{\Phi,k}) + 1$ edges and $\mathsf{a}_{IndSub(\,\Phi,k\to\star\,)}(\,H_k\,) \neq 0$. Set $r := \lceil k/(2(s+1)+1)\rceil$, and observe that

$$\binom{k}{2} - hw(f^{\Phi,k}) + 1 > \binom{k}{2} - sk > \binom{k}{2} - (s+1)k \geq \left(1 - \frac{1}{r}\right)\cdot\frac{1}{2}|V(H_k)|^2.$$

By Turán's Theorem (Fact 7.1), H_k hence contains K_{r+1} as a subgraph and, in particular, K_r as a minor. By Lemma 8.24 we can, given a graph G compute $\#Hom(\,H_k \to G\,)$ in linear time if we are given oracle access to $\#IndSub(\,\Phi, k \to \star\,)$. Adding Lemma 8.21, we can compute $\#Hom(\,K_r \to G\,)$ in linear time if we are given oracle access to $\#IndSub(\,\Phi, k \to \star\,)$. Note further that $\#Hom(\,K_r \to G\,)$ is at least 1 if and only if G contains a clique of size r.

We continue similarly to the proof of Lemma 10.15: Assume that there is an algorithm $\mathbb{A}$ that solves #IndSub(Φ) in time $g(k) \cdot |V(G)|^{o(k)}$ for some function g. We show that $\mathbb{A}$ can be used to solve the problem of *finding* a clique of size m in a graph G in time $p(m) \cdot |V(G)|^{o(m)}$ for some function p. Given m and G, search for the smallest $k \in \mathcal{K}_\Phi$ with $m \leq r$,. Observe that we can find such a k in time depending only on m as Φ is semidecidable. Further, see that $k \in O(m)$ as s is a constant and $\mathcal{K}_\Phi$ is dense.

As in Lemma 10.15, we pad G to a graph G_r by adding r times a new vertex to G and connecting it to all other (possibly new) vertices of G. Again, it is easy to see that G has a clique of size m if and only if G_r has a clique of size r. By the analysis and assumptions above, we can decide whether G_r indeed has an r-clique by using $\mathbb{A}$ in time $g(k)\cdot|V(G_r)|^{o(k)}$. As $|V(G_r)| \in O(|V(G)|)$ and $k \in O(m)$, the overall time to decide whether G has a clique of size m is hence bounded by $p(m)\cdot|V(G)|^{o(m)}$ for some function p, which is impossible, unless ETH fails (Fact 8.11). The results for #IndSub($\overline{\Phi}$) and #IndSub($\neg\Phi$) follow by Lemma 7.9. ◿

11 Counting and Finding Homomorphisms is Universal

In this chapter, we show that every parameterized counting problem $P \in \#\boldsymbol{W}[1]$ is interreducible with a problem $\#\textsc{Hom}(\mathcal{H}_P \to \mathcal{G}_P)$ under parameterized Turing-reductions. Further, the proof shows that the corresponding statement holds for (parameterized) decision problems in $\boldsymbol{W}[1]$ and a problem $\textsc{Hom}(\mathcal{H} \to \mathcal{G})$. Our starting point is Lemma 8.19—Indeed, in this chapter we finally see the main reason, why we exported more technical details from the hardness proof of $\#\textsc{Hom}(\mathcal{H} \to \top)$.

The main result of this chapter reads as follows.

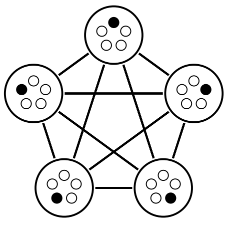
Theorem 11.1. *For any problem (P, κ) in $\#\boldsymbol{W}[1]$, there are graph classes $\mathcal{H}_P$ and $\mathcal{G}_P$ such that*

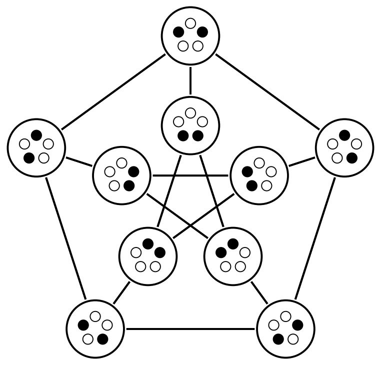
$$(P, \kappa) \equiv_T^{fpt} \#\textsc{Hom}(\mathcal{H}_P \to \mathcal{G}_P).$$

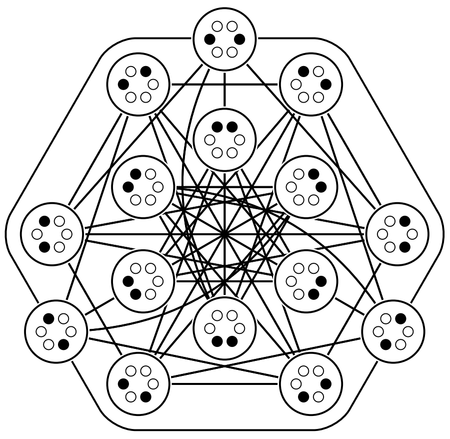
Further, $\mathcal{H}_P$ is recursively enumerable and $\mathcal{G}_P$ is recursive.

In particular, we show that for any problem (P, κ) in $\#\boldsymbol{W}[1]$, we can construct graph classes $\mathcal{H}_P$ and $\mathcal{G}_P$ such that we have for any graphs $H \in \mathcal{H}_P$ and $G \in \mathcal{G}_P$:

1. If $\#Hom(H \to G) \neq 0$, the pair (H, G) corresponds to exactly one instance x of the problem (P, κ), and we can obtain both x and $\kappa(x)$ from H and G.
2. If $\#Hom(H \to G) = 0$, the pair (H, G) does not correspond to an instance of (P, κ).


(a) $K(5,1) = K_5$.


(b) $K(5,2)$, or *Petersen graph.*


(c) $K(6,2)$.

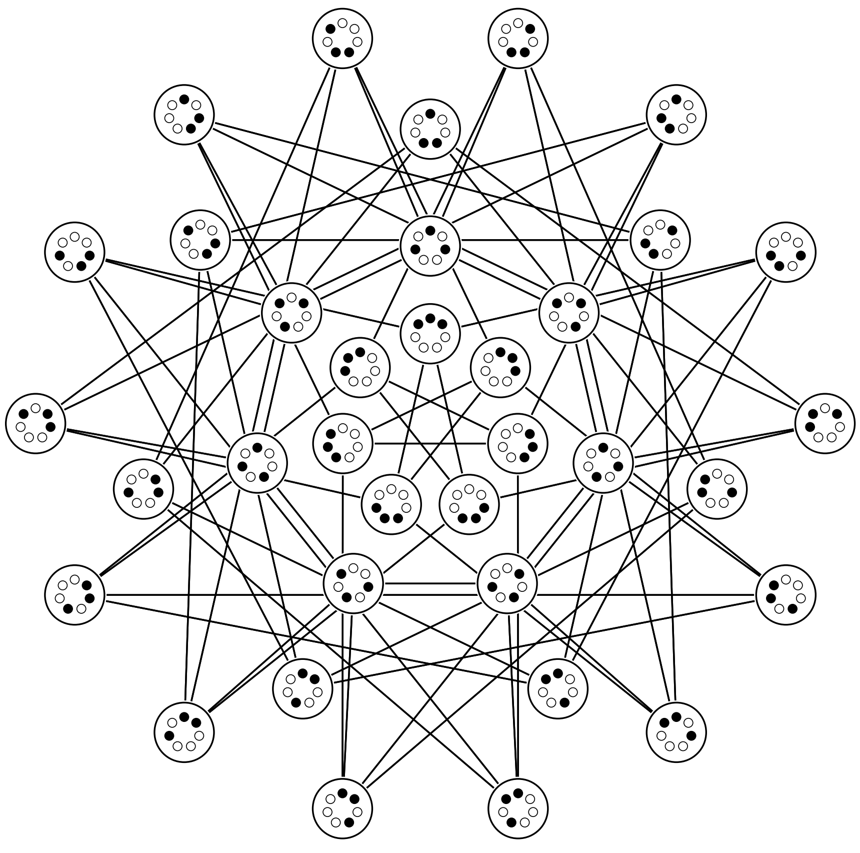

(d) $K(7,3) = K(3)$.

FIGURE 11.1. Examples for Kneser graphs. The set corresponding to each vertex is depicted inside of each vertex—a black dot denotes the presence of an element in a set, while a white dot denotes the absence of an element in a set.

11.1 Counting Homomorphisms Between Kneser Graphs

By Lemma 8.19, the problem $\#\textsc{Hom}(\mathcal{H} \to \top)$ is $\#W[1]$-hard for *any* (computable) class of connected cores. In particular, it is known that this is the case for (subclasses) of the class of *Kneser graphs*. Further, Kneser graphs have other nice properties which we exploit in the proof of Theorem 11.1. Formally, we start with the following definition; consider also Figure 11.1 for examples of Kneser graphs.

☞ Definition 11.2 ([see 57, Chapter 3]). *For $r, s \in \mathbb{N} \cup \{0\}$, the* Kneser graph $\mathrm{K}(r,s)$ *is the graph with*

$$V(\mathrm{K}(r,s)) := \binom{[\,r\,]}{s}$$
$$E(\mathrm{K}(r,s)) := \{\{U, W\} \mid U \cap W = \emptyset\},$$

that is, the vertices of $\mathrm{K}(r,s)$ are the subsets of size s of $[\,r\,]$ and $\mathrm{K}(r,s)$ has edges exactly between vertices that correspond to disjoint sets. ⌟

For an integer $n \geq 3$, we set $\mathrm{K}(n) := \mathrm{K}((2n+1)(n-2), n(n-2))$. With this choice of r and s, we can use the following results;

☞ Fact 11.3 ([82] and Propositions 3.13, 3.14 in [57]). *For any integer $n \geq 3$, all of the following hold:*

1 $\mathrm{K}(n)$ *has chromatic number n.*
2 $\mathrm{K}(n)$ *has odd girth $2n + 1$.*
3 $\mathrm{K}(n)$ *is a core; $\#Hom(\mathrm{K}(n) \to \mathrm{K}(n)) = \#Aut(\mathrm{K}(n))$.* ⌟

☞ Fact 11.4 (Folklore, [see 116]). *For non-negative integers r, s, the graph $\mathrm{K}(r,s)$ is connected if and only if $r > 2s$.*
Hence, $\mathrm{K}(n)$ is connected. ⌟

It is of crucial importance to us that Kneser graphs form an antichain with respect to the homomorphism order.

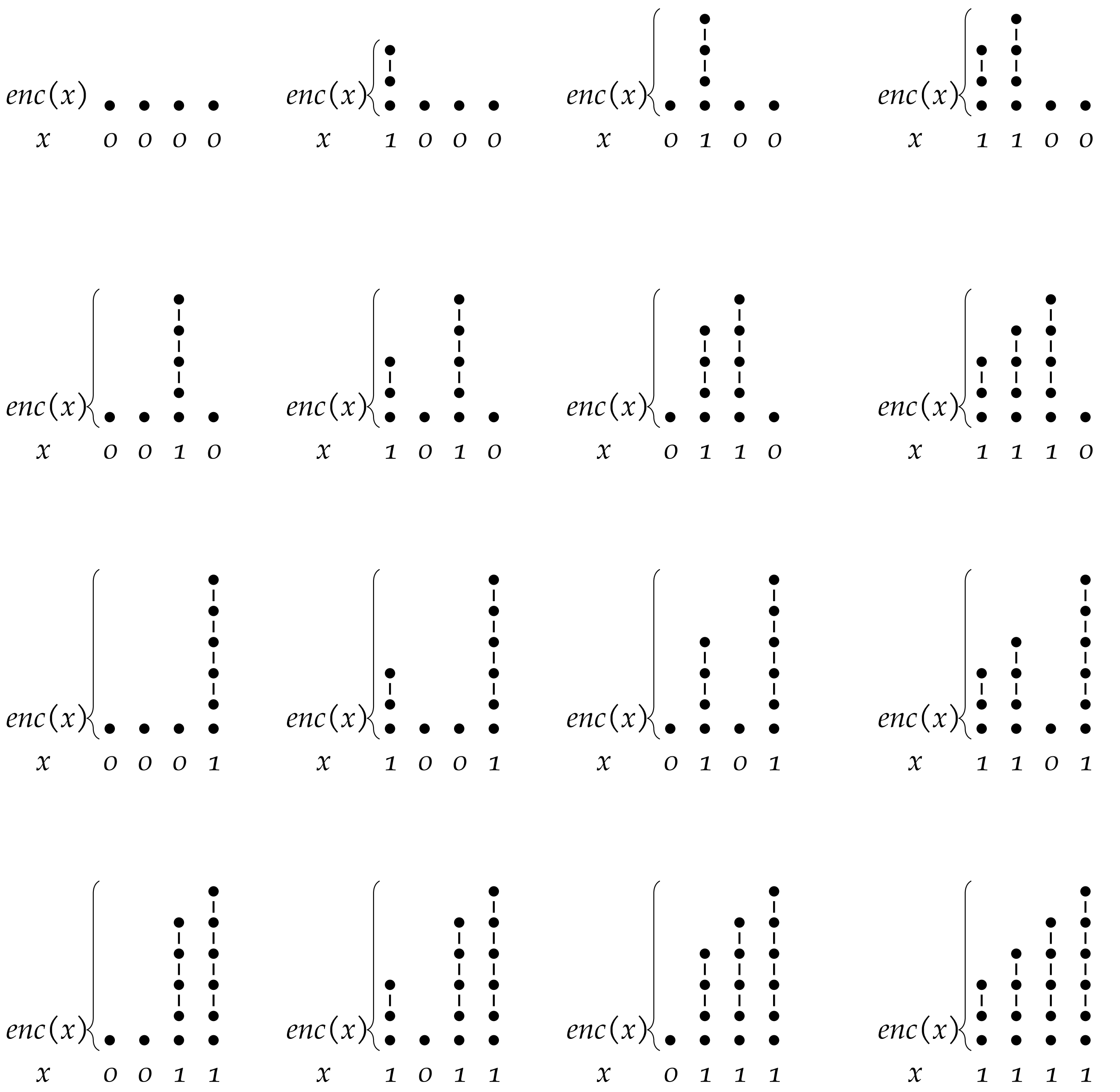


FIGURE 11.2. Examples of strings and their encoding into a graph.

LEMMA 11.5. *For distinct integers $n, m \geq 3$, we have*

$$\#Hom(\mathrm{K}(m) \to \mathrm{K}(n)) = 0.$$

PROOF. Follows from Lemmas 7.3 and 7.4 and Fact 11.3.

Now, let $\mathcal{K}_{even}$ denote the set of all graphs $\mathrm{K}(n)$ with even n and let $\mathcal{K}_{odd}$ denote the set of all graphs $\mathrm{K}(n)$ with odd n.

11.2 Encoding Problems into Graphs Classes

A central tool for the proof of Theorem 11.1 is an encoding of arbitrary strings into graphs, which we discuss next. In particular, we use a disjoint union of paths for the encoding. Given a string $x = x[1]x[2]\cdots x[n] \in \{0,1\}^*$, we write $enc(x)$ for the graph that is the disjoint union of paths P_{i+1} for all $i \leq |x|$ with $x_i = 1$,[113] as well as of $|x|$ isolated vertices. Consider Figure 11.2 for a visualization.

113. Recall that P_i is the path with i vertices and $i-1$ edges.

Next, we show how to use the encoding *enc* to (reversibly) encode an instance of an arbitrary problem in $\#\boldsymbol{W}[1]$ into a pair of graphs. To that end, let (P, κ) denote any problem in $\#\boldsymbol{W}[1]$. By Lemma 8.19, we have $(P,\kappa) \leq^{w\text{-}fpt} \#\mathrm{HOM}(\mathcal{K}_{even} \to \top)$, that is, there is an algorithm $\mathbb{A} = \mathbb{A}_P$ and a triple (f, g, s) of computable functions such that for all $x \in \{0,1\}^*$ all of the following hold:

1 The algorithm $\mathbb{A}$ computes a pair of graphs $\mathbb{A}(x) = (H_x, G_x)$, where $H_x \in \mathcal{K}_{even}$ and G_x is connected and H_x-colorable.
2 The answer to the instance x of the problem (P,κ) can be computed as $P(x) = g(x) \cdot \#Hom(H_x \to G_x)$.
3 The problem (g, κ) is FPT in time $p(\kappa(x)) \cdot |x|^{O(1)}$.
4 The algorithm $\mathbb{A}$ runs in time $f(\kappa(x)) \cdot |x|^{O(1)}$.
5 The size of the computed graph $|V(H_x)|$ is at most $s(\kappa(x))$.

Now, fix an instance x to (P,κ). Using the graphs $\mathbb{A}(x) = (H_x, G_x)$ computed by the algorithm $\mathbb{A}$, we construct a pair of new

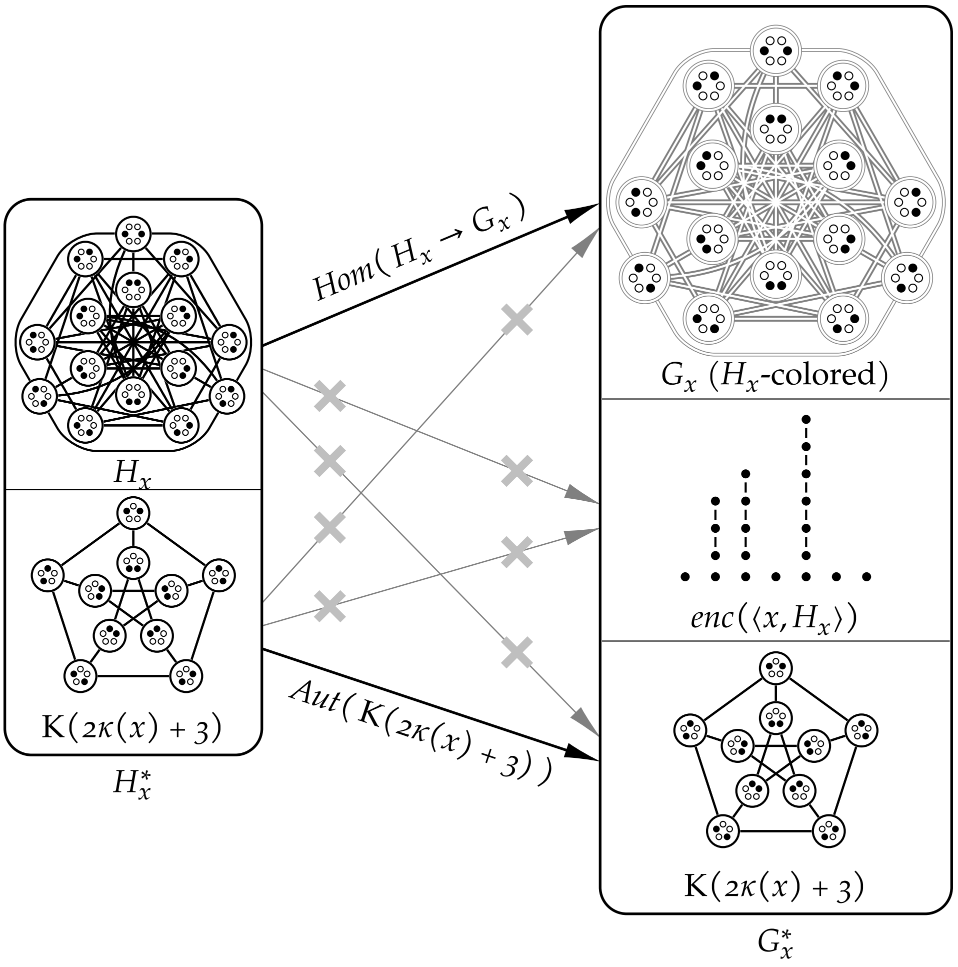


FIGURE 11.3. Lemma 11.6 illustrated. A cross denotes that no homomorphisms between the parts of the graphs exist. Note that the Kneser graphs used in the lemma differ from the ones depicted.

graphs, that additionally encode the original instance x as well as its parameter $\kappa(x)$ by setting

$$H_x^* := H_x \cup \mathrm{K}(2\kappa(x) + 3), \text{ and} \tag{11.1}$$
$$G_x^* := G_x \cup enc(\langle x, H_x \rangle) \cup \mathrm{K}(2\kappa(x) + 3), \tag{11.2}$$

where $\langle x, H_x \rangle$ is any efficient encoding of the pair (x, H_x). We proceed to show that the constructed graphs H_x^* and G_x^* behave as intended; also consider Figure 11.3 for a visualization.

LEMMA 11.6. *For any $(P, \kappa) \in \#\mathbf{W}[1]$ and any instance x to (P, κ), let the graphs H_x^* and G_x^* be defined as in Equations (11.1) and (11.2). Then,*

1. *The graph G_x^* is H_x^*-colored and*
2. *$\#Hom(H_x^* \to G_x^*) = \#Hom(H_x \to G_x) \cdot \#Aut(\mathrm{K}(2\kappa(x) + 3))$.*

PROOF. By definition, we have $H_x^* := H_x \cup \mathrm{K}(2\kappa(x) + 3)$ and $G_x^* := G_x \cup enc(\langle x, H_x \rangle) \cup \mathrm{K}(2\kappa(x) + 3)$, where $enc(\star)$ encodes a string into a disjoint set of paths and isolated vertices.

Now, for the homomorphism from G_x^* to H_x^*, note that the graph G_x is H_x-colored (by Lemma 8.19 and in particular Item 1). Further, the graph $\mathrm{K}(2\kappa(x) + 3)$ has an automorphism and contains at least one edge, so there is a homomorphism from the graph $enc(\langle x, H_x \rangle) \cup \mathrm{K}(2\kappa(x) + 3)$ into the graph $\mathrm{K}(2\kappa(x) + 3)$. In total, this completes the proof that G_x^* is H_x^*-colored.

For the number of homomorphisms from H_x^* to G_x^*, note that there are $\#Hom(H_x \to G_x)$ many homomorphisms from H_x to G_x and $\#Aut(\mathrm{K}(2\kappa(x)+3))$ many homomorphisms from $\mathrm{K}(2\kappa(x)+3)$ to itself. As the graphs H_x^* and G_x^* consist of the disjoint union of H_x and $\mathrm{K}(2\kappa(x) + 3)$, and G_x and $\mathrm{K}(2\kappa(x) + 3)$, respectively, we directly obtain a lower bound for the number of homomorphisms:

$$\#Hom(H_x^* \to G_x^*) \geq \#Hom(H_x \to G_x) \cdot \#Aut(\mathrm{K}(2\kappa(x)+3)). \tag{11.3}$$

To prove the upper bound, observe the following.

1 The graph H_x cannot be mapped homomorphically to the graph $\mathrm{K}(2\kappa(x)+3)$, as H_x is contained in $\mathcal{K}_{even}$ and $\mathrm{K}(2\kappa(x)+3)$ is contained in $\mathcal{K}_{odd}$; hence both are distinct Kneser graphs and by Lemma 11.5 no homomorphisms between them are possible.

2 The graph $\mathrm{K}(2\kappa(x)+3)$ cannot be mapped homomorphically to the graph G_x. Suppose otherwise, that an homomorphism $h : \mathrm{K}(2\kappa(x)+3) \to G_x$ existed. As G_x is H_x-colorable, there is a homomorphism $c : G_x \to H_x$. Composing the homomorphisms h and c yields a homomorphism $h \circ c : \mathrm{K}(2\kappa(x)+3) \to H_x$, that is, a homomorphism from a graph in $\mathcal{K}_{odd}$ to a graph in $\mathcal{K}_{even}$, which, again, is not possible by Lemma 11.5.

3 None of the graphs H_x and $\mathrm{K}(2\kappa(x)+3)$ can be mapped homomorphically to the graph $enc(\langle x, H_x\rangle)$, as paths have a chromatic number of at most 2, and both graphs H_x and $\mathrm{K}(2\kappa(x)+3)$ have a chromatic number of at least 3.

Hence, the homomorphisms counted in the lower bound (11.3) are already *all* homomorphisms from H_x^* to G_x^*:

$$\#Hom(\, H_x^* \to G_x^* \,) = \#Hom(\, H_x \to G_x \,) \cdot \#Aut(\, \mathrm{K}(2\kappa(x)+3) \,).$$

This completes the proof. ◻

Now write $\mathcal{H}_P$ and $\mathcal{G}_P$ for the graph class of all graphs H_x^* and G_x^*, respectively, that correspond to instances x to (P,κ) for which the function g is non-zero. For the classes $\mathcal{H}_P$ and $\mathcal{G}_P$ to be useful to us, we need to show that the only pairs of graphs $H \in \mathcal{H}_P$ and $G \in \mathcal{G}_P$ that admit a homomorphism from H to G are those, that correspond to the same pair $(x, \kappa(x))$. Formally, consider the following lemma.

LEMMA 11.7. *Let (P,κ) denote a problem in $\#\mathbf{W}[1]$ with the corresponding graph classes $\mathcal{H}_P$ and $\mathcal{G}_P$. For any graphs $H \in \mathcal{H}_P$ and $G \in \mathcal{G}_P$, if there is a homomorphism from H to G, then there is an instance x to (P,κ) that corresponds to both H and G, that is, $H = H_x^*$ and $G = G_x^*$.*

Proof. It suffices to show that there are no homomorphisms from H to G if the graphs H and G *do not* correspond to the same instance of (P, κ). Hence, assume that the graphs $H_x^* \in \mathcal{H}_P$ and $G_y^* \in \mathcal{G}_P$ correspond to distinct instances x and y of (P, κ). For the sake of contradiction, further assume that there is a homomorphism h from H_x^* to G_y^*. By Equations (11.1) and (11.2), for an even integer a and odd integers b and c, we have

$$\begin{aligned} H_x^* &= H_x \cup \mathrm{K}(2\kappa(x) + 3) \\ &=: \mathrm{K}(a) \cup \mathrm{K}(b) \text{ and} \\ G_y^* &= G_y \cup enc(\langle y, H_y \rangle) \cup \mathrm{K}(2\kappa(y) + 3) \\ &=: G_y \cup enc(\langle y, H_y \rangle) \cup \mathrm{K}(c), \end{aligned}$$

where G_y is a connected graph that is H_y-colored.

Similar to Item 3 of the proof of Lemma 11.6 we can show that $\#Hom(\,\mathrm{K}(a) \to enc(\langle x, H_x \rangle)\,) = \#Hom(\,\mathrm{K}(b) \to enc(\langle x, H_x \rangle)\,) = 0$.

Now, first assume $b = c$. In this case, we have $\mathrm{K}(b) = \mathrm{K}(c)$, and hence $\kappa(x) = \kappa(y)$. Further, by Lemma 11.5, the h maps $\mathrm{K}(a)$ to G_y, as $\mathrm{K}(a)$ and $\mathrm{K}(c)$ are different Kneser graphs. Combining this homomorphism from $\mathrm{K}(a)$ to G_y with the homomorphism from G_y to H_y (which exists as G_1 is H_y-colorable) yields a homomorphism from $\mathrm{K}(a)$ to H_y. However, as $\mathrm{K}(a) = H_x$ and H_y are both Kneser graphs, a homomorphism between them is possible only if they are the same Kneser graph. This in turn, means that the instances x and y are the same, which is a contradiction.

Second, consider the case where the numbers a, b, and c are pairwise distinct. By Lemma 11.5, we obtain $\#Hom(\,\mathrm{K}(a) \to \mathrm{K}(c)\,) = 0$ and $\#Hom(\,\mathrm{K}(b) \to \mathrm{K}(c)\,) = 0$. Hence, the homomorphism h maps both graphs $\mathrm{K}(a)$ and $\mathrm{K}(b)$ to the graph G_y. Now, as before, we obtain a homomorphism from the graph H_x to the graph H_y and hence (by Lemma 11.5) $H_x = \mathrm{K}(a) = H_y = \mathrm{K}(b)$, which is a contradiction as $a \neq b$.

In total, if H and G do not correspond to the same instance x, there is no homomorphism from H to G. ◻

11.3 The Main Reductions

Fix a problem $(P, \kappa) \in \#\mathbf{W}[1]$ and the corresponding graph classes $\mathcal{H}_P$ and $\mathcal{G}_P$. Using Lemmas 11.6 and 11.7, we proceed to show that the problems (P, κ) and $\#\text{Hom}(\mathcal{H}_P \to \mathcal{G}_P)$ are interreducible with respect to parameterized Turing reductions.[114] We start with the more involved direction.

114. Note that the reductions are *almost* weakly parsimonious—The main problem lies in obtaining a *trivial* instance of the problem P. In Lemma 11.8, if we obtain malformed input, we output 0 in the Turing reduction—For a (weakly) parsimonious reduction we would need to output an instance c such that $g(c) \cdot P(c) = 0$. A task that is easily doable for most natural problems P, but not in general.

Lemma 11.8. *We have* $\#\text{Hom}(\mathcal{H}_P \to \mathcal{G}_P) \leq_T^{fpt} (P, \kappa)$.

Proof. Given graphs H and G and an oracle $\mathbb{O}$ for (P, κ), we wish to compute the number of homomorphisms $\#Hom(H \to G)$ if the promise $H \in \mathcal{H}_P$ and $G \in \mathcal{G}_P$ is fulfilled. Consider the following algorithm $\mathbb{B}$.

1. Verify that the graph H is the union of two Kneser graphs $\mathrm{K}(a) \in \mathcal{K}_{even}$ and $\mathrm{K}(b) \in \mathcal{K}_{odd}$. If this is not the case, output 0.
2. Verify that the graph G is the union of a set of paths $\mathcal{P}$ (and isolated vertices) and two connected components G_1 and G_2 that are not paths. If this is not the case, output 0.
3. Verify that either $G_1 = \mathrm{K}(b)$ or $G_2 = \mathrm{K}(b)$ holds, otherwise output 0. Now, assume that $G_2 = \mathrm{K}(b)$.[115]
4. Find a pair (c, H_c) such that $enc(\langle c, H_c \rangle) = \mathcal{P}$ or report that no decoding is possible.[116] If the decoding failed, output 0.
5. Compute the parameter $\kappa(c)$ of the instance c. If we have $2\kappa(c) + 3 \neq b$, output 0.
6. Verify that the graphs H_c and $\mathrm{K}(a)$ are isomorphic. If they are not isomorphic, output 0.
7. Compute the value $g(c)$. If we have $g(c) = 0$, output 0.
8. Query the oracle $\mathbb{O}$ on input c and obtain $\mathbb{O}(c)$. Output $\mathbb{O}(c) \cdot \#\mathsf{Aut}(\mathrm{K}(2\kappa(c) + 3))/g(c)$.

115. Otherwise relabel G_1 and G_2.

116. For instance if the set of paths $\mathcal{P}$ is empty, contains the same path multiple times or the number of isolated vertices does not match.

We first prove the required bound on the running time of the algorithm $\mathbb{B}$. On input H and G, Step 1 takes time depending only on $|V(H)|$; Step 2 can be done in time polynomial in $|V(G)|$. Step 3 takes again time depending only on $|V(H)|$. Considering Step 4, we observe that by the definition of the encoding enc and by the assumption that $\langle \star, \star \rangle$ is an efficient encoding of pairs, the decoding can be done in time polynomial in $|V(\mathcal{P})| \leq |V(G)|$. Step 5 can also be done in time polynomial in $|V(G)|$, as the function κ is computable in polynomial time in $|c|$. As the encoding $enc(\langle c, H\rangle)$ contains an isolated vertex for every bit of the string c, we have

$$|c| \le |V(G)| \tag{11.4}$$

and hence the claimed running time for Step 5. Similarly to Step 3, we can perform Step 6 in time depending only on $|V(H)|$. Now assume Step 7 is reached. In this case, $2\kappa(c) + 3 = b$ and hence

$$\kappa(c) \le |V(H)|, \tag{11.5}$$

as the graph $\mathrm{K}(b)$ is a component of H and we have $|V(\mathrm{K}(b))| \ge b$. Note that Steps 7 and 8 take time $p(\kappa(c)) \cdot |c|^{O(1)}$, as the problem (g, κ) is FPT (see 3). We may assume that p is monotonically increasing and thus Equation (11.5) yields a running time bound of $p(|V(H)|) \cdot |V(G)|^{O(1)}$; recall Equation (11.4), that is $|c| \le |V(G)|$.[117]

Next, we prove the correctness of $\mathbb{B}$. To that end, assume that the promise is fulfilled, that is, $H \in \mathcal{H}_P$ and $G \in \mathcal{G}_P$.[118]

Hence, for instances x and y, we have

$$\begin{aligned} H = H_y^* &= \mathrm{K}(a) \cup \mathrm{K}(b) \\ &= H_y \cup \mathrm{K}(2\kappa(y) + 3), \\ G = G_x^* &= G_1 \cup \mathcal{P} \cup G_2 \\ &= G_x \cup enc(\langle x, H_x\rangle) \cup \mathrm{K}(2\kappa(x) + 3). \end{aligned}$$

Further, by construction we have $g(x) \ne 0$. We consider three cases.

1 $H_x \ne H_y$: The instances x and y are different. By Lemma 11.7, there are no homomorphisms from H to G.
In the algorithm $\mathbb{B}$, in this case, the test in Step 6 fails, and $\mathbb{B}$ outputs 0, which is correct.

2 $H_x = H_y$ and $\kappa(x) \ne \kappa(y)$.[119] Indeed, in this case the instances x and y differ and so, again by Lemma 11.7, there are no homomorphisms from the graph H to the graph G.

117. Note that the last argument also shows that the parameter $\kappa(c)$ of the oracle query $\mathbb{O}(c)$ is bounded by $|V(H)|$.

118. If the promise is not fulfilled, we are not required to compute a correct output.

119. Note that $H_x = H_y$ does not imply that the corresponding instances are the same; the algorithm $\mathbb{A}$ is not necessarily injective.

In the algorithm $\mathbb{B}$, in this case, the test in Step 3 fails, and $\mathbb{B}$ outputs 0, which is correct.

3 $H_x = H_y$ and $\kappa(x) = \kappa(y)$: In this case, we have $H_y^* = H_x^*$. Hence, by Lemma 11.6, the number of homomorphisms from H to G is

$$\#Hom(\,H \to G\,) = \#Hom(\,H_x \to G_x\,) \cdot \#Aut(\,\mathrm{K}(2\kappa(x)+3)\,). \tag{11.6}$$

Note that the oracle $\mathbb{O}$ on input x computes the number

$$\mathbb{O}(x) = g(x) \cdot \#Hom(\,H_x \to G_x\,), \tag{11.7}$$

and we may assume that $g(x) \neq 0$ by construction. Hence, combining Equation (11.6) and Equation (11.7) yields that we can compute the number of homomorphisms from H to G by

$$\#Hom(\,H \to G\,) = \mathbb{O}(x) \cdot \#Aut(\,\mathrm{K}(2\kappa(x)+3)\,)/g(x).$$

In the algorithm $\mathbb{B}$, it is easy to verify that Steps 3 to 7 succeed and that in Step 8, we do return $\mathbb{O}(x) \cdot \#Aut(\,\mathrm{K}(2\kappa(x)+3)\,)/g(x)$. Hence, the algorithm is correct in this case as well.

In total, the algorithm $\mathbb{B}$ correctly solves $\#\textsc{Hom}(\mathcal{H}_P \to \mathcal{G}_P)$. ⌟

Finally, we construct and verify the easy reduction.

Lemma 11.9. *We have* $(P,\kappa) \leq_T^{fpt} \#\textsc{Hom}(\mathcal{H}_P \to \mathcal{G}_P)$.

Proof. Given an instance x to (P,κ) and an oracle $\mathbb{O}$ for the problem $\#\textsc{Hom}(\mathcal{H}_P \to \mathcal{G}_P)$, we wish to compute the number $P(x)$. By Lemma 8.19, we have $(P,\kappa) \leq^{w\text{-}fpt} \#\textsc{Hom}(\mathcal{K}_{even} \to \top)$; let $\mathbb{A}$ denote the corresponding algorithm. For the graphs $(H_x, G_x) := \mathbb{A}(x)$, we have $P(x) = g(x) \cdot \#Hom(\,H_x \to G_x\,)$.

Now, to compute the result $P(x)$, we first compute the value $g(x)$ in FPT time with respect to κ. If we observe $g(x) = 0$, we output 0. Otherwise, we simulate the algorithm $\mathbb{A}$ and obtain graphs H_x and G_x in time $f(\kappa(x)) \cdot |x|^{O(1)}$. After that, we can compute the graphs H_x^* and G_x^* in time $p(\kappa(x)) \cdot |x|^{O(1)}$: The construction of the encoding $enc(\langle x, H_x \rangle)$ can be done in polynomial time in $|x|$ and $|V(H_x)|$. Note that $|V(H_x)|$ is bounded by $s(\kappa(x))$ and that the construction of $\mathrm{K}(2\kappa(x)+3)$ clearly takes time depending only on $\kappa(x)$. In particular, we have the size of the graph H_x^* depends only on $\kappa(x)$. Hence, we can query $\mathbb{O}$ on (H_x^*, G_x^*) and obtain $\#Hom(\, H_x^* \to G_x^* \,)$.

Recall that by Lemma 11.6, we have

$$\begin{aligned} \#Hom(\, H_x \to G_x \,) &= \#\mathsf{Aut}(\mathrm{K}(2\kappa(x)+3))^{-1} \cdot \#Hom(\, H_x^* \to G_x^* \,) \\ &= \#\mathsf{Aut}(\mathrm{K}(2\kappa(x)+3))^{-1} \cdot \mathbb{O}(H_x^*, G_x^*). \end{aligned}$$

Hence, to compute the result $P(x) = g(x) \cdot \#Hom(\, H_x \to G_x \,)$, we can compute the number $\#\mathsf{Aut}(\mathrm{K}(2\kappa(x)+3))^{-1}$ in time depending only on $\kappa(x)$ and multiply it with the result of the oracle query and the result previous computation of the value $g(x)$. ▟

THEOREM 11.1. *For any problem (P, κ) in $\#\mathbf{W}[1]$, there are graph classes $\mathcal{H}_P$ and $\mathcal{G}_P$ such that*

$$(P, \kappa) \equiv_T^{fpt} \#\textsc{Hom}(\mathcal{H}_P \to \mathcal{G}_P).$$

Further, $\mathcal{H}_P$ is recursively enumerable and $\mathcal{G}_P$ is recursive.

PROOF. Follows from Lemmas 11.8 and 11.9. ▟

Using Corollary 8.20 as a starting point, we also obtain:

THEOREM 11.10. *For any problem (P, κ) in $\mathbf{W}[1]$, there are graph classes $\mathcal{H}_P$ and $\mathcal{G}_P$ such that*

$$(P, \kappa) \equiv_T^{fpt} \textsc{Hom}(\mathcal{H}_P \to \mathcal{G}_P).$$

Further, $\mathcal{H}_P$ is recursively enumerable and $\mathcal{G}_P$ is recursive. ▟

Conclusions and Open Questions

The contributions of this part were two-fold. First, we studied the problem #Hom$(\mathcal{H} \to \mathcal{G})$ for graph classes $\mathcal{H}$ and $\mathcal{G}$. We discussed *complexity dichotomies* for two specific pairs of graph classes. Further, we convinced ourselves that we may not hope for clean dichotomy for all pairs of graph classes—We can *reversibly* encode any problem in #$W[1]$ into classes of Kneser graphs, thereby obtaining a universality result for #$W[1]$. In particular, we relied on (specific) Kneser graphs forming an *infinite anti-chain* with respect to the homomorphism order.[120] There are other (sets of) graphs with this property—Can we obtain a similar universality result for them? Are such anti-chains the only obstacle for obtaining complexity dichotomy results for #Hom$(\mathcal{H} \to \mathcal{G})$?

120. To some extent we also needed that Kneser graphs cannot be mapped homomorphically to collections of paths. However, this seems to be the less-important property.

Second, we studied the problem #IndSub(Φ) for a graph property Φ. Exploiting that said problem morphs to evaluating linear combinations of homomorphism counts and known hardness results for evaluating said linear combinations, we could obtain hardness for a variety of properties Φ by analyzing the so-called f-vector of Φ. In particular, assuming ETH, we obtained (almost) tight lower bounds for all monotone, that is subgraph-closed, properties and properties that only depend on the number of edges of a graph. Even more general, we proved that a sufficiently *sparse* f-vector guarantees hardness. Together with the long line of previous work on #IndSub(Φ) [62, 63, 87, 64, 100, 40], our results provide further evidence that for all non-trivial Φ, the problem #IndSub(Φ) is hard. This motivates the following conjecture.

Conjecture. *For any non-trivial Φ, the problem #IndSub(Φ) is #$W[1]$-complete and, assuming ETH, cannot be solved in time $f(k) \cdot |V(G)|^{o(k)}$ for any function f.*

As next steps toward proving this conjecture, it seems natural to consider properties with *non-spare f*-vectors. As a natural example, consider properties closed under *induced* subgraphs or *hereditary* properties. In [102], we did prove first results in that direction—a comprehensive study of such properties is still missing though. Similarly, for properties closed under edge-removal, the above conjecture holds on odd cycles [100] and for properties that are nontrivial on bipartite graphs [40]. Again, the general case is still not resolved, though.

As a final concluding remark, when generalizing graph properties to *graph functions*—that is allowing arbitrary reals as values of Φ—it is fathomable that we can capture many more natural problems. While the graph motif parameter framework naturally generalizes, proving non-vanishing of the corresponding coefficients a does require new ideas.

Bibliography

1 A. Abboud, A. Backurs, and V. V. Williams. If the current clique algorithms are optimal, so is Valiant's parser. In *56th Annual Symposium on Foundations of Computer Science, FOCS'15*, pages 98–117, 2015. doi: 10.1109/FOCS.2015.16.

2 K. R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987. doi: 10.1137/0216067.

3 M. O. Albertson and K. L. Collins. Homomorphisms of 3-chromatic Graphs. *Discrete Mathematics*, 54(2):127–132, 1985. doi: 10.1016/0012-365X(85)90073-1.

4 S. Alstrup, G. S. Brodal, and T. Rauhe. Pattern matching in dynamic texts. In D. B. Shmoys, editor, *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, pages 819–828. ACM/SIAM, 2000.

5 A. Amir, M. Lewenstein, and E. Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2): 257–275, 2004. doi: 10.1016/S0196-6774(03)00097-X.

6 M. Babenko, P. Gawrychowski, T. Kociumaka, I. Kolesnichenko, and T. Starikovskaya. Computing minimal and maximal suffixes of a substring. *Theoretical Computer Science*, 638:112–121, 2016. doi: 10.1016/j.tcs.2015.08.023.

7 A. Backurs and P. Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (Unless SETH is False). *SIAM J. Comput.*, 47(3):1087–1097, 2018. doi: 10.1137/15M1053128.

8 L. W. Beineke. Characterizations of Derived Graphs. *Journal of Combinatorial Theory*, 9(2):129–135, 1970. ISSN 0021-9800. doi: https://doi.org/10.1016/S0021-9800(70)80019-9.

9 M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, pages 88–94, 2000. doi: 10.1007/10719839_9.

10 A. Bhattacharyya, S. Ghoshal, Karthik C. S., and P. Manurangsi. Parameterized Intractability of Even Set and Shortest Vector Problem from Gap-ETH. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 17:1–17:15, 2018. doi: 10.4230/LIPIcs.ICALP.2018.17.

11 A. Bhattacharyya, S. Ghoshal, Karthik C. S., and P. Manurangsi. Parameterized Intractability of Even Set and Shortest Vector Problem from Gap-ETH. *CoRR*, abs/1803.09717, 2018.

12 P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. R. Satti, and O. Weimann. Random Access to Grammar-Compressed Strings and Trees. *SIAM J. Comput.*, 44(3):513–539, 2015. doi: 10.1137/130936889.

13 L. Billera and A. Björner. Face Numbers of Polytopes and Complexes. In *Handbook of discrete and computational geometry*, pages 291–310. CRC Press, Inc., 1997.

14 M. Bodirsky and M. Grohe. Non-dichotomies in constraint satisfaction complexity. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, pages 184–196, 2008. doi: 10.1007/978-3-540-70583-3_16.

15 H. L. Bodlaender. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM J. Comput.*, 25(6): 1305–1317, 1996. doi: 10.1137/S0097539793251219.

16 C. Borgs, J. Chayes, L. Lovász, V. T. Sós, and K. Vesztergombi. Counting graph homomorphisms. In M. Klazar, J. Kratochvíl, M. Loebl, J. Matoušek, P. Valtr, and R. Thomas, editors, *Topics in Discrete Mathematics*, pages 315–371, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33700-3.

17 R. S. Boyer and J. S. Moore. MJRTY—a fast majority vote algorithm. In R. S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Automated Reasoning Series, pages 105–117. Kluwer Academic Publishers, 1991. doi: 10.1007/978-94-011-3488-0_5.

18 K. Bringmann and P. Wellnitz. Clique-based lower bounds for parsing tree-adjoining grammars. In J. Kärkkäinen, J. Radoszewski, and W. Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi: 10.4230/LIPIcs.CPM.2017.12.

19 K. Bringmann and P. Wellnitz. On Near-Linear-Time Algorithms for Dense Subset Sum. In D. Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1777–1796. SIAM, 2021. doi: 10.1137/1.9781611976465.107.

20 K. Bringmann, M. Künnemann, and P. Wellnitz. Few matches or almost periodicity: Faster pattern matching with mismatches in compressed texts. In *30th Annual ACM-SIAM Symposium on Discrete Algorithmsm, SODA 2019*, San Diego, CA, USA, 2019. SIAM.

21 K. Bringmann, N. Fischer, D. Hermelin, D. Shabtay, and P. Wellnitz. Faster Minimization of Tardy Processing Time on a Single Machine. In A. Czumaj, A. Dawar, and E. Merelli, editors,

47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference), volume 168 of *LIPIcs*, pages 19:1–19:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi: 10.4230/LIPIcs.ICALP.2020.19.

22 M. Burrows and D. Wheeler. A block-sorting lossless data compression algorithm. Technical report, DIGITAL SRC RESEARCH REPORT, 1994.

23 T. M. Chan, S. Golan, T. Kociumaka, T. Kopelowitz, and E. Porat. Approximating Text-to-Pattern Hamming Distances, 2020. To appear at STOC'20.

24 L. S. Chandran and C. R. Subramanian. Girth and treewidth. *J. Comb. Theory, Ser. B*, 93(1):23–32, 2005. doi: 10.1016/j.jctb.2004.05.004.

25 P. Charalampopoulos. *Data Structures for Strings in the Internal and Dynamic Settings*. PhD thesis, King's College London, 2021.

26 P. Charalampopoulos, T. Kociumaka, and P. Wellnitz. Faster Approximate Pattern Matching: A Unified Approach. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 978–989. IEEE, 2020. doi: 10.1109/FOCS46700.2020.00095.

27 H. Chen and S. Mengel. Counting Answers to Existential Positive Queries: A Complexity Classification. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 315–326, 2016. doi: 10.1145/2902251.2902279.

28 J. Chen, B. Chor, M. Fellows, X. Huang, D. W. Juedes, I. A. Kanj, and G. Xia. Tight lower bounds for certain parameterized NP-hard problems. *Inf. Comput.*, 201(2):216–231, 2005. doi: 10.1016/j.ic.2005.05.001.

29 J. Chen, X. Huang, I. A. Kanj, and G. Xia. Strong computational lower bounds via parameterized complexity. *J. Comput. Syst. Sci.*, 72(8):1346–1367, 2006. doi: 10.1016/j.jcss.2006.04.007.

30 Y. Chen, M. Thurley, and M. Weyer. Understanding the Complexity of Induced Subgraph Isomorphisms. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, pages 587–596, 2008. doi: 10.1007/978-3-540-70575-8_48.

31 M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas. The Strong Perfect Graph Theorem. *Annals of Mathematics*, 164 (1):51–229, 2006. ISSN 0003486X.

32 R. Clifford, A. Fontaine, E. Porat, B. Sach, and T. Starikovskaya. The k-mismatch problem revisited. In R. Krauthgamer, editor, *27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 2039–2052. SIAM, 2016. doi: 10.1137/1.9781611974331.ch142.

33 R. Cole and R. Hariharan. Approximate String Matching: A Simpler Faster Algorithm. *SIAM J. Comput.*, 31(6):1761–1782, 2002. doi: 10.1137/S0097539700370527.

34 R. Curticapean and D. Marx. Complexity of Counting Subgraphs: Only the Boundedness of the Vertex-Cover Number Counts. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 130–139, 2014. doi: 10.1109/FOCS.2014.22.

35 R. Curticapean, H. Dell, and D. Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223, 2017. doi: 10.1145/3055399.3055502.

36 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized*

Algorithms. Springer, 2015. ISBN 978-3-319-21274-6. doi: 10.1007/978-3-319-21275-3.

37 V. Dalmau and P. Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329 (1-3):315–323, 2004. doi: 10.1016/j.tcs.2004.08.008.

38 H. Dell, M. Roth, and P. Wellnitz. Counting answers to existential questions. In C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 113:1–113:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi: 10.4230/LIPIcs.ICALP.2019.113.

39 J. Díaz, M. J. Serna, and D. M. Thilikos. Counting H-colorings of partial k-trees. *Theor. Comput. Sci.*, 281(1-2):291–309, 2002. doi: 10.1016/S0304-3975(02)00017-8.

40 J. Dörfler, M. Roth, J. Schmitt, and P. Wellnitz. Counting Induced Subgraphs: An Algebraic Approach to #W[1]-hardness. In P. Rossmanith, P. Heggernes, and J. Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 26:1–26:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi: 10.4230/LIPIcs.MFCS.2019.26.

41 R. Downey and M. Fellows. Fixed-parameter tractability and completeness III: Some structural aspects of the W hierarchy. In *Complexity theory*, pages 191–225. Cambridge University Press, 1993.

42 M. E. Dyer and C. S. Greenhill. The complexity of counting graph homomorphisms. *Random Struct. Algorithms*, 17(3-4): 260–289, 2000.

43 J. Edmonds. Paths, Trees, and Flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi: 10.4153/CJM-1965-045-4.

44 P. Erdös and G. Szckeres. A combinatorial problem in geometry. In I. Gessel and G.-C. Rota, editors, *Classic Papers in Combinatorics*, pages 49–56, Boston, MA, 1987. Birkhäuser Boston. ISBN 978-0-8176-4842-8. doi: 10.1007/978-0-8176-4842-8_3.

45 M. Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 137–143, 1997. doi: 10.1109/SFCS.1997.646102.

46 J. Flum and M. Grohe. The Parameterized Complexity of Counting Problems. *SIAM J. Comput.*, 33(4):892–922, 2004. doi: 10.1137/S0097539703427203.

47 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. ISBN 978-3-540-29952-3. doi: 10.1007/3-540-29953-X.

48 M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $o(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984. doi: 10.1145/828.1884.

49 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN 0-7167-1044-7.

50 M. R. Garey, D. S. Johnson, and H. C. So. An Application of Graph Coloring to Printed Circuit Testing (Working Paper). In *16th Annual Symposium on Foundations of Computer Science, Berkeley, California, USA, October 13-15, 1975*, pages 178–183, 1975. doi: 10.1109/SFCS.1975.3.

51 P. Gawrychowski and P. Uznański. Towards unified approximate pattern matching for Hamming and L_1 distance. In I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, editors, *Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 62:1–62:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. doi: 10.4230/LIPIcs.ICALP.2018.62.

52 P. Gawrychowski, A. Karczmarz, T. Kociumaka, J. Lacki, and P. Sankowski. Optimal dynamic strings. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1509–1528, 2018. doi: 10.1137/1.9781611975031.99. Full version available at `arXiv:1511.02612`.

53 O. Goldreich. *Computational Complexity - A Conceptual Perspective*. Cambridge University Press, 2008. ISBN 978-0-521-88473-0.

54 J. A. Grochow and M. Kellis. Network Motif Discovery Using Subgraph Enumeration and Symmetry-Breaking. In T. Speed and H. Huang, editors, *Research in Computational Molecular Biology*, pages 92–106, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-71681-5.

55 M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1): 1:1–1:24, 2007. doi: 10.1145/1206035.1206036.

56 M. Grohe, T. Schwentick, and L. Segoufin. When is the evaluation of conjunctive queries tractable? In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 657–666, 2001. doi: 10.1145/380752.380867.

57 G. Hahn and C. Tardif. Graph homomorphisms: Structure and Symmetry. In *Graph symmetry*, pages 107–166. Springer, 1997.

58 F. Harary. *Graph theory*. Addison-Wesley series in mathematics. Addison-Wesley Pub. Co., 1969.

59 P. Hell and J. Nesetril. On the Complexity of *H*-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990. doi: 10.1016/0095-8956(90)90132-J.

60 T. I. Longest common extensions with recompression. In J. Kärkkäinen, J. Radoszewski, and W. Rytter, editors, *28th*

Annual Symposium on Combinatorial Pattern Matching, CPM 2017, volume 78 of *LIPIcs*, pages 18:1–18:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi: 10.4230/LIPIcs.CPM.2017.18.

61 R. Impagliazzo and R. Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi: 10.1006/jcss.2000.1727.

62 M. Jerrum and K. Meeks. The parameterised complexity of counting connected subgraphs and graph motifs. *J. Comput. Syst. Sci.*, 81(4):702–716, 2015. doi: 10.1016/j.jcss.2014.11.015.

63 M. Jerrum and K. Meeks. Some Hard Families of Parameterized Counting Problems. *TOCT*, 7(3):11:1–11:18, 2015. doi: 10.1145/2786017.

64 M. Jerrum and K. Meeks. The parameterised complexity of counting even and odd induced subgraphs. *Combinatorica*, 37(5):965–990, 2017. doi: 10.1007/s00493-016-3338-5.

65 A. Jeż. Faster fully compressed pattern matching by recompression. *ACM Transactions on Algorithms*, 11(3):20:1–20:43, 2015. doi: 10.1145/2631920.

66 A. Jeż. Recompression: A simple and powerful technique for word equations. *Journal of the ACM*, 63(1):4:1–4:51, 2016. doi: 10.1145/2743014.

67 P. W. Kasteleyn. The statistics of dimers on a lattice: I. The number of dimer arrangements on a quadratic lattice. *Physica*, 27(12):1209–1225, 1961. doi: 10.1016/0031-8914(61)90063-5.

68 P. W. Kasteleyn. Dimer Statistics and Phase Transitions. *Journal of Mathematical Physics*, 4(2):287–293, 1963. doi: 10.1063/1.1703953.

69 D. Kempa and T. Kociumaka. Resolution of the burrows-wheeler transform conjecture. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC,*

USA, November 16-19, 2020, pages 1002–1013. IEEE, 2020. doi: 10.1109/FOCS46700.2020.00097.

70 D. Kempa and N. Prezza. At the roots of dictionary compression: string attractors. In I. Diakonikolas, D. Kempe, and M. Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 827–840. ACM, 2018. doi: 10.1145/3188745.3188814.

71 S. Khot and V. Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theor. Comput. Sci.*, 289(2):997–1008, 2002. doi: 10.1016/S0304-3975(01)00414-5.

72 T. Kociumaka. *Efficient Data Structures for Internal Queries in Texts*. PhD thesis, University of Warsaw, Oct. 2018.

73 T. Kociumaka, J. Radoszewski, W. Rytter, and T. Walen. Internal pattern matching queries in a text and applications. In P. Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 532–551. SIAM, 2015. doi: 10.1137/1.9781611973730.36.

74 S. Kosaraju. Efficient string matching. Manuscript, 1987.

75 A. V. Kostochka. Lower bound of the Hadwiger number of graphs by their average degree. *Combinatorica*, 4(4):307–316, 1984. doi: 10.1007/BF02579141.

76 R. E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, 1975. doi: 10.1145/321864.321877.

77 G. M. Landau and U. Vishkin. Efficient string matching with k mismatches. *Theor. Comput. Sci.*, 43:239–249, 1986. doi: 10.1016/0304-3975(86)90178-7.

78 G. M. Landau and U. Vishkin. Fast parallel and serial approximate string matching. *J. Algorithms*, 10(2):157–169, 1989. doi: 10.1016/0196-6774(89)90010-2.

79 N. Larsson and A. Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000. doi: 10.1109/5.892708.

80 P. G. H. Lehot. An Optimal Algorithm to Detect a Line Graph and Output Its Root Graph. *J. ACM*, 21(4):569–575, Oct. 1974. ISSN 0004-5411. doi: 10.1145/321850.321853.

81 A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976.

82 L. Lovász. Kneser's Conjecture, Chromatic Number, and Homotopy. *J. Comb. Theory, Ser. A*, 25(3):319–324, 1978. doi: 10.1016/0097-3165(78)90022-5.

83 L. Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, 2012. ISBN 978-0-8218-9085-1.

84 D. Marx. Can You Beat Treewidth? *Theory of Computing*, 6(1): 85–112, 2010. doi: 10.4086/toc.2010.v006a005.

85 H. A. Maurer, I. H. Sudborough, and E. Welzl. On the Complexity of the General Coloring Problem. *Information and Control*, 51(2):128–145, 1981. doi: 10.1016/S0019-9958(81)90226-6.

86 C. McCartin. Parameterized counting problems. *Ann. Pure Appl. Logic*, 138(1-3):147–182, 2006. doi: 10.1016/j.apal.2005.06.010.

87 K. Meeks. The challenges of unbounded treewidth in parameterised subgraph counting problems. *Discrete Applied Mathematics*, 198:170–194, 2016. doi: 10.1016/j.dam.2015.06.019.

88 K. Mehlhorn, R. Sundar, and C. Uhrig. Maintaining Dynamic Sequences under Equality Tests in Polylogarithmic Time. *Algorithmica*, 17(2):183–198, 1997. doi: 10.1007/BF02522825.

89 R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network Motifs: Simple Building Blocks of Complex Networks. *Science*, 298(5594):824–827, 2002. ISSN 0036-8075. doi: 10.1126/science.298.5594.824.

90 G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33:2001, 2001.

91 S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970. ISSN 0022-2836. doi: https://doi.org/10.1016/0022-2836(70)90057-4.

92 C. G. Nevill-Manning and I. H. Witten. Compression and explanation using hierarchical grammars. *The Computer Journal*, 40(2 and 3):103–116, 1997.

93 R. Niedermeier. Invitation to fixed-parameter algorithms. *Oxford Lecture Series in Mathematics and its Applications*, 31, 2002.

94 G. Pólya. Bemerkung zur Interpolation und zur Näherungstheorie der Balkenbiegung. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 11(6):445–449, 1931.

95 F. P. Ramsey. On a Problem of Formal Logic. *Proceedings of the London Mathematical Society*, s2-30(1):264–286, 1930. doi: 10.1112/plms/s2-30.1.264.

96 N. Robertson and P. D. Seymour. Graph minors. V. Excluding a planar graph. *J. Comb. Theory, Ser. B*, 41(1):92–114, 1986. doi: 10.1016/0095-8956(86)90030-4.

97 N. Robertson, P. D. Seymour, and R. Thomas. Quickly Excluding a Planar Graph. *J. Comb. Theory, Ser. B*, 62(2):323–348, 1994. doi: 10.1006/jctb.1994.1073.

98 G.-C. Rota. On the foundations of combinatorial theory I. Theory of Möbius functions. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 2(4):340–368, 1964.

99 M. Roth. *Counting Problems on Quantum Graphs: Parameterized and Exact Complexity Classifications*. PhD thesis, Saarland University, 2019.

100 M. Roth and J. Schmitt. Counting induced subgraphs: A Topological Approach to #W[1]-hardness. In *13th International Symposium on Parameterized and Exact Computation, IPEC 2018, August 20-24, 2018, Helsinki, Finland*, pages 24:1–24:14, 2018. doi: 10.4230/LIPIcs.IPEC.2018.24.

101 M. Roth and P. Wellnitz. Counting and Finding Homomorphisms is Universal for Parameterized Complexity Theory. In S. Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2161–2180. SIAM, 2020. doi: 10.1137/1.9781611975994.133.

102 M. Roth, J. Schmitt, and P. Wellnitz. Counting Small Induced Subgraphs Satisfying Monotone Properties. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1356–1367. IEEE, 2020. doi: 10.1109/FOCS46700.2020.00128. Accepted (subject to minor revision) to the SICOMP Special Issue.

103 M. Roth, J. Schmitt, and P. Wellnitz. Detecting and counting small subgraphs, and evaluating a parameterized tutte polynomial: Lower bounds via toroidal grids and cayley graph expanders. In N. Bansal, E. Merelli, and J. Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 108:1–108:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi: 10.4230/LIPIcs.ICALP.2021.108.

104 W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003. doi: 10.1016/S0304-3975(02)00777-6.

105 S. C. Sahinalp and U. Vishkin. Approximate pattern matching using locally consistent parsing. Manuscript, 1997.

106 I. J. Schoenberg. On Hermite-Birkhoff Interpolation. *Journal of Mathematical Analysis and Applications*, 16(3):538–543, 1966. ISSN 0022-247X. doi: 10.1016/0022-247X(66)90160-0.

107 F. Schreiber and H. Schwöbbermeyer. Frequency Concepts and Pattern Detection for the Analysis of Motifs in Networks. In C. Priami, E. Merelli, P. Gonzalez, and A. Omicini, editors, *Transactions on Computational Systems Biology III*, pages 89–104, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31446-2.

108 P. H. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1(4):359–373, 1980. ISSN 0196-6774. doi: https://doi.org/10.1016/0196-6774(80)90016-4.

109 Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa. Byte pair encoding: A text compression scheme that accelerates pattern matching. Technical report, Technical Report DOI-TR-161, Department of Informatics, Kyushu University, 1999.

110 J. Spencer. Ramsey's theorem—a new lower bound. *Journal of Combinatorial Theory, Series A*, 18(1):108–115, 1975. ISSN 0097-3165. doi: 10.1016/0097-3165(75)90071-0.

111 R. P. Stanley. *Enumerative Combinatorics: Volume 1*. Cambridge University Press, 2011. ISBN 978-1-107-60202-5.

112 R. Sundar and R. E. Tarjan. Unique binary-search-tree representations and equality testing of sets and sequences. *SIAM Journal on Computing*, 23(1):24–44, 1994.

113 H. N. V. Temperley and M. E. Fisher. Dimer problem in statistical mechanics-an exact result. *The Philosophical Magazine: A Journal of Theoretical Experimental and Applied Physics*, 6(68): 1061–1063, 1961. doi: 10.1080/14786436108243366.

114 A. Thomason. An extremal function for contractions of graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 95(2):261–265, 1984. doi: 10.1017/S0305004100061521.

115 S. Toda. PP is as Hard as the Polynomial-Time Hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi: 10.1137/0220053.

116 M. Valencia-Pabon and J.-C. Vera. On the diameter of kneser graphs. *Discrete Mathematics*, 305(1):383–385, 2005. ISSN 0012-365X. doi: https://doi.org/10.1016/j.disc.2005.10.001.

117 L. G. Valiant. The Complexity of Computing the Permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. doi: 10.1016/0304-3975(79)90044-6.

118 L. G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.*, 8(3):410–421, 1979. doi: 10.1137/0208032.

119 T. Welch. A technique for high-performance data compression. *Computer*, 17:8–19, 1984.

120 H. Whitney. Congruent Graphs and the Connectivity of Graphs. In *Hassler Whitney Collected Papers*, pages 61–79. Birkhäuser Boston, 1992. doi: https://doi.org/10.1007/978-1-4612-2972-8_4.

121 J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23 (3):337–343, 1977.

Index