Julian Pape-Lange

Combinatorial Properties of Periodic Patterns in Compressed Strings
Julian Pape-Lange

Combinatorial Properties of Periodic Patterns in Compressed Strings

Universitätsverlag Chemnitz
2023
Acknowledgments

First of all I want to thank Professor Laurent Bartholdi without whom this PhD thesis would not have been possible. He introduced me to the topics of string compression and computation on compressed strings for my master’s thesis, which was also the starting point of my PhD research. His way of teaching greatly helped forming my mathematical intuition. I am very grateful for our interesting talks during my master’s studies and the time of my PhD research.

Moreover, I would like to extend my sincere thanks to Professor Andreas Goerdt for his support, encouragement and trust in me during my time in Chemnitz. Especially our talks on the Burrows-Wheeler-transform and the fast Fourier transform were very fruitful.

Special thanks also go to Professor Klaus Meer for his careful examination of my thesis with many helpful comments for the final version.

Additionally, I want to thank my friends, colleagues, fellow students and professors for their advice and the many inspiring and motivating talks about mathematics and computer science.

Lastly, I am very grateful to my family, especially my grandmother, for their support and for always having an open ear for my problems.
# Contents

1 Introduction ........................................... 1
   1.1 Motivation ........................................... 1
   1.2 Periodic Patterns .................................... 2
   1.3 Scope of this Thesis ................................. 4
   1.4 Structure of this Thesis ............................. 5

2 Preliminaries ........................................... 7
   2.1 Strings and Their Substructures ..................... 7
      2.1.1 Substrings ....................................... 7
      2.1.2 Subsequences .................................... 9
   2.2 Periodic Patterns .................................... 10
      2.2.1 Powers .......................................... 10
      2.2.2 Periods .......................................... 11
      2.2.3 Maximal (Sub)-Repetitions .................... 12
      2.2.4 Maximal Pairs and Maximal Repeats ............ 13
      2.2.5 Cadences and Their Variants ................... 16
   2.3 Compressions and Complexity Measures ............... 17
      2.3.1 String Attractors ................................ 17
      2.3.2 LZ77-Type Compressions ......................... 17
      2.3.3 Grammar-Based Compressions .................... 18
      2.3.4 Compacted Directed Acyclic Word Graph ........ 20
      2.3.5 RLBWT .......................................... 21
      2.3.6 Substring Complexity ............................ 22
      2.3.7 Relationships Between the Complexity Measures .. 25
   2.4 Morphisms ........................................... 26
   2.5 Model of Computation ................................ 27

3 Non-Rectangular Convolution ......................... 29
   3.1 Rectangular Convolution ............................. 30
   3.2 Triangular Convolution ................................ 31
   3.3 Convex Convolution .................................. 34
CONTENTS

3.4 Non-Convex Convolution 36

4 Alphabet Reduction 39

5 Maximal (Sub-)Repetitions 45
  5.1 Related Work 46
    5.1.1 Squares 46
    5.1.2 Higher Powers 48
    5.1.3 Lower Powers 48
    5.1.4 Maximal $\delta$-(Sub-)Repetitions 50
    5.1.5 Algorithms 51
  5.2 Compressed Upper Bounds 51
    5.2.1 Upper Bound for Maximal $\delta$-Repetitions 54
    5.2.2 Upper Bound for Maximal $\delta$-Subrepetitions 59
  5.3 Tightness 64
  5.4 Conclusion 68

6 Cadences 69
  6.1 Related Work 70
  6.2 Bounds 71
  6.3 NP-Complete Cadence Problems 78
  6.4 Algorithms 86
    6.4.1 Naïve Algorithms 86
    6.4.2 Convolution-Based Algorithms 87
    6.4.3 Regularity-Based Algorithms 91
  6.5 Longer (Sub-)Cadences 110

7 Maximal Pairs 111
  7.1 Related Work 112
    7.1.1 Combinatorial Properties and CDAWG  112
    7.1.2 RLBWT 113
    7.1.3 Algorithms 114
  7.2 Compressed Upper Bounds 115
    7.2.1 Non-Highly-Periodic Maximal Pairs 117
    7.2.2 Highly-Periodic Maximal Pairs 128
    7.2.3 Derived Upper Bounds 131
  7.3 Tightness 135
  7.4 Conclusion 140

A Propositions 153
Chapter 1

Introduction

1.1 Motivation

Strings are one of the most fundamental data types in computer science. With the increasing size of the available data, it becomes more and more important to develop a good understanding of the underlying structure of the data.

However, even finding a good approximation of the amount of information a string contains is difficult. Shannon’s entropy is a good measure for strings whose characters are independent and identically distributed but it cannot differentiate between, for example, $a^n b^n$ and a string that contains $n$ as and $n$ bs in a random order.

The other extreme is the Kolmogorov complexity which in theory exactly shows the amount of information in a given string. However, the Kolmogorov complexity is not useful in practice since it is inherently uncomputable.

In the last years many complexity measures were proposed. Additionally, each compression scheme can serve as a complexity measure. For example, we can measure the complexity of a string by the number of LZ77-factors that we need to compress this string, the size of the smallest grammar or the number of runs of the run-length encoded Burrows-Wheeler transform. Since each of the complexity measures created this way only show the compressibility with regard to the corresponding compression scheme, it is not clear that these measures are related at all.

For example, Christodoulakis proves in [21] that there are strings with arbitrarily many LZ77-factors and arbitrarily large grammar but only two runs in the run-length encoded Burrows-Wheeler transform.

With the introduction of the string attractor in [59] by Kempa and Prezza, it was shown that many of these strong compression schemes can only differ by a few logarithmic factors with respect to length of the underlying string.
CHAPTER 1. INTRODUCTION

These bounds were subsequently improved with the recent introduction of the substring complexity by Christiansen in [20].

Initially, the main motivation of this research was to fit the size of the compacted directed acyclic word graph and the number of runs of the run-length encoded Burrows-Wheeler transform somewhere into this set of similar complexity measures. Since the compacted directed acyclic word graph is unable to compress high powers, we can only hope for an upper bound for non-highly-periodic strings.

Additionally, we hope that a better understanding of the most basic periodic patterns that are studied in this thesis may lead to a better understanding of repetitiveness in general and can thereby help with the creation of algorithms that work directly on compressed strings or data structures that are derived from uncompressed strings.

For example, cadences work as toy model that already led to an efficient algorithm for equidistant subsequence detection for patterns of size 3 by Funakoshi in [40]. These subsequences can in turn be used to analyze one-dimensional subpatterns in multi-dimensional data. It is well-known that \(k\)-subcadences cannot be avoided in sufficiently long strings. And even though van der Waerden’s bounds are a rich and active field with many nontrivial calculations and proofs, the concept of the subcadence is at least easy to grasp and thereby somewhat accessible. If it becomes possible to analyze \(k\)-subcadences with \(k \geq 4\) in subquadratic time, the subcadences may become helpful for the preliminary analysis of periodic substructures of strings with errors.

1.2 Periodic Patterns

In this thesis, we analyze maximal \(\delta\)-(sub-)repetitions, \(k\)-cadences and maximal pairs and some of their variants. The corresponding non-maximal variants can be seen in Figure 1.1.

\(\delta\)-Repetitions are substrings that are at least \(2 + \delta\) times as long as their minimum period. Similarly, \(\delta\)-subrepetitions are substrings that are at least \(1 + \delta\) times as long as their minimum period. These \(\delta\)-(sub-)repetitions are maximal if extending the substring to either side would increase the minimum period.

\(k\)-Subcadences are sequences \(\{i, i + d, i + 2d, \ldots, i + (k - 1)d\}\) whose corresponding characters in the underlying string are equal. A \(k\)-cadence additionally requires that the indices \(i - d\) and \(i + kd\) are outside the underlying string.

A maximal pair is a pair of equal substrings whose predecessors are distinct
1.2. PERIODIC PATTERNS

$\delta$-Repetition

$\sigma \quad T \quad \sigma \quad T' \quad \sigma$

3-Subcadence

$p = |P| + |P'|$

Repetition

$P \quad P'' \quad P'' \quad \cdots \cdots$

Square

$p = |P| + |P''|$

$\delta$-Subrepetition

$p = |P| + |P''| \geq \delta p$

$\frac{1}{2}$-Gapped Repeat

$P \quad P'' \quad P''$

Pair of Repeats

$P \quad T \quad P$

Figure 1.1: Periodic string patterns sorted by descending required periodicity.
and whose successors are distinct.

Even the non-maximal variants of these patterns naturally give rise to a substring of the underlying string and a corresponding period. This is most obvious for the $\delta$-repetition which is defined as a substring that is at least $2 + \delta$ times its minimum period. However, even a pair of repeats gives rise to a period defined by the distance of the two occurrences of its corresponding maximal repeat. The subcadence is a special case because it only requires the periodicity in one subsequence instead of a full substring.

Figure 1.1 provides an example of the non-maximal variants of all periodic string patterns that are analyzed in thesis as well as the repetition, the square and the $\frac{1}{3}$-gapped repeat for completeness. Note that except for the 3-subcadence, each of these patterns naturally gives rise to all patterns below itself.

1.3 Scope of this Thesis

Our most significant original contribution to the field of stringology are tight upper bounds for the number of distinct extended maximal $\delta$-(sub-)repetitions and several subsets of distinct maximal pairs.

We also provide new insights about the relationship between the runs of the run-length encoded Burrows-Wheeler transform and these maximal pairs.

In terms of string cadences, we introduce new algorithms for quasi-linear 3-cadence detection and counting. Along the way, we also present a new variant of the discrete acyclic convolution that extends the underlying geometry from a rectangle to arbitrary polygons and can be calculated in quasi-linear time.

We also present a strongly invertible, uniform and cube-free morphism from an arbitrary alphabet $\Sigma$ to $\{a, b\}$.

Most results have already published in our journal paper or in one of our five conference papers:

- “Tight upper bounds on distinct maximal (sub-)repetitions in highly compressible strings” by Julian Pape-Lange to appear in the International Journal of Foundations of Computer Science (IJFCS) [82]

- “Non-rectangular convolutions and (sub-)cadences with three elements” by Mitsuru Funakoshi and Julian Pape-Lange at the 37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020) [41]
1.4. STRUCTURE OF THIS THESIS

- “Upper bounds on distinct maximal (sub-)repetitions in compressed strings” by Julian Pape-Lange at Developments in Language Theory (DLT 2021) [81]

- “Cadences in grammar-compressed strings” by Julian Pape-Lange at Language and Automata Theory and Applications (LATA 2020 & 2021) [80]

- “On Extensions of Maximal Repeats in Compressed Strings” by Julian Pape-Lange at the 31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020) [78]

- “On Maximal Repeats in Compressed Strings” by Julian Pape-Lange at the 30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019) [77]

We also present some new results.

In this thesis, we prove that no non-trivial bounds for \( k \)-cadences and their variants exist, even if we have the compressed size as additional parameter. We also show that for fixed \( k \geq 3 \) even the \( k \)-subcadence detection with a given character is \( \mathcal{NP} \)-hard. Thereby, we also show that the equidistant subsequence problem is \( \mathcal{NP} \)-hard even for patterns of length 3 and binary strings.

We think that the strongly invertible and uniform morphism from an arbitrary alphabet \( \Sigma \) to \( \{a, b\} \) is of independent interest. Therefore, we improve it to be even cube-free and analyze it in an additional chapter.

The upper bounds in Chapter 7 were dependent on the number of LZ77-factors. These results are now improved to use the size of the string attractor instead.

The algorithm to find all maximal repeats were the main goal of my Master’s thesis [76] and is therefore beyond the scope of this thesis.

An efficient algorithm for reporting all squares and runs of a given compressed string was already found by I et al. in [55] and therefore also beyond the scope of this thesis.

Note that whenever a result by I is stated, i.e. whenever [54] or [55] is cited, “I” is the surname of Tomohiro I and not the author of this thesis.

1.4 Structure of this Thesis

In Chapter 2, we will introduce the notation and important definitions of this thesis. We will also present some papers which show that most complexity measures mentioned in this thesis are of similar size.
In Chapter 3 and Chapter 4, we will present tools that we will use in later chapters but which are also of independent interest. The non-rectangular convolution presented in Chapter 3 extends underlying geometry of the convolution from rectangles to arbitrary convex polygons and can still be computed efficiently. In Chapter 4, we will show that there is a strongly invertible, uniform and cube-free morphism from an arbitrary alphabet to a binary alphabet. This morphism naturally translates maximal $\delta$-repetitions and maximal pairs in the preimage into maximal $\delta$-repetitions and maximal pairs in the image. This morphism is therefore very useful to translate the tightness results from the later chapters into the corresponding theorems on binary strings.

In Chapter 5, we will prove tight upper bounds for the number of distinct extended maximal $\delta$-(sub-)repetitions. The proof of the tightness on binary strings relies on the alphabet reduction from Chapter 4.

In Chapter 6, we will show that there are no non-trivial upper bounds for the number of $k$-subcadences and their variants. We also present some efficient algorithms for the detection, counting and reporting of 3-subcadences and their variants and we will show that all even slightly harder problems are $\text{NP}$-complete on grammar-compressed strings. For the convolution based 3-cadence algorithms, we will need the non-rectangular convolution from Chapter 3.

Lastly, in Chapter 7, we will prove several upper bounds for different subsets of distinct maximal pairs. These bounds also lead to upper bounds for the size of the compacted directed acyclic word graph and the number of runs of the run-length encoded Burrows-Wheeler transform. For the analysis of the distinct non-extendable $\frac{1}{4}$-highly periodic maximal pairs, we use the upper bound for the number of maximal $\delta$-repetitions from Chapter 5. For the tightness of the upper bound of maximal repeats without $q$-th powers in binary string, we will use the alphabet reduction from Chapter 4.
Chapter 2

Preliminaries

2.1 Strings and Their Substructures

A string $S$ of length $|S|$ is the concatenation


of $|S|$ characters of an alphabet $\Sigma$. For a given character $S[i]$ in the string $S$, the predecessor of $S[i]$ is the character $S[i - 1]$ and the successor of $S[i]$ is the character $S[i + 1]$. We define $S[-1] = $ and $S[|S|] = $ with characters $\not\in \Sigma$ so that each character in $S$ has a predecessor and a successor.

2.1.1 Substrings

For $-1 \leq l \leq r \leq |S|$, the substring $S[l..r]$ of $S$ is the concatenation


For $l > r$, the substring $S[l..r]$ is the empty string of length 0. If $l = 0$ holds, the substring $S[l..r]$ is a prefix and if $r = |S| - 1$ holds, the substring $S[l..r]$ is a suffix.

In this thesis, we are interested in both the content of substrings and their relationship to the underlying string. If we are only interested in the characters that make up a given substring, we call this substring an unpositioned substring. If we are also interested in the position of the substring in the underlying string, we call this substring a positioned substring. Formally, a positioned substring is a pair $(l,r)$ of indices and the content of the positioned substring is the substring $S[l..r]$. However, in this thesis, we will slightly abuse the notation and denote the positioned substrings like unpositioned substrings with $S[l..r]$. 
For two strings $S$ and $P$, the positioned substring $S[l..r]$ is an occurrence of $P$, if $S[l..r]$ and $P$ are equal as concatenation of characters. I.e. for each index $k$, the characters $(S[l..r])[k]$ and $P[k]$ are equal. Two occurrences of the same substring $P$ are copies of each other.

(Sub-)strings can be different in two ways. Firstly, two (sub-)strings $S[l..r]$ and $S[l'..r']$ are distinct if they have a different length or if for some index $k$ their $k$-th characters $(S[l..r])[k]$ and $(S[l'..r'])[k]$ are different. Secondly, two positioned substrings $S[l..r]$ and $S[l'..r']$ are different if the starting indices $l$ and $l'$ or if the ending indices $r$ and $r'$ differ. Therefore, two substrings are different if they are either distinct strings or if they are different occurrences of the same string. In particular, the notion of distinct (sub-)strings is stronger than the notion of different substrings.

The intersection of two positioned substrings $S[l..r]$ and $S[l'..r']$ of $S$ is the substring $S[\max(l,l')..\min(r,r')]$ containing the characters which lie in both substrings.

By definition, a substring is allowed to contain the prepended $S_l$ and the appended $S_r$. This allows us to define extensions of substrings which lie entirely in $S$. For $0 \leq l \leq r \leq |S| - 1$ the extended substring of the substring $S[l..r]$ of $S$ is the substring $S[l-1..r+1]$.

Figure 2.1 shows the string $S = \text{abaababaabab}$ with all substrings of length 2. Even though $S_l a$ and $b S_r$ are considered to be substrings of $S$, the prefix and suffix of $S$ with length 2 are still the first $ab$ and the last $ab$, respectively.
The string $S$ has 14 different substrings of length 2 but only 5 distinct substrings of length 2. The two substrings $S[0..1] = S[3..4] = S[5..6] = S[8..9] = S[11..12]$, have only one occurrence each. The substring $ab$ has the occurrences


the substring $ba$ has the occurrences


and the substring $aa$ has the occurrences


### 2.1.2 Subsequences

For a set $M = \{i_1, i_2, i_3, \ldots, i_{|M|}\}$ with $i_1 < i_2 < i_3 < \cdots < i_{|M|}$ and a string $S$ the subsequence $S[M]$ of length $|M|$ is defined by the concatenation

$$S[M] = S[i_1]S[i_2]S[i_3] \ldots S[i_{|M|}].$$

If $M$ is of the form $\{i, i + d, i + 2d, \ldots, i + (k - 1)d\}$, the subsequence $S[M]$ is an arithmetic subsequence with starting index $i$, distance $d$ and length $k$. In order to obtain a unique triple from a given set of indices, we will only consider subsequences with length of at least 2 in this thesis. Therefore, an arithmetic subsequence is given by the triple $(i, d, k)$ with $d \geq 1$ and $k \geq 2$. We can extend arithmetic subsequences to the left which yields the arithmetic subsequence $(i - d, d, k + 1)$ and to the right which yields the arithmetic subsequence $(i, d, k + 1)$. An arithmetic subsequence is maximal if neither of these two extensions is contained in $S$. Therefore, an arithmetic subsequence is maximal if and only if both $i - d < 0$ and $i + kd \geq |S|$ hold.

Similarly to the case of substrings, we differentiate between unpositioned subsequences and positioned subsequences. Therefore, we also have notion of distinct subsequences which differ in the string they represent and different subsequences which differ in their underlying set of indices.

Figure 2.2 shows the string $S = \text{aaaaaa}$ with some of its arithmetic subsequences of length 3. The subsequences $S[\{0, 3, 6\}]$ and $S[\{2, 5, 8\}]$ are maximal since neither of their extensions lies in $S$. The arithmetic subsequences $S[\{2, 4, 6\}]$ is not maximal and can be canonically extended to $S[\{0, 2, 4, 6, 8\}] = \text{aaaa}$. For the sake of brevity, we denote the set of all even/odd numbers of the set $M$ by $M_{\text{even}} := M \cap 2\mathbb{Z}$ and $M_{\text{odd}} := M \cap (2\mathbb{Z} + 1)$, respectively.
2.2 Periodic Patterns

2.2.1 Powers

The simplest periodic pattern is the square. A square is the concatenation \( PP \) of a string \( P \) with itself. Similarly, a cube is the concatenation \( PPP \) of three copies of a string \( P \). This concept can be generalized to arbitrary natural powers. For \( q \in \mathbb{N}_{\geq 0} \), the \( q \)-th power of \( P \) is the concatenation \( P^q \) of \( q \) copies of the same string \( P \). Even more general, for a rational number \( q \in \frac{\mathbb{N}_{\geq 0}}{|P|} \), the \( q \)-th power of \( P \), denoted by \( P^q \), is defined by the concatenation

\[
PP \ldots PP[1..q|P| \mod |P|]
\]

of \( |q| \) copies of \( P \) and the prefix of \( P \) with length \((q - |q|)|P|\).

For example, for the string \( P = ab \), the square of \( P \) is \( P^2 = abab \), the cube of \( P \) is \( P^3 = ababab \) and the \( \frac{5}{2} \)th power of \( P \) is given by the string \( P^2 = P^2P^2 = (abab)a = ababa \).

A string of the form \( P^q \) for an integer \( q \in \mathbb{N}_{\geq 2} \) is a natural power. A string which is not a natural power is primitive and a square \( P^2 \) with a primitive root \( P \) is a primitively rooted square.

Furthermore, for \( q \in \mathbb{N} \), a string \( S \) is \( q \)-th power-free if it has no non-empty substring of the form \( P^q \).
2.2. PERIODIC PATTERNS

2.2.2 Periods

A string $S$ has a period $p$ if all characters in $S$ with distance $p$ are equal. For brevity, we call such a $p$-periodic. The minimum period of $S$ is the minimum of all periods of $S$.

Powers are closely related to periods since a string $S$ is a $q$-th power if and only if $S$ is $\frac{|S|}{q}$-periodic. For example $(ab)\frac{5}{2} = ababa$ is a $\frac{5}{2}$th power and therefore has the period $\frac{|S|}{q} = \frac{5}{2} = 2$.

For calculations, the periods are very useful because the well-known periodicity lemma by Fine and Wilf from [36] shows that sufficiently long strings with two given periods also have their greatest common divisor as period.

**Theorem 2.2.1** (Periodicity Lemma [36]). Let $p_1$ and $p_2$ be two positive natural numbers and $S$ a $p_1$-periodic and $p_2$-periodic string with a length of at least $p_1 + p_2 - \gcd(p_1, p_2)$.

Then $S$ is $\gcd(p_1, p_2)$-periodic.

A string $S$ is $\frac{1}{2q}$-highly-periodic, if it is a power with an exponent of at least $q$. Equivalently, this holds if $S$ has a period of at most $\frac{1}{q}|S|$.

For example, the strings $aaaa = a^4$, $aaaaa = a^5$ as well as the string $ababababa = (ab)^4a = (ab)^4.5$ are $\frac{1}{24}$-highly-periodic, but $aaaaac = a^4c$ and $abababa = (ab)^3.5$ are not $\frac{1}{24}$-highly-periodic.

For a given period $p$ and a $p$-periodic positioned substring $S[l..r]$ with length of at least $p$, the $p$-periodic extension of this occurrence is the largest substring of $S$ that contains $S[l..r]$ and is still $p$-periodic. This extension is given by $S[l'..r']$ such that

- $l' \leq l \leq r \leq r'$,
- $S[l'..r']$ is $p$-periodic,
- $S[l' - 1..r']$ is not $p$-periodic and
- $S[l'..r' + 1]$ is not $p$-periodic.

With this notation, the substring $S[l' - 1..r' + 1]$ is the extended $p$-periodic extension. For brevity, we will call the (extended) $p$-periodic extension of a substring $S[l..r]$ with minimum period $p$ just (extended) periodic extension.

Substrings are maximal with respect to their periodicity, if they are equal to their periodic extension. This concept will be explored in the next subsection. However, it is also very useful to relax the concept of maximality to strings which can be extended by at most one minimum period. This
relaxed maximality gives rise to the following definition of extendability. A substring $S[l..r]$ with minimum period $p$ is extendable if the periodic extension is at least $p + 1$ characters longer than $S[l..r]$.

For example, in Figure 2.3, we have the string $S = \text{ababxababab}$. The substrings $S[0..1] = \text{ab}$ and $S[6..9] = \text{baba}$, each with minimum period 2, are not extendable, since their maximal periodic extensions $S[0..3] = \text{abab}$ and $S[5..10] = \text{ababab}$, respectively are only 2 characters longer. On the other hand, the substring $S[7..8] = \text{ab}$ with minimum period 2 is extendable, since it has the maximal periodic extension $S[5..10] = \text{ababab}$ which is 4 characters longer. We can verify that the extendable substrings of $S$ are exactly the nine 2-periodic substrings of the substring $S[5..10]$ with length less than 4.

### 2.2.3 Maximal (Sub)-Repetitions

For $\delta > 0$, a substring $S[l..r]$ of $S$ with minimum period $p$ is

- a $\delta$-subrepetition, if $\frac{|S[l..r]|}{p} \geq 1 + \delta$ holds,
- a repetition, if $\frac{|S[l..r]|}{p} \geq 2$ holds and
- a $\delta$-repetition, if $\frac{|S[l..r]|}{p} \geq 2 + \delta$ holds.
2.2. PERIODIC PATTERNS

A δ-(sub-)repetition $P$ is maximal if it has an occurrence $S[l..r] = P$ such that both its left extension $S[l - 1..r]$ and its right extension $S[l..r + 1]$ have a greater minimum period. I.e. the minimum period neither extends to the left nor to the right. The extended maximal δ-(sub-)repetition of a positioned maximal δ-(sub-)repetition $S[l..r]$ is the substring $S[l - 1..r + 1]$.

For example, the string $S = \text{ababxababa}$ in Figure 2.4 has the following four maximal $\frac{2}{3}$-subrepetitions. We have two 2-periodic $\frac{2}{3}$-subrepetitions $S[0..3] = \text{abab}$ and $S[5..9] = \text{ababa}$, which are even a repetition as well as a $\frac{1}{2}$-repetition, respectively. We also have the 3-periodic $\frac{2}{3}$-subrepetitions $S[2..6] = \text{abxab}$ and the 4-periodic $\frac{2}{3}$-subrepetitions $S[0..8] = \text{ababxababa}$. The 7-periodic substring $S[0..9] = \text{ababxababa}$ is not a $\frac{2}{3}$-subrepetition, since this string is only a power with exponent $\frac{10}{7}$. Also, for example, $S[6..9] = \text{baba}$ with minimum period 2 is not maximal $\frac{2}{3}$-subrepetition, since it can be periodically extended to the left.

Note that all δ-subrepetitions contain their first character and their last character twice and therefore cannot contain either $S_l$ or $S_r$.

2.2.4 Maximal Pairs and Maximal Repeats

A maximal pair with length $k$ of $S$ is a pair of two copies $S[n..n + k - 1]$ and $S[m..m + k - 1]$ of a substring $P$ of $S$ such that both left extensions $S[n - 1..n + k - 1]$ and $S[m - 1..m + k - 1]$ are distinct and both right extensions $S[n..n + k]$ and $S[m..m + k]$ are distinct. We can describe such a maximal pair with the pair $(n, m)$ of its starting indices.
Note that while Gusfield represents the maximal pairs in [49] by the triple 
\((n, m, k)\) of the two starting indices and the length of the maximal pair, the 
length can be obtained from the starting positions and the underlying string. 
Therefore, we may exclude the length from the representation of the maximal 
pair. Also, to keep the examples simple, we also require \(n < m\). However, the 
proofs in Chapter 7 will not rely on this requirement and therefore will count 
each maximal pair exactly twice.

The distance \(d\) of a maximal pair \((n, m)\) is the distance \(d = |n - m|\) of 
the two starting indices. By construction, for all maximal pairs, this distance 
is a positive integer.

A maximal repeat of a string \(S\) is a substring \(S[n..n + k - 1]\) such that 
there is a maximal pair \((n, m)\) of length \(k\) for some starting indices \(n, m\). 
Conversely, for a maximal pair \((n, m)\) of length \(k\), the two positioned strings 
\(S[n..n + k - 1]\) and \(S[m..m + k - 1]\) are the corresponding maximal repeats 
of this maximal pair.

For example, in the string \(ababxababab\) in Figure 2.3, the substring \(bab\) 
is not a maximal repeat because every occurrence of \(bab\) is preceded by \(a\). The 
substring \(abab\), however, is a maximal repeat with maximal pair given 
by, for example, the pair \((0, 5)\). This maximal pair has length 4 and distance 
5. There are two more maximal pairs for this maximal repeat. There is the 
maximal pair \((0, 7)\) and the maximal pair \((5, 7)\). Note that in the maximal 
pair \((5, 7)\), it is allowed that both occurrences of \(abab\) are overlapping.

For a positioned maximal repeat \(S[n..n + k - 1]\), the right extension of this 
maximal repeat is the substring \(S[n..n + k]\) which is obtained by extending the 
maximal repeat by its successor. Similarly, the extension is \(S[n - 1..n + k]\).

For example, in the string \(ababxababab\) in Figure 2.3, the maximal repeat 
\(S[0..3] = abab\) has the right extension \(S[0..4] = ababx\) and the extension 
\(S[-1..4] = ababx\).

Similarly to the substrings, we define distinct and different maximal pairs. 
However, since we are not only interested in the maximal repeats but also 
in their extensions, we define the distinctness of maximal pairs to take these 
extensions into account. Two maximal pairs \((n_1, m_1)\) and \((n_2, m_2)\) are distinct, 
if the two extensions of the corresponding maximal repeats of \((n_1, m_1)\) are 
equal to the two extensions of the corresponding maximal repeats of \((n_2, m_2)\).

If two maximal pairs are not distinct, they are copies of each other. Note 
that we also do not care about the order of the two strings. Therefore, two 
maximal pairs \((n, m_1)\) and \((n_2, m_2)\) of length \(k\) are also copies of each other if 
\(S[n_1 - 1..n_1 + k] = S[m_2 - 1..m_2 + k]\) and \(S[m_1 - 1..m_1 + k] = S[n_2 - 1..n_2 + k]\) 
hold.

Two maximal pairs \((n_1, m_1)\) and \((n_2, m_2)\) are different, if either \(n_1 \neq n_2\) 
or \(m_1 \neq m_2\) hold.
2.2. PERIODIC PATTERNS

Figure 2.5: The string \( S = \text{abcbabcdb} \) with its extensions of the maximal repeat \( b \).

For example, in the string \( S = \text{abcbabcdb} \) in Figure 2.5, there are 4 different extensions of the maximal repeat \( b \). Of these 4 substrings only 3 are distinct. The string \( S \) has 4 different maximal pairs with the corresponding maximal repeat \( b \). The first two maximal pairs are \((1, 3)\) and \((3, 5)\) whose corresponding maximal repeats have the extensions \( abc \) and \( cba \). The other two maximal pairs are \((1, 7)\) and \((5, 7)\) whose corresponding maximal repeats have the extensions \( abc \) and \( cbd \). Therefore, this string has 2 distinct maximal pairs with the corresponding maximal repeat \( b \).

The periodicity and the extendability of a maximal pair is defined by the corresponding maximal repeats. A maximal pair is \( \geq q \)-highly-periodic if the corresponding maximal repeat is \( \geq q \)-highly-periodic. Also, a maximal pair is extendable, if both occurrences of the corresponding maximal repeat are extendable. Otherwise, it is non-extendable.

For example, in the string \( S = \text{ababxababab} \) in Figure 2.3 the maximal pair \((5, 7)\) with corresponding maximal repeat \( \text{abab} \) has length 4 and minimal period 2. Therefore it is \( \frac{1}{2} \)-highly-periodic but not \( \frac{1}{2q} \)-highly-periodic for any \( q > 2 \).

Since the maximal repeat in \( \text{ab} \) is 2-periodic, the maximal pairs with this corresponding maximal repeat are extendable if and only if both occurrences of the maximal repeat can be periodically extended by at least 3 characters. Therefore, the maximal pair \((5, 9)\) is extendable, while the maximal pairs \((0, 2)\), \((0, 9)\) and \((2, 5)\) are not extendable.
2.2.5 Cadences and Their Variants

Another pattern we will examine in this thesis is the subcadence and its variants. In this thesis, we will use the most restrictive definition of the cadence which was introduced by Amir et al. in [4].

The most general variant of the cadence is the subcadence. A subcadence of a string $S$ is a positioned arithmetic subsequence $(i, d, k)$ in which all characters are equal. This definition of subcades coincides with the definition of cadences by Gardelle and Guilbaud in [45] and arithmetic cadence as defined by Pin in [70, Chapter 3.3].

A subcadence is maximal if neither the extension of the arithmetic subsequence to the left $(i - d, d, k + 1)$ nor the extension of the arithmetic subsequence to the right $(i, d, k + 1)$ forms a subcadence.

A cadence is a subcadence $(i, d, k)$ that also satisfies $i - d < 0$ and $i + kd \geq |S|$. Therefore, a cadence is a structurally maximal subcadence in the sense that it is not only maximal as a subcadence but also maximal as an arithmetic subsequence.

In this thesis, we are mostly interested in (sub-)cadences $(i, d, k)$ with a fixed variable $k$. Therefore, we define these (sub-)cadences for the sake of brevity to be $k$-(sub-)cadences.

For example, in the string $S = a^9 = 	ext{aaaaaaaa}$ in Figure 2.2 all three marked arithmetic subsequences are 3-subcades but only the first two of them are 3-cadences.

It might be tempting to understand $k$-cadences as $k$-subcades which start sufficiently close to the start of the string and end sufficiently close to the end of the string. However, the previous example shows that the allowed indices of the starting index of the $k$-cadence depend on the ending index and vice versa. Therefore, we also study $L$-$R$-$k$-cadences which simplify the $k$-cadences by removing this dependence. For two intervals $L$ and $R$ an $L$-$R$-$k$-cadence is a $k$-subcadence $(i, d, k)$ whose starting index $i$ is contained in $L$ and whose ending index $i + (k - 1)d$ is contained in $R$.

For each 3-subcadence $(i, d, k)$, the first index $i$ has the same parity as the last index $i + 2d$. It will be very useful to consider both possible parities separately. If the first index $i$ of a subcadence is even, then the subcadence is even. Otherwise, the subcadence is odd. These definitions extend to all other variants of the subcadence as well.

In order to analyze the cadences, we will use maximal repetitions with minimum period 1. For the sake of brevity, we will call these maximal repetitions runs and if the character of the run $\sigma^k$ is important, we call them $\sigma$-runs.
2.3 Compressions and Complexity Measures

2.3.1 String Attractors

The most important complexity measure of this thesis is the size of the smallest string attractor which was introduced in 2018 by Kempa and Prezza in [59].

While this complexity measure does not directly correspond to a string compression algorithm, we will see that its size is closely linked to several other complexity measures.

A string attractor of the string $S$ is a subset $\Gamma$ of $\{0, 1, 2, \ldots, |S| - 1\}$ such that for each substring $S[l..r]$ of $S$ with $0 \leq l \leq r \leq |S| - 1$, there is a copy $S[l'..r'] = S[l..r]$ which contains at least one index $i \in \Gamma$.

For example, in the string $S = \text{ababxababab}$ in Figure 2.3, a string attractor is given by $\Gamma = \{4, 9, 10\}$. Each other substring is also a substring of $\text{ababab}$ and therefore has a copy which either ends with $S[9]$ or ends with $S[10]$. Therefore, the size of the smallest attractor is at most 3. On the other hand, in order to contain each substring of size 1, the string attractor has to include at least one copy of each character in the underlying string. Therefore, the size of the smallest string attractor of $S = \text{ababxababab}$ is exactly 3.

The string attractor will allow us to extend local upper bounds on different patterns to global upper bounds on distinct patterns. In particular, we will use the string attractor for all global upper bounds in Chapter 5 and Chapter 7.

Since the extensions of maximal $\delta$-(sub-)repetitions and maximal pairs in $S$ can extend to $S_l$ and $S_r$, we will use more commonly a string attractor of $S_lS_r$. By construction, for each string attractor $\Gamma$ of $S$, the set $\Gamma \cup \{-1, |S|\}$ is a string attractor of $S_lS_r$.

Kempa and Prezza prove in [59] that the smallest string attractor problem is $\mathcal{NP}$-complete. However, we will see in Subsection 2.3.7, that there is a good string attractor which can be calculated fast.

2.3.2 LZ77-Type Compressions

The next complexity measure that is actually a compression method is given by the LZ77-type decomposition. An LZ77-type decomposition of a string $S$ is the factorization $S = F_1F_2\ldots F_z$ into LZ77-factors $F_i$ such that each factor is either

- a single character or
- a substring of $S[0..|F_1F_2\ldots F_i| - 2]$. 
For example the factorization \texttt{banana} = b \cdot a \cdot n \cdot anana is an LZ77-decomposition since the last LZ77-factor \( S[3..7] = \texttt{anana} \) already occurs at \( S[1..5] \). In particular, an LZ77-factor may overlap with the string it is copied from.

LZ77-type decompositions naturally give rise to a compression. If an LZ77-factor is a single character, we simply write it down. Otherwise, we write down the index pair of a previous occurrence of the LZ77-factor. Even though the single characters need less space than the pairs of integers, we only consider the number of the LZ77-factors in a given LZ77-decomposition for the corresponding complexity measure.

Using the decomposition \texttt{banana} = b \cdot a \cdot n \cdot anana from above, we get the compression \( b, a, n, (1, 5) \). If we want to decompress this string, we evaluate the LZ77-factors from left to right. The first three characters are straightforward and yield \( S[0..2] = \texttt{ban} \). The last factor yields that the next 5 characters, which will be the string \( S[3..7] \) are the substring \( S[1..5] \). While we don’t have that substring yet, we do have the first characters of the substring. Therefore, we can use that \( S[3..7] = S[1..5] \) implies \( S[3] = S[1], S[4] = S[2], S[5] = S[3], S[6] = S[4] \) and \( S[7] = S[5] \). These equations then yield \( S[3..7] = \texttt{anana} \) which in turn shows \( S = \texttt{banana} \).

It is well-known, that the greedy algorithm, which always chooses the longest possible LZ77-factor, minimizes the number of LZ77-factors. Therefore the minimal number of LZ77-factors can be efficiently computed.

Let \( S \) be a string with factorization \( S = F_1 F_2 \ldots F_z \) into LZ77-factors. Let \( S[l..r] \) be the first occurrence of a substring of \( S \). By construction of the LZ77-decomposition, the occurrence \( S[l..r] \) cannot be contained in an LZ77-factor with length greater than one. Thus, either \( S[l..r] \) is equal to an LZ-factor of length one or it crosses the borders of LZ77-factors. In either case, it contains a first character of an LZ77-factor. Therefore, each LZ77-type decomposition leads to a string attractor of equal size.

While the upper bounds in Chapter 5 and in Chapter 7 are given in terms of the smallest string attractor, we will use strings with given LZ77-decomposition to prove that these bounds are tight.

### 2.3.3 Grammar-Based Compressions

We can also compress strings by using context-free grammars. A \textit{context-free grammar} is a tuple \( G = (V, \Sigma, S, \text{rhs}) \) of nonterminals \( V = \{v_1, v_2, \ldots, v_{|V|}\} \), an alphabet \( \Sigma \), a start symbol \( S \) and production rules \( \text{rhs} \).

In order to ensure that each grammar only encodes a single string, we restrict the context-free grammars to straight-line grammars. A \textit{straight-line grammar} is a context-free grammar such that for each nonterminal \( v_i \), there is
only one production rule and the nonterminals in the right-hand side \( \text{rhs}(v_i) \) of \( v_i \) have indices strictly less than \( i \). Therefore, straight-line grammars do not have any branches or loops and thereby each nonterminal \( v_i \) encodes a single string \( \text{val}(v_i) \).

A context-free grammar is in Chomsky normal form, if each right-hand side either contains exactly one character or exactly two nonterminals. It is well-known that straight-line grammars can be transformed into an equivalent straight-line grammar in Chomsky normal form in linear time.

The **size of a production rule** \( \text{rhs}(v_i) \) of a nonterminal \( v_i \) is the number of nonterminals and characters of the right-hand side of this rule. The **size of a straight-line grammar** \( G \) of a string \( S \), denoted by \( |G| \), is the sum of the sizes of all production rules of the grammar. It is easy to see that the size of the grammar is at least logarithmic with regard to the length of the uncompressed string.

For example, the string \( S = \text{abaababaabaab} \) can be encoded by the straight-line grammar \( (\{F_0, F_1, \ldots, F_5\}, \{a, b\}, F_5, \text{rhs}) \) with

\[
\begin{align*}
\text{rhs}(F_0) &= a & \text{val}(F_0) &= a \\
\text{rhs}(F_1) &= ab & \text{val}(F_1) &= ab \\
\text{rhs}(F_2) &= F_1 F_0 & \text{val}(F_2) &= aba \\
\text{rhs}(F_3) &= F_2 F_1 & \text{val}(F_3) &= abaab \\
\text{rhs}(F_4) &= F_3 F_2 & \text{val}(F_4) &= abababa \\
\text{rhs}(F_5) &= F_4 F_3 & \text{val}(F_5) &= abababaabaab.
\end{align*}
\]

This straight-line grammar has size

\[
|\text{rhs}(F_0)| + |\text{rhs}(F_1)| + |\text{rhs}(F_2)| + |\text{rhs}(F_3)| + |\text{rhs}(F_4)| + |\text{rhs}(F_5)|
= 1 + 2 + 2 + 2 + 2 + 2 = 11.
\]

Lehman shows in his PhD-thesis [65, Section 3.1] that unless \( \mathcal{P} = \mathcal{NP} \) holds, there is no polynomial-time approximation algorithm for finding a grammar with a size of at most \( \frac{8569}{8568} \) times the size of the smallest grammar. In the paper [19] Charikar et al. this proof can also be found. In [18] Casel et al. prove that even on strings over an alphabet of size 17, it is still \( \mathcal{NP} \)-complete to determine whether a grammar with a given size exists.

While there is no similar result for binary strings and Casel et al. state in [18] that the alphabet size in their approach cannot be substantially decreased, the use of the smallest grammar as a complexity measure is limited. However, small grammars are very useful for algorithms on very repetitive strings.
Lohrey presents in [68] many problems on strings which can be solved on grammar-compressed strings without decompressing the strings.

For example, we can solve fully compressed pattern matching in polynomial time. In fully compressed pattern matching, we have two strings $S$ and $P$ given by their grammars $G_S$ and $G_P$, respectively, and we want a binary string $R$ which signifies the starting indices of $P$ in $S$ with

$$R[i] = 1 \iff S[i..i + |P| - 1] = P.$$ 

There are many algorithms for this problem. The best known algorithm was found by Jeż who proves in [56] that this problem can be solved in $O((|G_S| + |G_P|) \log(|P|))$ time if $|P|$ fits in $O(1)$ machine words.

Given a grammar $G$ for a string $S$, we can also construct a grammar $G'$ for a given substring $S[l..r]$. Hagenah shows in pages 99-104 of [50] (written in German) that each substring of $S$ can be written as concatenation of at most $2|G|$ nonterminals. Since these nonterminals can efficiently be found, it follows that the substring grammar has a size of at most $3|G|$ and can be found in linear time.

Given a character $\sigma \in \Sigma$ and a grammar $G$ for a string $S$, we can calculate for each nonterminal $v_i$ the following values:

- The size of rhs($v_i$),
- the index of the first $\sigma$ in rhs($v_i$) and
- the index of the last $\sigma$ in rhs($v_i$).

By traversing the $v_i$ in descending order, we can calculate these values for all $v_i$ in linear time combined.

Also, given a grammar in Chomsky normal form, we can construct for each nonterminal $v_i$ the nonterminals $v_{i,even}$ and $v_{i,odd}$ which generate the even subsequence and the odd subsequence of rhs($v_i$), respectively. This can be done in linear time by traversing the $v_i$ in descending order.

We will use these algorithms on straight-line programs in Chapter 6 to create detection algorithms for cadences and their variants in grammar-compressed strings.

### 2.3.4 Compacted Directed Acyclic Word Graph

Next, we consider the compacted directed acyclic word graph (CDAWG). The CDAWG is both the compactification of the minimal deterministic automaton that detects all suffixes of the underlying string as well as the minimization
2.3. COMPRESSIONS AND COMPLEXITY MEASURES

of the suffix tree. These two definitions have been proven to be equivalent by Crochemore and Vérin in [28].

The CDAWG of a string is a useful data structure which was introduced by Blumer et al. in [13] and has most advantages of suffix trees and acyclic directed word graphs while usually being much smaller than each of them. The CDAWG is therefore a powerful tool for string processing.

One might hope that well-compressible strings have highly structured suffix trees and thereby small CDAWGs. This, however, is unfortunately not the case. Even the arguably best compressible string, $a^{q-1}$, which does have the simple looking suffix tree shown in Figure 2.6, also has the CDAWG shown in Figure 2.7 with $q-2$ internal nodes.

In Chapter 7, we will present an upper bound for the number of arcs in the CDAWG of a string $S$ that polynomially depends on the smallest string attractor $S$, the highest power in $S$ and the logarithm of $S$.

2.3.5 RLBWT

The next complexity measure and last compression method of this thesis is derived from the Burrows-Wheeler transform.

For the Burrows-Wheeler transform, we need the the alphabet $\Sigma$ to be sorted. We also define $\sigma := \sigma_l = \sigma_r < \sigma$ for all $\sigma \in \Sigma$. A string $S$ is lexicographically smaller than a string $S'$ if $S$ is either a prefix of $S'$ or if for the smallest $k$ with $S[k] \neq S'[k]$ the inequality $S[k] < S'[k]$ holds.
Let $\pi_i \in \{0, 1, 2, \ldots, |S|\}$ be given by the lexicographic order of the cyclic rotations $S[\pi_i..|S|]S[0..\pi_i - 1]$ of $S$. The Burrows-Wheeler transform defined in [16] is given by the last characters of those strings, and, since $S[-1] = S = S[|S|]$ hold by definition, these characters are given by $S[\pi_i - 1]$.

The Burrows-Wheeler transform has several interesting properties:

Firstly, repeating substrings tend to get transformed to runs, i.e. maximal repetitions with period 1. This allows the Burrows-Wheeler transformed string to be compressed more easily.

Secondly and quite surprisingly, the Burrows-Wheeler transform is invertible. Therefore, we can recover the original string from the compressed Burrows-Wheeler transformed string.

Also, the Burrows-Wheeler transform gives rise to several text indices like the FM-index. Gagie et al. prove in [44] that the Burrows-Wheeler transform thereby leads to a time-efficient and space-efficient pattern matching algorithm for uncompressed patterns.

Since the compressible Burrows-Wheeler transformed strings nicely decompose into runs and we can represent each run by the underlying character and its length, the Burrows-Wheeler transform naturally gives rise to the run-length encoded Burrows-Wheeler transform (RLBWT).

For example, in the string $S\$_r = \text{abaababaabab}_r$, the cyclic rotations are given by Figure 2.8. In Figure 2.9, these cyclic rotations are lexicographically sorted. Thereby, this figure shows that the Burrows-Wheeler transform of $S\$_r is given by bbbbaab$_r, aaaaa$. This string can easily compressed with run-length encoding leaving (b, 4), (a, 2), (b, 1), (_r, 1), (a, 6).

In Chapter 7, we will present an upper bound for the number of runs in the RLBWT of a string $S$ that polynomially depends on the smallest string attractor $S$ and the logarithm of $S$.

### 2.3.6 Substring Complexity

The last and newest complexity measure used in this thesis is the substring complexity which was introduced by Christiansen et al. in [20] in 2021.

The substring complexity given by $\delta = \max_{m=1}^{|S|} \frac{1}{m}|S_m|$ where $|S_m|$ is the number of substrings of $S$ with length $m$.

Christiansen et al. prove in [20] that the substring complexity can, unlike the size of the smallest string attractor and the size of the smallest grammar, be efficiently computed in linear time. In the next subsection, we will also see that the substring complexity is good at approximating several other complexity measures.
Figure 2.8: The string $S = \text{abaababaabaab}$ with its unsorted cyclic rotations.
Figure 2.9: The string $S = \text{abaababaabaab}$ with its lexicographically sorted cyclic rotations.
2.3. COMPRESSIONS AND COMPLEXITY MEASURES

2.3.7 Relationships Between the Complexity Measures

The number of LZ77-factors proved itself to be a very useful indicator of the complexity of strings in the past. For example, Charikar et al. proved in [19] that even if we do not allow self-referential LZ77-factors, the number of these non-self-referential LZ77-factors is a lower bound for the smallest grammar compression. Conversely, Charikar et al. in [19] and Rytter in [89] have proven independently that the size of the smallest grammar and the minimal number of non-self-referential LZ77-factors differ by a logarithmic factor only.

The self-referential version of LZ77 was used for example by Tanimura in [95] in order to show that the size of the $t$-truncated suffix tree is bounded by $t$ times the number of LZ77-factors.

Also, it is obvious that the minimal number of non-self-referential LZ77-factors is bounded from below by the minimal number of self-referential LZ77-factors. Furthermore, each self-referential LZ77-factor can be replaced by at most $\log(|S|)$ non-self-referential factors.

However, compression schemes like LZ77, grammar-compression, the RLBWT and similar schemes which are not covered in this thesis like bidirectional macro schemes, run-length grammars and collage systems “just” measure the compressibility with regard to themselves. Therefore, we want a more definitive measure of repetitiveness.

In [59], Kempa and Prezza introduce the smallest string attractor. The underlying idea of this measure is that it considers all distinct substrings of a given string with respect to the given string. Kempa and Prezza therefore suggest that it captures the complexity of the set of distinct substrings.

Kempa and Prezza prove the size of the smallest string attractor is, up to a constant factor, a lower bound for the size of the aforementioned compression schemes in this subsection. On the other hand, they also prove that except for the number of runs in the RLBWT, all these compression schemes are in $O(|\Gamma| (\log |S|)^2)$ for strings $S$ with string attractor $\Gamma$. A corresponding result for the number of runs in the RLBWT was subsequently shown by Kempa and Kociumaka in [59].

These results also show that, up to some $(\log |S|)$-factors, all complexity measures in this subsection are similar.

Unfortunately, it is $\mathcal{NP}$-hard to determine the size of the smallest string attractor.

Therefore, Christiansen et al. present in [20] the substring complexity as new complexity measure. This measure can be calculated in linear time and is, as proven by Kociumaka et al. in [60], generally even smaller than the string attractor. Nevertheless, the aforementioned complexity measures
cannot be much greater than the substring complexity.

The last paragraph of the following theorem is proven by Kempa and
Kociumaka in [58] and the remainder is proven by Kociumaka et al. in [60].

**Theorem 2.3.1.** Let $S$ be a string with substring complexity $\delta$.
Then, the size of the following complexity measures is in $O\left(\delta \log \frac{|S|}{\delta}\right)$:

- The smallest string attractor of $S$,
- the minimal number of LZ77-factors of $S$,
- the smallest bidirectional macro schemes of $S$ and
- the smallest collage system of $S$.

Also, the size of the smallest grammar of $S$ and the size of the smallest
run-length grammar of $S$ are in $O\left(\delta \left(\log \frac{|S|}{\delta}\right)^2\right)$.

Furthermore, the number of runs in the RLBWT is in

$$O\left(\delta (\log \delta) \max \left(1, \log \frac{|S|}{\delta (\log \delta)}\right)\right).$$

Since all these measures are at least as large as the substring complexity,
we can thereby retrieve relationships between all these complexity measures.

Since the CDAWG of the well-compressible string $a^n$ has size $\Theta(n)$, it
has no nice relationship to the other complexity measures. However, we will
show in Chapter 7 that it can be bounded in strings without high powers.
Conversely, Belazzougui et al. show in [10] that the CDAWG is at least as
large as the minimum number of self-referential LZ77-factors.

## 2.4 Morphisms

A **morphism** is a function $\varphi : \Sigma^* \to \Sigma'^*$ which is given by its function values
for single characters and $\varphi(SS') = \varphi(S)\varphi(S')$.

A morphism $\varphi : \Sigma^* \to \Sigma'^*$ is **uniform** if there is a natural number $k$ such
that for each $\sigma \in \Sigma$ the equation $|\varphi(\sigma)| = k$ holds. If $k$ is relevant, we also
write $k$-uniform.

A morphism is **invertible** unless there are strings $S \neq S'$ with $\varphi(S) = \varphi(S')$.
If a morphism is invertible, we can retrieve the preimage of each image of the
morphism. Since we are dealing with substrings, we want a slightly stronger
definition of invertibility. Note that for a given string $S$, each substring
of $\varphi(S)$ is of the form $(\text{suf}(\varphi(\sigma_l)))\varphi(P)(\text{pre}(\varphi(\sigma_r)))$ or a substring of some
2.5 Model of Computation

\( \varphi(\sigma_i) \), where \( \text{suf}(\varphi(\sigma_i)) \) is a proper suffix of the image of a character from \( \Sigma \), \( \text{pre}(\varphi(\sigma_i)) \) is a proper prefix of the image of a character from \( \Sigma \) and \( P \) is a (possibly empty) substring of \( S \). We call a morphism strongly invertible if for each substring of each image, we can retrieve the corresponding string \( P \).

For example, the morphism \( \varphi : \{a, b, c\} \rightarrow \{0, 1\} \) given by \( \varphi(a) = 00 \), \( \varphi(b) = 01 \) and \( \varphi(c) = 10 \) is 2-uniform and invertible since no image of a character is a prefix of the image of another character. However, it is not strongly invertible, because, for example, the string 0101 can be both the image of bb and a substring of \( \varphi(ccc) = 101010 \). Thus, we cannot determine the “original” preimage of 0101.

On the other hand, the morphism \( \varphi : \{a, b, c\} \rightarrow \{0, 1\} \) given by the images \( \varphi(a) = 010 \), \( \varphi(b) = 0110 \) and \( \varphi(c) = 01110 \) is not uniform. It is, however, strongly invertible since we can retrieve the preimage of each substring of the image. For example, in the substring \( T = 1001100111001 \), the first 0 belongs to an incomplete character, since no image of a character contains two adjacent 0s. Similarly, the last 0 belongs to an incomplete character. Now, we are left with \( T' = 011001110 \) which can only occur as the image of bc.

We will see in Chapter 5 and Chapter 7 that strongly invertible, uniform morphism do not decrease the number of maximal \( \delta \)-repetitions and maximal repeats.

We also need that the images of the used morphisms do not contain larger powers than their preimages.

A morphism is \( k \)-power free if it maps each string without \( k \)-th powers to a string without \( k \)-th powers. For \( k = 2 \), the morphism is square-free and for \( k = 3 \), the morphism is cube-free.

Note that it is not obvious, and for \( k = 2 \) not even true, that \( k \)-power free morphisms are \((k + 1)\)-power free. In [9], Bean et al. present a square-free morphism that is not cube-free. However, for uniform morphisms and \( k \geq 3 \), Wlazinski prove in [99] that \( k \)-power free morphism are also \((k + 1)\)-power free.

2.5 Model of Computation

All time complexities in this thesis are based on the word RAM model.

In particular, we assume that for each input string \( S \) over \( \Sigma \), we can do execute arithmetic operations with \( \mathcal{O}(\log |S|) \) bits in constant time. We also assume that each character of the alphabet is represented by \( \mathcal{O}(\log |S|) \) bits.

The first assumption allows us to get the remainder of a division by a prime \( p < 2(4|S|\log(4|S|))^2 \) in constant time. Bach and Sorenson prove in [7]
that if the generalized Riemann hypothesis holds, then for each $n$, there is a prime $p < 2(n \log(n))^2$ with $1 = p \mod n$. Agarwal and Burrus prove in [2] that if $n$ is a power of 2, we can efficiently calculate the cyclic convolution modulo $p$ of two vectors of length $n$. Ankeny also proves in [6] that if the generalized Riemann hypothesis holds, we can efficiently find a suitable root for this calculation. Therefore, with a given suitable prime, we can calculate the discrete acyclic convolution of two integer vectors in $O(|S| \log |S|)$ time.

In [41], Funakoshi and the author of this thesis show that if the generalized Riemann hypothesis holds, we can find such a suitable prime in $O\left(|S| (\log |S|)^2 (\log \log |S|)\right)$ time. They also summarize how the modern multiplication algorithms which rely on the fast convolution solve or circumvent the problem of finding a suitable prime for the fast number theoretic transform.

Since we assume that each character in $\Sigma$ is represented by $O(\log |S|)$ bits, we can access each character of an uncompressed string in constant time. Also, there is a natural order of the alphabet and we can compare two characters in constant time.

We call an algorithm linear/polynomial time algorithm if the necessary time for that algorithm is bounded by a linear/polynomial function with respect to its input’s length. Note that for grammar compressed strings the size of the input is at least logarithmic with respect to the length of the underlying grammar.
Chapter 3

Non-Rectangular Convolution

In this chapter, we will present the non-rectangular convolution. This generalization of the discrete acyclic convolution was published by Funakoshi and the author of this thesis in [41] at the 37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).

In particular, we will show that for a given convex polygon $P$ and two given sequences $a = (a_0, a_1, a_2, \ldots, a_n)$ and $b = (b_0, b_1, b_2, \ldots, b_n)$ we can calculate the sequence $c = (c_0, c_1, c_2, \ldots, c_{2n})$ with

$$c_k = \sum_{i+j=k \atop (i,j) \in P \cap \mathbb{Z}^2} a_i b_j$$

in $O(k + p(\log p)^2 \log k)$ time, where $k$ is the number of vertices of $P$ and $p$ is the perimeter of $P$.

This is a generalization of the discrete acyclic convolution which is a function that takes two sequences $a = (a_0, a_1, a_2, \ldots, a_n)$ and $b = (b_0, b_1, b_2, \ldots, b_n)$ and returns the sequence $c = (c_0, c_1, c_2, \ldots, c_{2n})$ with

$$c_k = \sum_{i+j=k} a_i b_j.$$

The convolution has many applications like the multiplication of polynomials (see [23, p. 905]) and large numbers (e.g. [92] (written in German) and [51]) in quasi-linear time. It is also used in signal processing for blurring, sharpening and edge detection in images. Furthermore, the convolution forms the basis of more elaborate algorithms like the conversion of images taken in daylight into night scenes presented in [96], feature detection as presented in [67] and convolutional neural networks as used in [1].

It is well known that the discrete acyclic convolution can be computed in $O(n \log n)$ operations using either the fast Fourier transform or similar transformation algorithms as the number theoretic transform. In [41], Funakoshi
and the author of this thesis provide an algorithm to efficiently find a suitable prime number for the number theoretic transform under the assumption that the generalized Riemann hypothesis is true.

The algorithms in this chapter can easily be translated to other homomorphisms that take sequences as inputs and that preserve additions, subtractions and translations of the inputs. However, since we will only use the convolution in this thesis, other functions are beyond the scope of this thesis. In particular, we will use the convex convolution to detect and count 3-cadences in Chapter 6.

Since we only consider discrete acyclic convolutions in this thesis, we will omit the words “discrete” and “acyclic” in this chapter.

3.1 Rectangular Convolution

In this section, we will show that if the underlying polygon is an axis-aligned rectangle, we can compute the corresponding convolution by a single convolution.

By construction, the convolution can be interpreted geometrically: Let $a = (a_0, a_1, a_2, \ldots, a_n)$ and $b = (b_0, b_1, b_2, \ldots, b_n)$ be two sequences. Then, the convolution calculates the partial sums

$$
\sum_{i+j=k \atop (i,j) \in P \cap \mathbb{Z}^2} a_ib_j,
$$

where $P$ is the square given by $\{(x, y) : 0 \leq x, y \leq n\}$.

Now consider the integers $0 \leq x_l \leq x_u \leq n$ and $0 \leq y_l \leq y_u \leq n$ and the corresponding axis-aligned rectangle

$$
P = \{(x, y) : x_l \leq x \leq x_u \wedge y_l \leq y \leq y_u\}.
$$

We define the shortened and translated sequences $a'$ and $b'$ by

$$
a'_i = a_{x_l+i} \quad \text{for } 0 \leq i \leq x_u - x_l \quad \text{and}
$$

$$
b'_j = b_{y_l+j} \quad \text{for } 0 \leq j \leq y_u - y_l
$$

and pad the shorter sequence with additional zeros at the end. We can now define the convolution $c'$ of $a'$ and $b'$. We then translate $c'$ to obtain

$$
c_k' = \begin{cases} 
c'_{k-(x_l+y_l)} & \text{if } x_l + y_l \leq k \leq x_u + y_u \\
0 & \text{otherwise.}
\end{cases}
$$
3.2. TRIANGULAR CONVOLUTION

With this definition, we get for $x_l + y_l \leq k \leq x_u + y_u$

$$c_k = c'_{k-(x_l+y_l)} = \sum_{(i-x_l)+(j-y_l) = k-(x_l+y_l)} a'_{i-x_l} \cdot b'_{j-y_l} = \sum_{i+j=k} a_i b_j$$

and otherwise

$$c_k = 0 = \sum_{i+j=k} a_i b_j$$

In either case, we can compute the rectangular convolution of an axis aligned rectangle with perimeter $p = 2((x_u - x_l) + (y_u - y_l))$ by a single convolution in $O(p \log p)$ time.

### 3.2 Triangular Convolution

In this section, we will extend the convolution to arbitrary triangles with perimeter $p$ and show that the corresponding convolution can be done in $O(p \log^2 p)$ time.

Firstly, we consider right-angled triangles with a vertical cathetus and a horizontal cathetus. We will trisect the right-angled triangles as shown in Figure 3.1 into two similar triangles with half the height and width of the original triangle and a rectangle. The triangles are handled recursively while the rectangle correspond to a normal convolution which can be computed as shown in the last section. This will then lead to the following lemma.

**Lemma 3.2.1.** Let $P$ be a right-angled triangle with a vertical cathetus and a horizontal cathetus and perimeter $p$. Let $a = (a_0, a_1, a_2, \ldots, a_n)$ and $b = (b_0, b_1, b_2, \ldots, b_n)$ be two sequences.
Then, the partial sums
\[ c_k = \sum_{i+j=k \atop (i,j) \in P \cap \mathbb{Z}^2} a_i b_j \]
can be calculated in \( O(p(\log p)^2) \) time.

Proof. For simplicity, we assume that \( P \) is oriented as in Figure 3.1. By symmetry, the proof can easily be adapted to the other three orientations.

Since the only non-zero values of the output sequence have indices between \( x_l + y_l \) and \( x_u + y_u \), it is sufficient to return the corresponding subsequence. Hence, we initialize that subsequence \( c = (c_{x_l+y_l}, c_{x_l+y_l+1}, c_{x_l+y_l+2}, \ldots, c_{x_u+y_u}) \) with zeros in \( O(p) \) time.

We assume that both catheti are included in the triangle and that the hypotenuse and its endpoints are not included. If this is not the expected behavior for any edge of the triangle, we can traverse this edge in \( O(p) \) time and for each integer point \((i,j)\) on the edge, we can decrease or increase the corresponding \( c_{i+j} \) by \( a_i b_j \) if necessary.

If \( p \leq 1 \) holds, the triangle \( P \) contains at most one integer point \((i,j)\). If such a point exists, we only have to return \( c_{i+j} = a_i b_j \). Otherwise, we return 0.

If \( p > 1 \) holds, we will separate the triangle \( P \) into the three disjoint parts that can be seen in Figure 3.1.

- The triangle \( P' \) of points with x-coordinate of at least \( \lceil \frac{x_l+x_u}{2} \rceil \),
- the triangle \( P'' \) of points with y-coordinate of at least \( \lceil \frac{y_l+y_u}{2} \rceil \) and
- the green rectangle of points with x-coordinate of at most \( \lceil \frac{x_l+x_u}{2} \rceil - 1 \) and y-coordinate of at most \( \lceil \frac{y_l+y_u}{2} \rceil - 1 \).

There are no integers strictly between \( \lceil \frac{x_l+x_u}{2} \rceil - 1 \) and \( \lceil \frac{x_l+x_u}{2} \rceil \) nor strictly between \( \lfloor \frac{y_l+y_u}{2} \rfloor \) and \( \lfloor \frac{y_l+y_u}{2} \rfloor \). Therefore, each integer point in \( P \) is in exactly one of the three parts.

The partial sums corresponding to the green rectangle can be computed with a single convolution in \( O(p \log p) \) time. The partial sums corresponding to \( P' \) and \( P'' \) are computed recursively. For each \( i \) with \( x_l + y_l \leq i \leq x_u + y_u \), the final value \( c_i \) is then given by the sum of the three partial values for \( c_i \). For the whole sequence, these sums can be calculated in \( O(p) \) time.

For each depth \( d \), we have \( 2^d \) triangles with perimeter \( \frac{p}{2^d} \). Each of them requires \( O \left( \frac{p}{2^d} \log \frac{p}{2^d} \right) \) additional time for the initialization of \( c \), the convolution
corresponding to the rectangle and the summation of the final return value. Hence, the algorithm takes

$$
\sum_{d=0}^{\log_2 p} 2^d \cdot O \left( \frac{p}{2^d} \log \frac{p}{2^d} \right) + O \left( \frac{p \log p}{\log \log p} \right) \subseteq O \left( \sum_{d=0}^{\log p} p \log p \right) = O \left( p (\log p)^2 \right)
$$
time.

We will now further extend this result to arbitrary triangles.

**Lemma 3.2.2.** Let $P$ be a triangle with perimeter $p$. Let $a = (a_0, a_1, \ldots, a_n)$ and $b = (b_0, b_1, \ldots, b_n)$ be two sequences. Then, the partial sums

$$
c_k = \sum_{i+j=k, (i,j) \in P \cap \mathbb{Z}^2} a_i b_j
$$
can be calculated in $O \left( p (\log p)^2 \right)$ time.

**Proof.** Similarly to the proof of Lemma 3.2.1, we define $x_l, y_l, x_u, y_u$ as the minimal and maximal x-coordinates and y-coordinates of the three vertices of the polygon $P$, respectively. Also, we first initialize the output vector $c = (c_{x_l+y_l}, c_{x_l+y_l+1}, c_{x_l+y_l+2}, \ldots, c_{x_u+y_u})$ with zeros.

Since we can remove/add edges and vertices in linear time by traversing them and finding all integer coordinates, we will ignore the edges and vertices for the sake of simplicity.

We consider the rectangle $R = \{(x, y)|x_l < x < x_u \land y_l < y < y_u\}$. Our goal is to compute the triangular convolution of $P$ as the difference of the rectangular convolution of $R$ and some other rectangular convolutions and triangular convolutions with right-angled triangles.
By construction, each edge of $R$ contains at least one vertex of $P$. Since $R$ has four edges but $P$ only has three vertices, at least one of the vertices of $P$ is also a vertex of $R$. Without loss of generality, this vertex is $(x_l, y_l)$.

We now consider the two remaining edges in $R$ which correspond to $x_u$ and $y_u$. We also consider the two remaining vertices of $P$.

If both vertices of $P$ lie on the edges of $R$, the underlying geometry of $P$ and $R$ looks similar to the geometry depicted on the left-hand side of Figure 3.2. In this case, the rectangle $R$ is the disjoint union of $P$ and three (possibly degenerated) right-angled triangles. Therefore, we can compute the partial sums for the triangular convolution of $P$ by the difference of the convolution of $R$ and the convolutions of the three triangles. Therefore, the convolution of $P$ can be computed in $O(p(\log p)^2)$ time.

Otherwise, one of the remaining vertices of $P$ has to coincide with the intersection of the two remaining edges of $R$. Therefore, the underlying geometry of $P$ and $R$ looks similar to the geometry depicted on the right-hand side of Figure 3.2. In this case, the rectangle $R$ is the disjoint union of $P$, three right-angled triangles and a smaller rectangle. Therefore, we can compute the partial sums for the triangular convolution of $P$ by the difference of the convolution of $R$ and the four convolutions corresponding to the red areas in the figure. Therefore, the convolution of $P$ can be computed in $O(p(\log p)^2)$ time.

Since both cases require $O(p(\log p)^2)$ time, this concludes the proof.

3.3 Convex Convolution

In this section, we will extend the convolution further to arbitrary convex polygons with $k$ vertices and perimeter $p$ and show that this convolution can be done in $O(k + p(\log p)^2 \log k)$ time.

In order to do this, we will dissect $P$ into $k - 2$ triangles by adding $k - 3$ chords. Since the time of the triangular convolution depends on the perimeter, we want the additional chords to be short. However, we also want a fast triangulation. We will show that we can find a triangulation which only increases the perimeter by the factor $O(\log k)$ in $O(k)$ time.

**Lemma 3.3.1.** Let $P$ be a convex polygon with $k$ vertices and perimeter $p$. Let $a = (a_0, a_1, a_2, \ldots, a_n)$ and $b = (b_0, b_1, b_2, \ldots, b_n)$ be two sequences.

Then, the partial sums

$$c_k = \sum_{i+j=k} a_i b_j$$

for $(i,j) \in P \cap \mathbb{Z}^2$. 

3.3. CONVEX CONVOLUTION

Figure 3.3: Two possible convex polygons $P$ with more than 3 vertices in Lemma 3.3.1.

can be calculated in $\mathcal{O}(k + p(\log p)^2 \log k)$ time.

Proof. If $P$ is a triangle, this lemma is equivalent to Lemma 3.2.2. Therefore, we assume that $P$ has at least 4 vertices.

As in the lemmata of the last section, we define $x_l, y_l, x_u, y_u$ to be the minimal and maximal x-coordinates and y-coordinates of the $k$ vertices of $P$.

Since we can add edges and vertices in linear time, by traversing them and finding all integer coordinates, we assume without loss of generality that none of the edges and vertices of $P$ is included in $P$.

If $P$ is a quadrilateral, we add one of the diagonals to partition $P$ into two triangles as shown on the left-hand side of Figure 3.3. Note that the diagonal has to be included in exactly one of the two resulting triangles. By triangle inequality, both triangles have a perimeter of at most $p$. Therefore, the partial sums corresponding to $P$ can be computed in $\mathcal{O}(p(\log p)^2)$ time.

Otherwise, we label the vertices of $P$ counterclockwise $V_1, V_2, \ldots, V_k$. We add the chords $V_1V_3, V_3V_5, \ldots, V_{2\left\lceil \frac{k}{4} \right\rceil - 3}V_{2\left\lceil \frac{k}{4} \right\rceil - 1}$ as shown on the right-hand side of Figure 3.3, and, if $k$ is even, $V_{k-1}V_1$.

Using the triangle inequality, these new chords have a combined length of at most $p$. Therefore, the resulting triangles have a combined perimeter of $\sum p_i \leq 2p$. The inequality

$$\sum \mathcal{O} \left( \max \left( (1, p_i(\log p_i)^2) \right) \right) \leq \mathcal{O} \left( k + \sum \left( p_i(\log p)^2 \right) \right) \leq \mathcal{O} \left( k + p(\log p)^2 \right)$$

therefore shows that we can compute the partial sums corresponding to the triangles in $\mathcal{O}(k + p(\log p)^2)$ time.

This procedure is repeated, until the inner polygon is either a triangle or a quadrilateral. Since each step almost halves the number of vertices, this is achieved in $\mathcal{O}(\log k)$ steps.
CHAPTER 3. NON-RECTANGULAR CONVOLUTION

Figure 3.4: A non-convex polygon with the chords that would arise from the algorithm in Lemma 3.3.1.

Taking the sum over all steps, this leads to a total time complexity of

$$\sum_{i=0}^{O(\log k)} O \left( \left( \frac{k}{2^i} + 1 \right) + p(\log p)^2 \right) \subseteq O \left( k + p(\log p)^2 \log k \right)$$

for the convolution of $P$.

3.4 Non-Convex Convolution

Lastly, we will use a result of Levcopoulos and Lingas in [66] to extend the convex convolution to non-convex polygons. In particular, we will show in this section, that for a given simple polygon with $k$ vertices and perimeter $p$, we can calculate its convolution in $O \left( (k \log k) + p(\log p)^2 \log k \right)$ time.

The na"ive application of the algorithm in Lemma 3.3.1 on non-convex polygons can result in non-simple polygons as shown in Figure 3.4. This makes it very difficult to adapt this algorithm to non-convex polygons directly.

However, in [66], Levcopoulos and Lingas present an algorithm to partition arbitrary simple polygons into convex polygons. They also prove that this algorithm increases the perimeter only by factor $O(\log k)$ and the number of vertices by a constant factor. Furthermore, their algorithm only takes $O(k \log k)$ time.

**Theorem 3.4.1.** Let $P$ be a polygon with $k$ vertices and perimeter $p$. Let $a = (a_0, a_1, a_2, \ldots, a_n)$ and $b = (b_0, b_1, b_2, \ldots, b_n)$ be two sequences.

Then, the partial sums

$$c_k = \sum_{i+j=k \atop (i,j) \in P \cap \mathbb{Z}^2} a_i b_j$$

can be calculated in $O \left( (k \log k) + p(\log p)^2 (\log k)^2 \right)$ time.
3.4. NON-CONVEX CONVOLUTION

Proof. Consider the partition of $P$ into the convex polygons $P_i$ given by the algorithm in [66]. For each $i$, let $p_i$ be the perimeter of $P_i$ and $k_i$ the number of vertices of $P_i$.

The algorithm guarantees, that the sum $\sum p_i$ of all perimeters is bounded by $O(p \log k)$ and the sum $\sum k_i$ of all vertices is bounded by $O(k)$. Therefore, the time for the corresponding convex convolutions is given by

$$\sum O\left(k_i + p_i (\log p_i)^2 \log k_i\right)$$
$$\subseteq O\left(\left(\sum k_i\right) + \left(\sum (p_i (\log p_i))^2 \log (k_i)\right)\right)$$
$$\subseteq O\left(\left(\sum k_i\right) + \left(\sum p_i \left(\log \left(\sum p_i\right)\right)^2 \log \left(\sum k_i\right)\right)\right)$$
$$\subseteq O\left(\left(\sum k_i\right) + \left(\sum p_i \left(\log \left(\sum p_i\right)\right)^2 \log \left(\sum k_i\right)\right)\right)$$
$$\subseteq O\left(k + p ((\log p) + (\log \log k))^2 (\log k)\right)$$
$$\subseteq O\left(k + p (\log p)^2 (\log k)^2\right)$$

Since the partition algorithm needs $O(k \log k)$ time, we can compute the convolution of $P$ in $O((k \log k) + p(\log p)^2(\log k)^2)$ time. 

This theorem proves that we can restrict the convolution to the integer pairs given by arbitrary simple polygons. Furthermore, this restricted convolution is still efficiently computable in quasi-linear time.

In Chapter 6, we use this non-rectangular convolution to count 3-cadences.
Chapter 4

Alphabet Reduction

In this chapter, we will prove that there is a strongly invertible, uniform, cube-free morphism from an arbitrary alphabet to a binary alphabet.

We will see in Chapter 5 and Chapter 7 that with these morphisms, distinct maximal repetitions and distinct maximal pairs in arbitrary strings naturally translate into distinct maximal repetitions and distinct maximal pairs in the corresponding binary strings. Thereby, we will show that the upper bounds stated in these chapters are, up to a constant factor, tight, even if the underlying alphabet only contains two characters.

The first morphism is already published by the author of this thesis in [82] in a special issue of the International Journal of Foundations of Computer Science (IJFCS). However, in this thesis, we will also prove that the morphism is cube-free.

The second morphism is different than the corresponding morphism in the aforementioned article, since that morphism was not cube-free. The second morphism is derived from the morphism presented by the paper [86] of Rampersad et al. who use a similar morphism to show that there are arbitrarily long cube-free strings without long squares. However, since their prerequisites are different from the prerequisites used in this chapter, we do our own analysis of the corresponding morphism.

We will also prove that the two morphisms increase both the logarithm of the length of the underlying string and its number of LZ77-factors only by a constant factor.

Furthermore, if the string contains each character of the underlying alphabet at least once, the two implicit constants do not depend on the underlying alphabet.

Firstly, we reduce arbitrary alphabets to ternary alphabets. However, before we can reduce the alphabet size, we have to introduce the Prouhet-Thue-Morse words.
Definition 4.0.1. The Prouhet-Thue-Morse words are the words $T_i$ defined by the following equations.

$$
\begin{align*}
T_0 &= 0 \\
\overline{T}_0 &= 1 \\
T_1 &= T_0\overline{T}_0 = 01 \\
\overline{T}_1 &= \overline{T}_0T_0 = 10 \\
&\vdots \\
T_i &= T_{i-1}\overline{T}_{i-1} \\
\overline{T}_i &= \overline{T}_{i-1}T_{i-1}.
\end{align*}
$$

Note that the size of $T_i$ is twice the size of $T_{i-1}$ and that this definition provides a grammar whose size is linear in $i$. In particular, its compressed size is logarithmic with respect to the uncompressed size.

Lemma 4.0.2. Let $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_{|\Sigma|}\}$ be an alphabet. Define the alphabet $\Sigma_3 = \{0, 1, 2\}$ and let $T$ be the Prouhet-Thue-Morse word of length $8 \cdot 2^{\lceil \log(|\Sigma| + 1) \rceil}$.

Consider the morphism $\varphi : \Sigma^* \to \Sigma_3^*$ defined for all characters $\sigma_i \in \Sigma$ and indices $0 \leq j < |T| + 2$ by:

$$
\varphi(\sigma_i)[j] = \begin{cases} 
2 & \text{if } j \text{ is the index of the } (i+1)-\text{th } 0 \text{ in } T \\
2 & \text{if } j \text{ is the index of the } (i+1)-\text{th}-\text{last } 0 \text{ in } T \\
2 & \text{if } j \geq T \text{ holds} \\
T[j] & \text{otherwise}.
\end{cases}
$$

Then, $\varphi$ is a strongly invertible, $(|T| + 2)$-uniform, cube-free morphism.

Proof. By construction, the morphism $\varphi$ maps each character of $\Sigma$ to a string consisting of exactly $|T| + 2$ characters of $\Sigma_3$. Thus, the morphism $\varphi$ is $(|T| + 2)$-uniform.

Note that for each $\varphi(\zeta)$, the string $\varphi(\zeta)$ contains the character 2 exactly 4 times. The first 2 is at most the $(|\Sigma| + 1)$-th 0 of $T$. Therefore, its index is at most $2 \cdot 2^{\lceil \log(|\Sigma| + 1) \rceil} - 1$. Similarly, the index of the second 2 in $\varphi(\zeta)$ is at least $6 \cdot 2^{\lceil \log(|\Sigma| + 1) \rceil}$.

Next, we have to prove that $\varphi$ is strongly invertible.

Let $P' = (\text{suffix}(\varphi(\zeta_0))) \varphi(\zeta_1) \varphi(\zeta_2) \ldots \varphi(\zeta_i) \text{ (prefix}(\varphi(\zeta_{i+1})))$ be the substring of an image of $\varphi$. Since the substring 22 only occurs in $P'$ as suffix of the substrings $\varphi(\zeta_i)$, we can decide, which characters in $P'$ correspond to which character in $\zeta_0 \zeta_1 \zeta_2 \ldots \zeta_i \zeta_{i+1}$. While the suffix of $\varphi(\zeta_0)$ in $P'$ and the prefix of
\(\varphi(\varsigma_{i+1})\) in \(P^t\) may be too short to retrieve \(\varsigma_0\) and \(\varsigma_{i+1}\), we are able to retrieve \(\varsigma_1, \ldots, \varsigma_i\) from the string \(P^t\) and we can check that the images of \(\varsigma_0\) and \(\varsigma_{i+1}\) are incomplete. Thus, we can invert the morphism \(\varphi\) even on substrings of its image.

Lastly, we will show that \(\varphi\) is cube-free.

Let \(P \in \Sigma^*\) be a cube-free string and let \(P' = \varphi(P)[s..e]\) be a substring of \(\varphi(P)\). We will show that \(P'\) is not a cube. In order to do this, we will consider five cases.

Case 1: The number of occurrences of the character 2 in \(P'\) is not divisible by 3: In this case, the string \(P'\) is not a cube.

Case 2: \(P'\) does not contain the character 2: In this case, the string \(P'\) is a substring of the Prouhet-Thue-Morse word and is therefore cube-free.

Case 3: \(P'\) does contain the character 2 exactly three times: The string \(\varphi(P)\) does not contain 222. Therefore, if \(P'\) contains the substring 22, it is not a cube. Otherwise, the string \(P'\) is a substring of \(2\varphi(\varsigma)\) for some \(\varsigma \in \Sigma\) and it also contains the first two 2s of \(\varphi(\varsigma)\). Since the distance of these two adjacent 2s is at least \(4 \cdot 2^{\log(|\Sigma|+1)}\) while the length of \(P'\) is at most \(8 \cdot 2^{\log(|\Sigma|+1)} + 2\), the string \(P'\) is not a cube.

Case 4: \(P'\) does contain the character 2 exactly 6 times: Then, the string \(P'\) is a substring of \(2\varphi(\varsigma_1)\varphi(\varsigma_2)\) for some \(\varsigma_1, \varsigma_2 \in \Sigma\). Also, the string \(P'\) contains the first 2 and the third 2 of \(\varphi(\varsigma)\) for one \(\varsigma \in \{\varsigma_1, \varsigma_2\}\). These two 2s with exactly one 2 between them have a distance of at least \(6 \cdot 2^{\log(|\Sigma|+1)}\) while \(P'\) contains at most \(16 \cdot 2^{\log(|\Sigma|+1)} + 1\) characters. Hence, the string \(P'\) is not a cube.

Case 5: \(P'\) does contain the character 2 at least 9 times: Then, the string \(P'\) contains the substring \(\varphi(\varsigma)\) for some \(\varsigma \in \Sigma\). Thus, we can write \(P'\) as \((\text{suf}(\varphi(\varsigma_0)))\varphi(\varsigma_1)\varphi(\varsigma_2)\ldots\varphi(\varsigma_i)(\text{pre}(\varphi(\varsigma_{i+1})))\) with either \(i \geq 2\) or both \(i = 1\) and \(|\text{suf}(\varphi(\varsigma_0))| \geq 2\). Since the substring 22 only occurs at the last two characters of each \(\varphi(\varsigma_i)\), the minimum period of \(P'\) is a multiple of \(|T| + 2\). Since \(\varphi\) is \(|T| + 2\)-uniform, powers in the preimage naturally translate into powers in the image of \(\varphi\). Since \(P\) is cube-free, the string \(\varphi(\varsigma_1)\varphi(\varsigma_2)\ldots\varphi(\varsigma_i)\) is cube-free as well. Similarly, prepending more than the last \(2 \cdot 2^{\log(|\Sigma|+1)}\) characters of \(\text{suf}(\varphi(\varsigma_0))\) or appending more than the first \(2 \cdot 2^{\log(|\Sigma|+1)}\) characters of \(\text{pre}(\varphi(\varsigma_{i+1}))\) does not yield a cube. On the other hand, if we add less than \(8 \cdot 2^{\log(|\Sigma|+1)}\) characters to \(\varphi(\varsigma_1)\varphi(\varsigma_2)\ldots\varphi(\varsigma_i)\), these additional characters correspond to adding a fractional character to \(\varsigma_1\varsigma_2\ldots\varsigma_i\) and therefore do not yield a cube either.

In either case, the string \(P'\) is cube-free. Since we chose \(P'\) arbitrarily, the morphism \(\varphi\) is also cube-free.

Next, we further reduce the ternary alphabet to a binary alphabet.
Lemma 4.0.3. Let $\Sigma_3 = \{0, 1, 2\}$ be an alphabet.

Define the alphabet $\Sigma_2 = \{a, b\}$.

Consider the morphism $\psi : \Sigma_3^* \rightarrow \Sigma_2^*$ defined by:

- $\psi(0) = \text{abaabb} \cdot \text{ababba} \cdot \text{ababba} \cdot \text{abaabb}$
- $\psi(1) = \text{abaabb} \cdot \text{abbaab} \cdot \text{abbaab} \cdot \text{abaabb}$
- $\psi(2) = \text{abaabb} \cdot \text{abbaba} \cdot \text{abbaba} \cdot \text{abaabb}$

Then, $\psi$ is a strongly invertible, 24-uniform, cube-free morphism.

Proof. By construction, the morphism $\psi$ maps each character of $\Sigma_3$ to a string consisting of exactly 24 characters of $\Sigma_2$. Thus, the morphism $\psi$ is 24-uniform.

Note that each occurrence of $\text{abaabb}$ in the image of $\psi$ is either the first 6 characters of $\psi(\varsigma)$ with $\varsigma \in \Sigma_3$, the last 6 characters of $\psi(\varsigma)$ with $\varsigma \in \Sigma_3$ or the middle 6 characters in $\psi(2)$. Also, if we have either the preceeding 6 characters or the succeeding 6 characters, we can decide to which of the cases the occurrence belongs.

Next, we have to prove that $\psi$ is strongly invertible.

Let $P' = (\text{suf}(\psi(\varsigma_0))) \psi(\varsigma_1) \psi(\varsigma_2) \ldots \psi(\varsigma_i) (\text{pre}(\psi(\varsigma_{i+1})))$ be the substring of an image of $\psi$. If $P'$ contains the factor $\text{abaabb}$ and either the preceeding 6 characters or the succeeding 6 characters, we can determine for each character in $P'$, to which character in its preimage it belongs. Otherwise, the preimage of $P'$ does not contain a full character.

Lastly, we will show that $\psi$ is cube-free.

Let $P \in \Sigma_3^*$ be a cube-free string and let $P' = \psi(P)[s..e]$ be a substring of $\psi(P)$. We will show that $P'$ is not a cube.

Since the alphabet $\Sigma_3$ only contains 3 characters, it is feasible to check analyze all substrings of the strings $\psi(\varsigma_1) \psi(\varsigma_2) \psi(\varsigma_3) \psi(\varsigma_4)$. In particular, we can show that the power of substrings $P'$ with minimum period of less than 24 is at most $\frac{8}{3}$. This can be done in a few seconds by Listing 4.1.

If $P'$ has a minimum period of at least 24 and has length $|P'| < 72$, then $P'$ is not a cube.

Otherwise, the string $P'$ has a length of at least 72 characters and thereby contains the substring $\text{abaabb} \cdot \text{abaabb}$ at least twice. Since each occurrence of $\text{abaabb} \cdot \text{abaabb}$ starts with the last 6 characters of a $\psi(\varsigma)$ with $\varsigma \in \Sigma_3$, the minimum period of $P'$ is a multiple of 24. Since $\psi$ is 24-uniform, powers in the preimage naturally translate into powers in the image of $\psi$. Since each substring of $P$ is cube-free, their images are cube-free as well. Similarly, prepending more than 11 characters of a suffix of $\phi(\varsigma)$ for any $\varsigma \in \Sigma_3$ or appending at more than 11 characters of a prefix of some $\phi(\varsigma)$ for any $\varsigma \in \Sigma_3$
does not yield a cube. On the other hand, if we add less than 24 characters to the image of a substring of \( P \), these additional characters correspond to adding a fractional character to the corresponding substring and therefore do not yield a cube either.

In either case, the string \( P' \) is cube-free. Since we chose \( P' \) arbitrarily, the morphism \( \psi \) is also cube-free.

By concatenating the two morphisms, we get a corresponding morphism from an arbitrary alphabet to a binary alphabet.

**Theorem 4.0.4.** Let \( \Sigma \) be an alphabet and define the alphabet \( \Sigma_2 = \{a, b\} \).

Let \( S \) be a string in \( \Sigma^* \) with \( z \) LZ77-factors which contains each character of the alphabet at least once.

Then, \( \psi \circ \varphi \) is a strongly invertible, uniform, cube-free morphism from the alphabet \( \Sigma \) to \( \Sigma_2 \). The morphism is also \( k \)-power-free for all integers \( k \geq 4 \).

Furthermore, the image \( \psi(\varphi(S)) \) has length \( \mathcal{O}(|S|^2) \) and \( \mathcal{O}(z) \) LZ77-factors. The implicit constants do not depend on the underlying alphabet \( \Sigma \).

**Proof.** Since both \( \psi \) and \( \varphi \) are strongly invertible, uniform, cube-free morphisms, the same is true for their composition.

Wlazinski proves in [99] that uniform cube-free morphisms are also \( k \)-power-free for all integers \( k \geq 4 \). Hence \( \psi \circ \varphi \) is \( k \)-power-free for all integers \( k \geq 4 \).

Since \( \psi \) is 24-uniform, its application only increases the string by a constant factor. The function \( \varphi \) is \((|T| + 2)\)-uniform, where \( T \) is a string of length \( 8 \cdot 2^{\lfloor \log(|\Sigma|+1) \rfloor} < 16(|\Sigma|+1) \leq 16(|S|+1) \). Thus, the morphism \( \psi \circ \varphi \) increases the length of \( S \) uniformly by a factor of at most \( 384(|S|+1) \).

Next we consider a factorization of \( S \) into \( z \) LZ77-factors. Substrings of \( S \) naturally correspond to substrings of the images of \( S \). Hence, we can translate each LZ77-factor which does not correspond to the first occurrence of a character in \( S \) by a single LZ77-factor of the image of \( S \).

The Prouhet-Thue-Morse word can be encoded with logarithmically many LZ77-factors and by prerequisite, the alphabet \( \Sigma \) contains at most \( z \) characters. Since each string \( \varphi(\varsigma) \) is equal to the Prouhet-Thue-Morse word except for exactly 4 characters, the first of these strings can be encoded with \( \mathcal{O}(\log z) \) LZ77-factors. Afterwards, each other \( \varphi(\varsigma') \) differs from \( \varphi(\varsigma) \) in at most 4 indices. Therefore, each other character can be encoded with at most 9 LZ77-factors. Therefore, \( \varphi(S) \) can be encoded with \( \mathcal{O}(z) \) LZ77-factors.

Similar analysis shows then that \( \psi \varphi(S) \) can be encoded with \( \mathcal{O}(z) \) LZ77-factors.
CHAPTER 4. ALPHABET REDUCTION

Listing 4.1: This listing finds the maximal power with period less than 24 in the substring of the strings $\psi(\varsigma_1)\psi(\varsigma_2)\psi(\varsigma_3)\psi(\varsigma_4)$ with $\varsigma_1, \varsigma_2, \varsigma_3, \varsigma_4 \in \Sigma_3$.

```python
from itertools import product

def checkPeriod(string, period):
    unshifted = string[0:len(string) - period]
    shifted = string[period:len(string)]
    return unshifted == shifted

def get_power(string, max_period):
    max_period = min(len(string), max_period)
    for period in range(1, max_period):
        if checkPeriod(string, period):
            return len(string)/period
    return 1

code = ["abaabbababbaababbaabbaabb", 
        "abaabbababbaababbaababaabb", 
        "abaabbabbabaababbaabaabb"]

max_power = 1

for raw_string in product(code, repeat=4):
    string = "".join(raw_string)
    substrings = [string[i: j]
                  for i in range(len(string))
                  for j in range(i + 1, len(string) + 1)]
    for substring in substrings:
        power = get_power(substring, 24)
        if max_power < power:
            max_power = power

print(max_power)
```
Chapter 5
Maximal (Sub-)Repetitions

The first periodic patterns we consider in this thesis are maximal repetitions and their variants, $\delta$-repetitions and $\delta$-subrepetitions for positive real numbers $\delta$.

In this chapter, we will present upper bounds for the number of maximal $\delta$-(sub-)repetitions. We will also show that for fixed $\delta$, the bound for the number of maximal $\delta$-repetitions is tight up to a constant factor even if the underlying alphabet contains only two characters and the underlying string does not contain powers with large exponent.

Up to a few optimizations, these results are already published by the author of this thesis in [82] in a special issue of the International Journal of Foundations of Computer Science (IJFCS). Weaker versions of these results are also published by the author of this thesis in [81] at the International Conference on Developments in Language Theory (DLT 2021).

Repetitions are substrings which are at least twice as long as their minimum period. Similarly, $\delta$-repetitions (and $\delta$-subrepetitions) are at least $2 + \delta$ times (and $1 + \delta$ times, respectively) as long as their minimum period. In this way, repetitions and their variants are generalizations of the square which is a substring that has a period which is exactly half of its length.

The square is the most basic string pattern and the $\delta$-subrepetition is a very versatile generalization of the square. It is therefore not surprising that many fields which study one-dimensional data also consider $\delta$-subrepetitions. For example, subrepetitions have been used in the analysis of Greek literature [83], DNA analysis [33] and the analysis of musical scores [24].

However, the number of $\delta$-(sub-)repetitions may be quadratic with respect to the length of the string. Therefore, we restrict ourselves to maximal repetitions and their variants. A $\delta$-(sub-)repetition is maximal if it has an occurrence in the underlying string which cannot be extended in any direction without increasing the minimum period. Also, we will consider extended max-
mal δ-(sub-)repetitions which are the maximal δ-(sub-)repetitions extended by the preceding character and the succeeding character for some occurrence in the underlying string.

It may seem arbitrary to discriminate between repetitions and subrepetitions based on whether the exponent is at least 2. However, while the repetitions yield a compact representation of the squares in the underlying string, the subrepetitions which are not repetitions may be square-free and therefore provide no additional information about the squares. Furthermore, the subrepetitions which are not repetitions have the form $PP'$ where the minimum period does not yield copies of the characters in $P'$ while the minimum period in repetitions provide a copy of each character.

Maximal repetitions also provide succinct information about the powers in a string as shown in [25]. They also give rise to more elaborate patterns like two-dimensional maximal repetitions as defined in [5].

\section{Related Work}

\subsection{Squares}

Recall that a square is a string $PP$ consisting of exactly two copies of the same substring $P$.

Since each square is given by the pair of its starting index and its length, it is obvious that the number of different squares in a string is at most quadratic with regard to the length of the underlying string.

Conversely, in the string $a^n$, each occurrence of a substring $a^{2i} = a^i a^i$ of even length is a square. Hence, there is no subquadratic upper bound for the number of squares. Therefore, it is useful to restrict ourselves to distinct squares and primitively rooted squares.

Fraenkel and Simpson prove in [38] that the number of distinct squares in a string of length $|S|$ is bounded from above by $2|S|$. They further conjecture that this bound could be improved to $|S|$. An improvement of this bound has recently been proven in [15] by Brlek and Li.

Regarding primitively rooted squares, Kolpakov and Kucherov point out in [61] that the three squares lemma presented in [26] implies that the number of primitively rooted squares is in $O(|S| \log |S|)$.

The bounds stated in the last paragraph are tight up to a constant factor. The following lemma shows with $q = 2$ that there are arbitrarily long strings $S$ with $\Omega(|S|)$ distinct squares and $\Omega(|S| \log |S|)$ primitively rooted squares.

\begin{lemma}
Let $q \geq 2$ be a fixed natural number.

The strings $S_i$ given by $S_1 = \sigma_1^{q+1}$ and $S_i = (S_{i-1}[0..|S_{i-1}| - 2] \sigma_i)^{q+1}$ have
\end{lemma}
5.1. RELATED WORK

- length \((q + 1)^i\),
- \(\Omega \left( |S_i| \log |S_i| \right)\) primitively rooted \(q\)-th powers and
- \(\Omega \left( |S_i| \right)\) distinct \(q\)-th powers.

**Proof.** By construction, the string \(S_1\) has length \(q + 1\) and for each \(i\), the string \(S_{i+1}\) has length \((q + 1)|S_i|\). Hence, the length of each \(S_i\) is given by \((q + 1)^i\).

Also, each of the \(\frac{1}{q} |S_i|\) distinct cyclic rotations of

\[ S_i^{q+1} = (S_{i-1}[0..|S_{i-1}| - 2|\sigma_i|]q \]

is a \(q\)-th power and a substring of \(S_i\). Therefore, the string \(S_i\) contains \(\Omega \left( |S_i| \right)\) distinct \(q\)-th powers. Since the bases of these \(q\)-th powers contain the character \(\sigma_i\) exactly once, these \(q\)-th powers are primitively rooted.

Similarly, for each \(j\) with \(2 \leq j \leq i\), the strings

\[ S_j^{q+1} = (S_{j-1}[0..|S_{j-1}| - 2|\sigma_j|]q \]

and their cyclic rotations are both primitively rooted \(q\)-th powers and substrings of \(S_i\). These strings have length \(q(q + 1)^{j-1}\). Of these \(q(q + 1)^{j-1}\) possible cyclic rotations only \((q + 1)^{j-1}\) are distinct. Furthermore, each of these substrings has at least \((q + 1)^{i-j}\) occurrences.

Summation over these primitively rooted \(q\)-th powers shows that there are at least

\[ \sum_{j=2}^{i} (q + 1)^{j-1} \cdot (q + 1)^{i-j} = \sum_{j=2}^{i} (q + 1)^{i-1} \in \Omega \left( |S_i| \log_{q+1} |S_i| \right) \]

primitively rooted \(q\)-th powers.

Regarding the lower bounds, Fraenkel and Simpson show in [37] that there are arbitrary long binary strings with only three distinct squares. If we count all squares, Kucherov et al. prove in [64] that the number of all squares in a binary string of length \(|S|\) is bounded from below by 0.55\(|S|\).

However, if we add a third character to the alphabet, we can find arbitrarily long strings without squares. Thus, squares are avoidable in ternary strings. This was proven by Thue in [97] (written in German) which was translated into English by Berstel [12]. Furthermore, using numerical methods, Shur obtains in [93] that the number of ternary square-free words of length \(|S|\) is both in \(\Omega(1.301759^{|S|})\) and in \(O(1.301762^{|S|})\).
5.1.2 Higher Powers

Cubes are strings of the form \( P^3 = PPP \) consisting of exactly three copies of the same substring \( P \).

Since each cube \( P^3 \) contains the square \( P^2 \), all upper bounds for the number of squares are also upper bounds for the number of cubes.

Similarly to the case of squares, in the string \( a^n \), each occurrence of a substring \( a^{3i} = a^i a^i a^i \) of length divisible by 3 is a cube. Hence, there is no subquadratic upper bound for the number of cubes in a string. Therefore, we restrict ourselves to distinct cubes and primitively rooted cubes.

On the other hand, Lemma 5.1.1 shows with \( q = 3 \) that there are arbitrarily long strings \( S \) with \( \Omega(|S|) \) distinct cubes and \( \Omega(|S| \log |S|) \) primitively rooted cubes.

Therefore, the upper bounds for the number of squares in the last subsection also provide upper bounds for the number of cubes that are tight up to a constant factor.

Similarly, for each fixed exponent \( q \in \mathbb{N}_{\geq 2} \), Lemma 5.1.1 yields arbitrarily long strings \( S \) with \( \Omega(|S|) \) distinct \( q \)-th powers and \( \Omega(|S| \log |S|) \) primitively rooted \( q \)-th powers. Since each \( q \)-th power \( P^q \) contains \( P^2 \), the maximal number of distinct \( q \)-th powers in a string of length \( |S| \) is in \( \Theta(|S|) \) and the maximal number of primitively rooted \( q \)-th powers in a string of length \( |S| \) is in \( \Theta(|S| \log |S|) \). Note that the implicit constant depends on \( q \).

Unlike squares, higher powers are easier to avoid. We say that a string has an overlap if it contains a substring of the form \( PTPTP \) with \( |P| \geq 1 \). With this definition, an overlap-free string is allowed to contain squares but is forbidden to contain any greater rational power.

For example, in [97] (translated by Berstel in [12]), Thue (re-)defines the Prouhet-Thue-Morse sequence given by its finite prefixes length \( 2^i \) given by Definition 4.0.1. Morse and Hedlund prove in [73] that this (infinite) string is overlap-free. Therefore, all of its finite substrings are overlap-free as well.

5.1.3 Lower Powers

We have seen that while squares are unavoidable in sufficiently long strings, there are arbitrarily long binary strings that contain no higher powers than squares. We have further seen that there are arbitrarily long ternary strings without squares.

An important part of the studies of strings is if certain patterns are avoidable or if sufficiently long strings cannot avoid that pattern. In Chapter 6, we will consider another, famously unavoidable, pattern — the subcadence.

In terms of powers, we are interested in the repetitive threshold.
5.1. RELATED WORK

The repetitive threshold of an alphabet Σ is the number \( r(|\Sigma|) \) such that for each \( \varepsilon > 0 \),

- each sufficiently long string over \( \Sigma \) contains a power of exponent greater than \( r(|\Sigma|) - \varepsilon \) and
- there are arbitrary long strings over \( \Sigma \) without powers of exponent greater than \( r(|\Sigma|) + \varepsilon \).

While the term repetitive threshold was first introduced by Brändenburg in [14], the concept was already studied by Dejean in [32] (written in French) who conjectured:

\[
\begin{align*}
 r(|\Sigma|) &= 2 & \text{if } |\Sigma| = 2 \\
 r(|\Sigma|) &= \frac{7}{4} & \text{if } |\Sigma| = 3 \\
 r(|\Sigma|) &= \frac{5}{4} & \text{if } |\Sigma| = 4 \\
 r(|\Sigma|) &\geq \frac{|\Sigma|}{|\Sigma| - 1} & \text{if } |\Sigma| \geq 5
\end{align*}
\]

It is easy to show that a string of length \( |\Sigma| + 2 \) over the alphabet \( \Sigma \) has at least one power with exponent of at least \( \frac{|\Sigma|}{|\Sigma| - 1} \).

If the string has two equal characters \( S[i] = S[j] \) with \( 0 < j - i < |\Sigma| - 1 \) then the substring \( S[i..j] \) of length \( j - i + 1 \leq |\Sigma| \) has a minimum period of at most \( j - i + 1 - 1 = j - i \leq |\Sigma| - 1 \). Therefore, in order to avoid these powers with exponent of at least \( \frac{|\Sigma|}{|\Sigma| - 1} \), neither of the substrings \( S[0..|\Sigma| - 1] \), \( S[1..|\Sigma|] \) and \( S[2..|\Sigma| + 1] \) contains a character twice. Hence, the equations \( S[0] = S[|\Sigma|] \) and \( S[1] = S[|\Sigma| + 1] \) hold and \( S[0..|\Sigma| + 1] \) is a power with exponent of at least \( \frac{|\Sigma| + 2}{|\Sigma| - 1} \).

This proves that for all \( \Sigma \), powers with an exponent of at least \( \frac{|\Sigma|}{|\Sigma| - 1} \) are unavoidable. Therefore, the repetitive threshold \( r(|\Sigma|) \) is at least \( \frac{|\Sigma|}{|\Sigma| - 1} \). However, proving that for \( |\Sigma| \not\in \{3, 4\} \), these lower bounds are also the upper bound is much more challenging.

Since Prouhet-Thue-Morse words as defined in Definition 4.0.1 are overlap-free, we also have \( r(2) \leq 2 \) which proves \( r(2) = 2 \). The equality for other small alphabet sizes is proven by Dejean in [32] for \( |\Sigma| = 3 \), Pansiot in [75] (written in French) for \( |\Sigma| = 4 \), Ollagnier in [74] for \( 5 \leq |\Sigma| \leq 11 \) and Mohammad-Noori and Currie in [72] for \( 12 \leq |\Sigma| \leq 14 \).

The equality for large alphabet sizes is proven by Carpi in [17] for \( |\Sigma| \geq 33 \) and by Currie and Rampersad in [30] for \( |\Sigma| \geq 30 \) and in [29] for \( |\Sigma| \geq 27 \).

The gap of \( 15 \leq |\Sigma| \leq 26 \) was finally closed independently by Currie and Rampersad in [31] and Rao in [87]. Therefore, Dejean’s conjecture holds.
5.1.4 Maximal $\delta$-(Sub-)Repetitions

Recall that a \textit{repetition} is the occurrence of a power with exponent of at least 2. A repetition is \textit{maximal} if its minimum period neither extends to the left nor to the right.

As we have seen in Subsection 5.1.1 and Subsection 5.1.2, the number of squares and higher powers can be quadratic and even if we are only interested in primitively rooted powers, their upper bound is still not linear.

In [61], Kolpakov and Kucherov prove that the upper bound for the number of maximal repetitions is linear with regard to the length $|S|$ of the underlying string. While their proof does not provide an implicit constant, they claim that computer experiments suggest that the number of maximal repetitions is actually bounded by $|S|$. This “Runs” Theorem is proven by Bannai et al. in [8]. For binary strings, this result is further improved by Holub in [52] which shows that the number of maximal repetitions is bounded by $\frac{183}{193}|S|$.

In [39], Franek presents the strings with up to 35 characters which have the maximal amount of maximal repetitions for their length. Note that neither of the strings contains more than three different characters and all strings with at least length 17 are binary. Holub conjectures that the maximal number of maximal repetitions can always be achieved by binary strings. For squares, Manea and Seki prove in [71] that the maximal density of squares can be achieved over a binary alphabet. However, no such result is proven for maximal repetitions.

Similarly to repetitions, we define $\delta$-\textit{subrepetitions} which are occurrences of powers with exponent of at least $1 + \delta$ and $\delta$-\textit{repetitions} which are occurrences of powers with exponent of at least $2 + \delta$. Similarly to the repetition, these variants are \textit{maximal} if their minimum period neither extends to the left nor to the right. For example, in the string $(a^k b)^n$, each $a^k$ is a maximal $(k - 2)$-repetition.

It is easy to see that for each $\delta > 0$, the number of maximal $\delta$-repetitions is still at most linear. More surprisingly, the upper bound for the number of $\delta$-subrepetitions is also (just) linear. Kolpakov et al. show in [63] that for $\delta > 0$, the number of maximal $\delta$-subrepetitions is in $O\left(\frac{|S|}{\delta^2}\right)$. This upper bound was improved independently by Crochemore et al. in [27] and Gawrychowski et al. in [46] who prove that the maximal number of maximal $\delta$-subrepetitions is even in $O\left(\frac{|S|}{\delta^2}\right)$. The implicit constant was calculated by I and Köppl in [54] to be at most $3\left(\frac{\pi^2}{6} + \frac{5}{2}\right)$.

Conversely, Crochemore et al. also prove in [27] that, at least over an unbounded alphabet, this upper bound is tight up to a constant factor.
5.2. COMPRESSED UPPER BOUNDS

5.1.5 Algorithms

In [34], Ellert and Fischer showed that there is a linear time algorithm to report all maximal repetitions in strings over arbitrary linearly-sortable alphabets.

Surprisingly, for fixed $\delta$, we can even report all maximal $\delta$-subrepetitions in linear time. In [62], Kolpakov proves that all maximal $\delta$-subrepetitions in the string $S$ can be found in $O \left( \frac{|S|}{\delta} \log \frac{1}{\delta} \right)$ time. Since there are strings $S$ with $\Omega \left( \frac{|S|}{\delta} \right)$ $\delta$-subrepetitions, we cannot expect an algorithm that lists all $\delta$-subrepetitions in $o \left( \frac{|S|}{\delta} \right)$ time. Hence, this algorithm can only be improved by a factor $O \left( \log \frac{1}{\delta} \right)$.

In compressed strings, the upper bounds which we will prove in the next section show that we cannot expect a polynomial time algorithm for reporting all $\delta$-subrepetitions with $\delta \leq 1$.

For sufficiently large powers, the detection is straightforward. Given a string $S$ compressed by a context-free grammar $G = (V, \Sigma, S, \text{rhs})$, it is easy to find, up to cyclic rotation, all bases of powers with exponent $4|V|$. This was shown in the master's thesis of the author of this thesis in [76].

In [55], I et al. present an algorithm that gives a compact representation of all squares and runs of $S$ in $O(|V|^4)$ time. This algorithm can also be extended to count the number of squares and runs with the same time complexity.

5.2 Compressed Upper Bounds

In this section, we will present and prove upper bounds for the number of maximal $\delta$-(sub-)repetitions in highly compressible strings.

In particular, we will prove the following two theorems.

**Theorem 5.2.1.** Let $S$ be string with string attractor $\Gamma$ and let $\delta > 0$ be a real number.

Then, the string $S$ contains at most

$$((|\Gamma| + 2) \left\lfloor 3 + \frac{\delta}{6} \right\rfloor \cdot \left\lfloor \log_{1+\frac{\delta}{4}}(|S|) \right\rfloor)$$

distinct extended maximal $\delta$-repetitions.

**Theorem 5.2.2.** Let $S$ be string with string attractor $\Gamma$. Let further $\delta > 0$ be a real number and $q \geq 2$ be a natural number.
Then, the string $S$ contains at most

$$\left( |\Gamma| + 2 \right) \left[ 3 + \frac{4}{\delta} \right] \cdot \left[ \log_{1+\frac{1}{2\delta}} (|S|) \right]$$

distinct extended maximal $\delta$-subrepetitions without $q$-th powers.

These theorems are slightly stronger versions of theorems that were originally published by the author of this thesis in [81] and its extended journal article in [82].

Since the number of $q$-th powers can be quadratic with respect to the underlying string, we restricted ourselves to distinct or primitively rooted powers in order to keep their number (almost) linear. This restriction was not necessary for maximal $\delta$-(sub-)repetitions in uncompressed strings.

Consider String $S = (a^{2+\lceil \delta \rceil}b)^k$ with length $k(3 + \lceil \delta \rceil)$ and $k+1$ occurrences of the maximal $\delta$-repetition $a^{2+\lceil \delta \rceil}$. Since the string decomposes into just 4 LZ77-factors, we cannot expect the number of maximal $\delta$-repetitions to be sublinear even if the number of LZ77-factors of the string is constant.

We consider strings that have small compressed sizes with respect to a strong repetition-based compression scheme like LZ77 and therefore small string attractors. We should expect that such strings have few maximal $\delta$-(sub-)repetitions which are distinct as factors of the string but instead contain many copies of those few distinct $\delta$-(sub-)repetitions. Therefore, in the compressed case, it is more natural to consider distinct maximal $\delta$-(sub-)repetitions than to consider all different maximal $\delta$-(sub-)repetitions.

Also, since it doesn’t significantly increases the size of the upper bounds, we will consider extended maximal $\delta$-(sub-)repetitions which do not only include the $\delta$-(sub-)repetition itself but also the first character in both directions which break the periodicity.
5.2. COMPRESSED UPPER BOUNDS

However, even if we restrict ourselves to strings with only 4 LZ77-factors and distinct maximal repetitions, their number can still be linear with respect to the length of the string. See for example the string \((ab)^k a^2\) in Figure 5.1 with its linearly many distinct maximal repetitions.

However, if we set the threshold for the minimal exponent in the maximal repetitions slightly higher than 2, we do get the meaningful upper bound given by 5.2.1. Surprisingly, if we also require that the string does not contain \(q\)-th powers we also get a meaningful upper bound for the number of maximal \(\delta\)-subrepetitions which is given by 5.2.2.

We consider a given string attractor \(\Gamma\) of the string \(S\). In order to prove these upper bounds, we use that each substring of \(S\) has an occurrence containing an index in \(\Gamma\). We therefore want to count for each index \(\gamma \in \Gamma\) the maximal number of maximal \(\delta\)-(sub-)repetitions in \(S\) that can contain \(\gamma\).

This technique has three main disadvantages. Firstly, multiple occurrences of the same \(\delta\)-(sub-)repetition can contain different indices in \(\Gamma\) and thereby be counted multiple times. Secondly, even one occurrence of a long \(\delta\)-(sub-)repetitions can contain multiple indices in \(\Gamma\). Thirdly, while each maximal \(\delta\)-(sub-)repetition has an occurrence which contains an index in \(\Gamma\), it is possible that there are maximal \(\delta\)-(sub-)repetitions such that neither of these occurrences is maximal.

For example, in the string \(aaabaaaacaaa\), each minimal string attractor \(\Gamma\) contains the index of the \(b\), the index of the \(c\) and the index of one of the four \(a\)s in the middle of the string. The maximal repetition \(aaa\), however is only maximal in \(S[aaab]\) and \(caaaS_r\) but not in \(baaaac\). Therefore, the maximal repetition \(aaa\) has no occurrence containing an index in \(\Gamma\) in which \(aaa\) is maximal.

We solve the third problem by extending the maximal \(\delta\)-(sub-)repetitions by a single character to the left and to the right. We thereby get occurrences of the extended maximal \(\delta\)-(sub-)repetitions which prove their own maximality without further knowledge of the underlying string. However, these extended maximal \(\delta\)-(sub-)repetitions may contain the characters \(S[-1] = S_l\) and \(S[|S|] = S_r\).

This solution comes with a price. Firstly, we have to add \(-1\) and \(|S|\) to our string attractor. Since the corresponding strings are either prefixes or suffixes, we could handle them separately, but for the sake of simplicity we will just increase the size of the string attractor by two. Secondly, by counting all distinct extended maximal \(\delta\)-(sub-)repetitions, we might have to count some of them up to \(|\Sigma|^2 + 2\) times.

Fortunately, at least for fixed values \(\delta\) and \(q = \lfloor \delta + 2 \rfloor\), these disadvantages only contribute a constant factor to the upper bounds. We will show in this
chapter that the upper bounds given by Theorem 5.2.1 and Theorem 5.2.2 are, up to a constant factor, tight for fixed values of $\delta$ and $q = \lfloor \delta + 2 \rfloor$.

In order to translate local upper bounds for the number of maximal $\delta$-(sub-)repetitions to global upper bounds for the number of distinct maximal $\delta$-(sub-)repetitions, we need the following lemma.

**Lemma 5.2.3.** Let $S$ be a string such that for each index $i$ with $-1 \leq i \leq |S|$, the number of different extended maximal $\delta$-(sub-)repetitions containing the index $i$ is at most $c$. Let $\Gamma$ be a string attractor of $S$.

Then, the number of distinct extended maximal $\delta$-(sub-)repetitions in $S$ is at most $c(|\Gamma| + 2)$.

**Proof.** By definition of the string attractor $\Gamma$, each of the extended maximal $\delta$-(sub-)repetitions has an occurrence that contains an index in the string attractor, the character $S[-1] = \$_l$ or the character $S[|S|] = \$_r$.

Therefore, the upper bound for the number of different extended maximal $\delta$-(sub-)repetitions containing any of these indices naturally translates into an upper bound for the number of distinct extended maximal $\delta$-(sub-)repetitions in $S$. \qed

### 5.2.1 Upper Bound for Maximal $\delta$-Repetitions

In this subsection, we will prove the upper bound given in Theorem 5.2.1 for the number of distinct extended maximal $\delta$-repetitions in a string $S$ with string attractor $\Gamma$.

**Theorem 5.2.1.** Let $S$ be string with string attractor $\Gamma$ and let $\delta > 0$ be a real number.

Then, the string $S$ contains at most

$$\left( |\Gamma| + 2 \right) \left\lfloor 3 + \frac{6}{\delta} \right\rfloor \cdot \left\lfloor \log_{1 + \frac{2}{\delta}}(|S|) \right\rfloor$$

distinct extended maximal $\delta$-repetitions.

We will first find a local upper bound for the number of different extended maximal $\delta$-repetitions and will then use Lemma 5.2.3 to derive this global upper bound.

**Theorem 5.2.4.** Let $S$ be a string and $-1 \leq i \leq |S|$ an index. Let further $\delta > 0$ be a real number.

Then, there are at most

$$\left\lfloor 3 + \frac{6}{\delta} \right\rfloor \cdot \left\lfloor \log_{1 + \frac{2}{\delta}}(|S|) \right\rfloor$$
different maximal $\delta$-repetitions $S[s_k..e_k]$ whose extensions contain the index $i$.

Note that this theorem is slightly stronger than both Theorem 4 in [81] and Theorem 5 in [82].

In order to prove this stronger result for $\delta \leq 8$, we will have to consider the two possible maximal $\delta$-repetitions with minimum period 1 separately.

**Lemma 5.2.5.** Let $S$ be a string and $-1 \leq i \leq |S|$ an index. Let further $\delta > 0$ be a real number.

Then, there are at most 2 different maximal 1-periodic $\delta$-(sub-)repetitions whose extensions contain the index $i$.

**Proof.** By definition, each $\delta$-(sub-)repetition contains at least 2 characters. Therefore, if an extended $\delta$-(sub-)repetition contains the index $i$, the corresponding $\delta$-(sub-)repetition without its extension contains the index $i - 1$ or the index $i + 1$.

On the other hand, for a given index $i'$, the only possible maximal 1-periodic $\delta$-(sub-)repetitions which contains $i'$ is given by the 1-periodic extension of $S[i']$. Hence, the only 2 possible extended maximal 1-periodic $\delta$-(sub-)repetitions which contain the index $i$ are the 1-periodic extensions of $S[i - 1]$ and $S[i + 1]$. $\blacksquare$

In order to bound the number of maximal $\delta$-repetitions with minimum period greater than 1, we will use the pigeonhole principle to show that if more of these maximal $\delta$-repetitions existed, there would be two of them which have a long intersection and similar minimum periods. Note that apart from the specific bounds on what is considered a long intersection and what is considered similar minimum periods, this is basically the definition of neighboring runs given by Rytter in [90]. This definition was used to prove that the number of different maximal repetitions in $S$ is at most $5|S|$.

The following lemma will show that maximal $\delta$-repetitions cannot be that closely neighbored.

**Lemma 5.2.6.** Let $S$ be a string and $\delta > 0$ be a real number. Let $S[s_1..e_1]$ and $S[s_2..e_2]$ be different maximal $\delta$-repetitions of $S$ with minimum periods $p_1$ and $p_2$, respectively. Let further $L$ be a real number such that both $p_1$ and $p_2$ are contained in the interval $[L, (1 + \frac{\delta}{4}) L]$.

Then, the intersection $S[\max(s_1, s_2)..\min(e_1, e_2)]$ of $S[s_1..e_1]$ and $S[s_2..e_2]$ contains less than $(2 + \frac{\delta}{4}) L - 1$ characters.

**Proof.** Let $P = S[\max(s_1, s_2)..\min(e_1, e_2)]$ be the intersection of $S[s_1..e_1]$ and $S[s_2..e_2]$. Note that $P$ is both $p_1$-periodic and $p_2$-periodic.

Assume that $P$ contains at least $p_1 + p_2 - 1$ characters.
CHAPTER 5. MAXIMAL (SUB-)REPETITIONS

Since \( P \) contains at least \( p_1 \)-characters, the maximal \( \delta \)-repetition \( S[s_1..e_1] \) is the \( p_1 \)-periodic extension of its substring \( P \). Also, since \( p_1 \) is the minimum period of \( S[s_1..e_1] \), no proper divisor of \( p_1 \) is a period of \( P \). Similarly, \( S[s_2..e_2] \) is the \( p_2 \)-periodic extension of \( P \) and no proper divisor of \( p_2 \) is a period of \( P \).

This implies that the minimum periods \( p_1 \) and \( p_2 \) are different.

However, the periodicity lemma stated in Theorem 2.2.1 proves that \( P \) is \( \gcd(p_1, p_2) \)-periodic. Since no proper divisor of \( p_1 \) or \( p_2 \) is a period of \( P \), this implies that the minimum periods \( p_1 \) and \( p_2 \) are equal. This, in turn, contradicts the assumption.

Therefore, the intersection \( P \) contains less than

\[
p_1 + p_2 - 1 \leq \left( 2 + \frac{\delta}{2} \right) L - 1
\]

characters. \( \square \)

With this lemma, we can prove the local upper bound for the number of maximal \( \delta \)-repetitions.

Proof of Lemma 5.2.4. By contradiction:

Assume that there are at least \( \left\lceil 3 + \frac{6}{\delta} \right\rceil \cdot \left\lfloor \log_{1 + \frac{\delta}{4}}(|S|) \right\rfloor + 1 \) different maximal \( \delta \)-repetitions \( S[s_k..e_k] \) whose extensions contain the index \( i \).

For each \( k \), we define \( p_k \) as the minimum period of \( S[s_k..e_k] \) and \( q_k \geq 2 + \delta \) as the corresponding exponent.

We will use the pigeonhole principle twice to show that there is a real number \( L \) such that of these maximal \( \delta \)-repetitions

- at least \( \left\lceil 3 + \frac{6}{\delta} \right\rceil + 1 \) have minimum periods \( L \leq p_k < \left( 1 + \frac{\delta}{4} \right) L \) and then
- at least 2 have both minimum periods \( L \leq p_k < \left( 1 + \frac{\delta}{4} \right) L \) and an intersection with at least \( (2 + \frac{\delta}{2}) L - 1 \) characters.

This, however, is ruled out by Lemma 5.2.6.

For each \( k \), the inequality \( 1 \leq p_k \) holds. This implies that

\[
0 = \log_{1 + \frac{\delta}{4}}(1) \leq \log_{1 + \frac{\delta}{4}}(p_k)
\]

holds.

On the other hand, the length of the maximal \( \delta \)-repetition \( S[s_k..e_k] \) is at least \( p_k \left( 2 + \delta \right) \) and at most \( |S| \). Hence, the inequality \( p_k \leq \frac{|S|}{2 + \delta} \) holds. This implies that the inequality

\[
\log_{1 + \frac{\delta}{4}}(p_k) \leq \log_{1 + \frac{\delta}{4}}\left( \frac{|S|}{2 + \delta} \right) \leq \left\lceil \log_{1 + \frac{\delta}{4}}(|S|) \right\rceil - \log_{1 + \frac{\delta}{4}}(2 + \delta)
\]
5.2. COMPRESSED UPPER BOUNDS

holds.

For $\delta \leq 8$, we get

$$
\left(1 + \frac{\delta}{4}\right)^2 = 1 + \frac{\delta}{2} + \frac{\delta^2}{16} = 1 + \frac{\delta}{2} + \frac{\delta \cdot \delta}{8} \leq 1 + \frac{\delta}{2} + \frac{\delta}{8} < 2 + \delta
$$

while for $\delta > 8$, we get

$$
\left(1 + \frac{\delta}{4}\right)^1 < 2 + \delta.
$$

Thus, we obtain the inequality

$$
\log_{1 + \frac{\delta}{4}}(p_k) < \begin{cases} 
\left\lfloor \log_{1 + \frac{\delta}{4}}(|S|) \right\rfloor - 1 & \text{if } \delta \leq 8 \\
\log_{1 + \frac{\delta}{4}}(|S|) & \text{if } \delta > 8.
\end{cases}
$$

We can now sort the maximal $\delta$-repetitions with regard to the values $\log_{1 + \frac{\delta}{4}}(p_k)$ into the $\left\lfloor \log_{1 + \frac{\delta}{4}}(|S|) \right\rfloor$ intervals

$$[n, n + 1)$$

with $n \in \{0, 1, 2, \ldots, \left\lfloor \log_{1 + \frac{\delta}{4}}(|S|) \right\rfloor - 1\}$.

If $\delta \leq 8$ holds, the last interval is empty and we use the additional interval to divide the interval $[0, 1)$ into $\{0\}$ and $(0, 1)$.

Since we divided the $\left\lfloor 3 + \frac{6}{\delta} \right\rfloor \cdot \left\lfloor \log_{1 + \frac{\delta}{4}}(|S|) \right\rfloor + 1$ maximal $\delta$-repetitions into $\left\lfloor \log_{1 + \frac{\delta}{4}}(|S|) \right\rfloor$ sets, the pigeonhole principle proves that there is at least one of these sets that contains at least

$$\left\lfloor \frac{\left\lfloor 3 + \frac{6}{\delta} \right\rfloor \cdot \left\lfloor \log_{1 + \frac{\delta}{4}}(|S|) \right\rfloor + 1}{\left\lfloor \log_{1 + \frac{\delta}{4}}(|S|) \right\rfloor} \right\rfloor = \left\lfloor 3 + \frac{6}{\delta} \right\rfloor + 1
$$

maximal $\delta$-repetitions.

Lemma 5.2.5 shows that this set is not the set of maximal $\delta$-repetitions with minimum period equal to 1. Therefore, if $\delta \leq 8$ holds, we can assume that the minimum periods are at least 2.

By definition of the sets, there is a natural number $L'$ such that the inequality $L' \leq \log_{1 + \frac{\delta}{4}}(p_k) < L' + 1$ holds for at least $\left\lfloor 3 + \frac{6}{\delta} \right\rfloor + 1$ maximal $\delta$-repetitions. Exponentiating this inequality yields that there is a real number

$L = (1 + \frac{\delta}{4})^{L'}$ such that $L \leq p_k < (1 + \frac{\delta}{4})L$ holds for these $\left\lfloor 3 + \frac{6}{\delta} \right\rfloor + 1$ maximal $\delta$-repetitions.

It remains to be proven that of these maximal $\delta$-repetitions, there are at least 2 that have an intersection with at least $(2 + \frac{\delta}{2})L - 1$ characters.
We define
\[ s_k' := \begin{cases} 
  i - \lfloor (2 + \delta)L \rfloor & \text{if } s_k < i - \lfloor (2 + \delta)L \rfloor \\
  s_k & \text{if } s_k \geq i - \lfloor (2 + \delta)L \rfloor .
\end{cases} \]

Since each maximal \( \delta \)-repetition contains at least \( p_k(2 + \delta) \) characters and \( L \leq p_k \) holds, we get that each maximal \( \delta \)-repetition has at least \( (2 + \delta)L \) characters. Therefore, for each \( k \), the string \( S[s_k..e_k] \) contains the substring \( S[s_k..s_k + \lfloor (2 + \delta)L \rfloor - 1] \).

Since the extensions of the maximal \( \delta \)-repetitions contain the index \( i \), we also obtain that for each \( k \), the string \( S[s_k..e_k] \) contains \( S[s_k..i - 1] \).

Therefore, for each \( k \), the string \( S[s_k'..s_k'] + \lfloor (2 + \delta)L \rfloor - 1 \) is \( p_k \)-periodic.

For each \( k \), the index \( s_k' \) is in the interval \([ i - \lfloor (2 + \delta)L \rfloor ..i + 1] \) of length \( \lfloor (2 + \delta)L \rfloor + 2 \).

We divide this interval into \([ 3 + \frac{6}{\delta} ] \) subintervals. If \( \delta > 8 \) holds, we obtain \( 4 < \frac{1}{3} \delta \). Without loss of generality, the inequality \( L \geq 1 \) holds and the subintervals have lengths
\[ \frac{\lfloor (2 + \delta)L \rfloor + 2}{\lfloor 3 + \frac{6}{\delta} \rfloor} < \frac{(2 + \delta)L + 2}{3} \leq \frac{(4 + \delta)L}{3} < \frac{(\frac{3}{2} \delta)L}{3} = \frac{\delta}{2} L. \]

Otherwise, \( \delta \leq 8 \) holds. In this case, the minimum periods are at least \( 2 \) and thus we can assume \( L \geq 2 \). In this case, the the subintervals have lengths
\[ \frac{\lfloor (2 + \delta)L \rfloor + 2}{\lfloor 3 + \frac{6}{\delta} \rfloor} < \frac{(2 + \delta)L + 2}{2 + \frac{6}{\delta}} \leq \frac{(3 + \delta)L}{(3 + \frac{\delta}{2})} = \frac{\delta}{2} L. \]
In either case, the subintervals have lengths less than \( \frac{\delta}{2} L \).

We sort the \([ 3 + \frac{6}{\delta} ] +1 \) maximal \( \delta \)-repetitions depending on their respective value \( s_k' \) into one of the \([ 3 + \frac{6}{\delta} ] \) subintervals. Since we have more maximal \( \delta \)-repetitions than subintervals, there is at least one subinterval that contains at least 2 maximal \( \delta \)-repetitions.

Let two of these maximal \( \delta \)-repetitions be \( S[s_m..e_m] \) and \( S[s_n..e_n] \). We can assume that \( s'_m \geq s'_n \) holds and thus \( 0 \leq s'_m - s'_n < \frac{\delta}{2} L \) holds as well.

By construction, the intersection of these two maximal \( \delta \)-repetitions contains the substring \( S[s_m'..s_m' + \lfloor (2 + \delta)L \rfloor - 1] \) of length
\[ s'_n + \lfloor (2 + \delta)L \rfloor - 1 - s'_m + 1 > \lfloor (2 + \delta)L \rfloor - \frac{\delta}{2} L \geq \left( 2 + \frac{\delta}{2} \right) L - 1. \]

This, however, contradicts Lemma 5.2.6. Therefore, there are at most \([ 3 + \frac{6}{\delta} ] \cdot \lfloor \log_{1 + \frac{4}{\delta}}(|S|) \rfloor \) different maximal \( \delta \)-repetitions \( S[s_k..e_k] \) whose extensions contain the index \( i \).

With the local upper bound of Theorem 5.2.4 and Lemma 5.2.3, we now get the global upper bound given by Theorem 5.2.1.
5.2. COMPRESSED UPPER BOUNDS

5.2.2 Upper Bound for Maximal $\delta$-Subrepetitions

In this subsection, we will prove the upper bound given in Theorem 5.2.2 for the number of distinct maximal $\delta$-subrepetitions without $q$-th powers in a string $S$ with string attractor $\Gamma$.

**Theorem 5.2.2.** Let $S$ be string with string attractor $\Gamma$. Let further $\delta > 0$ be a real number and $q \geq 2$ be a natural number.

Then, the string $S$ contains at most

$$((|\Gamma| + 2) \left[ 3 + \frac{4}{\delta} \right] \cdot \left\lfloor \log_{1 + \frac{\delta}{2q}}(|S|) \right\rfloor$$

distinct extended maximal $\delta$-subrepetitions without $q$-th powers.

The proof of Theorem 5.2.2 is, up to two exceptions, identical to the proof of Theorem 5.2.1. Firstly, the calculations use different numbers. Secondly, we have to replace Lemma 5.2.6 which caused the contradiction in the last subsection by a lemma that takes the highest power into account. We therefore need a variant of this lemma.

We will first prove the local version of this theorem.

**Theorem 5.2.7.** Let $S$ be a string and $-1 \leq i \leq |S|$ an index. Let further $\delta > 0$ be a real number and $q \geq 2$ be a natural number.

Then, there are at most

$$\left\lfloor 3 + \frac{4}{\delta} \right\rfloor \cdot \left\lfloor \log_{1 + \frac{\delta}{2q}}(|S|) \right\rfloor$$
different $q$-th power-free maximal $\delta$-subrepetitions $S[s_k..e_k]$ whose extensions contain the index $i$.

Similarly to the last subsection, this theorem is slightly stronger than both Theorem 5 in [81] and Theorem 8 in [82]. Also, similarly to the last subsection, we have to consider the two possible maximal $\delta$-subrepetitions with minimum period 1 separately.

As in the last subsection, we will again use the pigeonhole principle in order to show that if more of these maximal $\delta$-subrepetitions existed, two of them would be neighbors. Since the subrepetitions are allowed to have exponents of 2 or less, we cannot expect the intersections of two maximal $\delta$-subrepetitions to be at least twice as long as the longer minimum period. Thus, the difference between the two minimum periods have to be much smaller in this subsection than it was in the last subsection.

More formally, we will construct two maximal $\delta$-subrepetitions whose minimum periods are in the interval $(L, \left(1 + \frac{\delta}{2q}\right)L)$ for some real number $L$ and whose intersection contains at least $(1 + \frac{\delta}{2})L - 1$ characters.

However, we have already seen in Figure 5.1 that there are strings with only 4 LZ77-factors and linearly many maximal repetitions. Furthermore,
the maximal repetitions in these strings have similar minimum periods and large intersections. Thus, we cannot expect that these neighboring maximal $\delta$-subrepetitions lead to a contradiction if we allow exponents of 2 or less. Instead, we will show that the neighboring maximal $\delta$-subrepetitions only occur if we allow highly periodic substrings in these subrepetitions.

**Lemma 5.2.8.** Let $S$ be a string and $\delta > 0$ a real number. Let $S[s_1..e_1]$ and $S[s_2..e_2]$ be different maximal $\delta$-subrepetitions of $S$ with minimum periods $p_1$ and $p_2$, respectively.

Let $L$ be a real number and $q$ be a positive natural number such that both periods $p_1$ and $p_2$ are contained in the interval $\left[L, \left(1 + \frac{\delta}{2q}\right)L\right]$ and such that the intersection $S[\max(s_1, s_2)\ldots \min(e_1, e_2)]$ of $S[s_1..e_1]$ and $S[s_2..e_2]$ contains at least $\left(1 + \frac{\delta}{2}\right)L - 1$ characters.

Then, the intersection of $S[s_1..e_1]$ and $S[s_2..e_2]$ contains a $q$-th power.

**Proof.** For $q = 1$, there is nothing to prove. Thus, we only consider $q \geq 2$ in the remainder of the proof.

Define $s_{\text{max}} = \max(s_1, s_2)$ and $e_{\text{min}} = \min(e_1, e_2)$. Then, the intersection $S[\max(s_1, s_2)\ldots \min(e_1, e_2)]$ of $S[s_1..e_1]$ and $S[s_2..e_2]$ is given by $S[s_{\text{max}}..e_{\text{min}}]$. We further define $p_{\text{max}} = \max(p_1, p_2)$ and $p_{\Delta} = \max(p_1, p_2) - \min(p_1, p_2)$.

Let $i$ be a natural number in the interval $\left\{0, 1, 2, \ldots, \left\lceil \frac{q\delta - \delta}{2q}L \right\rceil - 2\right\}$. We will show that the three indices $s_{\text{max}} + i$, $s_{\text{max}} + i + p_{\Delta}$ and $s_{\text{max}} + i + p_{\text{max}}$ are contained in the intersection $S[s_{\text{max}}..e_{\text{min}}]$. Using the periodicity of the intersection, these indices therefore correspond to equal characters.

Firstly, we note that

$$s_{\text{max}} \leq s_{\text{max}} + i \leq s_{\text{max}} + i + p_{\Delta} < s_{\text{max}} + i + p_{\text{max}}$$

hold.

On the other hand, we also have the inequality

$$s_{\text{max}} + i + p_{\text{max}} \leq s_{\text{max}} + \left\lceil \frac{q\delta - \delta}{2q}L \right\rceil - 2 + \left\lceil \left(1 + \frac{\delta}{2q}\right)L \right\rceil - 1$$

$$\leq s_{\text{max}} + \left\lceil \left(\frac{q\delta - \delta}{2q} + 1 + \frac{\delta}{2q}\right)L \right\rceil - 2$$

$$\leq s_{\text{max}} + \left\lceil \left(1 + \frac{\delta}{2}\right)L \right\rceil - 2$$

$$\leq e_{\text{min}}.$$

This implies that the first $\left\lceil \frac{q\delta - \delta}{2q}L \right\rceil - 1 + p_{\Delta}$ characters of the intersection $S[s_{\text{max}}..e_{\text{min}}]$ are $p_{\Delta}$-periodic. Also, since $p_1$ and $p_2$ are contained in the interval $\left[L, \left(1 + \frac{\delta}{2q}\right)L\right]$, the difference $p_{\Delta}$ is bounded from above by $\left\lceil \frac{\delta}{2q}L \right\rceil - 1 \geq 1$. 

Also, we have

\[
|S[s_{\max}..e_{\min}]| \geq \left(1 + \frac{\delta}{2}\right)L - 1 \geq \left(1 + \frac{\delta}{2q}\right)L - 1 \geq p_{\max}.
\]

This implies that both \(\delta\)-subrepetitions are given by the \(p_1\)-periodic extension and the \(p_2\)-periodic extension of the intersection, respectively. In particular, the minimum periods \(p_1\) and \(p_2\) are different.

Hence, the first \(\frac{q\delta - \delta}{2q}L\) - 1 + \(p_\Delta\) characters of \(S[s_{\max}..e_{\min}]\) are a power with exponent of at least

\[
\frac{q\delta - \delta}{2q}L - 1 + p_\Delta = \frac{(q-1)\delta}{2q}L - 1 + 1 \\
\geq \frac{\left(\frac{\delta}{2q}L\right) - 1 + 1 - 1}{\frac{\delta}{2q}L} + 1 \\
= q.
\]

Hence, the intersection of \(S[s_1..e_1]\) and \(S[s_2..e_2]\) contains a \(q\)-th power. \(\square\)

With this lemma, we can prove the local upper bound for the number of maximal \(\delta\)-subrepetitions.

**Proof of Lemma 5.2.7.** By contradiction:

Assume that there are at least \(\left[3 + \frac{4}{\delta}\right] \cdot \left\lfloor \log_{1+\frac{\delta}{2q}}(|S|) \right\rfloor + 1\) different \(q\)-th power-free maximal \(\delta\)-subrepetitions \(S[s_k..e_k]\) whose extensions contain the index \(i\).

For each \(k\), we define \(p_k\) as the minimum period of \(S[s_k..e_k]\) and \(q_k \geq 1 + \delta\) as the corresponding exponent.

We will use the pigeonhole principle twice to show that there is a real number \(L\) such that of these maximal \(\delta\)-subrepetitions

- at least \(\left[3 + \frac{4}{\delta}\right] + 1\) have minimum periods \(L \leq p_k < \left(1 + \frac{\delta}{2q}\right)L\) and then
- at least 2 have both minimum periods \(L \leq p_k < \left(1 + \frac{\delta}{2q}\right)L\) and an intersection with at least \((1 + \frac{\delta}{2})L - 1\) characters.

This, however, is ruled out by Lemma 5.2.8.

For each \(k\), the inequality \(1 \leq p_k\) holds. This implies that

\[
0 = \log_{1+\frac{\delta}{2q}}(1) \leq \log_{1+\frac{\delta}{2q}}(p_k)
\]
holds.

On the other hand, the length of the maximal $\delta$-subrepetition $S[s_k..e_k]$ is at least $p_k(1 + \delta)$ and at most $|S|$. Hence the inequality $p_k \leq \frac{|S|}{1+\delta}$ holds. This implies that the inequality

$$\log_{1+\frac{\delta}{2q}}(p_k) \leq \log_{1+\frac{\delta}{2q}}\left(\frac{|S|}{1+\delta}\right) \leq \left[\log_{1+\frac{\delta}{2q}}(|S|)\right] - \log_{1+\frac{\delta}{2q}}(1+\delta)$$

holds.

For $\delta \leq 6$, we get with $q \geq 2$

$$\left(1 + \frac{\delta}{2q}\right)^2 = 1 + \delta + \frac{\delta^2}{4q^2} \leq 1 + \frac{\delta}{2} + \frac{\delta}{2} \cdot \frac{\delta}{4q} < 1 + \frac{\delta}{2} + \frac{\delta}{2} = 1 + \delta$$

while for $\delta > 6$, we get

$$\left(1 + \frac{\delta}{2q}\right) < 1 + \delta.$$

Thus, we obtain the inequality

$$\log_{1+\frac{\delta}{2q}}(p_k) < \begin{cases} \left[\log_{1+\frac{\delta}{2q}}(|S|)\right] - 1 & \text{if } \delta \leq 6 \\ \left[\log_{1+\frac{\delta}{2q}}(|S|)\right] & \text{if } \delta > 6. \end{cases}$$

We can now sort the maximal $\delta$-subrepetitions with regard to the values $\log_{1+\frac{\delta}{2q}}(p_k)$ into the $\left[\log_{1+\frac{\delta}{2q}}(|S|)\right]$ intervals

$$[n, n+1) \text{ with } n \in \{0, 1, 2, \ldots, \left[\log_{1+\frac{\delta}{2q}}(|S|)\right] - 1\}.$$

If $\delta \leq 6$ holds, the last interval is empty and we use the additional interval to divide the interval $[0, 1)$ into $\{0\}$ and $(0, 1)$.

Since we divided the $\left[3 + \frac{4}{\delta}\right] \cdot \left[\log_{1+\frac{\delta}{2q}}(|S|)\right] + 1$ maximal $\delta$-subrepetitions into $\left[\log_{1+\frac{\delta}{2q}}(|S|)\right]$ sets, the pigeonhole principle proves that there is at least one of these sets that contains at least

$$\left\lfloor\frac{\left[3 + \frac{4}{\delta}\right] \cdot \left[\log_{1+\frac{\delta}{2q}}(|S|)\right] + 1}{\left[\log_{1+\frac{\delta}{2q}}(|S|)\right]}\right\rfloor = \left\lfloor3 + \frac{4}{\delta}\right\rfloor + 1$$

maximal $\delta$-subrepetitions.

Lemma 5.2.5 shows that this set is not the set of maximal $\delta$-repetitions with minimum period equal to 1. Therefore, if $\delta \leq 6$ holds, we can assume that the minimum periods are at least 2.
By definition of the sets, there is a natural number $L'$ such that the inequality $L' \leq \log_{1 + \frac{\delta}{2q}} (p_k) < L' + 1$ holds for at least $\left\lfloor 3 + \frac{4}{\delta} \right\rfloor + 1$ maximal $\delta$-subrepetitions. Exponentiating this inequality yields that there is a real number $L = \left(1 + \frac{\delta}{2q}\right)^{L'}$ such that $L \leq p_k < \left(1 + \frac{\delta}{2q}\right) L$ holds for these $\left\lfloor 3 + \frac{4}{\delta} \right\rfloor + 1$ maximal $\delta$-subrepetitions.

It remains to be proven that of these maximal $\delta$-subrepetitions, there are at least 2 that have an intersection with at least $(1 + \frac{\delta}{2}) L - 1$ characters.

We define

$$s'_k := \begin{cases} 
i - \left\lfloor (1 + \delta) L \right\rfloor & \text{if } s_k < i - \left\lfloor (1 + \delta) L \right\rfloor, \\ s_k & \text{if } s_k \geq i - \left\lfloor (1 + \delta) L \right\rfloor. \end{cases}$$

Since each maximal $\delta$-subrepetition contains at least $p_k(1 + \delta)$ characters and $L \leq p_k$ holds, we get that each maximal $\delta$-subrepetition has at least $(1 + \delta) L$ characters. Therefore, for each $k$, the string $S[s_k..e_k]$ contains the substring $S[s_k..s_k + \left\lfloor (1 + \delta) L \right\rfloor - 1]$.

Since the extensions of the maximal $\delta$-subrepetitions contain the index $i$, we also obtain that for each $k$, the string $S[s_k..e_k]$ contains $S[s_k..i - 1]$.

Therefore, for each $k$, the index $s'_k$ is in the interval $[i - \left\lfloor (1 + \delta) L \right\rfloor..i + 1]$ of length $\left\lfloor (1 + \delta) L \right\rfloor + 2$.

We divide this interval into $\left\lfloor 3 + \frac{4}{\delta} \right\rfloor$ subintervals. If $\delta > 6$ holds, we obtain $3 < \frac{1}{2} \delta$. Without loss of generality, the inequality $L \geq 1$ holds and the subintervals have lengths

$$\frac{\left\lfloor (1 + \delta) L \right\rfloor + 2}{\left\lfloor 3 + \frac{4}{\delta} \right\rfloor} < \frac{(1 + \delta) L + 2}{3} \leq \frac{(3 + \delta) L}{3} < \frac{(\frac{3}{2} \delta) L}{3} = \frac{\delta}{2} L.$$

Otherwise, $\delta \leq 6$ holds. In this case, the minimum periods are at least 2 and thus we can assume $L \geq 2$. In this case, the the subintervals have lengths

$$\frac{\left\lfloor (1 + \delta) L \right\rfloor + 2}{\left\lfloor 3 + \frac{4}{\delta} \right\rfloor} < \frac{(1 + \delta) L + 2}{2 + \frac{4}{\delta}} \leq \frac{(2 + \delta) L}{(2 + \delta) \frac{2}{3}} = \frac{\delta}{2} L.$$

In either case, the subintervals have lengths less than $\frac{\delta}{2} L$.

We sort the $\left\lfloor 3 + \frac{4}{\delta} \right\rfloor + 1$ maximal $\delta$-subrepetitions depending on their respective value $s'_k$ into one of the $\left\lfloor 3 + \frac{4}{\delta} \right\rfloor$ subintervals. Since we have more maximal $\delta$-repetitions than subintervals, there is at least one subinterval that contains at least 2 maximal $\delta$-subrepetitions.

Let two of these maximal $\delta$-subrepetitions be $S[s_m..e_m]$ and $S[s_n..e_n]$. We can assume that $s'_m \geq s'_n$ holds and thus $0 \leq s'_m - s'_n < \frac{\delta}{2} L$ holds as well.
By construction, the intersection of these two maximal \( \delta \)-subrepetitions contains the substring \( S[s'_m..s'_n + [(1 + \delta)L] - 1] \) of length

\[
s'_n + [(1 + \delta)L] - 1 - s'_m + 1 > [(1 + \delta)L] - \frac{\delta}{2}L \geq \left( 1 + \frac{\delta}{2} \right) L - 1.
\]

By prerequisite, the intersection is \( q \)-th power-free. This, however, contradicts Lemma 5.2.8. Therefore, there are at most

\[
\left\lfloor 3 + \frac{q}{\delta} \right\rfloor \cdot \left\lfloor \log_{1 + \frac{\delta}{2q}}(|S|) \right\rfloor
\]
different maximal \( \delta \)-subrepetitions \( S[s_k..e_k] \) whose extensions contain the index \( i \).

With the local upper bound of Theorem 5.2.7 and Lemma 5.2.3, we now get the global upper bound given by Theorem 5.2.2.

### 5.3 Tightness

In this section, we will prove that for fixed \( \delta \) and \( q \geq \lfloor \delta + 2 \rfloor \), the upper bounds given by the Theorems 5.2.1 and 5.2.2 are, up to a constant factor, tight. We further show that this tightness also holds for binary strings.

Slightly weaker versions of the results of this section were originally published by the author of this thesis in [81] and its extended journal article in [82]. The results in this thesis avoid powers with slightly lesser exponent and are also explicitly extended to subrepetitions.

Since it is more convenient to use the LZ77-factors \( z \) instead of the string attractor \( \Gamma \), the results in this section use the number of LZ77-factors. Since \( z \in \mathcal{O}(\Gamma) \) holds, this also leads to slightly stronger results.

We first prove the tightness for an unbounded alphabet. Afterwards, we will use Theorem 4.0.4 to translate the tightness to binary strings.

**Lemma 5.3.1.** Let \( \delta \) be a positive real number.

For all positive integers \( n \) and \( c \) with \( \log_2(n) \leq c \leq n \), there is a string \( S \)

- with \( \log(|S|) \in \mathcal{O}(c) \),
- with \( \mathcal{O}(n) \) LZ77-factors,
- with \( \Omega(cn) \) distinct maximal \( \delta \)-subrepetitions and
- without \( \lfloor \delta + 2 \rfloor \)-th powers.

The implicit constants only depend on \( \delta \).
If we consider both $\delta$ and the maximal power in the Theorems 5.2.1 and 5.2.2 to be a constant, both upper bounds simplify to
\[
\#(\text{extended maximal } \delta\text{-}(\text{sub-})\text{repetitions}) \\
\in \mathcal{O}(\#(\text{LZ77-factors}) \cdot \log(\text{string length})).
\]

Therefore, Lemma 5.3.1 proves that both Theorems are, up to a constant factor, tight.

**Proof of Lemma 5.3.1.** Define the exponent $q = \lfloor \delta + 2 \rfloor$ and the strings
\[
R_1 = (\sigma_1)^q \\
R_{i+1} = (\hat{R}_i \sigma_i)^q
\]

By construction, the strings $\hat{R}_i$ are constructed from $i$ characters and $i$ exponentiations and therefore decompose into $2^i$ LZ77-factors. The length of $\hat{R}_i$ is bounded by $\hat{R}_i < q^i$. Also, these strings do not contain any $q$-th powers.

Furthermore, the minimum period of $\hat{R}_i$ is $\hat{R}_{i-1} + 1$ and its length is $q \left( \hat{R}_{i-1} + 1 \right) - 1$. Therefore, both the quotient $\frac{|\hat{R}_{i+1}|}{|\hat{R}_i|}$ and the exponent of $\hat{R}_i$ converge to $q = \lfloor \delta + 2 \rfloor$.

Thus, for $\varepsilon = \frac{q^{-1}(1+\delta)}{2}$, there is an integer $i_0$ which only depends on $\delta$, such that for all $i \geq i_0$, the inequalities $|\hat{R}_i| \geq \frac{1}{\varepsilon}$ and $\frac{|\hat{R}_{i+1}|}{|\hat{R}_i|} \geq 1 + \delta + \varepsilon$ hold and the string $\hat{R}_i$ is a power with exponent of at least $1 + \delta + \varepsilon$.

In particular, for each non-negative integer $i \geq 0$, the inequality
\[
|\hat{R}_{i_0+i}| \geq |\hat{R}_{i_0+i}|(1 + \delta + \varepsilon)^i \geq \frac{1}{\varepsilon}(1 + \delta + \varepsilon)^i
\]
holds.

Define $k = \lceil \log_{1+\delta+\varepsilon}(2) \rceil$. Then for each non-negative integer $i \geq 0$, the inequality
\[
\varepsilon|\hat{R}_{i_0+kc+i}| \geq (1 + \delta + \varepsilon)^{kc+i} \geq 2^c \geq n
\]
holds. Since the strings $\hat{R}_{i_0+kc+i}$ are powers with exponents of at least $1 + \delta + \varepsilon$, we can remove up to the first $n$ characters and still have powers with exponent of at least $1 + \delta$.

Define $\hat{R} = \hat{R}_{i_0+kc+c}$. Then, for each starting index $s$ with $0 \leq s < n$, the string
\[
\hat{R}[s..|\hat{R}| - 1]
\]
contains for all $i$ with $0 \leq i < c$, the maximal $\delta$-subrepetitions

$$\hat{R}_{i_0 + k c + i}[s..|\hat{R}_{i_0 + k c + i}| - 1].$$

For each string $\hat{R}_{i_0 + k c + i}[s..|\hat{R}_{i_0 + k c + i}| - 1]$, we can retrieve the number $i$ from the last character in the string and then we can retrieve the number $s$ from its length. Therefore, the $cn$ maximal $\delta$-subrepetitions defined in the last paragraph are distinct.

Now we define

$$S = $0\hat{R}s_1 \prod_{j=1}^{n-1} \hat{R}[j..|\hat{R}| - 1]$.$$

By construction, the string $S$ does not contain any $[\delta + 2]$-th powers. The separators $s_i$ with $i \in \{0, 1, \ldots, n - 1\}$ guarantee that the all off the maximal $\delta$-subrepetitions in $\hat{R}[s..|\hat{R}| - 1]$ are still maximal in $S$. Hence, the string $S$ contains at least $cn$ distinct maximal $\delta$-subrepetitions. Since each separator and each substring needs exactly one LZ77-factor, the string $S$ can be built with $2(i_0 + k c + c) + (n + 1) + (n - 1)$ LZ77-factors. Since $k$ is a constant and $c \leq n$ holds, the number of LZ77-factors is in $O(n)$. Lastly, the length of $S$ is at most $n \cdot q^{i_0 + k c + c} + n + 1$. Therefore, we obtain the bound

$$\log_2(|S|) \leq 2 \log_2(q^{i_0 + k c + c}) = 2(i_0 + (k + 1)c) \log_2(q) \in O(c).$$

Since $\delta$-repetitions are $(\delta + 1)$-subrepetitions, we can easily translate this lemma to $\delta$-repetitions. Note that the following lemma avoids even smaller powers than the corresponding theorems in [81] and [82]. However, the implicit constants in this thesis are worse.

**Corollary 5.3.2.** Let $\delta$ be a positive real number.

For all positive integers $n$ and $c$ with $\log_2(n) \leq c \leq n$, there is a string $S$

- with $\log(|S|) \in O(c)$,
- with $O(n)$ LZ77-factors,
- with $\Omega(cn)$ distinct maximal $\delta$-repetitions and
- without $[\delta + 3]$-th powers.

The implicit constants only depend on $\delta$. 

\[\]
Next, we will use the morphisms defined in Chapter 4 in order to translate this lemma to binary strings.

Firstly, we show that strongly invertible, uniform morphisms do not decrease the number of extended maximal $\delta$-repetitions.

**Lemma 5.3.3.** Let $S$ be a string over the alphabet $\Sigma$ and let $\varphi$ be strongly invertible, $k$-uniform morphism from $\Sigma$ to an alphabet $\Sigma'$.

Then distinct maximal $\delta$-repetitions in $S$ correspond to distinct maximal $\delta$-repetitions in the image of $\varphi(S)$.

**Proof.** The uniformity of $\varphi$ guarantees that $\delta$-repetitions with minimum period $p$ in $S$ naturally translate into $\delta$-repetitions with period $kp$ in $\varphi(S)$. Note that the period $kp$ is not necessarily minimal. For example, for $\varphi(a) = 00$, the minimum period of $\varphi(a^3)$ is only 1.

However, since the period $kp$ is less than half of the length $|\varphi(S)|$, the periodicity lemma guarantees that the minimum period of $|\varphi(S)|$ is a divisor of $kp$.

Note that the images of maximal $\delta$-repetitions are not necessarily maximal. Consider for example the morphism $\varphi(a) = 01$, $\varphi(b) = 02$. The maximal 1-repetition $aaa$ in the string $baaab$ is mapped to the maximal 1-repetition $010101$ in $0201010102$ which has the periodic extension $0101010$. Thus, the image of the maximal 1-repetition $aa$ is not maximal.

Let $\sigma_lP\sigma_r$ be an extended maximal $\delta$-repetition with minimum period $p$ in $S$. Then, the image $\varphi(P)$ has a minimum period that is a divisor of $kp$. Consider the periodic extension of $\varphi(P)$ in $\varphi(\sigma_lP\sigma_r)$. Since $\varphi$ is strongly invertible, we can retrieve the preimage of this extension. Furthermore, by uniformity, this extension is $p$-periodic. Hence the preimage of the extension is $P$.

Since we can retrieve maximal $\delta$-repetitions from the periodic extension of their image, distinct maximal $\delta$-repetitions in $S$ naturally correspond to distinct maximal $\delta$-repetitions in $\varphi(S)$. \hfill $\square$

Theorem 4.0.4 now translates Lemma 5.3.1 from strings over an unbounded alphabet to strings over a binary alphabet.

**Theorem 5.3.4.** Let $\delta$ be a positive real number.

For all positive integers $n$ and $c$ with $\log_2(n) \leq c \leq n$, there is a binary string $S$

- with $\log(|S|) \in O(c)$,
- with $O(n)$ LZ77-factors,
• with $\Omega(cn)$ distinct maximal $\delta$-repetitions and
• without $\lfloor \delta + 3 \rfloor$-th powers.

The implicit constants only depend on $\delta$.

5.4 Conclusion

In this chapter, we introduced and proved upper bounds for the number of distinct extended maximal $\delta$-(sub-)repetitions. For fixed $\delta > 0$ and $q = \lfloor \delta + 3 \rfloor$ both the maximal number of distinct maximal $\delta$-repetitions and the maximal number of distinct maximal $\delta$-subrepetitions without $q$-th powers is in

$$\Theta\left(\#(LZ77-factors) \cdot \log(\text{string length})\right).$$

We have seen in Chapter 4 that the implicit constants given by the alphabet reduction do not depend on the size of the alphabet of the preimage. Therefore, the bound given in the last paragraph holds for all alphabets with at least two characters and their implicit constants only depend on $\delta$.

For variable $\delta > 0$ and $q \geq 2$, we get the upper bounds

$$\left(\left|\Gamma\right| + 2\right) \left[3 + \frac{6}{\delta}\right] \cdot \left[\log_{1+\frac{1}{\delta}}(\left|S\right|)\right]$$

for the number of distinct extended maximal $\delta$-repetitions and

$$\left(\left|\Gamma\right| + 2\right) \left[3 + \frac{4}{\delta}\right] \cdot \left[\log_{1+\frac{1}{\delta}}(\left|S\right|)\right]$$

for the number of distinct extended maximal $\delta$-subrepetitions without $q$-th powers in a string $S$ with string attractor $\Gamma$.

Since $\log(1 + x) \approx x$ holds for small values of $x$. We obtain that for small $\delta$ and large $q$ these upper bounds are $O \left(\frac{\Gamma \log(\left|S\right|)}{\delta^2}\right)$ for the extended maximal $\delta$-repetitions and $O \left(\frac{q^{\Gamma \log(\left|S\right|)}}{\delta^2}\right)$ for the extended maximal $\delta$-subrepetitions.

In the uncompressed case, the upper bound for the number of maximal $\delta$-subrepetitions as proven in [63] had divisor of $\delta^2$ as well. In [27] and [46], this was improved to $\delta$. It might be possible to improve this divisor in the compressed upper bounds as well. However, in order to improve this bound, it seems not to be sufficient to study the relationships of pairs of maximal $\delta$-(sub-)repetitions with the Lemmata 5.2.6 and 5.2.8.

Also, there is currently no efficient algorithm for reporting all maximal $\delta$-subrepetitions in compressed strings. While such an algorithm with polynomial time with respect to the input is not possible, there might be an algorithm with polynomial time with respect to its output.
Chapter 6

Cadences

The second periodic pattern we consider in this thesis is the subcadence. We will also consider some of its variants like $k$-subcadences, maximal subcadences, even/odd subcadences, $L$-$R$-cadences, cadences and some combinations of these variants.

In this chapter, we will present counting and reporting algorithms on uncompressed strings that were published by Funakoshi and the author of this thesis in [41] at the 37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020). We will also present some hardness results and detection algorithms on grammar-compressed strings that were published by the author of this thesis in [80] at the 14th-15th International Conference on Language and Automata Theory and Applications (LATA 2020 & 2021). Furthermore, we will prove some as yet unpublished results as the $NP$-hardness of $k$-subcadence detection in compressed strings and bounds for $k$-subcadences and some of their variants.

Subcadences are arithmetic subsequences in which all characters are equal. We will only consider $k$-subcadences which have a fixed length $k$. Most results in this thesis consider 3-subcadences and their variants.

For example, in the string $S = 010111$ in Figure 6.1, the arithmetic subsequences $(1, 2, 3)$ and $(3, 1, 3)$ corresponding to the index sets $\{1, 3, 5\}$ and $\{3, 4, 5\}$, respectively, are both 3-subcadences with underlying character 1. The characters which are not represented by an index of the arithmetic progression are irrelevant.

A subcadence is maximal if neither the extension to the left nor the extension to the right is a subcadence. In this thesis, we will consider the cadence, a special case of the maximal subcadence in which neither of the two extensions is contained in the underlying string. In this way, the cadence is structurally maximal.

In the example above, neither of the extensions $(1, 2, 4)$ and $(3, 1, 4)$ to
Figure 6.1: The string $S = 010111$ with its maximal 3-subcadences $(1, 2, 3)$ and $(3, 1, 3)$, their extensions $(1, 2, 4)$ and $(3, 1, 4)$ to the right and their extensions $(-1, 2, 4)$ and $(2, 1, 4)$ to the left.

the right is a subcadence, since neither of the extensions is contained in the string. Similarly, the extension $(-1, 2, 4)$ to the left is not contained in the string. The extension $(2, 1, 4)$ is contained in the string, but $S[2] = 0$ holds. Therefore, neither of the extensions is a subcadence. This also implies that both $(1, 2, 3)$ and $(3, 1, 3)$ are maximal subcadences but only $(1, 2, 3)$ is a cadence.

As a useful simplification, we will also consider $L$-$R$-subcadences which are subcadences that start in a given interval $L$ and end in a given interval $R$. In terms of 3-subcadences, it will turn out that it is very useful to separately consider the even 3-subcadences which start with an even index and odd 3-subcadences which start with an odd index.

6.1 Related Work

Subcadences were already studied in 1927 by van der Waerden who proved in [98] (written in German) that for fixed alphabet size $|\Sigma|$ and fixed $k$, every sufficiently long string contains a $k$-subcadence. We denote the smallest length that guarantees a $k$-subcadence by $m(k, |\Sigma|)$.

Gowers proves in [47] that the upper bound

$$m(k, |\Sigma|) \leq 2^{2^{2k^2}}2^{k+9}$$

holds. On the other hand, for $|\Sigma| = 2$ and some constant $\varepsilon > 0$, we have the lower bound $m(k, 2) \in \Omega \left( \frac{2^k}{k^\varepsilon} \right)$ as shown by Szabó in [94]. If $k - 1$ is prime, we also have Berlekamp’s slightly stronger lower bound $m(k, 2) \geq (k - 1)2^{k-1}$ from [11].
Since this gap is very large it seems reasonable to study the off-diagonal van der Waerden number $m(k_1, k_2; 2)$, which is the smallest length that guarantees either a $k_1$-subcadence with character 0 or a $k_2$-subcadence with character 1 in a binary string. It thereby is a generalization of $m(k, 2)$. Very recently, the bounds for $m(3, k_2; 2)$ were improved. Schoen proves in [91] that $m(3, k_2; 2)$ is bounded by $2^{O(k^c)}$ for some $c < 1$. On the other hand, Green proves in [48] that $m(3, k_2; 2)$ is bounded by $k^{\Omega(\sqrt{\log \log k})}$. This lower bound was already improved by Hunter who proves in [53] that $m(3, k_2; 2)$ is even bounded by $k^{\Omega(\log \log k)}$.

Van der Waerden considered the subcadences as arithmetic progression in subsets of the natural numbers. The term “cadence” in the context of string patterns was first used about 40 years later by Gardelle and Guilbaud in [45] (written in French). The “cadence” was redefined by Pin in [70, Chapter 3.3] and Amir et al. in [4]. In this thesis, we will use the definition of Amir which is the most restrictive version.

Subcadences can also naturally be generalized to arithmetic subsequences whose corresponding characters should match a given pattern. These equidistant subsequences were recently introduced and studied by Funakoshi et al. in [40]. They also show that some algorithms for the detection of subcadences and cadences can be adapted to detect equidistant subsequences with a given pattern.

6.2 Bounds

In this section, we will present the lower bounds and upper bounds for $k$-subcadences, $L-R-k$-cadences and $k$-cadences with fixed $k$ in a string $S$ over the alphabet $|\Sigma|$.

In particular, we will show that these bounds all depend on the length of $S$ and they can in general not be improved by the additional knowledge about the compressed size of the string.

Since each $k$-subcadence is given by its starting index $i$ and its distance $d$, the number of $k$-subcadences is bounded from above by $|S|^2$.

We will now prove that for fixed length $k$ and fixed alphabet size $|\Sigma|$, the number of $k$-subcadences is always quadratic with respect to the length of the underlying string.

We consider van der Waerden’s bound $m = m(k, |\Sigma|)$ such that each string of length $m$ has at least one $k$-subcadence. We define the set $Q$ of arithmetic subsequences of length $m$ in $S$ with starting index $0 \leq i' < \frac{|S|}{m+1}$ and distance $1 \leq d' < \frac{|S|}{m+1}$. Since arithmetic subsequences of these arithmetic subsequences
of \( S \) are also arithmetic subsequences of \( S \) directly, \( k \)-subcadences of the arithmetic subsequences in \( Q \) are also \( k \)-subcadences of \( S \).

Let \((i, d, k)\) be a \( k \)-subcadence in \( S \) that is also in the arithmetic subsequence \((i', d', m)\) in \( Q \). Let \( i \) be the \( s \)-th character in \((i', d', m)\) and \( i + d \) be the \( e \)-th character in \((i', d', m)\). Since \((i', d', m)\) has length \( m \), the inequality \( 0 \leq s < e < m \) holds. On the other hand for given values \( s \) and \( e \) with \( 0 \leq s < e < m \), both

\[
i' + sd' = i \\
i' + ed' = i + d
\]

hold. Therefore, we can reobtain \( d' = \frac{d}{e-s} \) and then \( i' = i - sd' \). This implies that \((i, d, k)\) is contained in at most \( m^2 \) arithmetic subsequences in \( Q \).

The set \( Q \) contains \( \Theta \left( \frac{|S|}{m^2} \right) \) arithmetic subsequences and each of these subsequences contains at least one \( k \)-subcadence. Furthermore, each \( k \)-subcadence is in at most \( m^2 \) arithmetic subsequences in \( Q \). Therefore, the string \( S \) contains \( \Omega \left( \frac{|S|}{m^2} \right) \) \( k \)-subcadences.

Therefore, the following theorem holds.

**Theorem 6.2.1.** For a fixed length \( k \geq 2 \) and alphabet size \( |\Sigma| \geq 1 \), the number of \( k \)-subcadences in a string \( S \) over \( \Sigma \) is in \( \Theta(|S|^2) \).

For a fixed character, \( k \)-subcadences can clearly be avoided by avoiding that character. If the fixed character has \( n \) occurrences, the number of 2-subcadences with the character is \( \binom{n}{2} \) since each pair of equal characters is a 2-subcadence.

Somewhat surprisingly, 3-subcadences with a given character can be avoided even if the character occurs \( \Theta(|S| \log_3(2)) \) times. Consider the following string which is inspired by the Cantor set. Figure 6.2 shows the first five Cantor strings scaled to the same width. By marking all occurrences of the character 1 in the Cantor strings, these strings naturally correspond to the sets that are given by the iterative creation of the Cantor set.

**Definition 6.2.2.** For a given integer \( n \) we define the Cantor string \( C(n) \) by

\[
C(0) = 1 \\
c(i+1) = C(i)Z(i)C(i) \\
Z(0) = 0 \\
Z(i+1) = Z(i)Z(i)Z(i)
\]

**Theorem 6.2.3.** For each \( i \), the Cantor string \( C(i) \) contains the character 1 exactly \( |C(i)|^{\log_3(2)} \) times and does not contain any 3-subcadence with the character 1.
6.2. BOUNDS

Figure 6.2: The Cantor strings $C(i)$ for $0 \leq i \leq 4$. The character 0 is denoted by an empty rectangle and the character 1 is denoted by a filled black rectangle. The strings are scaled to the same width regardless of their actual length.

Proof. By induction, it is easy to show that for each $i$, the strings $C(i)$ and $A(i)$ have the length $3^i$ and that $C(i)$ contains $2^i$ times the character 1. Thus, the Cantor string contains the character 1 exactly $|C(i)| \log_3(2)$ times.

In order to show that the Cantor string does not contain any 3-subcadence, we will show that for two indices with corresponding character 1 their mean does not correspond to a 1.

Let $s < e$ be two indices corresponding to a character 1. Consider the string $S[s..e]$ and let $i'$ be the smallest integer such that a copy of $S[s..e]$ is a substring of $C(i' - 1)$. By construction, the indices $s$ and $e$ cannot do not belong to the same copy of $C(i' - 1)$. Therefore, we obtain the inequality $0 \leq s < |C(i' - 1)|$ and since $|C(i' - 1)| = |Z(i' - 1)|$ holds also $2|C(i' - 1)| \leq e < 3|C(i' - 1)|$. Therefore, the mean of $s$ and $e$ fulfills the inequality $|C(i' - 1)| \leq \frac{s + e}{2} < 2|C(i' - 1)|$. This implies that even if $\frac{s + e}{2}$ is an integer, it corresponds to a character in $Z(i' - 1)$ and therefore not to a 1. Hence, the string $C(i)$ does not contain a 3-subcadence with the character 1.

Since $k$-cadences and $L\text{-}R\text{-}k$-cadences are $k$-subcadences with additional properties, they are even easier to avoid than $k$-subcadences.

On the other hand, consider the string $S = 1^{[S]}$. Each $k$-subcadence $(i, d, k)$ with $0 \leq i < \frac{|S|}{k}$ and $\frac{|S|}{k} \leq d < \frac{|S|}{k} + \frac{|S|}{k^2}$ fulfills the two inequalities $i - d < 0 \leq i$ and $i + (k - 1)d < |S| \leq i + kd$. Therefore, each of these $\Theta\left(\left(\frac{|S|}{k^2}\right)^2\right)$ $k$-subcadences is a $k$-cadence. Hence, for fixed $k$, there can be $\Theta(|S|^2)$ $k$-cadences.
By construction of these $k$-cadences, the characters with indices between $|S|/k^2$ and $|S|/k$ were not considered. Therefore, the strings of the form

$$1^{|S|/k^2}T\left(\left\lfloor \frac{|S|}{k} \right\rfloor - \left\lceil \frac{|S|}{k^2} \right\rceil \right) 1^{|S| - \left\lfloor \frac{|S|}{k^2} \right\rfloor}$$

where $T\left(\left\lfloor \frac{|S|}{k} \right\rfloor - \left\lceil \frac{|S|}{k^2} \right\rceil \right)$ is an arbitrary string of length $\left\lfloor \frac{|S|}{k} \right\rfloor - \left\lceil \frac{|S|}{k^2} \right\rceil$ also have $\Theta\left(\frac{|S|^2}{k}\right)$ $k$-cadences.

Furthermore, for a given candidate $(i, d, k)$ for a $k$-cadences in a random string, the probability that this candidate is a $k$-cadence with the character $1 \in \Sigma$ is $\frac{1}{|\Sigma|}$. Since there are $\Theta(|S|^2)$ of these candidates for fixed $k$ and fixed $\Sigma$, we should expect $\Theta(|S|^2)$ $k$-cadences in a random string $S$.

Since the string $1^{|S|}$ can be compressed with two LZ77-factors while random strings are generally incompressible, we obtain the following theorem.

**Theorem 6.2.4.** For a fixed length $k \geq 2$ and alphabet size $|\Sigma| \geq 1$, the number of $k$-cadences with a given character in a string $S$ over $\Sigma$ is in $O(|S|^2)$.

This bound cannot be improved by using the compressed size of $S$.

Next, we consider $L-R-k$-cadences. Since each $k$-subcadence is uniquely defined by its starting index and its ending index, a string can have at most $|L| \cdot |R|$ $L-R-k$-cadences.

Now consider the same string $1^n$ with the non-overlapping intervals $L$ and $R = \{r_{\min}, \ldots, r_{\max}\}$. For each $k$-subcadence $(i, d, k)$ with $i \in L$ and $\frac{r_{\min}-i}{k-1} \leq d \leq \frac{r_{\max}-i}{k-1}$, we have the inequality $r_{\min} \leq i + (k-1)d \leq r_{\max}$.

Therefore, each of these $\Theta\left(|L| \cdot \frac{|R|}{k}\right)$ $k$-subcadences is an $L-R-k$-cadence. Hence, for fixed $k$, there can be $\Theta\left(|L| \cdot |R|\right)$ $L-R-k$-cadences with the character 1.

Similarly to the $k$-cadences, for a random string, we should expect that of the $\Theta\left(|L| \cdot \frac{|R|}{k}\right)$ candidates for $L-R-k$-cadences given by the last paragraph, there are $\Theta\left(|L| \cdot \frac{|R|}{k} \cdot \frac{1}{|\Sigma|^2}\right)$ $L-R-k$-cadences with the character 1.

Therefore, there are no nontrivial upper bounds for the number of $L-R-k$-cadences in compressible strings.

**Theorem 6.2.5.** For a fixed length $k \geq 2$ and alphabet size $|\Sigma| \geq 1$ as well as two non-overlapping intervals $L$ and $R$, the number of $L-R-k$-cadences with a given character in a string $S$ over $\Sigma$ is in $O(|L| \cdot |R|)$.

This bound cannot be improved by using the compressed size of $S$.

In the remainder of the section, we will show that both $k$-cadences and $L-R-k$-cadences with an arbitrary character are avoidable. In particular,
we will show that this is true both in strings with a constant number of LZ77-factors and, unless both $k$ and the alphabet size $|\Sigma|$ are too small, in strings in which a constant portion of the string is incompressible.

By construction, the starting index of a $k$-cadence is in the first $k$-th of the string and its ending index is in the last $k$-th. Therefore, for $k \geq 3$, the string

$$S = 0^\left\lfloor \frac{|S|}{k} \right\rfloor T \left( |S| - 2 \left\lfloor \frac{|S|}{k} \right\rfloor \right)^1 1^\left\lceil \frac{|S|}{k} \right\rceil$$

where $T \left( |S| - 2 \left\lfloor \frac{|S|}{k} \right\rfloor \right)$ is an arbitrary string of length $S - 2 \left\lfloor \frac{|S|}{k} \right\rfloor$ does not contain any $k$-cadences. Also, if $T \left( |S| - 2 \left\lfloor \frac{|S|}{k} \right\rfloor \right)$ is constant, then $S$ only has 4 LZ77-factors. If $T \left( |S| - 2 \left\lfloor \frac{|S|}{k} \right\rfloor \right)$ is random, then a constant portion of the $S$ is generally incompressible.

**Theorem 6.2.6.** For a fixed length $k \geq 3$ and alphabet size $|\Sigma| \geq 2$, the number of $k$-cadences with an arbitrary character in a string $S$ over $\Sigma$ can be 0.

There is a constant $c$ which depends on $k$, $\Sigma$ and the chosen compression such that this bound cannot be improved by using the compressed size of $S$ unless the compressed size of $S$ is at least $c|S|$.

For $k \geq 2$ and $\Sigma \geq 3$, we consider the string

$$S = 0^\left\lfloor \frac{|S|}{k} \right\rfloor T \left( |S| - \left\lfloor \frac{|S|}{k} \right\rfloor \right)^1$$

where $T \left( |S| - \left\lfloor \frac{|S|}{k} \right\rfloor \right)$ is an arbitrary string of length $S - \left\lfloor \frac{|S|}{k} \right\rfloor$ over the alphabet $\Sigma \setminus \{0\}$. By construction, the string $S$ does not contain $k$-cadences. Also, if $T \left( |S| - \left\lfloor \frac{|S|}{k} \right\rfloor \right)$ is constant, then $S$ only has 4 LZ77-factors. On the other hand, if $T \left( |S| - \left\lfloor \frac{|S|}{k} \right\rfloor \right)$ is random, a constant portion of the $S$ is generally incompressible.

**Theorem 6.2.7.** For a fixed length $k \geq 2$ and alphabet size $|\Sigma| \geq 3$, the number of $k$-cadences with an arbitrary character in a string $S$ over $\Sigma$ can be 0.

There is a constant $c$ which depends on $k$, $\Sigma$ and the chosen compression such that this bound cannot be improved by using the compressed size of $S$ unless the compressed size of $S$ is at least $c|S|$.

For the $L$-$R$-$k$-cadences, we first consider $k \geq 4$. Let $|S|$ be an integer and $L = \{l_{\min}, \ldots, l_{\max}\}$ and $R = \{r_{\min}, \ldots, r_{\max}\}$ be two non-overlapping
intervals. Let \( m \) be an index such that \( l_{\text{max}} < m \leq r_{\text{min}} \) holds. We define
\[
\Delta_l = \min \left( \left\lfloor \frac{|S|}{3} \right\rfloor , m - 1 \right)
\]
and \( \Delta_r = \min \left( \left\lfloor \frac{|S|}{3} \right\rfloor , |S| - m \right) \) and consider the string
\[
S = T(m - 1 - \Delta_l) \cdot 0^\Delta_l \cdot 1^\Delta_r \cdot T(|S| - m - \Delta_r)
\]
where \( T(m - 1 - \Delta_l) \) and \( T(|S| - m - \Delta_r) \) are arbitrary strings with lengths \( m - 1 - \Delta_l \) and \( |S| - m - \Delta_r \), respectively.

By definition, no \( L-R-k \)-cadence can contain both a 0 and a 1. Thus, by position of \( i \), each \( L-R-k \)-cadence have to skip over the whole substring \( 0^{\Delta_l} \) or over the substring \( 1^{\Delta_r} \). I.e. the \( L-R-k \)-cadence has to contain both at least one character preceding the substring and at least one character succeeding the substring but no character from the substring itself. If one of the minima \( \Delta_l \) and \( \Delta_r \) is not \( \left\lfloor \frac{|S|}{3} \right\rfloor \), the corresponding substring cannot be skipped, because there are either no preceding characters or no succeeding characters.

Therefore, the distance of an \( L-R-k \)-cadence has to be at least \( \left\lfloor \frac{|S|}{3} \right\rfloor + 1 \).

Since \( k \geq 4 \) holds, this implies that for a given starting \( i \), the ending index is at least \( i + 3 \left( \left\lfloor \frac{|S|}{3} \right\rfloor + 1 \right) \geq S \). Therefore, the string \( S \) which contains at least \( \left\lfloor \frac{|S|}{3} \right\rfloor \) arbitrary characters, cannot contain an \( L-R-k \)-cadence. If the two substrings \( T(m - 1 - \Delta_l) \) and \( T(|S| - m - \Delta_r) \) are random, then a constant portion of \( S \) is generally incompressible.

**Theorem 6.2.8.** For a fixed length \( k \geq 4 \) and alphabet size \( |\Sigma| \geq 2 \), the number of \( L-R-k \)-cadences with an arbitrary character in a string \( S \) over \( \Sigma \) can be 0.

There is a constant \( c \) which depends on \( k, \Sigma \) and the chosen compression such that this bound cannot be improved by using the compressed size of \( S \) unless the compressed size of \( S \) is at least \( c|S| \).

Let \( L = \{l_{\text{min}}, \ldots, l_{\text{max}}\} \) and \( R \) be two non-overlapping intervals. Without loss of generality \( |L| \leq |R| \) holds. Let further \( k \geq 2 \) and \( |\Sigma| \geq 3 \) be two integers.

Consider the string \( S \) with \( S[L] = 0^{|L|} \) and the two substrings \( S[0..l_{\text{min}} - 1] \) and \( S[l_{\text{max}} + 1..|S| - 1] \) being arbitrary strings over the alphabet \( \Sigma \setminus \{0\} \). By construction, the string \( S \) does not contain any \( L-R-k \)-cadences. If the two substrings \( S[0..l_{\text{min}} - 1] \) and \( S[l_{\text{max}} + 1..|S| - 1] \) are constant, then \( S \) has at most 6 LZ77-factors. If the two substrings are random, then a constant portion of \( S \) is generally incompressible.

**Theorem 6.2.9.** For a fixed length \( k \geq 2 \) and alphabet size \( |\Sigma| \geq 3 \), the number of \( L-R-k \)-cadences with an arbitrary character in a string \( S \) over \( \Sigma \) can be 0.
There is a constant $c$ which depends on $k$, $\Sigma$ and the chosen compression such that this bound cannot be improved by using the compressed size of $S$ unless the compressed size of $S$ is at least $c|S|$.

The missing cases are

- 2-cadences in strings over a binary alphabet,
- $L$-$R$-2-cadences in strings over a binary alphabet and
- $L$-$R$-3-cadences in strings over a binary alphabet.

We will show that in these variants of the subcadence can only be avoided in strings with a small compressed size.

The starting index 0 can potentially form a 2-cadence with each ending index of at least $|S|/2$. Therefore, in order to avoid 2-cadences, for each $i \geq \lfloor |S|/2 \rfloor$, the inequality $S[0] \neq S[i]$ has to hold. Similarly, for each $i \leq \lfloor |S|/2 \rfloor - 1$, the inequality $S[i] \neq S[|S| - 1]$ has to hold.

Thus, the only binary strings without 2-cadences are of the form

- $0^i1^i$ and
- $1^i0^i$

where $t$ is either the empty string, a single 0 or a single 1. In either case, the string consists of only 4 LZ77-factors.

Similarly, an $L$-$R$-2-cadence is a set of an index $l$ in $L$ and an index $r$ in $R$ such that $S[l] = S[r]$. Therefore, in order to avoid $L$-$R$-2-cadences in a binary string, the strings $S[L]$ and $S[R]$ must be of the form $0^i$ and $1^i$, respectively, or vice versa. Therefore, in a string without $L$-$R$-2-cadences, both $S[L]$ and $S[R]$ must be compressible by only 2 LZ77-factors. However, the remaining characters of $S$ do not matter. Therefore, unless the intervals $L$ and $R$ make up the majority of $S$, we cannot say anything about the compressed size of the underlying string $S$.

Since each $L$-$R$-3-cadence can be transformed into an $L$-$R$-2-cadence by removing the middle index, the strings described in the last paragraph also avoid $L$-$R$-3-cadences. Also, if, for example, the intervals $L$ and $R$ are given by $L = \{0\}$ and $R = \{1, \ldots, |S|\}$, we may consider the string $S = 01^iT(i)$ where is $T(i)$ is an arbitrary string of length $i$. By definition of $L$, each $L$-$R$-3-cadence has to use the character 0. However, each possible middle index lies in $1^i$. Therefore, the string $S$ does not contain any $L$-$R$-3-cadences.

We will now consider an even number $|S|$ and intervals $L = \{0, \ldots, |S|/2 - 1\}$ and $R = \{\lfloor |S|/2 \rfloor, \ldots, |S| - 1\}$. We will show that if $S$ does not contain any
even $L$-$R$-3-cadences, then $S_{even}$ can be written by at most 8 LZ77-factors. Categorizing strings without $L$-$R$-3-cadences will be one of the main objectives in Subsection 6.4.3. In particular, Corollary 6.4.8 will show that one of the following cases hold:

- $S[L_{even}]$ is constant,
- $S[L_{even}]$ and $S[R_{even}]$ are both of the form $0^i1''$,
- $S[L_{even}]$ and $S[R_{even}]$ are both of the form $1^i0''$,
- $S[R_{even}]$ is constant.

Since both $0^i1''$ and $1^i0''$ can be written by at most 4 LZ77-factors, we only need to consider the first case and the last case. It is sufficient to show that if $S[L_{even}]$ is constant, then $S[R_{even}]$ is of the form $0^i1''$ or of the form $1^i0''$. The last case follows by symmetry.

Let $S[L_{even}]$ be constant and without loss of generality, $S[L_{even}] = 0^{\lfloor |S|/2 \rfloor}$ holds. Let $r$ be an even index such that $\lfloor |S|/2 \rfloor \leq r < S - 2$ holds. Then, the index $\frac{l + r}{2}$ is an even number for either $l = 0$ or $l = 2$. Also, the number $\frac{l + r}{2}$ is less than $\frac{|S|}{2}$. Therefore, the arithmetic subsequence $\{l, \frac{l + r}{2}, r]\}$ has corresponding characters $S[l] = 0$ and $S[\frac{l + r}{2}] = 0$. Thus, in order to avoid $L$-$R$-3-cadences, the character $S[r]$ has to be 1. This implies that $S[R_{even}]$ is of the form $1^{\lfloor |S|/2 \rfloor - 1}\sigma$ with $\sigma \in \{0, 1\}$.

This implies that if the string $S$ does not contain any even $L$-$R$-3-cadences, the subsequence $S_{even}$ can be written by at most 8 LZ77-factors.

It can be similarly shown that $S_{odd}$ can also be written by at most 8 LZ77-factors. Also, by more careful analysis of the possible strings $S_{even}$ and $S_{odd}$, it can also be shown that the whole string $S$ can be written by at most 18 LZ77-factors: Since both $S[L_{even}]$ and $S[L_{odd}]$ are of the form $0^i1''$ or $1^i0''$, the string $S[L]$ is of the form $(\sigma_1\sigma_2)^j(\sigma_3\sigma_4)^j(\sigma_5\sigma_6)^j''$ which can be written by at most 9 LZ77-factors.

Therefore, the three cases mentioned above are the only cases in which nontrivial bounds are possible, if both the length of the string as well as its compressed size are given.

### 6.3 NP-Complete Cadence Problems

In this section, we will show that on grammar-compressed strings over the alphabet $\Sigma$ the detection of $k$-cadences is $NP$-complete if at least one of the following conditions holds:


• $k \geq 3$ and $|\Sigma| \geq 2$ and we only consider $k$-cadences with a given character,

• $k \geq 3$ and $|\Sigma| \geq 3$ or

• $k \geq 4$ and $|\Sigma| \geq 2$.

Also, the detection of $L-R-k$-cadences on grammar-compressed strings is \text{NP}
-complete for $k \geq 3$ and $|\Sigma| \geq 2$. Furthermore, somewhat surprisingly, the detection of $k$-subcadences with a given character is also \text{NP}-complete for $k \geq 3$ and $|\Sigma| \geq 2$.

These detection problems are defined by

\textbf{input:} A string $S$ over $\Sigma$ given by its grammar-compression.

\textbf{output:} Is there $k$-subcadence with the given restrictions?

Except for the $\text{NP}$-completeness of the $k$-subcadence detection problem, these results are already published in [80] by the author of this thesis.

Each subcadence is given by the triple of its starting index, its distance and its length. Also each of these variables is bounded by the length of the underlying string. Therefore, in uncompressed strings, the detection, counting and reporting of subcadences and their variants can be done in polynomial time just by exhaustive testing.

However, even problems that can be solved in linear time on uncompressed strings can become $\text{NP}$-hard if the input strings are given by their compression.

For example, in Theorem 3.13 of [69] Lohrey proves the following, seemingly simple, character equality problem with character 1 to be $\text{NP}$-complete.

\textbf{input:} Two strings $P$ and $P'$ over the alphabet \{0, 1\} given by their grammar-compression.

\textbf{output:} Is there an index $l$ with $P[l] = P'[l] = 1$?

In order to prove the $\text{NP}$-hardness of the cadence detection, we will reduce this character equality problem to the cadence detection problems.

On a more positive note, the main compression schemes we consider in this thesis, LZ77 and grammar-compression, allow random access in polynomial time with respect to the compressed size of the underlying string and the logarithm of the length of the string. Also, we can test for a given arithmetic subsequence whether it is a subcadence or any of their variants by accessing the underlying indices and their characters. Therefore, the detection of the subcadence variants mentioned in the introduction lies in $\text{NP}$ for a fixed parameter $k$, even if the input string is given by its compression.

Firstly, we consider $k$-cadences and $L-R-k$-cadences with character 1.

**Theorem 6.3.1.** Let $k \geq 3$ and we consider grammar-compressed strings over the alphabet $\Sigma$ with at least two elements.
Then, the detection problem of $k$-cadences with character 1 is $NP$-hard. Also, the detection problem of $L$-$R$-$k$-cadences with character 1 is $NP$-hard.

Proof. Let $P$ and $P'$ be two strings over the alphabet \{0, 1\} given by grammar-compression. We will construct a grammar compression for a string that contains a $k$-cadence if and only if there is an index $l$ with $P[l] = P'[l] = 1$.

For $k = 3$, the construction is most straightforward. Consider the string

$$S_3 = P_{rev} \cdot 0^{|P|+|P'|} \cdot 1 \cdot 0^{|P|+|P'|} \cdot P'$$

of length $3(|P| + |P'|) + 1$. An example can be found in Figure 6.3.

This string can be built from the grammar of $P$, the grammar of $P'$ and $O(\log(|P|+|P'|))$ many additional rules to generate $0^{|P|+|P'|} \cdot 1 \cdot 0^{|P|+|P'|}$. In particular, a grammar for $S_3$ can be found in linear time with respect to the combined size of the grammars of $P$ and $P'$.

Since the string is too short to have a 3-cadence with a distance of at least $2(|P| + |P'|) + 2$, each 3-cadence of $S_3$ with character 1 has to contain the 1 at index $|P| + |P'|$. On the other hand, each 3-subcadence that contains this 1 has a distance of at least $|P| + |P'| + 1 \geq \frac{3(|P|+|P'|)+1}{3} = \left|S_3\right|$ and is therefore a 3-cadence. In particular, the 3-cadences of $S_3$ with character 1 are exactly the 3-subcadences that use the 1 at index $2|P| + |P'|$ as their middle index.

For each 3-subcadence with middle index $2|P| + |P'|$ and distance $d$, we get a 1 at index $2|P| + |P'| + d$. Since $P'$ starts at index $|P| + 2(|P| + |P'|) + 1$ in $S_3$ and

$$2|P| + |P'| + d = (|P| + 2(|P| + |P'|) + 1) + (d - |P| - |P'| - 1)$$

holds, this yields a 1 at $P'[d - |P| - |P'| - 1]$. Similarly, we get a 1 at index $2|P| + |P'| - d$ which yields a 1 at $P_{rev}[2|P| + |P'| - d]$. With

$$2|P| + |P'| - d = |P| - 1 - (d - |P| - |P'| - 1)$$
we get a 1 at \( P[d - |P| - |P'| - 1] \). Hence, each 3-cadence in \( S_3 \) can be translated to a solution of the character equality problem of \( P \) and \( P' \) with character 1.

Conversely, each solution of the character equality problem of \( P \) and \( P' \) with character 1 has a corresponding 3-cadence in \( S_3 \).

Hence, the detection problem of 3-cadences with character 1 is \( \mathcal{NP} \)-hard.

Additionally, each 3-cadence in \( S_3 \) starts in \( P_{\text{rev}} \) and ends in \( P' \). Conversely, each 3-subcadence in \( S_3 \) that starts in \( P_{\text{rev}} \) and ends in \( P' \) is a 3-cadence.

Therefore, the detection problem of \( L-R \)-3-cadences with character 1 is \( \mathcal{NP} \)-hard.

The idea for \( k \geq 4 \) is very similar.

The first three indices will correspond to the 3-cadence in \( S_3 \). We will also append multiple copies of the character 1 to ensure that all indices except for the first three of the candidates for \( k \)-cadences always correspond to a 1. Additionally, we will have to pad the string with multiple copies of the character 0 to ensure that the resulting \( k \)-subcadences are indeed \( k \)-cadences and to avoid additional \( k \)-cadences that do not respond to solutions of the character equality problem with character 1.

Consider the string

\[
S_k = \left(0^{(k-1)(|P|+|P'|)} \cdot 0^{|P'|} \cdot 0^{k(|P|+|P'|)}) \cdot 0 \cdot 0^{k(|P|+|P'|)}) \cdot 0 \cdot \left(0^{k(|P|+|P'|)} \cdot 0^{(k-1)(|P|+|P'|)}) \cdot 0 \cdot \left(0^{k(|P|+|P'|)} \cdot 0^{k(|P|+|P'|)}) \cdot 0 \cdot \left(0^{k(|P|+|P'|)+1}) \cdot 1^{2k(|P|+|P'|)+1}) \right)^{k-3}
\]

of length \( k(2k(|P| + |P'|) + 1) = 2k^2(|P| + |P'|) + k \).

This string can be built from the grammar of \( P \), the grammar of \( P' \) and \( O(\log(k^2(|P| + |P'|))) \) many additional rules to generate both the padding and the \( k - 3 \) last large brackets. In particular, for fixed \( k \), a grammar for \( S_k \) can be found in linear time with respect to the combined size of the grammars of \( P \) and \( P' \).

The prefix of \( S_k \) given by the first 3 \( (2k(|P| + |P'|) + 1) \) characters (i.e. the first three large brackets) is very similar to \( S_3 \). The only difference is the extended padding with copies of the character 0. Therefore, the analysis of \( S_3 \) still holds and the solutions to the character equality problem with character 1 naturally translate to \( L-R \)-3-cadences with character 1 that start in the first large bracket, have their middle index at \( 3k(|P| + |P'|) + 1 \) and end in the third large bracket.
The distances of these \(L-R\)-3-cadences are at least \(2k(|P| + |P'|) + 2\) and at most \((2k + 1)(|P| + |P'|) + 1\). Hence, if we extend the \(L-R\)-cadences to the right, the last index is at least
\[
3k(|P| + |P'|) + 1 + (k - 2)(2k(|P| + |P'|) + 2)
\]
\[
= 2k^2(|P| + |P'|) - k(|P| + |P'|) + 2k - 3
\]
\[
\geq |S_k| - (2k(|P| + |P'|) + 1)
\]
and at most
\[
3k(|P| + |P'|) + 1 + (k - 2)((2k + 1)(|P| + |P'|) + 1)
\]
\[
= 2k^2(|P| + |P'|) - 2(|P| + |P'|) + k - 1
\]
\[
< |S_k|.
\]

Thus, the extensions of the \(L-R\)-3-cadences to the right are \(L-R\)-\(k\)-cadences that start in the first large bracket and end in the last large bracket.

Conversely, since all large brackets have the same length, each \(L-R\)-\(k\)-cadence that starts in the first large bracket and ends in the last large bracket has its \(k\)-th index in the \(k\)-th bracket.

Therefore, the \(L-R\)-\(k\)-cadences naturally translate into solutions of the character equality problem with character 1. Hence, the detection problem of \(L-R\)-\(k\)-cadences with character 1 is \(\mathcal{NP}\)-hard for \(k \geq 3\).

Since the distances of the considered \(L-R\)-\(k\)-cadences are greater than the length of a large bracket, it follows easily that the \(L-R\)-\(k\)-cadences are also \(k\)-cadences. Conversely, since all large brackets have the same length, each \(k\)-cadence has to start in the first large bracket and to end in the last large bracket.

This implies that the \(L-R\)-\(k\)-cadences that start in the first large bracket and end in the last large bracket are exactly the \(k\)-cadences. Therefore, the detection problem of \(k\)-cadences with character 1 is \(\mathcal{NP}\)-hard for \(k \geq 3\). \(\square\)

For \(k \geq 4\), the last large bracket of the string \(S_k\) in the previous proof does only contain the copies of the character 1. Since the considered \(k\)-cadences and \(L-R\)-\(k\)-cadences end in this bracket, this forces the \(k\)-subcadences to use the character 1. We can therefore drop this condition from the requirements.

**Corollary 6.3.2.** Let \(k \geq 4\) and we consider grammar-compressed strings over the alphabet \(\Sigma\) with at least two elements.

Then, the detection problem of \(k\)-cadences is \(\mathcal{NP}\)-hard. Also, the detection problem of \(L-R\)-\(k\)-cadences is \(\mathcal{NP}\)-hard.
If we have an alphabet with at least three characters, we can change the second large bracket of $S_k$ to $\left(2^{k(|P|+|P'|)} \cdot 1 \cdot 2^{k(|P|+|P'|)}\right)$. This results in the string

$$S'_k = \left(0^{(k-1)(|P|+|P'|)} \cdot 0^{P'_\text{rev}} \cdot 0 \cdot 0^{k(|P|+|P'|)}\right) \cdot \left(2^{k(|P|+|P'|)} \cdot 1 \cdot 2^{k(|P|+|P'|)}\right) \cdot \left(0^{k(|P|+|P'|)} \cdot 0 \cdot (P'0^{P'}) 0^{(k-1)(|P|+|P'|)}\right) \cdot \left(1^{2k(|P|+|P'|)+1}\right)^{k-3}.$$ 

All large brackets of $S'_k$ have equal length. Also, the $k$-cadences and $L-R-k$-cadences considered in Theorem 6.3.1 start in the first large bracket and end in the last large bracket. Hence, they either use the character 0 or the character 1. Additionally, the second index of these $k$-cadences and $L-R-k$-cadences lies in the second large bracket. Hence, they use either the character 1 or the character 2. This forces their used character to be 1.

**Corollary 6.3.3.** Let $k \geq 3$ and we consider grammar-compressed strings over the alphabet $\Sigma$ with at least three elements.

Then, the detection problem of $k$-cadences is $\mathcal{NP}$-hard. Also, the detection problem of $L-R-k$-cadences is $\mathcal{NP}$-hard.

For the $\mathcal{NP}$-hardness of the $L-R$-$k$-cadences it is left to show that the problem stays $\mathcal{NP}$-hard for $k = 3$ and $|\Sigma| = 2$, even if we do not require a specific character.

**Theorem 6.3.4.** We consider grammar-compressed strings over the alphabet $\Sigma$ with two elements.

Then, the detection problem of $L-R$-$3$-cadences is $\mathcal{NP}$-hard.

**Proof.** Let $P$ and $P'$ be two strings over the alphabet $\{0, 1\}$ given by grammar-compression. We will construct a string that contains an $L-R$-$3$-cadence if and only if there is an index $l$ with $P[l] = P'[l] = 1$.

Let $P''$ be the string that is obtained by duplicating each character in $P'$. For example, for $P' = 100$, we have $P'' = 110000$. This can be done by introducing the four nonterminals $D_0 \rightarrow S_0S_0$, $D_1 \rightarrow S_1S_1$, $S_0 \rightarrow 0$ and $S_1 \rightarrow 1$. We also have to replace all nonterminals that map to the single character $i$ and terminals $j$ in the right-hand sides of the grammar of $P'$ with $D_i$ and $S_j$, respectively.

Consider the string

$$S = 1 \cdot 0^{P|+|P'|} \cdot P \cdot 0^{P'} \cdot P''$$
and the intervals \( L = \{0\} \) which corresponds to the first 1 and

\[
R = \{1 + 2(|P| + |P'|), 1 + 2(|P| + |P'|) + 1, \ldots, |S| - 1\}
\]

which corresponds to \( P'' \) at the end of the string.

By construction, for each index \( 0 \leq i < |P| \), the equations

\[
P[i] = S[1 + (|P| + |P'|) + i] = S[1 + |P| + |P'| + i]
\]

and

\[
P'[i] = P''[2i + 1] = S[1 + 2(|P| + |P'|) + 2i + 1] = S[2(1 + |P| + |P'| + i)]
\]

hold.

Hence, for each index \( 0 \leq i < |P| \), the equation \( P[i] = 1 = P'[i] \) holds if and only if the equation \( 1 = S[0] = S[1 + |P| + |P'| + i] = S[2(1 + |P| + |P'| + i)] \) holds. Therefore, solutions of the character equality problem with character 1 naturally translates to \( L-R \)-3-cadences and vice versa.

This proves that even for \( k = 3 \) and \( |\Sigma| = 2 \), the detection problem of \( L-R \)-3-cadences is \( \mathcal{NP} \)-hard.

All reductions above used that we could force all cadences to use a fixed character of the string. However, surprisingly, if \( L \) and \( R \) have similar length, we can detect in linear time, whether an uncompressed binary string has an \( L-R \)-3-cadence, and we can detect in polynomial time, whether a compressed binary string has an \( L-R \)-3-cadence. We will prove these results in Section 6.4.

Surprisingly, while \( k \)-subcadences are unavoidable over a fixed alphabet and for a fixed \( k \), the detection of \( k \)-subcadences with a given character is \( \mathcal{NP} \)-complete for \( k \geq 3 \).

**Theorem 6.3.5.** We consider grammar-compressed strings over the alphabet \( \Sigma \) with at least two elements.

Then, the detection problem of \( k \)-subcadences with character 1 is \( \mathcal{NP} \)-hard.

**Proof.** Unfortunately, since arbitrary strings may contain \( k \)-subcadences, there seems to be no straightforward way to reduce this problem to the character equality problem with the character 1. However, in the reduction from the subsetsum problem to the character equality problem with the character 1 in Theorem 3.13 of [69], the strings can be easily modified to avoid \( k \)-subcadences for \( k \geq 3 \). We will therefore adapt the reduction to reduce the \( k \)-subcadence detection problem to the subsetsum problem directly.

The following subsetsum problem is known to be \( \mathcal{NP} \)-complete.
input: A set of positive integers \( \{w_1, w_2, \ldots, w_n\} \) and another positive integer \( t \).

output: Is there a subset \( I \subseteq \{1, 2, \ldots, n\} \) such that \( \sum_{i \in I} w_i = t \) holds?

Firstly, we will adapt the Cantor strings from Definition 6.2.2. We define the stretched Cantor strings \( C(n, m) \) of length \( m \cdot |C(n)| \) with

\[
C(n)[i] = 1 \iff C(n, m)[mi] = 1.
\]

Also, we consider modified Cantor strings in which for a given number \( i \), the strings \( C(i, m) \) are substituted with strings \( D(i, m, j) \) with length \( |C(i)| \) and at most one 1.

Similarly to the Cantor strings and for the same reason given in the proof of Theorem 6.2.3, both the stretched Cantor strings and the modified Cantor strings do not contain any \( k \)-subcadences for \( k \geq 3 \).

Let the integers \( \{w_1, w_2, \ldots, w_n\} \) and \( t \) be an instance of the subsetsum problem. Following Lohrey’s reduction, we will firstly create a string that encodes the candidates for the subsetsum problem. This string will have a common 1 with a suitable stretched Cantor string if and only if the subsetsum problem has a solution. We define \( s = w_1 + w_2 + \cdots + w_n \).

We consider the stretched Cantor string

\[
C(2, s) = \left(10^{s-1}0^{s-1}\right) \cdot \left(0^{3s}\right) \cdot \left(10^{s-1}0^{s-1}\right)
\]

Lohrey constructed a string in which the indices of the 1s modulo \( s \) correspond to the sums of each candidate. Since this may result in unwanted \( k \)-subcadences, we will instead decode the candidates into strings of length \( |C(2, s)| \) and we will adjust the distance between two candidates to prohibit accidental \( k \)-subcadence. We define for each candidate \( I \subseteq \{1, 2, \ldots, n\} \) the corresponding string

\[
D(2, s, I) = 0^{2s-t+\sum_{i \in I} w_i} 10^{7s+t-1-\sum_{i \in I} w_i}.
\]

Note that \( D(2, s, I) \) and \( C(2, s) \) both have length \( 9s \) and if and only if \( \sum_{i \in I} w_i = t \) holds, then both strings have a 1 with the same index (which is 2s).

Following Lohrey’s reduction, we define \( Z(i, s) = 0^{3s} \) and

\[
A(1) = 0^{2s-t}10^{7s+t-1}Z(2, s)0^{2s-t+w_1}1,
\]

\[
A(j + 1) = A(j)0^{7s+t-1-\sum_{i=1}^{j} w_i}Z(j + 2, s)0^{w_{j+1}}A(j)
\]

and finally

\[
A(n) = A(n - 1)0^{7s+t-1-\sum_{i=1}^{n-1} w_i}Z(n + 1, s)0^{w_n}A(n - 1)0^{7s+t-1-\sum_{i=1}^{n} w_i}.
\]
By construction, the final string $A(n)$ is a modified Cantor string with length $|C(n+2,s)|$. Furthermore, the strings $A(n)$ and $C(n+2,s)$ have a 1 with the same index if and only if the subsetsum has a solution.

Now we define $m = 3^{n+2}ks$ and consider the string

$$S = 10^m A(n)0^m C(n+2,2s)0^m C(n+2,3s) \ldots 0^m C(n+2,(k-1)s)$$

By construction, neither the modified Cantor string $A(n)$ nor any of the stretched Cantor strings contain a $k$-subcadence with character 1. Therefore, each $k$-subcadence with character 1 has to have a distance of at least $m$. Since $m$ is larger than the length of even the longest stretched Cantor string $C(n+2,(k-1)s)$, the modified Cantor string and each of the stretched Cantor strings can contain at most 1 element of a $k$-subcadence. Therefore, if a $k$-subcadence with character 1 exists, it has to contain the starting index 0.

Let $(0, m+l)$ be a $k$-subcadence in $S$ with character 1. Then, the string $A(n)$ has a 1 at index $l$. Similarly, the stretched Cantor string $C(n+2,js)$ has a 1 at index $jl$. By construction, this is possible if and only if the string $C(n+2,s)$ has a 1 at index $l$. This, however, is possible if and only if the subsetsum problem has a solution.

Since the number of distinct runs of 0s is polynomial and even the largest block $0^m$ can be constructed by a grammar of polynomial size, we can construct $S$ in polynomial time with respect to the input size.

This concludes the proof and thereby shows that $k$-subcadence detection is $NP$-hard.

By definition of equidistant subsequences by Funakoshi et al. in [40], $k$-subcadences are equidistant subsequences where all characters in the pattern are equal. Therefore, the equidistant subsequence matching problem is also $NP$-complete, even if the pattern has only three characters and the alphabet of the underlying string has only two elements.

### 6.4 Algorithms

In this section, we will discuss several algorithms for detecting, counting and reporting subcadences and their variants. We will consider both uncompressed strings and grammar-compressed strings.

#### 6.4.1 Naïve Algorithms

In this subsection, we will present the naïve algorithms for detection and counting of $k$-subcadences and their variants in uncompressed strings. In
particular, we will show that these algorithms take quadratic time for both fixed $k$ and arbitrary $k$.

Amir et al. show in [4] that all cadences in an uncompressed string can be reported in quadratic time with respect to the string’s length. They further show that we can detect 2-cadences by testing only the first occurrence and the last occurrence of each character. They report that this can be done in $O(|S| \log |S|)$ time by sorting the characters in the string. Also, if the size of the alphabet is constant, the 2-cadences can even be detected in linear time. Note that this algorithm also detects 2-cadences in $O(|\Sigma| \cdot |G|)$ time in strings given by a grammar $G$ over the alphabet $\Sigma$.

For each distance $d$, we can find and report all maximal subcadences by checking the $d$ maximal arithmetic subsequences with distance $d$ which each have roughly length $\frac{|S|}{d}$. This can be done in linear time. Since each maximal subcadence of length $l$ gives rise to \( \binom{l}{2} - l \) subcadences with the same distance and a length of at least 2, to $\max(0, l + 1 - k) \) $k$-subcadences with the same distance and to at most 1 cadence with the same distance, this allows for counting the subcadences, $k$-subcadences and cadences with distance $d$ in linear time.

By using this approach for each possible distance, we can count all subcadences, $k$-subcadences and cadences in quadratic time.

### 6.4.2 Convolution-Based Algorithms

In this subsection, we will show that the counting of 3-subcadences and their variants in uncompressed strings $S$ over a fixed alphabet can be efficiently reduced to discrete acyclic convolution. Therefore, in uncompressed strings, 3-subcadences and their variants can be detected and counted in quasi-linear time.

This approach was already done indirectly by Amir et al. in [4]. In their paper, Amir et al. reduce the detection of $L$-$R$-3-cadences with the intervals $L = \left\{0, \ldots, \left\lfloor \frac{|S|}{3} \right\rfloor - 1 \right\}$ and $R = \left\{\left\lfloor \frac{|S|}{3} \right\rfloor + 1, \ldots, |S| - 1 \right\}$ to a restricted variant of the 3SUM-problem. This variant can in turn be solved with discrete acyclic convolution in $O(|S| \log |S|)$ additions and multiplications.

More recently, in [41], Funakoshi and the author of this thesis refined this approach so that it allows not only the detection but also the counting of the $L$-$R$-3-cadences. We also generalized the convolution to the non-rectangular convolution that is presented in Chapter 3. This more generalized convolution allows counting more complex variants of the 3-subcadence like 3-cadences. In this subsection, we will present the resulting fast convolution algorithms for the counting of 3-subcadences and 3-cadences.
CHAPTER 6. CADENCES

Firstly, we will show how 3-subcadences with a given character 1 can be counted in $O(|S| \log |S|)$ time in uncompressed strings. The main idea will be to use the fast discrete acyclic convolution to count for each index $m$ with $S[m] = 1$, how many 3-subcadences with middle index $m$ exist in $S$.

**Theorem 6.4.1.** For each character $\sigma \in \Sigma$ in a string $S$, the 3-subcadences with character $\sigma$ can be counted in $O(|S| \log |S|)$ time. Also, if all $n_\sigma$ occurrences of $\sigma$ are known, the 3-subcadences with $\sigma$ can be counted in $O(n_\sigma^2)$ time.

**Proof.** In order to use the convolution, we first introduce a suitable sequence given by the indicator function for the character 1 in $S$. In particular, we define $\delta = (\delta_0, \delta_1, \delta_2, \ldots, \delta_{|S|-1})$ by the following equation:

$$
\delta_i := \begin{cases} 
1 & \text{if } S[i] = 1 \\
0 & \text{if } S[i] \neq 1 
\end{cases}
$$

With this definition, products of elements in $\delta$ are given by the common indicator function for the character 1 on the corresponding tuples. In particular, the equation

$$
\delta_i \delta_j := \begin{cases} 
1 & \text{if } S[i] = 1 \land S[j] = 1 \\
0 & \text{if } S[i] \neq 1 \lor S[j] \neq 1 
\end{cases}
$$

holds. By construction, the variable

$$
c_{m'} := \sum_{i+j=m'} (\delta_i \delta_j) = \# \{ i : S[i] = S[m'-i] = 1 \}
$$

counts the number of index pairs $(i, j)$ with $i + j = m'$ such that the two corresponding characters $S[i]$ and $S[j]$ are both 1. Note that all $c_{m'}$ with $m' \in \{0, 1, \ldots, 2|S| - 2\}$ can be simultaneously calculated in $O(|S| \log |S|)$ time with fast discrete acyclic convolution. Also, for a given index $i$ and an even integer $m'$, we naturally get an arithmetic sequence $\{ i, \frac{m'}{2}, m' - i \}$. This sequence has either length 1 or length 3, depending on whether $i = \frac{m'}{2}$ holds.

Let $m$ be an index in $S$. If $S[m] \neq 1$ holds, there is no 3-subcadence with character 1 and with middle element $m$.

Otherwise, the equation $S[m] = 1$ holds. Therefore, the variable $c_{m'}$ contains the addend $\delta_m \delta_m = 1$, which does not correspond to a 3-subcadence. Also, for each pair $(i, j)$ with $i \neq j$ that fulfills both $i + j = 2m$ and $\delta_i \delta_j = 1$, the variable $c_{2m}$ counts both addends $\delta_i \delta_j$ and $\delta_j \delta_i$. However, both addends correspond to the same arithmetic sequence $\{ i, m, 2m - i \}$. Therefore, we want to count the two corresponding addends in $c_{2m}$ for only one 3-subcadence.
6.4. ALGORITHMS

Therefore, the variables
\[ s_m := \begin{cases} \frac{c_{2m} - 1}{2} & \text{if } S[m] = 1 \\ 0 & \text{if } S[m] \neq 1 \end{cases} \]
count the 3-subcadences with character 1 and middle index \( m \). In particular, the number of all 3-subcadences with character 1 is given by the sum \( \sum_{m=1}^{\vert S \vert} s_m \).

With the given values \( c_{2m} \), each value of \( s_m \) can be calculated in constant time. Hence, the sum \( \sum_{m=1}^{\vert S \vert} s_m \) can be calculated in linear time with the values \( c_{2m} \). Therefore, the counting of the 3-subcadences with character 1 takes \( O(\vert S \vert \log \vert S \vert) \) time.

On the other hand, if all \( n_\sigma \) occurrences of the character \( \sigma \) are known, we can calculate all non-zero values \( c_k \) for this character in \( O(n_\sigma^2) \) by testing each pair of those occurrences.

If we want to count all 3-subcadences, we repeat the two approaches for each character which leads to the following theorem.

**Theorem 6.4.2.** The number of all 3-subcadences in a string \( S \) over the alphabet \( \Sigma \) can be counted in
\[ O\left( \min\left( \vert \Sigma \vert, \sqrt{\frac{\vert S \vert}{\log \vert S \vert}} \cdot \vert S \vert \log \vert S \vert \right) \right) \]
time.

**Proof.** By executing this algorithm once for each character in the alphabet, we can count all 3-subcadences in \( O(\vert \Sigma \vert \cdot \vert S \vert \log \vert S \vert) \) time which simplifies to \( O(\vert S \vert \log \vert S \vert) \) for fixed alphabet size.

Following an idea described by Amir et al. in [4], we can sort the input string in \( O(\vert S \vert \log \vert S \vert) \) time and thereby get all occurrences of each character. This implies that the counting can be done in
\[ O\left( \sum_{\sigma \in \Sigma} \min(n_\sigma^2, \vert S \vert \log \vert S \vert) \right) \subseteq O\left( \frac{\vert S \vert}{(\vert S \vert \log \vert S \vert)^{1/2}} \vert S \vert \log \vert S \vert \right) \]
\[ = O(\vert S \vert^{3/2}(\log \vert S \vert)^{1/2}) \]
time.

Since the convolution leads to the number of 3-subcadences for each possible middle index, we can easily report 3-subcadence each in linear time.
CHAPTER 6. CADENCES

Figure 6.4: All possible pairs which form 3-cadences. The dashed edges do not correspond to possible 3-cadences.

**Theorem 6.4.3.** After counting at least $x$ 3-subcadences in the string $S$, it is possible to report $x$ 3-subcadences in $O(x \cdot |S|)$ time.

We will now use the non-rectangular convolution to prove similar results for 3-cadences.

**Theorem 6.4.4.** Let $S$ be a string over the alphabet $\Sigma$ with character $\sigma \in \Sigma$.

- The 3-cadences with character $\sigma$ can be counted in $O(|S|(|S| \log |S|) \cdot |S|)$ time. Also, if all $n_\sigma$ occurrences of $\sigma$ in $S$ are known, the 3-cadences with $\sigma$ can be counted in $O(n_\sigma^2)$ time.

- The number of all 3-cadences in $S$ can be counted in

$$O\left(\min\left(|\Sigma|, \frac{\sqrt{|S|}}{\log |S|}\right) \cdot |S|(\log |S|)^2\right)$$

- After counting at least $x$ 3-cadences in the string $S$, it is possible to report $x$ 3-cadences in $O(x \cdot |S|)$ time.

**Proof.** We will virtually use the same algorithms as for the counting and reporting of 3-subcadences. However, the 3-cadences have the additional requirements that for the starting index $i$ and the distance $d$, the inequalities $i - d < 0$ and $i + 3d \geq |S|$ hold. These two inequalities together with $i \geq 0$ and $i + 2d < |S|$ form the quadrilateral given by Figure 6.4.
Since the perimeter of this quadrilateral is linear in $|S|$, the non-rectangular convolution finds the number of 3-cadences with each possible middle index in $O(|S|(\log |S|)^2)$ time.

Similarly to the 3-subcadences, if all $n_\sigma$ occurrences of the character $\sigma$ are known, we can calculate all non-zero values $c_k$ for this character in $O(n_\sigma^2)$ by testing each pair of those occurrences.

Therefore, we can count all 3-cadences in $S$ in

$$O\left(\sum_{\sigma \in \Sigma} \min (n_\sigma^2, |S| (\log |S|)^2)\right) \subseteq O\left(\frac{|S|}{(|S| (\log |S|)^2)^{1/2}} |S| \log |S| \right)$$

$$= O(|S|^{3/2} \log |S|)$$
time.

In [41], Funakoshi and the author of this thesis show that this approach can be extended to partial cadences with three elements and $k$-cadences with at most $k - 3$ errors. However, these variants of the 3-cadence are beyond the scope of this thesis.

6.4.3 Regularity-Based Algorithms

In this subsection, we show that binary strings which contain no $k$-subcadences or their variants with three elements are very regular and can thus be efficiently detected.

In particular, we show the following results for a binary string $S$.

- We can decide in constant time whether $S$ contains a $k$-subcadence for a fixed $k$.

- If $S$ is uncompressed, we can decide in linear time whether $S$ contains an $L$-$R$-3-cadence and whether it contains a 3-cadence.

- If the string $S$ is compressed by a grammar $G$, we can decide in $O(|G| \max \left(\frac{|R|}{|L|}, \frac{|L|}{|R|}\right))$ time whether $S$ contains an $L$-$R$-3-cadence and in $O(|G| \log |S|)$ time whether it contains a 3-cadence.

Except for the $k$-subcadence detection problem, these results are already published in [80] by the author of this thesis.

Chvátal shows in [22] that each binary string with at least 9 characters contains a 3-subcadence. We therefore get the following lemma for both uncompressed strings and grammar-compressed strings in Chomsky normal form.
Chapter 6. Cadences

Lemma 6.4.5. It can be decided in constant time, whether a binary string contains a 3-subcadence.

Note that in grammar-compressed strings, we are not even able to determine the character with a given index in constant time. However, since we don’t have rules of the form

\[
\text{Nonterminal} \rightarrow \text{Nonterminal}
\]

or

\[
\text{Nonterminal} \rightarrow \epsilon
\]

in the Chomsky normal form, we can decide after using at most 15 applications of the product rules whether a string contains at least 9 characters.

With van der Waerden’s theorem, we can extend this result to subcadences with arbitrary length. However, as stated in Section 6.1, the implicit constants are very large.

Theorem 6.4.6. For a fixed alphabet size \(|\Sigma|\) and a fixed length \(k\), it can be decided in constant time, whether a string over \(\Sigma\) contains a \(k\)-subcadence.

Next, we will show that \(L\text{-}R\text{-}3\)-cadences can be detected in \(O(|L| + |R|)\) time in uncompressed binary strings and in \(O\left(|G| \max \left(\frac{|R|}{|L|}, \frac{|L|}{|R|}\right)\right)\) time in binary strings given by the grammar \(G\).

We will consider even \(L\text{-}R\text{-}3\)-cadences and odd \(L\text{-}R\text{-}3\)-cadences separately. Since all observations and algorithms hold for both even \(L\text{-}R\text{-}3\)-cadences and odd \(L\text{-}R\text{-}3\)-cadences, we will only consider even \(L\text{-}R\text{-}3\)-cadences explicitly.

We will treat the three substrings given by the possible starting indices, middle indices and ending indices like three separate strings. We also want to avoid wrongfully reporting subsequences with distance 0 as \(L\text{-}R\text{-}3\)-cadences. Therefore, we start with the assumption that \(L\) and \(R\) are disjoint.

We will see that if a binary string does not contain an even \(L\text{-}R\text{-}3\)-cadence, at least one of the subsequences \(S[L_{even}]\) and \(S[R_{even}]\) is very structured. The following lemma implies that if \(S[L_{even}]\) contains the substring 01 and \(S[R_{even}]\) contains the substring 10 or vice versa, then \(S\) has an even \(L\text{-}R\text{-}3\)-cadence.

Lemma 6.4.7. Let \(S\) be a binary string and \(L\) and \(R\) be two disjoint intervals.

If there are even indices \(i\) and \(j\) with

- \(S[i] = S[j] \neq S[i + 2] = S[j - 2]\),
- \(i, i + 2 \in L\) and
- \(j, j - 2 \in R\),

then \(S\) contains an even \(L\text{-}R\text{-}3\)-cadence.
then $S$ has an $L$-$R$-3-cadence.

Proof. Since $i$ and $j$ are both even, the number $\frac{i+j}{2}$ is an integer. Furthermore, since $S$ is binary and $S[i] = S[j] \neq S[i+2] = S[j-2]$ holds, we either have $S[i] = S[\frac{i+j}{2}] = S[j]$ or $S[i+2] = S[\frac{i+j}{2}] = S[j-2]$. Therefore, there is at least one $L$-$R$-3-cadence. \qed

Therefore, in order to avoid $L$-$R$-3-cadences in a binary string $S$, there are only the four possibilities for the subsequences $S[L_{\text{even}}]$ and $S[R_{\text{even}}]$ given by the following lemma.

Corollary 6.4.8. Let $S$ be a binary string and $L$ and $R$ be two disjoint intervals such that $S$ has no $L$-$R$-3-cadences.

Then, one of the following cases hold,

1. $S[L_{\text{even}}]$ is constant,
2. $S[L_{\text{even}}]$ and $S[R_{\text{even}}]$ are both of the form $0^i1^i$,
3. $S[L_{\text{even}}]$ and $S[R_{\text{even}}]$ are both of the form $1^i0^i$,
4. $S[R_{\text{even}}]$ is constant.

In uncompressed strings, we can check in $O(|L| + |R|)$ time whether one of these cases holds.

As shown in the Subsection 2.3.3, in grammar-compressed strings, we can do the following operations in linear time with respect to the size of the grammar:

1. Convert the grammar to Chomsky normal form,
2. generate substrings given by their starting index and ending index,
3. generate the even subsequence and the odd subsequence and
4. find the indices of the first and last occurrence of any character.

Furthermore, the grammars of the substrings, the even subsequence and the odd subsequence are linear in the size of the grammar of the underlying string. We can therefore check for both subsequences $S[L_{\text{even}}]$ and $S[R_{\text{even}}]$ in linear time whether they are constant and whether they are of the form $0^i1^i$ or $1^i0^i$.

If both $S[L_{\text{even}}]$ and $S[R_{\text{even}}]$ are of the form $0^i1^i$, we can divide $L_{\text{even}}$ and $R_{\text{even}}$ into $L'_{\text{even}}$, $L''_{\text{even}}$, $R'_{\text{even}}$ and $R''_{\text{even}}$ such that $S[L'_{\text{even}}] = 0^i$, $S[L''_{\text{even}}] = 1^i$, $S[R'_{\text{even}}] = 0^j$ and $S[R''_{\text{even}}] = 1^j$. Furthermore, the subintervals $L'$, $L''$, $R'$
and $R''$ can be found in $O(|L| + |R|)$ time in uncompressed strings and in linear time in grammar-compressed strings.

By construction of the subintervals, there are no even $L''-R''$-3-cadences and no even $L''-R'$-3-cadences. Therefore, the only possible even $L-R$-3-cadences are even $L'-R'$-3-cadences and $L''-R''$-3-cadences. The following lemma shows that these subcadences can be detected in linear time.

**Lemma 6.4.9.** Let $S$ be a binary string and $L$ and $R$ be two disjoint intervals such that $S[L_{\text{even}}] = 0^i$ and $S[R_{\text{even}}] = 0^j$ hold for some integers $i$, $j$.

Then, we can decide whether the string $S$ contains an even $L-R$-3-cadence in $O(|L| + |R|)$ time if $S$ is uncompressed and in linear time with respect to the size of the grammar of $S$ if $S$ is grammar-compressed.

**Proof.** Let $l_{\text{min}}$, $l_{\text{max}}$, $r_{\text{min}}$ and $r_{\text{max}}$ be the minimal and maximal indices of $L_{\text{even}}$ and $R_{\text{even}}$, respectively.

If $S$ has an even $L$-$R$-3-cadence, the index of the middle element is in the interval $M := \{\frac{l_{\text{min}} + r_{\text{min}}}{2}, \frac{l_{\text{min}} + r_{\text{min}}}{2} + 1, \ldots, \frac{l_{\text{max}} + r_{\text{max}}}{2}\}$. Therefore, if $S[M]$ does not contain a 0, there is no even $L-R$-3-cadence.

If, on the other hand, $S[M]$ contains a 0 at index $m$, then we define $d := \min(m - l_{\text{min}}, r_{\text{max}} - m)$. We will show that the subsequence $(m - d, d, 3)$ is an even $L-R$-3-cadence.

Since $m - d = \max(l_{\text{min}}, 2m - r_{\text{max}})$ holds and both $l_{\text{min}}$ and $r_{\text{max}}$ are even, the starting index $m - d$ is even as well. An example can be seen in Figure 6.5.

If $\max(l_{\text{min}}, 2m - r_{\text{max}}) = l_{\text{min}}$ holds, the starting index $m - d = l_{\text{min}}$ is by definition in $L_{\text{even}}$. Otherwise, we get $l_{\text{min}} \leq 2m - r_{\text{max}} = m - d$ and $m - d = 2m - r_{\text{max}} \leq \frac{l_{\text{max}} + r_{\text{max}}}{2} - r_{\text{max}} = l_{\text{max}}$ and hence $l_{\text{min}} \leq m - d \leq l_{\text{max}}$. 

\[S[\{0, 1, \ldots, 15\}_{\text{even}}]: \quad \begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \end{array} \]

\[S[\{16, 17, \ldots, 30\}]: \quad \begin{array}{cccccccc} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ \end{array} \]

\[S[\{32, 33, \ldots, 47\}_{\text{even}}]: \quad \begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \end{array} \]

Figure 6.5: A string with 48 characters with intervals $L = \{0, 1, \ldots, 15\}$ and $R = \{32, 34, \ldots, 47\}$. For each index of $M$ as defined in Lemma 6.4.9 which correspond to a 0, we obtain at least one $L$-$R$-3-cadence.
Therefore, in either case, the starting index \( m - d \) is contained in \( L_{\text{even}} \) and by prerequisite, the equation \( S[m - d] = 0 \) holds.

Also, we have \( m + d = \min(2m - l_{\text{min}}, r_{\text{max}}) \). Thus, we can similarly prove that the ending index \( m + d \) is contained in \( R_{\text{even}} \) and by prerequisite, the equation \( S[m + d] = 0 \) holds.

Therefore, the subsequence \((m - d, d, 3)\) is an even \( L-R-3 \)-cadence.

We can check whether \( S[M] \) contains a 0, and thus whether \( S \) contains an even \( L-R-3 \)-cadence, in \( O(|L| + |R|) \) time in uncompressed strings and in linear time with respect to the size of the grammar in grammar-compressed strings.

Since Lemma 6.4.9 allows us to efficiently deal with the cases 2 and 3 of Lemma 6.4.8, we will now assume that at least one of the subsequences \( S[L_{\text{even}}] \) and \( S[R_{\text{even}}] \) is constant. Note that this case was also used in the proof of the \( \mathcal{NP} \)-hardness of \( L-R-3 \)-cadence detection in grammar-compressed strings. In particular, if the constant subsequence is as short as in Figure 6.6, we may have to check linearly many possible candidates in order to decide whether the string has an even \( L-R-3 \)-cadence. However, if the constant subsequence is as long as in Figure 6.7, it may be sufficient to check whether the first or the last 0 in the other subsequence is contained in an even \( L-R-3 \)-cadence.

If we have a constant even subsequences for both the set of starting indices and the set of ending indices, Lemma 6.4.9 allows us to efficiently determine, whether there is an even \( L-R-3 \)-cadence. Similarly, if we have a constant even subsequences for the set of starting indices and a constant substring for the middle indices, we can also efficiently determine, whether there is an even \( L-R-3 \)-cadence.
Figure 6.7: A string with 48 characters with intervals $L = \{0, 1, \ldots, 15\}$ and $R = \{32, 34, \ldots, 47\}$. For each index of $R_{\text{even}}$, there are eight candidates $(i, d, 3)$ for forming an $L$-$R$-$3$-cadence.

**Lemma 6.4.10.** Let $S$ be a binary string and $L$ and $R$ be two disjoint intervals and $M$ be an interval such that $S[L_{\text{even}}] = 0^i$ and $S[M] = 0^j$ hold for some integers $i, j$.

Then, we can decide whether $S$ contains an even $L$-$R$-$3$-cadence with middle element in $M$ in $O(|L| + |R|)$ if $S$ is uncompressed and in linear time with respect to the size of the grammar of $S$ if $S$ is grammar-compressed.

**Proof.** The proof follows the same idea as the proof of Lemma 6.4.9.

Let $l_{\text{min}}, l_{\text{max}}, m_{\text{min}}, m_{\text{max}}, r_{\text{min}}$ and $r_{\text{max}}$ be the minimal and maximal indices of $L_{\text{even}}, M$ and $R_{\text{even}}$, respectively.

By construction, each even $L$-$R$-$3$-cadence with middle element in $M$ has an ending index of at least $r'_{\text{min}} = \max(r_{\text{min}}, 2m_{\text{min}} - l_{\text{max}})$ and at most $r'_{\text{max}} = \min(r_{\text{max}}, 2m_{\text{max}} - l_{\text{min}})$.

We define the set $R'_{\text{even}} = \{r'_{\text{min}}, r'_{\text{min}} + 1, \ldots, r'_{\text{max}}\}$. Using this definition, if $S[R'_{\text{even}}]$ does not contain a 0, there is no even $L$-$R$-$3$-cadence with middle element in $M$.

If, on the other hand, $S[R'_{\text{even}}]$ contains a 0 at index $r$, then we define $l := \max(l_{\text{min}}, 2m_{\text{min}} - r)$. We will show that the subsequence $(l, \frac{l+r}{2}, 3)$ is an even $L$-$R$-$3$-cadence with middle element in $M$.

Since $l = \max(l_{\text{min}}, 2m_{\text{min}} - r)$ holds and both $l_{\text{min}}$ and $r$ are even, the starting index $l$ is even as well.

Furthermore, the inequality $r \geq r'_{\text{min}} = \max(r_{\text{min}}, 2m_{\text{min}} - l_{\text{max}})$ implies $l_{\text{max}} \geq 2m_{\text{min}} - r$ and thus $l_{\text{max}} \geq l = \max(l_{\text{min}}, 2m_{\text{min}} - r) \geq l_{\text{min}}$. Therefore, the index $l$ is an index in $L$. In particular, the character $S[l]$ is 0.

The middle index is given by $\frac{l+r}{2} = \max(l_{\text{min}}, 2m_{\text{min}} - l_{\text{max}})$.

Using $r \leq \min(r_{\text{max}}, 2m_{\text{max}} - l_{\text{min}}) \leq 2m_{\text{max}} - l_{\text{min}}$, we obtain $\frac{l_{\text{min}}+r}{2} \leq m_{\text{max}}$. 
Figure 6.8: A string with 48 characters after one pass of the while-loop of Algorithm 1 with \( L = \{0, 1, \ldots, 8\} \) and \( R = \{32, 33, \ldots, 47\} \). Firstly, the index \( r_0 = 32 \) is found. The minimal and maximal candidates for \( L-R-3 \)-cadences with ending index \( r_0 \) are marked with red. Secondly, the index \( m_0 = 21 \) is found. The minimal and maximal candidates for \( L-R-3 \)-cadences with middle index \( m_0 \) are marked with yellow. Afterwards, the gray characters are guaranteed not to form a 3-cadence with characters from the first block of the string.

Since both \( \frac{l_{\text{min}} + r}{2} \) and \( m_{\text{min}} \) are at most \( m_{\text{max}} \) and their maximum is at least \( m_{\text{min}} \), the middle index \( \frac{l_{\text{min}} + r}{2} \) is indeed in \( M \). In particular, the character \( S[\frac{l_{\text{min}} + r}{2}] \) is 0.

Hence, the subsequence \( (l, \frac{l_{\text{min}} + r}{2}, 3) \) is an even \( L-R-3 \)-cadence with middle element in \( M \).

We can check whether \( S[R'_{\text{even}}] \) contains a 0, and thereby whether \( S \) contains at least one even \( L-R-3 \)-cadence with middle element in \( M \), in \( O(|L| + |M|) \subseteq O(|L| + |R|) \) time in uncompressed strings and in linear time with respect to the size of the grammar in grammar-compressed strings.

Note that we only require the intervals \( L \) and \( R \) to be disjoint in order to avoid reporting subsequences with distance 0. We therefore do not require the stronger assumption that \( L \) and \( M \) are disjoint.

Lemma 6.4.11. Let \( S \) be a binary string and \( L \) and \( R \) be two disjoint intervals such that \( S[L_{\text{even}}] = 0^i \) holds for some integer \( i \).

Then, we can decide whether the string \( S \) contains an even \( L-R-3 \)-cadence in \( O(|L| + |R|) \) time if \( S \) is uncompressed and in \( O(|G|^{|R|/|E|}) \) time if \( S \) is given by the grammar \( G \).

Proof. We will show that Algorithm 1 has the proposed time complexity and finds an \( L-R-3 \)-cadence if an \( L-R-3 \)-cadence exists. The first iteration of this algorithm is presented in Figure 6.8.
Algorithm 1: Even L-R-3-cadence detection for constant $S_{[L_{\text{even}}]}$

**input**: A string $S$ and two disjoint Intervals $L$ and $R$ with $S_{[L_{\text{even}}]} = 0^i$

**output**: An even $L$-R-3-cadence $(i, d, 3)$ if possible, otherwise NONE

1. $l_{\text{min}} \leftarrow$ first even index in $L$;
2. $l_{\text{max}} \leftarrow$ last even index in $L$;
3. $r_{\text{max}} \leftarrow$ last even index in $R$;
4. while $R \neq \emptyset$ do
5.  \hspace{1em} $r_0 \leftarrow$ index of first 0 in $S_{[R_{\text{even}}]}$ (or NONE);
6.  \hspace{1em} if $r_0 = \text{NONE}$ then
7.  \hspace{2em} return NONE;
8.  \hspace{1em} end
9.  \hspace{1em} Use Lemma 6.4.9 to find an even $L$-{$r_0$}-3-cadence;
10. \hspace{1em} if $L$-{$r_0$}-3-cadence found then
11. \hspace{2em} return $L$-{$r_0$}-3-cadence;
12. \hspace{1em} end
13. \hspace{1em} $m_0 \leftarrow$ index of first 0 in $S$ after $\frac{l_{\text{max}}+r_0}{2}$ until $\frac{l_{\text{max}}+r_{\text{max}}}{2}$ (or NONE);
14. \hspace{1em} if $m_0 = \text{NONE}$ then
15. \hspace{2em} return NONE;
16. \hspace{1em} end
17. \hspace{1em} Use Lemma 6.4.10 to find an even $L$-R-3-cadence with middle index $m_0$;
18. \hspace{1em} if $L$-R-3-cadence found then
19. \hspace{2em} return $L$-R-3-cadence;
20. \hspace{1em} end
21. \hspace{1em} $R \leftarrow \{r \in R | r > 2m_0 - l_{\text{min}}\}$;
22. \hspace{1em} end
23. return NONE;
Firstly, we explain the algorithm and show that it is correct.

The algorithm finds the next 0 in \( R \) and \( M \) and uses the Lemmata 6.4.9 and 6.4.10 alternately until either the an \( L-R-3 \)-cadence is found or until it is shown that no \( L-R-3 \)-cadence exists. Hence, by construction, if the algorithm outputs an \( L-R-3 \)-cadence, there is indeed an \( L-R-3 \)-cadence in the string.

For the correctness it is left to show that if the string contains at least one \( L-R-3 \)-cadence, the algorithm outputs an \( L-R-3 \)-cadence. We will therefore assume that the algorithm does not yield an \( L-R-3 \)-cadence and show that the algorithm does not discard any \( L-R-3 \)-cadences in this case.

Since \( S[L_{\text{even}}] \) does only contain the character 0, each even \( L-R-3 \)-cadence has to use the character 0. Therefore, no even index lesser than \( r_0 \) can be the ending index of an even \( L-R-3 \)-cadence. Furthermore, if there is no index \( r_0 \in R_{\text{even}} \) with \( S[r_0] = 0 \), there is no even \( L-R-3 \)-cadence at all.

Since for each even \( L-R-3 \)-cadence, the starting index is at least \( l_{\text{min}} \) and the ending index is at least \( r_0 \), the middle index is at least \( \frac{l_{\text{min}} + r_0}{2} \). Note that if Lemma 6.4.9 does not find an even \( L-R-3 \)-cadence, there is no 0 in the interval \( \{ \frac{l_{\text{min}} + r_0}{2}, \frac{l_{\text{min}} + r_0}{2} + 1, \ldots, \frac{l_{\text{max}} + r_0}{2} \} \).

Since each even \( L-R-3 \)-cadence has to use the character 0, no index lesser than \( m_0 \) can be the middle index of an even \( L-R-3 \)-cadence. Furthermore, if there is no index \( m_0 \in \{ \frac{l_{\text{min}} + r_0}{2}, \frac{l_{\text{min}} + r_0}{2} + 1, \ldots, \frac{l_{\text{max}} + r_0}{2} \} \) with \( S[m_0] = 0 \), there is no even \( L-R-3 \)-cadence at all.

Since \( m_0 \) is the smallest index that can be the middle index of an even \( L-R-3 \)-cadence, no index lesser than \( l_{\text{max}} + 2(m_0 - l_{\text{max}}) = 2m_0 - l_{\text{max}} \) can be the ending index of an even \( L-R-3 \)-cadence. If Lemma 6.4.10 does not find an even \( L-R-3 \)-cadence, there is no 0 in \( \{ 2m_0 - l_{\text{max}}, 2m_0 - l_{\text{max}} + 1, \ldots, 2m_0 - l_{\text{min}} \} \). Therefore, there cannot be an even \( L-R-3 \)-cadence with ending index lesser than \( 2m_0 - l_{\text{min}} \). This implies that the algorithm does not discard any \( L-R-3 \)-cadence.

Next, we will calculate how much an iteration of the while loop reduces the size of \( R \). For the analysis, we ignore the last iteration of the while loop. Hence, we only consider the iterations of the while loop that leave the interval \( R \) nonempty. We define \( \Delta_r \) and \( \Delta_m \) by \( \Delta_r = r_0 - r_{\text{min}} \) and \( \Delta_m = m_0 - \frac{l_{\text{max}} + r_0}{2} - 1 \). In particular, the equation

\[
2m_0 = 2\Delta_m + l_{\text{max}} + r_0 + 2 = 2\Delta_m + l_{\text{max}} + \Delta_r + r_{\text{min}} + 2
\]

holds. Therefore, the number of indices that are removed from \( R \) are given by

\[
2m_0 - l_{\text{min}} - r_{\text{min}} + 1 = l_{\text{max}} - l_{\text{min}} + 3 + 2\Delta_m + \Delta_r > |L| + \Delta_m + \Delta_r.
\]

Therefore, each iteration of the while loop reduces the size of \( R \) by at least \( |L| \) characters.
This implies that the while loop is executed at most $\frac{|R|}{|L|}$ times. Since in grammar-compressed strings all operations inside the while loop can be executed in linear time, Algorithm 1 needs $O(|G| \frac{|R|}{|L|})$ time in grammar-compressed strings.

In uncompressed strings, the application of the Lemmata 6.4.9 and 6.4.10 in Algorithm 1 take $O(|L| + 1) = O(|L|)$ time. Finding $r_0$ and $m_0$ can be done by reading the string left-to-right. If unsuccessful, this takes $O(|L| + |R|)$ time and there is no further iteration of the while loop. If there are suitable indices $r_0$ and $m_0$, we find them in $O(\Delta_r)$ time and $O(\Delta_m)$ time, respectively. Since we remove at least $|L| + \Delta_m + \Delta_r$ indices from $R$, the time spent in each iteration of the while loop is linear with respect to the number of indices removed from $R$. Thus, in all iterations combined, we need $O(|L| + |R|)$ time in uncompressed strings.

By symmetry, the detection of odd $L$-$R$-3-cadences can also be done as the detection of even $L$-$R$-3-cadences.

**Lemma 6.4.12.** Let $S$ be a binary string and $L$ and $R$ be two disjoint intervals.

Then, we can decide whether the string $S$ contains an $L$-$R$-3-cadence in $O(|L| + |R|)$ time if $S$ is uncompressed and in $O(|G| \max \left( \frac{|R|}{|L|}, \frac{|L|}{|R|} \right))$ time if $S$ is given by the grammar $G$.

We will now allow overlapping intervals $L$ and $R$.

**Theorem 6.4.13.** Let $S$ be a binary string and $L$ and $R$ be two (not necessarily disjoint) intervals.

Then, we can decide whether the string $S$ contains an $L$-$R$-3-cadence in $O(|L| + |R|)$ time if $S$ is uncompressed and in $O(|G| \max \left( \frac{|R|}{|L|}, \frac{|L|}{|R|} \right))$ time if $S$ is given by the grammar $G$.

**Proof.** If $L$ and $R$ are disjoint, there is nothing left to prove. Otherwise, if the intervals $L$ and $R$ overlap by at least 9 characters, the underlying binary string contains a 3-subcadence on the substring defined by the intersection of $L$ and $R$ and thus an $L$-$R$-3-cadence.

Let now $L$ and $R$ be two intervals with an overlap of at most 8 characters. Define the overlapping part $O = L \cap R$ as well as the two non-overlapping parts $L' = L \setminus R$ and $R' = R \setminus L$.

By construction, each $L$-$R$-3-cadence is either

- an $L'$-$R'$-3-cadence,
- an $L'$-$O$-3-cadence,
• an $O$-$R'$-3-cadence or
• an $O$-3-cadence.

Since both $\frac{|L'|}{|O|}$ and $\frac{|R'|}{|O|}$ can be large, we use that each $L'$-$R'$-3-cadence is at least one of the following

• an $L'$-$R'$-3-cadence or an $L'$-$O$-3-cadence (i.e. a $L'$-$R'$-3-cadence),
• an $L'$-$R'$-3-cadence or an $O$-$R'$-3-cadence (i.e. a $L$-$O$-3-cadence),
• an $O$-$O$-3-cadence (i.e. a subcadence of $S[O]$).

If $L'$ (or $R'$) is empty, there are no $L'$-$R'$-3-cadences (or no $L$-$R'$-3-cadences, respectively).

Otherwise, the subinterval $L'$ (or $R'$) has at least length 1 while $O$ has at most length 8. In particular, the subinterval $O$ is at most 8 times as long as $L'$ (or $R'$). Since by construction $L = L' \cup O$ and $R = R' \cup O$ hold, we get $|L| \leq 9|L'|$ and $|R| \leq 9|R'|$. Therefore, the maxima $\max\left(\frac{|R|}{|L'|}, \frac{|L'|}{|R'|}\right)$ and $\max\left(\frac{|R'|}{|L|}, \frac{|L|}{|R'|}\right)$ are bounded by $9 \cdot \max\left(\frac{|R|}{|L'|}, \frac{|L'|}{|R'|}\right)$.

Therefore, both the existence of $L'$-$R'$-3-cadences and the existence of $L$-$R'$-3-cadence can be checked in, up to a constant factor, the same time as the existence of $L$-$R$-3-cadences.

The existence of subcadences of $S[O]$ can be checked by reading all of the at most 8 characters and checking whether this substring of bounded length contains a subcadence. This takes constant time in uncompressed strings and $O(|G|)$ time in grammar-compressed strings.

Next, we will show that 3-cadences can be detected in linear time in uncompressed strings and in $O(|G| \log |S|)$ time in strings $S$ given by the grammar $G$.

The possible index pairs of 3-cadence given by Figure 6.4 form a square and to axis-aligned right-angled triangles. Hence, the corresponding non-rectangular convolution can be seen as a sum of rectangular convolutions without subtractions. Since each rectangular convolution corresponds to two intervals $L$ and $R$ with their $L$-$R$-3-cadences, the 3-cadences can naturally be derived from $L$-$R$-3-cadences.

However, if the string $S$ does not contain a 3-cadence, this approach has to check for $\Theta(S)$ many pairs of intervals $L$, $R$ whether an $L$-$R$-3-cadence exists. This implies that this approach will not result in a polynomial time detection algorithm for 3-cadences in grammar-compressed strings. Therefore, we will adapt the Lemmata 6.4.9 and 6.4.10 to decide directly to the 3-cadence detection.
In this thesis, we consider this approach only for even 3-cadences. The case of odd 3-cadences is similar, but not quite as easy as replacing each occurrence of the word “even” by the word “odd”. Therefore, for the odd 3-cadences please refer to our prior paper [80]. Since in this paper, the first character of the string has index 1, the even 3-cadences in this paper naturally translate into the odd 3-cadences of this thesis.

Firstly, we note that it is sufficient to only consider even 3-cadences that start in one of the first two runs of $S_{\text{even}}$ or end in one of the two last runs of $S_{\text{even}}$.

Lemma 6.4.14. Let $S$ be a string.

Let $i$ be the index of the first character in the second run in $S_{\text{even}}$ and $j$ be the index of the last character in the third to last run in $S_{\text{even}}$. Let further $d = \frac{i-1}{2}$.

If both $i - d < 0$ and $i + 3d \geq n$ hold, then there is an even 3-cadence that starts in one of the first two runs of $S_{\text{even}}$ and ends in one of the three last runs of $S_{\text{even}}$.

Proof. Without loss of generality, the character $S[i]$ is 1 and therefore, the character $S[i - 2]$ is 0. Let $j' + 2$ be the first character of the last 0-run in $S_{\text{even}}$. Therefore, the character $S[j']$ is 1. We define $d'$ by $d' = \frac{j' + i}{2}$. By construction, the inequality $j' \geq j$ holds. Therefore, the inequality $d' \geq d$ also holds.

Similarly to Lemma 6.4.7 that proved the existence of an $L$-$R$-3-cadence in a similar situation, we will show that either $(i, d', 3)$ or $(i - 2, d' + 2, 3)$ is a 3-cadence.

By construction of $d'$, we have $j' = i + 2d'$. Therefore, if $S[i + d'] = 1$ holds, we have $S[i] = S[i + d'] = S[i + 2d'] = 1$, $i - d' \leq i - d < 0$ and $i + 3d' = j' + d' \geq j + d \geq n$. Therefore in this case, the arithmetic subsequence $(i, d', 3)$ is a 3-cadence.

Similarly, we have $j' + 2 = (i - 2) + 2(d' + 2)$. Therefore, if $S[i + d'] \neq 1$ holds, we have $S[(i - 2)] = S[(i - 2) + (d' + 2)] = S[(i - 2) + 2(d' + 2)] = 0$, $(i - 2) - (d' + 2) < i - d < 0$ and $(i - 2) + 3(d' + 2) = (j' + 2) + (d' + 2) > j + d \geq n$. Therefore in this case, the arithmetic subsequence $(i - 2, d' + 2, 3)$ is a 3-cadence.

In either case, we obtain a 3-cadence. \(\square\)

Note that this lemma is not symmetric, because on the left-hand side of the string, we only care about the length of the first run of $S_{\text{even}}$, while on the right-hand side of the string, we consider the length of the two last runs of $S_{\text{even}}$.

If Lemma 6.4.14 does not give us a 3-cadence, we have to consider the three following cases.
• 3-cadences that start in the first 0-run of $S_{\text{even}}$ and end in one of the two last 0-runs of $S_{\text{even}},$

• 3-cadences that start in the first 1-run of $S_{\text{even}}$ and end in one of the two last 1-runs of $S_{\text{even}},$

• 3-cadences that start in the first run of $S_{\text{even}}$ and end in any run of $S_{\text{even}}$ except for the last three runs.

The first two cases are similar to the case of Lemma 6.4.9 and can be similarly efficiently computed. Note that each 3-cadence starts in the first third of the string and ends in the last third of the string. Therefore, we can assume the first runs and the last runs to be disjoint by truncating them.

Lemma 6.4.15. Let $S$ be a binary string and $L$ and $R$ be two disjoint intervals such that $S[L_{\text{even}}] = 0^i$ and $S[R_{\text{even}}] = 0^j$ hold for some integers $i$, $j$.

Then, we can decide whether the string $S$ contains an even 3-cadence that starts in $L$ and ends in $R$ in $O(|L| + |R|)$ time if $S$ is uncompressed and in linear time with respect to the size of the grammar of $S$ if $S$ is grammar-compressed.

Proof. Similarly to the proof of Lemma 6.4.9, we will show that if and only if the substring of allowed middle indices contains a 0, then there is an even 3-cadence that starts in $L$ and ends in $R$.

Let $l_{\text{min}}, l_{\text{max}}, r_{\text{min}}$ and $r_{\text{max}}$ be the minimal and maximal indices of $L_{\text{even}}$ and $R_{\text{even}}$, respectively.

The proof of Lemma 6.4.9 shows that the middle index of any even $L$-$R$-3-cadence lies in the interval $M := \{l_{\text{min}} + r_{\text{min}} \frac{1}{2}, l_{\text{min}} + r_{\text{min}} + 1, \ldots, l_{\text{max}} + r_{\text{max}} \frac{1}{2}\}$. However, for the 3-cadence $(i, d, 3)$, we have the additional restrictions that $i - d < 0$ and $i + 3d \geq |S|$ have to hold.

For a given middle index $m$, the distance $d$ is bounded from above by $m - l_{\text{min}}$ and by $r_{\text{max}} - m$. Since we require both $m - 2d < 0$ and $m + 2d \geq |S|$, we obtain the inequalities

• $0 > m - 2d \geq m - 2(m - l_{\text{min}}) = 2l_{\text{min}} - m \Rightarrow m > 2l_{\text{min}},$

• $|S| \leq m + 2d \leq m + 2(m - l_{\text{min}}) = 3m - 2l_{\text{min}} \Rightarrow m \geq \frac{|S| + 2l_{\text{min}}}{3},$

• $0 > m - 2d \geq m - 2(r_{\text{max}} - m) = 3m - 2r_{\text{max}} \Rightarrow m < \frac{2r_{\text{max}}}{3}$ and

• $|S| \leq m + 2d \leq m + 2(r_{\text{max}} - m) = 2r_{\text{max}} - m \Rightarrow m \leq 2r_{\text{max}} - |S|.$
We therefore define
\[ m'_\text{min} = \max \left( \frac{l_{\text{min}} + r_{\text{min}}}{2}, 2l_{\text{min}} + 1, \frac{|S| + 2l_{\text{min}}}{3} \right) \]
and
\[ m'_\text{max} = \min \left( \frac{l_{\text{max}} + r_{\text{max}}}{2}, \frac{2r_{\text{max}}}{3} - 1, 2r_{\text{max}} - |S| \right) . \]

Therefore, if an even 3-cadence that starts in \( L \) and ends in \( R \) exists, then \( M' := \{ m'_\text{min}, m'_\text{min} + 1, \ldots, m'_\text{max} \} \) contains a 0.

Conversely, we will show that if \( M' \) contains a 0 at an index \( m \), then \( S \) contains a 3-cadence that starts in \( L \) and ends in \( R \). We therefore define the distance \( d := \min(m - l_{\text{min}}, r_{\text{max}} - m) \). The proof of Lemma 6.4.9 already proves that \((m - d, d, 3)\) is an even \( L-R \)-3-cadence. Thus, it is sufficient to show that both \( m - 2d < 0 \) and \( m + 2d \geq |S| \) hold.

Since both \( m > 2l_{\text{min}} \) and \( m < \frac{2r_{\text{max}}}{3} \) hold and \( d \) is either \( m - l_{\text{min}} \) or \( r_{\text{max}} - m \), we obtain either
\[ m - 2d = m - 2(m - l_{\text{min}}) = 2l_{\text{min}} - m < 0 \]
or
\[ m - 2d = m - 2(r_{\text{max}} - m) = 3m - 2r_{\text{max}} < 0. \]
In either case, we obtain the inequality \( m - 2d < 0 \).

Similarly, since both \( m \geq \frac{|S| + 2l_{\text{min}}}{3} \) and \( m \leq 2r_{\text{max}} - |S| \) hold, we obtain either the inequality
\[ m + 2d = m + 2(m - l_{\text{min}}) = 3m - 2l_{\text{min}} \geq |S| \]
or
\[ m + 2d = m + 2(r_{\text{max}} - m) = 2r_{\text{max}} - m \geq |S|. \]
In either case, we obtain \( m + 2d \geq |S| \).

Therefore, if \( M' \) contains a 0, we obtain a 3-cadence that starts in \( L \) and ends in \( R \).

We also want a fast algorithm to decide, whether there is a 3-cadence that starts in some constant subsequence \( L_{\text{even}} \) and has its middle index in a constant substring \( M \). The following lemma is therefore very similar to Lemma 6.4.10.

**Lemma 6.4.16.** Let \( S \) be a binary string and \( L \) and \( M \) be intervals such that \( S[L_{\text{even}}] = 0^i \) and \( S[M] = 0^j \) hold for some integers \( i, j \).

Then, we can decide whether \( S \) contains an even 3-cadence that starts in \( L \) and has its middle element in \( M \) in \( O(\|L| + |M|) \) if \( S \) is uncompressed and in linear time with respect to the size of the grammar of \( S \) if \( S \) is grammar-compressed.
Proof. Similarly to the proof of Lemma 6.4.10, we will show that if and only if the substring of allowed ending indices contains a 0, then there is an even 3-cadence that starts in L and has its middle index in M.

The translation of Lemma 6.4.10 to this lemma therefore follows the same path as the translation of Lemma 6.4.15 to the previous lemma. Hence, we will firstly restrict the set of possible ending positions. Afterwards, we will show that if and only if there is a 0 in this restricted subsequence, then there is an even 3-cadence that starts in L and has its middle index in M.

Let \( l_{\min}, l_{\max}, m_{\min}, m_{\max} \) be the minimal and maximal indices of \( L_{\text{even}} \) and \( M \), respectively.

In the proof of Lemma 6.4.10, we have already seen that each 3-subcadence that starts in \( L_{\text{even}} \) and has its middle element in \( M \), has its last element in the interval \( R := \{2m_{\min} - l_{\max}, 2m_{\min} - l_{\max} + 1, \ldots, 2m_{\max} - l_{\min}\} \). For the 3-cadence \((i, d, 3)\), we also require that \( i - d < 0 \) and \( i + 3d \geq |S| \) hold.

For a given ending index \( r \), the distance \( d \) is bounded from above by \( r - l_{\min} \) and by \( r - m_{\min} \). Since we require both \( r - 3d < 0 \) and \( r + d \geq |S| \) hold, we obtain the inequalities

- \( 0 > r - 3d \geq r - 3\left(\frac{r - l_{\min}}{2}\right) = \frac{3l_{\min} - r}{2} \Rightarrow r > 3l_{\min}, \)
- \( |S| \leq r + d \leq r + \left(\frac{r - l_{\min}}{2}\right) = \frac{3r - l_{\min}}{2} \Rightarrow r \geq \frac{2|S| + l_{\min}}{3}, \)
- \( 0 > r - 3d \geq r - 3(r - m_{\min}) = 3m_{\min} - 2r \Rightarrow r > \frac{3m_{\min}}{2} \) and
- \( |S| \leq r + d \leq r + (r - m_{\min}) = 2r - m_{\min} \Rightarrow r \geq \frac{|S| + m_{\min}}{2} \).

We therefore define

\[
 r'_{\min} = \max \left(2m_{\min} - l_{\max}, 3l_{\min} + 1, \frac{2|S| + l_{\min}}{3}, 3m_{\min} \frac{2}{2} + 1, \frac{|S| + m_{\min}}{2}\right)
\]

and

\[
 r'_{\max} = 2m_{\max} - l_{\min}.
\]

Therefore, if an even 3-cadence that starts in L has its middle index in M exists, then for \( R' := \{r'_{\min}, r'_{\min} + 1, \ldots, r'_{\max}\} \), the subsequence \( R'_{\text{even}} \) contains a 0.

Conversely, we will prove that if \( R' \) contains a 0 at an index \( r \), then \( S \) contains a 3-cadence that starts in L and has its middle index in M. In order to prove this, we define \( l := \max(l_{\min}, 2m_{\min} - r) \). The proof of Lemma 6.4.10 shows that \((l, r - l, 3)\) is an even L-R-3-cadence with middle element in M. Since the index \( r \) is in \( R' \), it is even an L-R'-3-cadence. Since \( r = l + 2 \left(\frac{r - l}{2}\right)\) holds, it is therefore left to show that both \( r - 3 \left(\frac{r - l}{2}\right) < 0 \) and \( r + \left(\frac{r - l}{2}\right) \geq |S| \) hold.
CHAPTER 6. CADENCES

Both \( r > 3l_{\min} \) and \( r \geq \frac{2n+l_{\min}}{3} \) hold. Thus, if \( l = l_{\min} \) holds, we obtain

\[
r - 3 \left( \frac{r - l}{2} \right) = r - 3 \left( \frac{r - l_{\min}}{2} \right) = \frac{3l_{\min} - r}{2} < 0
\]

and

\[
r + \left( \frac{r - l}{2} \right) = r + \left( \frac{r - l_{\min}}{2} \right) = \frac{3r - l_{\min}}{2} \geq |S|.
\]

Similarly, both \( r > \frac{3m_{\min}}{2} \) and \( r \geq \frac{n+m_{\min}}{2} \) hold. Hence, if \( l \neq l_{\min} \) holds, we obtain \( l = 2m_{\min} - r \) and hence

\[
r - 3 \left( \frac{r - l}{2} \right) = r - 3 (r - m_{\min}) = 3m_{\min} - 2r < 0
\]

and

\[
r + \left( \frac{r - l}{2} \right) = r + (r - m_{\min}) = 2r - m_{\min} \geq |S|.
\]

In both cases, we obtain the inequalities \( r - 3 \left( \frac{r - l}{2} \right) < 0 \) and \( |S| \leq r + \left( \frac{r - l}{2} \right) \).

Therefore, if \( R' \) contains a 0, we obtain a 3-cadence that starts in \( L \) has its middle index in \( M \). \qed

If Lemma 6.4.14 does not already yield a 3-cadence, we will now consider the 3-cadences that start in the first run of \( S_{\text{even}} \) and end in any run of \( S_{\text{even}} \) except for the last three runs.

Let \( L \) be an interval such that \( S[L_{\text{even}}] \) is the first run of \( S_{\text{even}} \) and let \( l_{\max} \) be the last even index in \( L \). Since Lemma 6.4.14 does not yield a 3-cadence, the index \( l_{\max} + 2 \) cannot be the starting index of a 3-cadence that ends in the third-to-last run of \( S_{\text{even}} \) or any of its preceding runs. Each 3-subcadence with distance of at least \( \frac{|S|}{3} \) is guaranteed to be a 3-cadence. Therefore, the index \( l_{\max} + 2 + \left\lceil \frac{2}{3}|S| \right\rceil \) lies in one of the two last runs of \( S_{\text{even}} \), and the index \( l_{\max} + \left\lceil \frac{2}{3}|S| \right\rceil \) lies in one of the three last runs of \( S_{\text{even}} \).

We will therefore use the Lemmata 6.4.15 and 6.4.16 to check for 3-cadences that start in \( S[L_{\text{even}}] \) and end with an index of at most \( l_{\max} + \left\lceil \frac{2}{3}|S| \right\rceil \). The resulting algorithm is given by Algorithm 2 and is very similar to Algorithm 1.

If a 3-cadence that starts in the first run of \( S_{\text{even}} \) and ends in any run of \( S_{\text{even}} \) except for the last three runs exist, Algorithm 2 is guaranteed to find a 3-cadence.

**Lemma 6.4.17.** Let \( S \) be a binary string and \( L \) be an interval such that \( S[L_{\text{even}}] \) is the first run of \( S_{\text{even}} \). Let further \( l_{\max} \) be the last even index in \( L \).

Then, we can decide whether the string \( S \) contains an even 3-cadence that starts in \( L \) and ends with an index of at most \( l_{\max} + \left\lceil \frac{2}{3}|S| \right\rceil \) in linear time if \( S \) is uncompressed and in \( \mathcal{O}(|G| \log |S|) \) time if \( S \) is given by the grammar \( G \).
Proof. We will show that Algorithm 2 has the proposed time complexity and finds a 3-cadence as described in the claim if such a 3-cadence exists.

The proof of the correctness is virtually identically to the proof of the correctness of Algorithm 1 in Lemma 6.4.11.

Next, we will calculate how much an iteration of the while loop increases the variable $r_{\text{min}}$.

The maximal index $m'_{\text{max}}$ of $M'$ in the application of Lemma 6.4.15 is given by

$$m'_{\text{max}} = \min \left( \frac{l_{\text{max}} + r_0}{2}, \frac{2r_0}{3} - 1, 2r_0 - |S| \right)$$

and maximal index $r'_{\text{max}}$ of $R'$ in the application of Lemma 6.4.16 is given by

$$r'_{\text{max}} = 2m_0 - l_{\text{min}}.$$

Since $r_0 \geq r_{\text{min}}$, $m_0 \geq m'_{\text{max}} + 1$ and $l_{\text{min}} = 0$ hold, we obtain

$$r'_{\text{max}} \geq 2 \min \left( \frac{l_{\text{max}} + r_{\text{min}}}{2}, \frac{2r_{\text{min}}}{3} - 1, 2r_{\text{min}} - |S| \right) + 2.$$

If the minimum is $\frac{l_{\text{max}} + r_{\text{min}}}{2}$, we get

$$r'_{\text{max}} > l_{\text{max}} + r_{\text{min}} \geq l_{\text{max}} + \left\lceil \frac{2}{3} |S| \right\rceil = r_{\text{max}}.$$

Therefore, this case finishes the while loop in a single iteration.

If the minimum is $\frac{2r_{\text{min}}}{3} - 1$, we get

$$r'_{\text{max}} \geq 4 \frac{r_{\text{min}}}{3} = r_{\text{min}} + \frac{r_{\text{min}}}{3} \geq r_{\text{min}} + \frac{2}{9} |S|.$$

Since we can assume that $l_{\text{min}} < \frac{1}{3} |S|$ holds, after at most 2 iterations of the while loop with this case, the while loop is finished.

Therefore, the interesting case is that the minimum is $2r_{\text{min}} - |S|$. In this case we get

$$r'_{\text{max}} \geq 4r'_{\text{min}} - 2 |S| + 2 = \frac{2}{3} |S| + 4 \left( r'_{\text{min}} - \frac{2}{3} |S| \right) + 2.$$

Thus, in each iteration of the while loop, the distance between $r_{\text{min}}$ and $\frac{2}{3} |S|$ quadruples. Therefore, after $\mathcal{O} (\log |S|)$ iterations of the while loop with this case, the while loop is finished.

In each combination of cases, we need at most $\mathcal{O} (\log |S|)$ iterations of the while loop.
CHAPTER 6. CADENCES

If $S$ is given by the grammar $G$, each application of the Lemmata 6.4.15 and 6.4.16 takes $\mathcal{O}(|G|)$ time. Therefore, Algorithm 2 takes $\mathcal{O}(|G| \log |S|)$ time.

If $S$ is uncompressed, the time complexities given by the Lemmata 6.4.15 and 6.4.16 do not lead to linear time complexity for Algorithm 2. However, in each iteration of the while loop, we read the characters with the following indices at most once.

- The even indices between $r_{\text{min}}$ and $r_{\text{max}}'$ and
- the indices between $\frac{l_{\text{min}}+r_{\text{min}}}{2}$ and $\frac{l_{\text{min}}+r_{\text{max}}'}{2}$.

These are $\mathcal{O} \left( r_{\text{max}}' - r_{\text{min}} \right)$ characters. Since $r_{\text{min}}$ increases by $r_{\text{max}}' - r_{\text{min}}$ with the iteration of the while loop, Algorithm 2 takes linear time on uncompressed strings.

We can now prove the time complexity for the detection of 3-cadences in binary strings.

**Theorem 6.4.18.** Let $S$ be a binary string.

Then, we can decide whether the string $S$ contains an 3-cadence in linear time if $S$ is uncompressed and in $\mathcal{O}(|G| \log |S|)$ time if $S$ is given by the grammar $G$.

**Proof.** Firstly, we check whether Lemma 6.4.14 already yields a 3-cadence. We can do this in linear time if $S$ is uncompressed and in $\mathcal{O}(|G|)$ time if $S$ is grammar-compressed.

If Lemma 6.4.14 does not give us a 3-cadence, we check the following three cases.

- 3-cadences that start in the first 0-run of $S_{\text{even}}$ and end in one of the two last 0-runs of $S_{\text{even}}$;
- 3-cadences that start in the first 1-run of $S_{\text{even}}$ and end in one of the two last 1-runs of $S_{\text{even}}$;
- 3-cadences that start in the first run of $S_{\text{even}}$ and end in any run of $S_{\text{even}}$ except for the last three runs.

The first two cases can checked with Lemma 6.4.15 in linear time if $S$ is uncompressed and in $\mathcal{O}(|G|)$ time if $S$ is grammar-compressed.

The last case can be checked with Algorithm 2 which needs linear time if $S$ is uncompressed and in $\mathcal{O}(|G| \log |S|)$ time if $S$ is grammar-compressed.

Combining all cases gives the wanted time complexity. \qed
Algorithm 2: Detection of even 3-cadences starting in the first run

**input**: A string $S$ and an interval $L$ such $S[L_{even}] = 0'$ is the first run of $S_{even}$

**output**: An even 3-cadence $(i, d, 3)$ starting in $L$ and with an ending index of at most $|L| + \frac{2}{3}|S|$ if possible, otherwise NONE

1. $l_{\text{min}} \leftarrow $ first even index in $L$;
2. $l_{\text{max}} \leftarrow $ last even index in $L$;
3. $r_{\text{min}} \leftarrow \lceil \frac{2}{3}|S| \rceil$;
4. $r_{\text{max}} \leftarrow l_{\text{max}} + \lceil \frac{2}{3}|S| \rceil$;
5. while $r_{\text{min}} \leq r_{\text{max}}$ do
6.     $r_0 \leftarrow $ index of first 0 in $S_{even}$ starting from $r_{\text{min}}$ (or NONE);
7.     if $r_0 = $ NONE then
8.         return NONE;
9.     end
10.     Use Lemma 6.4.15 to find an even 3-cadence starting in $L$
11.     and ending in $r_0$;
12.     if 3-cadence found then
13.         return 3-cadence;
14.     end
15.     $m'_{\text{max}} \leftarrow $ maximal index of $M'$ in the application of Lemma 6.4.15;
16.     $m_0 \leftarrow $ index of first 0 in $S$ after $m'_{\text{max}}$ (or NONE);
17.     if $m_0 = $ NONE then
18.         return NONE;
19.     end
20.     Use Lemma 6.4.16 to find an even 3-cadence starting in $L$
21.     with middle element $m_0$;
22.     if 3-cadence found then
23.         return 3-cadence;
24.     end
25.     $r'_{\text{max}} \leftarrow $ maximal index of $R'$ in the application of Lemma 6.4.16;
26.     $r_{\text{min}} \leftarrow r'_{\text{max}} + 1$;
27. end
28. return NONE;
6.5 Longer (Sub-)Cadences

In this chapter, we have shown several efficient algorithms for detecting, counting and reporting subcadences and their variants with three characters. Funakoshi et al. prove in [40] that for fixed $k$ we can count both $k$-subcadence and $k$-cadence in subquadratic time in the word RAM model.

In terms of bit complexity, it is still unknown whether we can detect, for example, 4-subcadences with a given character or 4-cadences with an arbitrary character with subquadratic operations. One might hope that the partial cadences which were analyzed in [41] by Funakoshi and the author of this thesis provide useful heuristics. However, the time complexity of the corresponding detection algorithms is not yet known and is beyond the scope of this thesis.
Chapter 7

Maximal Pairs

The last periodic pattern we consider in this thesis is the extended maximal pair.

In this chapter, we will present upper bounds for both non-highly-periodic extended maximal pairs and highly-periodic extended maximal pairs. These upper bounds also translate into upper bounds for the number of arcs of the CDAWG and the number of runs in the RLBWT. We will also show that the upper bound for non-$\frac{1}{2q}$-highly-periodic maximal pairs is tight up to a constant factor even if the underlying alphabet contains only two characters.

Similar results with upper bounds dependent on the number of LZ77-factors instead of the, generally smaller, size of the smallest string attractor are already published by the author of this thesis in [77] at the 30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019) and in [78] at the 31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020).

This chapter is also the continuation of the master’s thesis [76] of the author of this thesis. In particular, the upper bounds for the number of distinct non-highly-periodic extended maximal pairs given by Theorem 7.2.6 and its application for the CDAWG given by Theorem 7.2.14 are variants of Theorem 3.2.6 and Theorem 3.2.7 of [76], respectively.

In terms of highly-periodic extended maximal pairs the master’s thesis and this chapter follow different approaches. In this chapter, we will consider non-extendable maximal pairs which have an occurrence which cannot be periodically extended by more than one minimum period. In the master’s thesis the notion of “uninduced exponents” were introduced. These uninduced exponents were defined by the numbers of occurrences of the powers of a given substring.

A pair is a pair of equal substrings $S[n_k..n_k+l-1] = S[m_k..m_k+l-1]$ which is defined by the triple given by the two starting indices and the common
length. If both the preceding characters are distinct and the succeeding characters are distinct, the pair is maximal and the common string is a maximal repeat. Note that maximal pairs are already given by the pair of their starting indices. An extended maximal pair is a maximal pair including the preceding characters and their succeeding characters.

Maximal pairs and maximal repeats are a compact representation of all repeating substrings in a string. They therefore have a wide variety of use cases, for example in string compression [42] and genetics [3]. In the following sections, we will also see that they are tightly linked to some data structures like the CDAWG and other string compressions like the RLBWT.

Maximal pairs whose corresponding maximal repeats are high powers, naturally give rise to smaller maximal pairs. For example, in the string

$$ba^{10}ba^{20}bS = baaaaaabaabaaaaabaaaaabaaaaabaaaaab$$,

we have the maximal pairs with corresponding maximal repeats $$a^i$$ for the exponent $$1 \leq i \leq 19$$. However, each of these maximal pairs can be derived from the maximal pairs with corresponding maximal repeat $$a^9$$ and $$a^{19}$$. Thus, we are most interested in the maximal pairs whose corresponding maximal repeats have a similar size than one of their periodic extensions.

An occurrence of a maximal repeat is extendable if its minimum period $$p$$ extends to at least $$p + 1$$ additional surrounding characters. A maximal pair is extendable if both occurrences of the corresponding maximal repeat are extendable. Similarly, a maximal repeat is $$\frac{1}{\varepsilon q}$$-highly-periodic if it is at least $$q$$ times as long as its minimum period and a maximal pair is $$\frac{1}{\varepsilon q}$$-highly-periodic if its corresponding maximal repeat is $$\frac{1}{\varepsilon q}$$-highly-periodic.

We will see that these non-extendable maximal pairs can naturally be found in the RLBWT.

7.1 Related Work

7.1.1 Combinatorial Properties and CDAWGs

Since maximal pairs are given by the pairs of the starting indices of the corresponding occurrences of the maximal repeat, there are at most $$|S|^2$$ different maximal pairs in a string $$S$$. This implies that there are also at most $$|S|^2$$ different maximal repeats in a string $$S$$. The same upper bound can be found by the fact that each maximal repeat is a substring of $$S$$.

For the maximal pairs, we cannot expect a subquadratic upper bound for the number of maximal pairs if the alphabet is not constant. For example,
in the string $a_1a_2a_3a\ldots a_{n-1}a_n$, each pair of as is maximal. Note that all extended maximal pairs are not only different but even distinct.

Even the number of different maximal repeats can be quadratic. For example, in the string $a^n$, each proper substring $a^i$ with its $n+1-i$ occurrences is a maximal repeat. We therefore consider distinct maximal repeats.

Maximal repeats and their right extensions are closely related to the CDAWG which was introduced by Blumer et al. in [13]. The CDAWG is the compactification of the minimal deterministic automaton that detects all suffixes of the underlying string. Crochemore and Vérin proved in [28] that this construction is equivalent to the minimization of the suffix tree.

For the maximal repeats, Raffinot shows in [85] that the labels of the longest paths to the internal nodes of the CDAWG form a natural bijection from the internal nodes in a CDAWG to the set of maximal repeats. Since Blumer et al. prove in [13] that for a string $S$, the even the number of arcs of the CDAWG is bounded by $2|S|$, this shows that $S$ contains at most $2|S|$ distinct maximal repeats.

The result of Raffinot was improved by Belazzougui et al. in [10] which introduces the number of right extensions of maximal repeats as a measure for the repetitiveness of strings and shows that the set of right extensions of the maximal repeats plus the set of distinct characters in the string naturally corresponds to the arcs of the CDAWG. Therefore, even the number of distinct right extensions of maximal repeats is bounded by $2|S|$.

As already pointed out by Blumer in [13], this upper bound of $2|S|$ distinct right extensions of maximal repeats is achieved by the string $a^n$. Thus, even for highly compressible strings, this bound cannot be improved. This fact may also explain the apparent lack of research regarding the number of nodes in CDAWGs of general compressible strings.

However, some nontrivial bounds for the number of nodes in CDAWGs for special strings do exist. For example, Radoszewski and Rytter prove in [84] that the number of arcs of the CDAWG of Thue-Morse words is logarithmic in the size of the word. A similar result has been shown by Epifanio et al. in [35] for Sturmian words.

Regarding the lower bound, Belazzougui et al. prove in [10] that the number of edges in the CDAWG is bounded from below by the minimal number of self-referential LZ77-factors. This implies, that a string with $z$ LZ77-factors over the alphabet $\Sigma$ has at least $\frac{z}{|\Sigma|} - 1$ maximal repeats.

### 7.1.2 RLBWT

The Burrows-Wheeler-transform is a permutation of the characters of a string which aims to transform repeating substrings into runs of repeating characters.
These runs can be compressed by pairs of their corresponding characters and their lengths. This compression is \textit{run-length encoding}. The \textit{RLBWT} is the run-length encoding of the string after the Burrows-Wheeler-transform.

The relation between repeating substrings and the number of runs of the \textit{RLBWT} was formalized by Belazzougui et al. in [10]. This paper shows that the number of runs of the \textit{RLBWT} is bounded from above by the number of right extensions of maximal repeats plus the size of the used alphabet.

Since high powers generally induce many maximal repeats but are well-compressible by the \textit{RLBWT}, this lower bound is not a good lower bound for highly-periodic strings. For example, the string $a^n$ has $n - 1$ maximal repeats but only 1 run in the \textit{RLBWT}. More surprisingly, there are even strings without fourth powers with only 2 runs in the \textit{RLBWT} but arbitrarily many maximal repeats. For example, Christodoulakis et al. show in [21] that the Burrows-Wheeler transform of the $n$-th Fibonacci string $F_n$ is given by $b|F_n - 2|a|F_n - 1|$. However, each string $F_n$ has at least the maximal repeats $F_i$ with $i \leq i - 2$.

It can be shown that the number of runs of the \textit{RLBWT} is roughly as large as other strong string compression schemes and complexity measures.

One of these complexity measures is the substring complexity given by $\delta = \max_{m=1}^{\lvert S \rvert} \frac{1}{m} \lvert S_m \rvert$ where $\lvert S_m \rvert$ is the number of substrings of $S$ with length $m$. The substring complexity was recently introduced by Christiansen et al. in [20] and is derived from the complexity measure $\frac{1}{m} \lvert S_m \rvert$ used by Raskhodnikova in [88]. Christiansen et al. prove in [20] that for each string attractor $\Gamma$ of $S$, the inequality $\delta \leq \lvert \Gamma \rvert$ holds.

Kempa and Kociumaka prove that the number of runs of the \textit{RLBWT} of a string $S$ is in $O\left((\delta \log \delta) \max \left(1, \frac{n}{\delta \log \delta}\right)\right)$.

Conversely, in [43], Gagie et al. prove that the number of LZ77-factors of a string $S$ is in $\Omega\left(r \log(\frac{n}{r})\right)$ where $r$ is the number of runs of the \textit{RLBWT} of $S$.

\subsection{Algorithms}

As stated in the Subsection 7.1.1, Raffinot has shown in [85] that the maximal repeats naturally correspond to the nodes of the CDAWG of the underlying string.

There are several ways to build the CDAWG in linear time. For example Crochemore and Vérin show in [28] that McCreight’s algorithm for building suffix trees can be adapted to CDAWGs. By efficiently traversing all nodes of the CDAWG, we can determine for each node, how long the longest path label to this node is and we can find the left-most occurrence of the corresponding...
Therefore, we can find all distinct maximal repeats in linear time.

Similarly, we can obtain all right extensions from the children of a given node and all left extensions from the Weiner links. Thus, we can find all distinct maximal pairs in linear time with respect to the combined size of the string and the number of maximal pairs. We can also count the maximal pairs in linear time.

A special case of maximal pairs are maximal $\alpha$-gapped repeats for $\alpha \geq 1$. A maximal pair with two non-overlapping occurrences of the corresponding maximal repeat $P$ naturally forms a substring $PTP$. This substring is a maximal $\alpha$-gapped repeat if $|PT| \leq \alpha|P|$ holds. Crochemore et al. prove in [27] that all maximal $\alpha$-gapped repeats in a string $S$ can be found in $O(\alpha|S|)$ time.

In compressed strings, the upper bounds which we will prove in the next section show that we cannot expect a polynomial time algorithm for reporting all maximal repeats. However, the author of this thesis proves in his master’s thesis [76] that there is an algorithm that creates the CDAWG, and thereby finds all maximal repeats, which is polynomial in the combined size of the input and the output. In particular, the thesis proves that for a string $S$ over $\Sigma$ with grammar of size $g$ and without $q$-th powers, the CDAWG can be created in $O(qg^6(p + g)|\Sigma|)$ time.

### 7.2 Compressed Upper Bounds

In this section, we will present and prove upper bounds for the number of both non-highly-periodic maximal pairs and non-extendable highly-periodic maximal pairs.

In particular, we will prove the following two theorems.

**Theorem 7.2.1.** Let $S$ be a string with string attractor $\Gamma$. Let further $-1 \leq i, j \leq |S|$ be two indices and $q \geq 3$ be a natural number.

Then, there are at most

$$12 \left( 1 + 48 \frac{q}{\log_2 q} \right) \log_2(|S|)$$

different non-$\frac{1}{2^q}$-highly-periodic maximal pairs containing the index pair $(i, j)$.

Also, there are at most

$$((|\Gamma| + 2)^2 \cdot 12 \left( 1 + 48 \frac{q}{\log_2 q} \right) \log_2(|S|))$$

distinct non-$\frac{1}{2^q}$-highly-periodic maximal pairs.
**Theorem 7.2.2.** Let $S$ be a string with string attractor $\Gamma$. Let further $-1 \leq i, j \leq |S|$ be two indices.

Then, there are at most $24 \log_2(|S|)$ different pairs of extended maximal 2-repetitions that contain the indices $i$ and $j$, respectively, in which both maximal 2-repetitions have the same minimum period. Also, there are at most $120 \log_2(|S|)$ different non-extendable $\frac{1}{2^4}$-highly-periodic maximal pairs that contain the index pair $(i, j)$.

Furthermore, there are at most

$$(|\Gamma| + 2)^2 \cdot 120 \log_2(|S|)$$

distinct non-extendable $\frac{1}{2^4}$-highly-periodic maximal pairs.

These theorems are variants of the bounds presented in [77] and [78] which depended on the size of the LZ77-factorization instead of the slightly smaller string attractor.

Similarly to the maximal $\delta$-(sub-)repetitions in Chapter 5, we will not only consider the maximal pairs but their extensions. Also, for a given string attractor $\Gamma$ of the string $S$, we use that each extended maximal pair has a copy in which both extended maximal repeats contain a character in $\Gamma \cup \{-1, |S|\}$.

This allows us to find local upper bounds for the number of different extended maximal pairs. These upper bounds can then be translated to global upper bounds for the number of distinct maximal pairs. In order to do this, we will need the following lemma which is an adaption of Lemma 5.2.3 to pairs of substrings.

**Lemma 7.2.3.** Let $S$ be a string such that for each pair of indices $(i, j)$ with $-1 \leq i, j \leq |S|$, the number of different extended maximal pairs containing the index pair $(i, j)$ is at most $c$. Let $\Gamma$ be a string attractor of $S$.

Then, the number of distinct maximal pairs in $S$ is at most $c(|\Gamma| + 2)^2$.

**Proof.** By definition of the string attractor $\Gamma$, for each maximal pair, each of the two extended maximal repeats has an occurrence that contains an index in the string attractor, the character $S[-1] = $ or the character $S[|S|] = $.

Therefore, the upper bound for the number of different extended maximal pairs containing any of the possible index pairs in the Cartesian product $(\Gamma \cup \{-1, |S|\}) \times (\Gamma \cup \{-1, |S|\})$ naturally translates into an upper bound for the number of distinct maximal pairs.

Recall that a maximal pair is called $\frac{1}{2^q}$-highly-periodic, if the minimum period of its underlying maximal repeat is at most $\frac{1}{q}$-th of its length. Otherwise, the maximal pair is non-$\frac{1}{2^q}$-highly-periodic. We will consider non-$\frac{1}{2^q}$-highly-periodic maximal pairs and $\frac{1}{2^4}$-highly-periodic separately.
7.2. COMPRESSED UPPER BOUNDS

7.2.1 Non-Highly-Periodic Maximal Pairs

In this subsection, we will prove that for each integer \( q \geq 3 \), there are at most \( (|\Gamma| + 2)^2 \cdot 12 \left( 1 + 48 \frac{q}{\log_2 q} \right) \log_2(|S|) \) distinct non-\( \frac{1}{2q} \)-highly-periodic maximal pairs in a string \( S \) with string attractor \( \Gamma \).

In particular, we will show that the number of distinct non-\( \frac{1}{2q} \)-highly-periodic maximal pairs in a string \( S \) with string attractor \( \Gamma \) is bounded from above by \( (|\Gamma| + 2)^2 \cdot 1103 \log_2(|S|) \).

In order to prove this upper bound, we will split the corresponding maximal repeats into two parts. Note that for \( q \geq 3 \), the maximal repeats are non-\( \frac{1}{2q} \)-highly-periodic if and only if one of the following three cases hold:

- The longer of the two substrings is not \( \frac{1}{2q} \)-highly-periodic,
- the longer of the two substrings is \( \frac{1}{2q} \)-highly-periodic with a period that does not extend to the whole substring or
- the longer of the two substrings is \( \frac{1}{2q} \)-highly-periodic with a period that extends to the whole maximal repeat but is larger than \( \frac{1}{q} \)-th of the length of the maximal repeat.

For the first case, the proof of Theorem 8 in [77] by the author of this thesis provide an upper bound. This upper bound and the upper bound for the second case are also published by the author of this thesis in [78].

The third case was not considered in [78]. However, we can derive an upper bound by combining the upper bound for the number of maximal \( \delta \)-repetitions from Chapter 5 with another application of the pigeonhole principle.

Firstly, we consider maximal pairs in which the longer part of the corresponding maximal repeat is not \( \frac{1}{2q} \)-highly-periodic.

Therefore, we will show that intersecting maximal pairs imply periodic substrings. The following lemma shows that intersecting maximal pairs have different distances between the two occurrences of their maximal repeats.

**Lemma 7.2.4.** Let \( S \) be a string and let \((n_a, m_a)\) and \((n_b, m_b)\) be different maximal pairs in \( S \) with lengths \( l_a \) and \( l_b \), respectively. Let further \( c \) be an index such that \( c \in [n_a - 1, n_a + l_a] \) and \( c \in [n_b - 1, n_b + l_b] \) hold.

Then, the distances \( d_a = m_a - n_a \) and \( d_b = m_b - n_b \) are unequal.

**Proof.** Since the two maximal pairs are different, the pairs \((n_a, m_a)\) and \((n_b, m_b)\) of starting indices are different. Therefore, if \( n_a = n_b \) holds, then the inequality \( m_a \neq m_b \) holds. This, however, already implies that the distances \( d_a \) and \( d_b \) are unequal.
Therefore, we can assume that the starting indices \( n_a \) and \( n_b \) are unequal. Without loss of generality, we also assume that \( n_a > n_b \) holds.

We consider the character \( S[m_a - 1] \). By definition of the maximal pair \((n_a, m_a)\), we get \( S[m_a - 1] \neq S[n_a - 1] \).

By definition of \( c \), we obtain the inequality \( n_b \leq n_a - 1 \leq c \leq n_b + l_b \). Therefore, the index \( n_a - 1 \) is contained in the maximal repeat given by \((n_b, m_b)\) that starts with \( n_b \). By definition of the maximal pair \((n_a, m_a)\), this implies that \( S[n_a - 1] = S[m_b + (n_a - 1 - n_b)] \) holds.

By definition of the distances, we get \( m_b + (n_a - 1 - n_b) = m_a - 1 + (d_b - d_a) \). Hence we obtain

\[
S[m_a - 1] \neq S[n_a - 1] = S[m_b + (n_a - 1 - n_b)] = S[m_a - 1 + (d_b - d_a)].
\]

This, however, implies that \( d_a \) and \( d_b \) are unequal and thereby concludes the proof. \( \square \)

The next lemma shows that this difference between the distances of intersecting maximal pairs can be translated into a periodicity.

**Lemma 7.2.5.** Let \( S \) be a string and let \((n_a, m_a)\) and \((n_b, m_b)\) be different maximal pairs in \( S \) with lengths \( l_a \) and \( l_b \) and distances \( d_a = m_a - n_a \) and \( d_b = m_b - n_b \), respectively. Define the difference of the distances \( \Delta_d = |d_a - d_b| \).

Then, the intersection \( S[\max(n_a, n_b) \ldots \min(n_a + l_a, n_b + l_b) - 1]\) of the two maximal repeats \( S[n_a \ldots n_a + l_a - 1] \) and \( S[n_b \ldots n_b + l_b - 1]\) is \( \Delta_d \)-periodic.

**Proof.** We define \( n_{\min} = \min(n_a, n_b) \), \( n_{\max} = \max(n_a, n_b) \), \( d_{\min} = \min(d_a, d_b) \) and \( d_{\max} = \max(d_a, d_b) \). Note that with the definition of \( n_{\max} \), the intersection of the two maximal repeats starts with the index \( n_{\max} \) and has the length \( \min(n_a - n_{\max} + l_a, n_b - n_{\max} + l_b) \).

Let \( x \) be a natural number such that

\[
0 \leq x < x + \Delta_d < \min(n_a - n_{\max} + l_a, n_b - n_{\max} + l_b)
\]

holds. Therefore, the indices \( n_{\max} + x \) and \( n_{\max} + x + \Delta_d \) lie in the two maximal repeats starting with \( n_a \) and \( n_b \), respectively. Thus, the equations

\[
S[n_{\max} + x] = S[n_{\max} + x + d_{\max}]
\]

and

\[
S[n_{\max} + x + \Delta_d] = S[n_{\max} + x + \Delta_d + d_{\min}]
\]

hold.
With $\Delta_d + d_{\min} = d_{\max}$ we then get
\[
S[n_{\max} + x] = S[n_{\max} + x + d_{\max}]
= S[n_{\max} + x + \Delta_d + d_{\min}]
= S[n_{\max} + x + \Delta_d]
\]

Since for all integers $x$ with
\[
0 \leq x < x + \Delta_d < \min(n_a - n_{\max} + l_a, n_b - n_{\max} + l_b)
\]
the equation $S[n_{\max} + x] = S[n_{\max} + x + \Delta_d]$ holds, the intersection of $S[n_a..n_a + l_a - 1]$ and $S[n_b..n_b + l_b - 1]$ is $\Delta_d$-periodic.

Now, we will prove a local upper bound for distinct maximal pairs that do not contain certain $q$-th powers.

**Theorem 7.2.6.** Let $S$ be a string and $-1 \leq i, j \leq |S|$ be two indices. Let further $q \geq 2$ be a natural number.

Then, there are at most $18q \cdot \lceil \log_q(|S|) \rceil$ distinct maximal pairs $(n_k, m_k)$ with lengths $l_k$ such that

- the extensions of $(n_k, m_k)$ contain the index pair $(i, j)$ and
- the longer of the two substrings $S[n_k..i - 1]$ and $S[i..n_k + l_k - 1]$ is not $\frac{1}{2q}$-highly-periodic.

**Proof.** By contradiction:

Assume that there are at least $18q \cdot \lceil \log_q(|S|) \rceil + 1$ different maximal pairs with the restrictions given by the prerequisites.

We will use the pigeonhole principle multiple times to find two pairs of maximal pairs with a long intersection and similar distances in order to create a contradiction.

Firstly, we divide the maximal pairs into the two (not necessarily disjoint) sets depending on whether $|S[n_k..i - 1]| \leq |S[i..n_k + l_k - 1]|$ holds or if $|S[n_k..i - 1]| \geq |S[i..n_k + l_k - 1]|$ holds.

Using the pigeonhole principle, at least one of the two sets contain at least half of the $18q \cdot \lceil \log_q(|S|) \rceil + 1$ maximal pairs. By symmetry, we assume without loss of generality that there are at least $9q \cdot \lceil \log_q(|S|) \rceil + 1$ maximal pairs $(n_k, m_k)$ such that $S[n_k..i - 1]$ is at least as long as $S[i..n_k + l_k - 1]$.

Since the length of a maximal pair is at least 1 and less than $|S|$, the logarithm of the length is bounded by $0 = \log_q(1) \leq \log_q(l_k) < \log_q(|S|)$.

We can now sort the maximal pairs with regard to the values $\log_q(l_k)$ into the $[\log_q(|S|)]$ intervals

$$[h, h + 1)$$

with $h \in \{0, 1, 2, \ldots, [\log_q(|S|)] - 1\}$.
CHAPTER 7. MAXIMAL PAIRS

The pigeonhole principle shows that of the \(9q \cdot \lceil \log_q(|S|) \rceil + 1\) remaining maximal pairs, there is one of the \(\lceil \log_q(|S|) \rceil\) sets which contains at least

\[
\left\lceil \frac{9q \cdot \lceil \log_q(|S|) \rceil + 1}{\lceil \log_q(|S|) \rceil} \right\rceil = 9q + 1
\]

maximal pairs.

Therefore, there is a natural number \(L'\) such that for at least \(9q + 1\) maximal pairs the inequality \(L' \leq \log_q(l_k) < 1 + L'\) holds. Exponentiating this inequality yields that there is a natural number \(L = q^{L'}\) such that \(L \leq l_k < qL\) holds for these \(9q + 1\) maximal pairs. Since both \(q\) and \(L\) are natural numbers, this implies \(l_k \leq qL - 1\).

By prerequisite, for each of these maximal pairs \((n_k, m_k)\), we have the indices \(i \in [n_k - 1..n_k + l_k]\) and \(j \in [m_k - 1..m_k + l_k]\). Also, by the first assumption, the substring \(S[n_k..i - 1]\) is at least as long as \(S[i..n_k + l_k - 1]\). Therefore, the inequality \(n_k + \frac{l_k}{2} \leq i \leq m_k \leq j \leq m_k + l_k\). Hence, we can bound the distances \(d_k = m_k - n_k\) by

\[
j - i - \frac{l_k}{2} \leq (m_k + l_k) - \left(n_k + \frac{l_k}{2}\right) - \frac{l_k}{2} = d_k
\]

and

\[
d_k = (m_k - 1) - (n_k + l_k) + l_k + 1 \leq j - i + l_k + 1.
\]

We can now bisect the interval \([L, qL - 1]\) and find a real number \(\theta\) such that

- for at least \(3q + 1\) of these \(9q + 1\) maximal pairs \(L \leq l_k \leq \theta L\) holds and
- for at least \(6q + 1\) of these \(9q + 1\) maximal pairs \(\theta L \leq l_k \leq qL - 1\) holds.

Note that at least one maximal pair is in both set.

Of the \(6q + 1\) maximal pairs \((n_k, m_k)\) with \(\theta L \leq l_k < qL\), each \(d_k\) is in one of the \(6q\) intervals

\[
\left[j - i - \frac{qL}{2} + h \cdot \frac{1}{4}L, j - i - \frac{qL}{2} + (h + 1) \cdot \frac{1}{4}L\right] \text{ with } 0 \leq h < 6q
\]

of length \(\frac{1}{4}L\). Therefore, the pigeonhole principle yields that one of these subsets contains at least 2 maximal pairs. Hence, we obtain two maximal pairs \((n_a, m_a)\) and \((n_b, m_b)\) with distances \(d_a\) and \(d_b\), respectively, such that \(\Delta_a := |d_a - d_b| \leq \frac{1}{4}L\) holds. Without loss of generality, we also have \(n_a \geq n_b\).
Lemma 7.2.4 yields $0 < \Delta_a$. Since the $6q + 1$ maximal pairs $(n_k, m_k)$ with $\theta L \leq l_k < qL$ satisfy $n_k + \frac{\theta L}{2} \leq n_k + \frac{l_k}{2} \leq i$ and $i \leq n_k + l_k$, Lemma 7.2.5 yields that for the maximal pair $(n_a, m_a)$, the substring $S[n_a..i-1]$ has the period $0 < \Delta_a \leq \frac{1}{4}L$. Note that the inequality $n_a \leq i - \frac{q}{2}L$ holds.

Similarly, we can sort the $3q + 1$ maximal pairs $(n_k, m_k)$ with $L \leq l_k \leq \theta L$ into the $3q$ sets with length $\frac{\theta}{2q}L$ given by

$$\left[j - i - \frac{\theta L}{2} + h \cdot \frac{\theta}{2q}L, j - i - \frac{\theta L}{2} + (h + 1) \frac{\theta}{2q}L\right]$$

with $0 \leq h < q$

and

$$\left[j - i + 1 + h \cdot \frac{\theta}{2q}L, j - i + 1 + (h + 1) \frac{\theta}{2q}L\right]$$

with $0 \leq h < 2q$.

Then, the pigeonhole principle shows that of these maximal pairs, there is a maximal pair $(n_c, m_c)$ such that $n_c \leq i - \frac{1}{2}L$ holds and such that $S[n_c..i-1]$ has a period of length $0 < \Delta_c \leq \frac{\theta}{2q}L$.

By prerequisite, the string $S[n_a..i-1]$ is not a fractional power with exponent of at least $q$. Thus, the inequality $\frac{i - n_a}{\Delta_a} < q$ holds. With both $\frac{\theta L}{2} \leq \frac{l_k}{2} \leq i - n_a$ and $\Delta_a \leq \frac{1}{4}L$, we obtain $\theta < \frac{q}{2}$ and hence $\Delta_c < \frac{1}{4}L$.

By construction, the string $S[\max(n_a, n_c)..i-1]$ has length of at least $\frac{1}{2}L$ and is both $\Delta_a$-periodic as well as $\Delta_c$-periodic. Since $\Delta_a + \Delta_c < \frac{1}{2}L$ holds, the periodicity lemma is applicable and shows that $S[\max(n_a, n_c)..i-1]$ is $\gcd(\Delta_a, \Delta_c)$-periodic. Since the string $S[\max(n_a, n_c)..i-1]$ is longer than its period $\Delta_a$, the subperiod $\gcd(\Delta_a, \Delta_c)$ extends to the whole $\Delta_a$-periodic superstring $S[n_a..i-1]$.

However, the inequality $\gcd(\Delta_a, \Delta_c) \leq \Delta_c \leq \frac{\theta}{2q}L$ shows that the period of $S[n_a..i-1]$ is at most $\frac{\theta}{2}L$. Also, the string $S[n_a..i-1]$ has a length of at least $\frac{\theta}{2}L$. Therefore, the string $S[n_a..i-1]$ is a fractional power with exponent of at least $q$.

This however contradicts the assumption and thereby proves the theorem.

Secondly, using the notation of the previous theorem, we will now consider maximal pairs $(n, m)$, in which the longer of the substrings $S[n..i-1]$ and $S[i..n+l-1]$ is at least $\frac{1}{2}$-highly-periodic with a period that does not extend to the whole maximal repeat.

In order to prove the upper bound for the number of these maximal pairs, we will show that $\frac{1}{2}$-highly-periodic suffixes of similar size have the same minimum period.
Lemma 7.2.7. Let \( S \) be a string with \( \frac{1}{3} \)-highly-periodic prefixes \( P_1 \) and \( P_2 \) with \( |P_1| \leq |P_2| \leq 2|P_1| \).

Then, the prefixes \( P_1 \) and \( P_2 \) have the same minimum period.

Proof. Let \( p_1 \) and \( p_2 \) be the minimum periods of \( P_1 \) and \( P_2 \), respectively. Note that \( P_1 \) is both \( p_1 \)-periodic and \( p_2 \)-periodic.

Since both \( P_1 \) and \( P_2 \) are \( \frac{1}{3} \)-highly-periodic, we obtain the inequalities \( p_1 \leq \frac{1}{3}|P_1| \) and \( p_2 \leq \frac{1}{3}|P_2| \leq \frac{2}{3}|P_1| \). Therefore, the inequality \( p_1 + p_2 \leq |P_1| \) holds and the periodicity lemma proves that \( P_1 \) is gcd\((p_1, p_2)\)-periodic.

By minimality of \( p_1 \), we obtain \( p_1 = \text{gcd}(p_1, p_2) \). Thus, the period \( p_2 \) is a multiple of \( p_1 \). Since the string \( P_1 \subset P_2 \) and \( P_1 \) is longer than its period \( p_2 \), the subperiod \( p_1 \) extends to the whole \( p_2 \)-periodic superstring \( P_2 \).

Therefore \( p_1 = p_2 \) holds. \( \square \)

Next, we will prove a local upper bound for distinct maximal pairs whose longer part is at least a \( q \)-th power whose periodicity does not extend to the whole maximal repeat.

Theorem 7.2.8. Let \( S \) be a string and \(-1 \leq i,j \leq |S| \) be two indices.

Then, there are at most \( 12 \log_2(|S|) \) different maximal pairs \((n_k, m_k)\) with lengths \( l_k \) such that

- the extensions of \((n_k, m_k)\) contain the index pair \((i, j)\) and
- the longer of the substrings \( S[n_k..i-1] \) and \( S[i..n_k+l_k-1] \) with minimum period \( p_k \) are \( \frac{1}{3} \)-highly-periodic, but
- the substrings \( S[n_k..n_k+l_k-1] \) are not \( p_k \)-periodic.

Proof. By contradiction:

Assume that there are at least \( (12 \log_2(|S|)) + 1 \) different maximal pairs with the restrictions given by the prerequisites.

We will use the pigeonhole principle multiple times to find two maximal pairs with similar starting indices and a long intersection in order to create a contradiction.

Firstly, we divide the maximal pairs into the two (not necessarily disjoint) sets depending on whether \( |S[n_k..i-1]| \leq |S[i..n_k+l_k-1]| \) holds or if \( |S[n_k..i-1]| > |S[i..n_k+l_k-1]| \) holds.

Using the pigeonhole principle, at least one of the two sets contain at least half of the \( (12 \log_2(|S|)) + 1 \) maximal pairs. By symmetry, we assume without loss of generality that there are at least \( 6 \log_2(|S|) + 1 \) maximal pairs \((n_k, m_k)\) such that \( S[n_k..i-1] \) is at least as long as \( S[i..n_k+l_k-1] \).
Since \( S[n_k..i - 1] \) is \( \frac{1}{23} \)-highly-periodic, this substring has to contain at least three characters. On the other hand, the indices \( n_k \) and \( i \) are bounded by \( n_k \geq 0 \) and \( i \leq |S| \). Therefore, the inequality \( 3 \leq i - n_k \leq |S| \) holds.

Taking the logarithm yields
\[
1 < \log_2(3) \leq \log_2(i - n_k) \leq \log_2(|S|) \leq \lfloor \log_2(|S|) \rfloor.
\]

We can now sort the maximal pairs with regard to the values \( \log_2(i - n_k) \) into the \( \lfloor \log_2(|S|) \rfloor - 1 \) intervals
\[
[h, h + 1] \text{ with } h \in \{1, 2, \ldots, \lfloor \log_2(|S|) \rfloor - 1\}.
\]

The pigeonhole principle shows that of the \( (6 \log_2 |S|) + 1 \) remaining maximal pairs, there is one of the \( \lfloor \log_2(|S|) \rfloor - 1 < \log_2(|S|) \) sets which contains at least
\[
\left\lfloor \frac{(6 \log_2(|S|)) + 1}{\log_2(|S|)} \right\rfloor = 7
\]
maximal pairs.

By construction, there is an integer \( L' \) such that these maximal pairs satisfy \( L' \leq \log_2(i - n_k) \leq 1 + L' \). Therefore, for \( L = 2^{L'} \) this gives a natural number \( L \) such that \( L \leq i - n_k \leq 2L \) holds for each of these \( 7 \) maximal pairs. Note that Lemma 7.2.7 proves that the underlying substrings \( S[n_k..i - 1] \) of these seven maximal pairs have the same minimum period.

Since the inequality \( |S[n_k..i-1]| \geq |S[i..n_k+l_k-1]| \) holds for the remaining maximal pairs, we obtain \( n_k + \frac{l_k}{2} \leq i \). Thus, the inequality \( \frac{l_k}{2} \leq i - n_k \leq 2L \) holds. In particular, the length \( l_k \) is at most \( 4L \).

Since \( j \) is contained in the extended maximal repeats starting with \( m_k \), the inequality \( m_k - 1 \leq j \leq m_k + l_k \leq m_k + 4L \) holds for each \( k \). This implies that the \( m_k \) of the remaining seven maximal pairs are contained in the interval \( j - 4L \leq m_k \leq j + 1 \) of length \( 4L + 1 \).

Of the 7 maximal pairs, each \( l_k \) is in one of the 6 intervals
\[
\left[ j - 4L + h \cdot \left( \frac{2}{3}L + \frac{1}{6} \right), j - 4L + (h + 1) \cdot \left( \frac{2}{3}L + \frac{1}{6} \right) \right] \text{ with } 0 \leq h < 6
\]
of length \( \frac{2}{3}L + \frac{1}{6} \). Therefore, the pigeonhole principle yields that one of these subsets contains at least 2 maximal pairs. Hence, we obtain two maximal pairs \( (n_a, m_a) \) and \( (n_b, m_b) \), respectively, such that \( |m_a - m_b| \leq \frac{2}{3}L + \frac{1}{6} \) holds. Since both \( |m_a - m_b| \) and \( L \) are natural numbers, this implies \( |m_a - m_b| \leq \frac{2}{3}L \).

We know that the underlying substrings \( S[n_a..i - 1] \) and \( S[n_b..i - 1] \) of these two maximal pairs have the same minimum period \( p_n \). Also their \( p_a \)-periodic extensions end with the same index. Hence the corresponding
maximal repeats of the two maximal pairs are of the form \( l_aP^3sr_a \) and \( l_bP^3sr_b \) where \( l_aP^3 \) and \( l_bP^3 \) are the \( p_a \)-periodic parts left of \( i \), the substrings \( s \) is the \( p_a \)-periodic extension of \( S[n_a..i−1] \) to the right and \( r_a \) and \( r_b \) are the remaining characters of the maximal repeats. Note that \( r_a \) and \( r_b \) are by prerequisite not the empty string.

Next we will show that the \( p_a \)-periodic strings \( l_aP^3s \) and \( l_bP^3s \) starting with \( m_a \) and \( m_b \), respectively, have an intersection of at least \( p_a \) characters. We use that the two strings \( l_aP^3s \) and \( l_bP^3s \) have a length of at least \( \max(3p_a,L) \) and the difference \( |m_a − m_b| \) is at most \( \frac{2}{3}L \).

If \( p_a ≥ \frac{L}{3} \) holds, the intersection contains at least \( 3p_a − \frac{2}{3}L ≥ p_a \) characters. Otherwise, \( p_a < \frac{L}{3} \) holds and the intersection contains at least \( L − \frac{2}{3}L = \frac{L}{3} > p_a \) characters.

Therefore, the union of the occurrences of \( l_aP^3s \) and \( l_bP^3s \) starting at \( m_a \) and \( m_b \), respectively, is a substring of the \( p_a \)-periodic extension of their intersection and thereby \( p_a \)-periodic as well. By construction of \( s \) and the non-emptiness of \( r_a \) and \( r_b \), this implies that both substrings end with the same index.

Since \((n_a,m_a)\) and \((n_b,m_b)\) are different, this implies that \( l_a \) and \( l_b \) have different lengths. Without loss of generality, the inequality \( |l_a| < |l_b| \) holds. However, by periodicity of the two occurrences of \( l_bP^3s \) starting at \( n_b \) and \( m_b \), the characters \( S[n_a−1] \) and \( S[m_a−1] \) are equal. Hence, the pair \((n_a,m_a)\) is not maximal which contradicts the assumption and proves the theorem. \( \square \)

Thirdly and lastly, we consider the maximal pairs \((n,m)\), in which the longer of the substrings \( S[n..i−1] \) and \( S[i..n + l−1] \) is \( \frac{1}{2q} \)-highly-periodic with a period that does extend to the whole maximal repeat but in which the maximal repeat is too short to be \( \frac{1}{2q} \)-highly-periodic.

**Theorem 7.2.9.** Let \( S \) be a string and \(-1 ≤ i, j ≤ |S| \) be two indices. Let \( q ≥ 3 \) be a natural number.

Then, there are at most \( 5q \left[ 3 + \frac{6}{q^2} \right] \cdot \left\lfloor \log_{1+\frac{6}{q^2}}(|S|) \right\rfloor \) different maximal pairs \((n_k,m_k)\) with lengths \( l_k \) such that

- the extensions of \((n_k,m_k)\) contain the index pair \((i,j)\),
- the longer of the substrings \( S[n_k..i−1] \) and \( S[i..n_k + l_k−1] \) with minimum period \( p_k \) are \( \frac{1}{2q} \)-highly-periodic and
- the substrings \( S[n_k..n_k + l_k−1] \) are \( p_k \)-periodic, but
- the substrings \( S[n_k..n_k + l_k−1] \) are not \( \frac{1}{2q} \)-highly-periodic.
Proof. By contradiction:

Assume that there are at least \(5q \left\lfloor 3 + \frac{6}{q-2} \right\rfloor \cdot \left\lfloor \log_{1+\frac{2}{q-2}}(|S|) \right\rfloor + 1\) different maximal pairs with the restrictions given by the prerequisites.

We will use the pigeonhole principle multiple times to find maximal pairs with equal minimum periods and a long intersection in order to create a contradiction.

Since all considered maximal pairs have corresponding maximal repeats that are at least \(q\)-th powers, their periodic extensions are \((q-2)\)-repetitions. Theorem 5.2.4 proves that there are at most \(\left\lfloor 3 + \frac{6}{q-2} \right\rfloor \cdot \left\lfloor \log_{1+\frac{2}{q-2}}(|S|) \right\rfloor\) different maximal \((q-2)\)-repetitions whose extensions contain the index \(i\).

The pigeonhole principle therefore shows that there are at least

\[
\left\lceil \frac{5q \left\lfloor 3 + \frac{6}{q-2} \right\rfloor \cdot \left\lfloor \log_{1+\frac{2}{q-2}}(|S|) \right\rfloor + 1}{3 + \frac{6}{q-2}} \cdot \left\lfloor \log_{1+\frac{2}{q-2}}(|S|) \right\rfloor \right\rceil = 5q + 1
\]

maximal pairs whose corresponding maximal repeats have the same periodic extension and thus the same minimum period \(p\).

Even though the maximal repeats corresponding to the remaining \(5q + 1\) maximal pairs are \(p\)-periodic, the maximal pairs are not \(\frac{1}{2q}\)-highly-periodic. Thus, the lengths of the corresponding maximal repeats are less than \(2qp\). Since both \(q\) and \(p\) are integers, their lengths are at most \(2qp - 1\). Therefore, we get \(n_k - 1 \leq i \leq n_k + 2qp - 1\) and \(m_k - 1 \leq m_k + 2qp - 1\).

Thus, the distances \(d_k = m_k - n_k\) are bound by

\[j - i - 2qp \leq (m_k + 2qp - 1) - (n_k - 1) - 2qp = d_k\]

and

\[d_k = (m_k - 1) - (n_k + 2qp - 1) + 2qp \leq j - i + 2qp.\]

Of the \(5q + 1\) maximal pairs, each \(d_k\) is in one of the \(5q\) intervals

\[
\left[ j - i - 2qp + \frac{4}{5}p, j - i - 2qp + (h + 1)\frac{4}{5}p \right] \text{ with } 0 \leq h < 5q
\]

of length \(\frac{4}{5}p\). Therefore, the pigeonhole principle proves that there is at least one of these subsets contains at least 2 maximal pairs.

With Lemma 7.2.4 and Lemma 7.2.5, we obtain that the intersection of the two corresponding maximal repeats starting in \(n_k\) has the period \(0 < \Delta \leq \frac{4}{5}p\). Furthermore, since the corresponding maximal repeats have at least the length \(qp\), their intersection contains at least \(qp - \frac{4}{5}p > p + \frac{4}{5}p\) characters.
Hence, the periodicity lemma is applicable and proves, that the intersection is $\gcd(\Delta, p)$-periodic.

Since the intersection is longer than its period $p$, the subperiod $\gcd(\Delta, p)$ extends to the whole $p$-periodic maximal repeat. This, however, implies that $p$ is not the minimum period of the maximal repeat. Therefore, this contradicts the assumption and thereby proves the theorem.

We will use a slightly weaker, simplified upper bound.

**Corollary 7.2.10.** Let $S$ be a string and $-1 \leq i, j \leq |S|$ be two indices. Let $q \geq 3$ be a natural number.

Then, there are at most $540 \log_q(|S|)$ different maximal pairs $(n_k, m_k)$ with lengths $l_k$ such that

- the extensions of $(n_k, m_k)$ contain the index pair $(i, j)$,
- the longer of the substrings $S[n_k..i-1]$ and $S[i..n_k+l_k-1]$ with minimum period $p_k$ are $\frac{1}{2q}$ highly-periodic and
- the substrings $S[n_k..n_k+l_k-1]$ are $p_k$-periodic, but
- the substrings $S[n_k..n_k+l_k-1]$ are not $\frac{1}{2q}$-highly-periodic.

**Proof.** This corollary can be derived from Theorem 7.2.9 by using $q \geq 3$. In particular, this inequality implies both $q - 2 \geq 1$ and $q - 2 \geq \frac{q}{3}$.

Thus, the inequality $\left\lfloor 3 + \frac{6}{q-2} \right\rfloor \geq 9$ holds.

With the Bernoulli inequality, we obtain

$$\left(1 + \frac{q - 2}{4}\right)^{12} \geq 1 + 12 \frac{q - 2}{4} \geq 1 + q > q$$

and thereby $\log_{1+\frac{q-2}{4}}(q) < 12$

By using $\log_{1+\frac{q-2}{4}}(|S|) = \log_{1+\frac{q-2}{4}}(q) \log_q(|S|)$, we obtain the inequality

$$\left\lfloor \log_{1+\frac{q-2}{4}}(|S|) \right\rfloor > 12 \log_q(|S|).$$

Combining these inequality, we derive the upper bound

$$540 \log_q(|S|) > 5q \left\lfloor 3 + \frac{6}{q-2} \right\rfloor \cdot \left\lfloor \log_{1+\frac{q-2}{4}}(|S|) \right\rfloor.$$ 

Now, we can obtain the local upper bound for the number of different non-$\frac{1}{2q}$-highly-periodic maximal pairs which was stated at the beginning of the section. Also, we use Lemma 7.2.3 to derive the global upper bound for the number of distinct non-$\frac{1}{2q}$-highly-periodic maximal pairs.
Theorem 7.2.1. Let $S$ be a string with string attractor $\Gamma$. Let further $-1 \leq i, j \leq |S|$ be two indices and $q \geq 3$ be a natural number.

Then, there are at most

$$12 \left( 1 + 48 \frac{q}{\log_2 q} \right) \log_2(|S|)$$

different non-$\frac{1}{\geq 2q}$-highly-periodic maximal pairs containing the index pair $(i, j)$. Also, there are at most

$$(|\Gamma| + 2)^2 \cdot 12 \left( 1 + 48 \frac{q}{\log_2 q} \right) \log_2(|S|)$$

distinct non-$\frac{1}{\geq 2q}$-highly-periodic maximal pairs.

Proof. If a maximal repeat $S[n_k..n_k + l_k]$ is not $\frac{1}{\geq 2q}$-highly-periodic, then either the longer of the parts $S[n_k..i - 1]$ and $S[i..n_k + l_k - 1]$ is

- not $\frac{1}{\geq 2q}$-highly-periodic

- $\frac{1}{\geq 2q}$-highly-periodic but the corresponding periodicity does not extend to the whole maximal repeat $S[n_k..n_k + l_k]$ or

- $\frac{1}{\geq 2q}$-highly-periodic and the corresponding periodicity does extend to the whole maximal repeat $S[n_k..n_k + l_k]$ but the maximal repeat is too short to be $\frac{1}{\geq 2q}$-highly-periodic.

Therefore, the number of different maximal pairs which fulfill the prerequisites can be bound by Theorems 7.2.6 and 7.2.8 and Corollary 7.2.10.

We also assume without loss of generality that the inequality $q \leq |S|$ holds. Hence, the value $\log_q(|S|)$ is at least 1 and $18q \left[ \log_q(|S|) \right] \leq q36 \log_q(|S|)$ also holds. Thus, there are at most

$$18q \cdot \left[ \log_q(|S|) \right] + 12 \log_2(|S|) + 540q \log_q(|S|)$$

$$\leq 576q \log_q(|S|) + 12 \log_2(|S|)$$

$$= 12 \left( 1 + 48 \frac{q}{\log_2 q} \right) \log_2(|S|)$$

of those maximal pairs.

Lemma 7.2.3 therefore shows that there are at most

$$(|\Gamma| + 2)^2 \cdot 12 \left( 1 + 48 \frac{q}{\log_2 q} \right) \log_2(|S|)$$

distinct extensions of non-$\frac{1}{\geq 2q}$-highly-periodic maximal repeats. \qed

For $q = 3$ this proves that the number of distinct non-$\frac{1}{\geq 26}$-highly-periodic maximal pairs is bounded from above by $(|\Gamma| + 2)^2 \cdot 1103 \log_2(|S|))$. 
7.2.2 Highly-Periodic Maximal Pairs

In this subsection, we prove that the number of distinct non-extendable \( \frac{1}{\geq 4} \)-highly-periodic maximal pairs in a string \( S \) with a string attractor \( \Gamma \) is at bounded from above by \((|\Gamma| + 2)^2 \cdot 120 \log_2(|S|))\).

In [78], the author of this thesis proved a similar result. In this thesis, however, we simplify the proofs by using the results of Chapter 5 which results in a slightly higher constant. Also, we state the result in terms of the string attractor instead of the slightly larger number of LZ77-factors.

Since the two occurrences of the maximal repeats of \( \frac{1}{\geq 4} \)-highly-periodic maximal pairs are at least 4th powers, they are both naturally contained in 2-repetitions. Furthermore, by construction, the two minimum periods of the 2-repetitions are equal.

We will therefore count the pairs of maximal 2-repetitions with equal minimum period. Afterwards, we will show that each of these pairs give rise to at most 5 non-extendable maximal pairs with the same minimum period.

Firstly, we use Theorem 5.2.4 to give a local upper bound for the number of 2-repetitions.

**Corollary 7.2.11.** Let \( S \) be a string and \(-1 \leq i \leq |S|\) an index.

Then, there are at most \(12 \log_2(|S|)\) different maximal 2-repetitions whose extensions contain the index \( i \).

**Proof.** Theorem 5.2.4 shows that there are at most
\[
3 + \frac{6}{2} \cdot \left\lfloor \log_{1+\frac{2}{4}}(|S|) \right\rfloor
\]
different maximal 2-repetitions whose extensions contain the index \( i \).

Since \((1 + \frac{2}{4})^2 > 2\) holds, we obtain the inequality
\[
\left\lfloor \log_{1+\frac{2}{4}}(|S|) \right\rfloor \leq \log_{1+\frac{2}{4}}(|S|) = \log_{1+\frac{2}{4}}(2) \log_2(|S|) < 2 \log_2(|S|).
\]

Therefore, there are at most \(12 \log_2(|S|)\) different maximal 2-repetitions whose extensions contain the index \( i \).

Next, we show that there are at most two 2-repetitions with a given minimum period whose extensions contain a given index. The proof is similar to the proof of Lemma 5.2.5 which shows the same statement for \( \delta \)-subrepetitions with minimum period 1.

**Lemma 7.2.12.** Let \( S \) be a string, \(-1 \leq i \leq |S|\) an index and \( p \geq 1 \) a natural number.

Then, there are at most 2 different maximal 2-repetitions with minimum period \( p \) whose extensions contain the index \( i \).
7.2. COMPRESSED UPPER BOUNDS

Proof. Maximal 2-repetitions are at least 4th powers. Thus, each $p$-periodic maximal 2-repetition whose extension contains the index $i$ either contains the substring $S[i - p .. i - 1]$ or $S[i + 1 .. i + p]$.

By definition, for a given substring of length $p$, the $p$-periodic extension of this substring is the only possible maximal $p$-periodic 2-repetition which contains this substring.

Hence, there are at most 2 different $p$-periodic maximal 2-repetitions whose extensions contain the index $i$. \hfill \Box

Now we will show that each non-extendable $\frac{1}{\sqrt[4]{4}}$-highly-periodic maximal pair naturally arises from a pair of maximal 2-repetitions.

Lemma 7.2.13. Each pair of maximal 2-repetitions with minimum period $p$ gives rise to at most 5 distinct non-extendable maximal pairs with minimum period $p$.

Also each non-extendable $\frac{1}{\sqrt[4]{4}}$-highly-periodic maximal pair can be obtained this way.

Proof. By periodicity and maximality, a non-extendable maximal pair with minimum period $p$ has to be both the prefix of at least one of the two underlying 2-repetitions and a suffix of at least one of the two underlying 2-repetitions.

There are multiple cases, how the maximal repeats can occur in the two maximal 2-repetition. These cases are depicted in Figure 7.1.

If the underlying maximal repeat is equal to the shorter 2-repetition, we consider three cases.

Case 1: The other occurrence of the maximal repeat is the prefix of the other 2-repetition: Since the length of the maximal repeat is given, there is at most 1 non-extendable maximal pair with minimum period $p$ in this case.

Case 2: The other occurrence of the maximal repeat is the suffix of the other 2-repetition: Similarly to Case 1, there is at most 1 non-extendable maximal pair with minimum period $p$ in this case.

Case 3: The other occurrence of the maximal repeat is neither the prefix nor the suffix of the other 2-repetition: Similarly to Case 1, the length of the maximal repeat is given. Unlike Case 1, there may be multiple possible occurrences of the maximal repeat in the other 2-repetition in this case. However, the extensions of the maximal repeat in the other 2-repetition are given by its periodicity. Therefore, all non-extendable maximal pairs with minimum period $p$ in this case are copies of each other.

Otherwise, the underlying maximal repeat is unequal to both 2-repetitions: In this case, one underlying maximal repeat is a proper prefix of one maximal
2-repetition and the other underlying maximal repeat is a proper suffix of the other maximal 2-repetition.

In both possible pairings, the length of the maximal repeat is fixed, up to a multiple of the minimum period. By non-extendability, this allows only one possible length for the maximal repeat. Hence, there are at most 2 non-extendable maximal pairs with minimum period $p$ in this case.

This proves that each pair of maximal 2-repetitions with minimum period $p$ gives rise to at most 5 distinct non-extendable maximal pairs with minimum period $p$.

Conversely, we can consider a non-extendable $\frac{1}{4}$-highly-periodic maximal pair with minimum period $p$. Since the maximal repeat is at least a 4th power, the $p$-periodic extensions of the maximal repeats are maximal 2-repetitions.

Therefore, each non-extendable $\frac{1}{4}$-highly-periodic maximal pair can be obtained as one of the at most 5 maximal pairs of a given pair of maximal 2-repetitions.

By multiplying these upper bounds, we obtain the local upper bound for the number of different non-extendable $\frac{1}{4}$-highly-periodic maximal pairs which was stated at the beginning of the section. Also, we use Lemma 7.2.3 to derive a global upper bound for the number of distinct non-extendable $\frac{1}{4}$-highly-periodic maximal pairs.

**Theorem 7.2.2.** Let $S$ be a string with string attractor $\Gamma$. Let further $-1 \leq i, j \leq |S|$ be two indices.

Then, there are at most $24 \log_2(|S|)$ different pairs of extended maximal 2-repetitions that contain the indices $i$ and $j$, respectively, in which both maximal 2-repetitions have the same minimum period. Also, there are at most $120 \log_2(|S|)$ different non-extendable $\frac{1}{4}$-highly-periodic maximal pairs that contain the index pair $(i, j)$.

Furthermore, there are at most

$$(|\Gamma| + 2)^2 \cdot 120 \log_2(|S|)$$

distinct non-extendable $\frac{1}{4}$-highly-periodic maximal pairs.
7.2. COMPRESSED UPPER BOUNDS

7.2.3 Derived Upper Bounds

In this subsection, we will show that the upper bounds of the last two sections also lead to upper bounds for the number of runs of the RLBWT and the size of the compacted directed acyclic word graph (CDAWG).

Firstly, we prove the upper bound for the number of arcs in the CDAWG.

**Theorem 7.2.14.** Let $S$ be a string with string attractor $\Gamma$ and let $q \geq 2$ be a natural number such that $S$ does not contain $q$-th powers.

Then, the CDAWG of $S$ has at most $(|\Gamma| + 2)^2 \left( 36q \cdot \lceil \log_q(|S|) \rceil + 1 \right)$ arcs.

**Proof.** Belazzougui et al. show in [10] that the number of arcs in the CDAWG is equal to the number of right extensions of maximal repeats plus the size of the used alphabet.

By definition, each maximal repeat has a corresponding maximal pair.

Consider a right extension $R$ of a maximal repeat and a maximal pair that correspond to the maximal repeat without extension. If neither of the two right extensions given by the maximal pair is equal to $R$, we have an additional maximal pair given by the maximal repeat $R$ and the maximal repeat of the maximal pair which has a distinct preceding character.

Therefore, even each right extension of a maximal repeat has a corresponding maximal pair.

Conversely, each maximal pair contains two occurrences of the corresponding maximal repeat. Therefore, each maximal pair defines 2 right extensions of the corresponding maximal repeat.

Lemma 7.2.3 shows that each maximal pair has a copy whose extension contain an index pair of $(\Gamma \cup \{-1, |S|\}) \times (\Gamma \cup \{-1, |S|\})$. Thus, since $S$ does not contain $q$-th powers, Theorem 7.2.6 implies that there are at most $(|\Gamma| + 2)^2 \left( 18q \cdot \lceil \log_q(|S|) \rceil \right)$ distinct maximal pairs in $S$.

Multiplying by 2 and accounting for the at most $|\Gamma| + 2$ distinct characters in $S$, this proves that the number of arcs in the CDAWG is bounded by

$$(|\Gamma| + 2)^2 \left( 36q \cdot \lceil \log_q(|S|) \rceil + 1 \right).$$

In the remainder of the subsection, we will prove that the number of runs of the RLBWT of a string $S$ with string attractor $\Gamma$ is bounded from above by $(|\Gamma| + 2)^2 (1223 \log_2(|S|) + 1)$.

Belazzougui et al. show in [10] that the number of runs in the Burrows-Wheeler transform is bounded from above by the number of right extensions of the maximal repeats plus the size of the used alphabet. Together with the upper bound of the number of these right extensions proven by the author of
this thesis in [77], this led to the first nontrivial upper bound for the number of runs in the RLBWT in highly-compressible strings without highly-periodic substrings.

We consider extended maximal pairs instead, since they are easier to handle than maximal repeats and their right extensions. Note that by considering extended maximal pairs, we only lose a factor of $\Sigma^3$ in the worst-case.

However, in Subsection 7.2.1, we have seen that in highly-periodic strings, the number of maximal pairs can be large. Therefore, we will show that only non-$\frac{1}{25}$-highly-periodic maximal pairs and non-extendable $\frac{1}{24}$-highly-periodic maximal pairs can correspond to a change of the runs in the RLBWT. In this way, the Burrows-Wheeler transform does not suffer from high powers as the CDAWG does.

The proof presented here follows the prior publication [78] by the author of this thesis. Independently of this result, Kempa and Kociumaka proved a slightly stronger upper bound in [58]. The two corresponding arXiv-articles are [57] by Kempa and Kociumaka whose first version has proven a slightly weaker upper bound compared to their final publication and about four months later [79] by the author of this thesis.

**Lemma 7.2.15.** Let $S$ be a string and let $i$ be an index at which a new run in the Burrows-Wheeler transform starts. Let $(\pi_{i-1}, \pi_i)$ be the corresponding pair of lexicographically neighbored cyclic rotations of $S$.

Then, either $S[\pi_{i-1} - 1] \neq S[\pi_i - 1]$ holds or $(\pi_{i-1}, \pi_i)$ defines a non-extendable maximal pair.

Conversely, this extended maximal pair does not correspond to another index pair $(\pi_{j-1}, \pi_j)$ given by the Burrows-Wheeler-transform.

**Proof.** If $S[\pi_{i-1}] \neq S[\pi_i]$ holds, there is nothing left to prove. Therefore, let $S[\pi_{i-1}] = S[\pi_i]$ hold.

By construction, the two predecessors $S[\pi_{i-1} + |S| \mod |S| + 1]$ and $S[\pi_i + |S| \mod |S| + 1]$ with regard to the cyclic rotation of $S$ are unequal. Therefore, the predecessors $S[\pi_{i-1} - 1]$ and $S[\pi_i - 1]$ are unequal as well.

Note that the Strings $(S)[\pi_{i-1}..|S|]$ and $(S)[\pi_i..|S|]$ have different lengths. Since both strings contain the character $\$$ exactly once and since this occurrence is the last character of the string, the two strings have a mismatch such that $S[\pi_{i-1} + l] \neq S[\pi_i + l]$ holds. Let $l$ be the first of these mismatches.

Therefore, $S[\pi_{i-1}..\pi_{i-1} + l - 1] = S[\pi_{i-1}..\pi_i + l - 1]$ holds. In particular, the pair $(\pi_{i-1}, \pi_i)$ is a maximal pair of length $l$. Let $p$ be the minimum period of the corresponding maximal repeat $S[\pi_{i-1}..\pi_{i-1} + l - 1]$.

We will prove the lemma by contradiction. Assume that the maximal pair $(\pi_{i-1}, \pi_i)$ is extendable. By definition of extendability, this implies that both
occurrences \( S[\pi_i..\pi_i + l - 1] \) and \( S[\pi_i..\pi_i + l - 1] \) of the corresponding maximal repeat can be periodically extended by at least \( p + 1 \) characters.

Under this assumption, we will construct a cyclic rotation that is lexicographically between the two lexicographically adjacent cyclic rotations \( S[\pi_i..\pi_i + l - 1] \) and \( S[\pi_i..\pi_i + l - 1] \).

Since the two occurrences \( S[\pi_i..\pi_i + l - 1] \) and \( S[\pi_i..\pi_i + l - 1] \) of the maximal repeat are preceded by different characters, at most one of the two occurrences can be \( p \)-periodically extended to the left. Similarly, at most one of the two occurrences can be \( p \)-periodically extended to the right.

Since we assume that both occurrences are \( p \)-periodically extendable, exactly one of the two occurrence has to be \( p \)-periodically extendable to the left and the other occurrence has to be \( p \)-periodically extendable to the right.

Inverting the order of the alphabet including the character $ just reverses the output string of the Burrows-Wheeler-transform. In particular, the maximal pair \((\pi_i, \pi_i-1)\) correspond to a change of runs in the Burrows-Wheeler-transform. Hence, we can assume without loss of generality that the occurrence \( S[\pi_i..\pi_i + l - 1] \) is \( p \)-periodically extendable to the left. Therefore, the occurrence \( S[\pi_i..\pi_i + l - 1] \) is \( p \)-periodically extendable to the right. This implies that the strings \( S[\pi_i..\pi_i - 1 + p + l - 1] \) and \( S[\pi_i..\pi_i + p + l] \) are \( p \)-periodic.

By construction and definition of the Burrows-Wheeler-transform, we obtain the order \( S[\pi_i..\pi_i + l] < S[\pi_i..\pi_i + l] \). Using the periodicity of the string \( S[\pi_i..\pi_i + p + l] \), we also get \( S[\pi_i..\pi_i + l] = S[\pi_i..\pi_i + l] \). Furthermore, since both maximal repeats are equal, we get \( S[\pi_i..\pi_i + l] = S[\pi_i..\pi_i + l] \). Hence, the inequality \( S[\pi_i..\pi_i + l] < S[\pi_i..\pi_i + l] \) holds.

Using the periodicity of \( S[\pi_i..\pi_i - 1 + p + l - 1] \), we obtain the equation

\[
S[\pi_i..\pi_i - 1 + p + l - 1] = S[\pi_i..\pi_i - 1 + p + l - 1].
\]

Thus, we get the lexicographic order \( S[\pi_i..\pi_i - 1 + l] < S[\pi_i..\pi_i - 1 + p + l] \) and thereby the order

\[
S[\pi_i..\pi_i - 1 + l - 1] = S[\pi_i..\pi_i - 1 + p + l - 1].
\]

On the other hand, we can also use the periodicity of \( S[\pi_i..\pi_i + l + p] \) to obtain \( S[\pi_i..\pi_i + l] < S[\pi_i..\pi_i + l] \). Since both maximal repeats are equal, we also get

\[
S[\pi_i..\pi_i - 1 + p + l - 1] = S[\pi_i..\pi_i + l - 1].
\]

Thus, we get the lexicographic order \( S[\pi_i..\pi_i - 1 + p + l] < S[\pi_i..\pi_i + l] \) and thereby the order

\[
S[\pi_i..\pi_i - 1 + l - 1] = S[\pi_i..\pi_i + p + l - 1].
\]
However, by prerequisite and definition of the Burrows-Wheeler-transform, there is no cyclic rotation between the cyclic rotations $S[\pi_{i-1}..|S|]S[0..\pi_{i-1}-1]$ and $S[\pi_{i}..|S|]S[0..\pi_{i}-1]$. Therefore, the lexicographic order of the cyclic rotation $S[\pi_{i-1} - p..|S|]S[0..\pi_{i-1} - p - 1]$ with regard to the other two cyclic rotations contradicts the assumption and thereby concludes this part of the proof.

Conversely, we have
\[
S[\pi_{i-1}..\pi_{i-1} + l] = (S[\pi_{i-1}..|S|]S[0..\pi_{i-1} - 1]) [0..l - 1] \\
< (S[\pi_{i}..|S|]S[0..\pi_{i} - 1]) [0..l - 1] \\
= S[\pi_{i}..\pi_{i} + l]
\]
Since the cyclic rotations of the Burrows-Wheeler-transform are lexicographically sorted, this inequality in turn can be used to find the corresponding $i$ for the given extended maximal pair.

Hence, this extended maximal pair does not correspond to another index pair given by the Burrows-Wheeler-transform. Therefore, the maximal pairs given by different lexicographically neighbored cyclic rotations are distinct. □

By summing up all relevant maximal pairs, the size of the used alphabet and an additional 1 for the first run, we obtain the following upper bound for the runs in the RLBWT.

**Theorem 7.2.16.** Let $S$ be a string with string attractor $\Gamma$.

Then, there are at most $(|\Gamma| + 2)^2 (1223 \log_2(|S|) + 1)$ runs in the RLBWT.

**Proof.** Each index pair $(\pi_{i-1}, \pi_i)$ where a new run starts corresponds to at least one of the following:

- the extended empty maximal pair,
- a non-$\frac{1}{26}$-highly-periodic extended maximal pair or
- a non-extendable $\frac{1}{24}$-highly-periodic extended maximal pair.

Since $S$ contains at most $|\Gamma| + 1$ distinct characters, there are at most $(|\Gamma| + 1)^2$ distinct extensions of the empty string and thus $(|\Gamma| + 1)^4$ distinct extensions of the empty maximal pair.

However, since we only care about the extensions in which the cyclic rotation with starting $\pi_i$ introduces a new character, there are at most $|\Gamma| = |\Gamma| + 1 - 1$ extensions which correspond to a new run.

Theorem 7.2.1 implies that there are at most $(|\Gamma| + 2)^2 \cdot 1103 \log_2(|S|)$ distinct non-$\frac{1}{26}$-highly-periodic extended maximal pairs.
Theorem 7.2.2 states that there are at most \((|\Gamma| + 2)^2 \cdot 120 \log_2(|S|)\) distinct non-extendable \(\frac{1}{z^2}\)-highly-periodic maximal pairs.

Additionally, we add 1 to account for the first run.

Therefore, there are at most
\[
|\Gamma| + (|\Gamma| + 2)^2 \cdot 1103 \log_2(|S|) + (|\Gamma| + 2)^2 \cdot 120 \log_2(|S|) + 1
\leq (|\Gamma| + 2)^2 \left(1223 \log_2(|S|) + 1\right)
\]
runs in the RLBWT. \(\square\)

### 7.3 Tightness

In this section, we will prove that for every \(z, q\) with \(z \leq q\) there are binary strings with \(O(z)\) LZ77-factors without \(q\)-th powers which have \(\Omega(qz^3)\) distinct maximal repeats. This proves that the upper bound given in Theorem 7.2.6 cannot be improved by more than a constant factor. We further show that this tightness also holds for binary strings.

In order to prove the tightness of Theorem 7.2.6, we will consider the following strings in this section.

**Definition 7.3.1.** Let \(v, d, q\) and \(c\) with \(v \geq 1, c \leq q\) and \(d < q\) be natural numbers.

We define symmetric nested powers \(V_{v,c,q}\) recursively by

\[
V_{0,c,q} := \sigma_0 \quad \text{and} \quad V_{v,c,q} := (V_{v-1,c,q})^c \sigma_v(V_{v-1,c,q})^c.
\]

We define their suffixes \(L_{v,c,q}\), their prefixes \(R_{v,c,q}\) and their symmetric infixes \(C_{v,c,q}\) by

\[
L_{v,c,q} := (V_{v-1,c,q})^c \sigma_v(V_{v-1,c,q})^q, \\
R_{v,c,q} := (V_{v-1,c,q})^q \sigma_v(V_{v-1,c,q})^c \quad \text{and} \\
C_{v,c,q} := L_{1,c,q}L_{2,c,q} \cdots L_{v-1,c,q}V_{v,c,q}R_{v-1,c,q} \cdots R_{2,c,q}R_{1,c,q}
\]

And finally, we define the strings \(S_{v,d,q}\) by

\[
S_{v,d,q} := V_{v,q,q} \left( \prod_{i=1}^{d} S_i C_{v,q-i,q} \right).
\]

Firstly, we will show that the strings \(C_{v,c,q}\) are indeed proper substrings of \(V_{v,q,q}\).
Lemma 7.3.2. For \( w \geq 1 \) and \( c \leq q - 1 \) the string \( L_{1,c,q}L_{2,c,q} \ldots L_{w,c,q} \) is a proper suffix of \( V_{w,q,q} \) and the string \( R_{w,c,q} \ldots R_{2,c,q}R_{1,c,q} \) is a proper prefix of the string \( V_{w,q,q} \).

Proof. We will only show that \( L_{1,c,q}L_{2,c,q} \ldots L_{w,c,q} \) is a proper suffix of \( V_{w,q,q} \). It then follows by symmetry that \( R_{w,c,q} \ldots R_{2,c,q}R_{1,c,q} \) is a proper prefix of the string \( V_{w,q,q} \).

For \( w = 1 \), we have

\[
L_{1,c,q} = (V_{0,q,q})^c \sigma_1 (V_{0,q,q})^q = \sigma_0^c \sigma_1 \sigma_0^q
\]

and

\[
V_{w,q,q} = V_{1,q,q} = \sigma_0^q \sigma_1 \sigma_0^q.
\]

Thus, the string \( L_{1,c,q}L_{2,c,q} \ldots L_{w,c,q} \) is a proper suffix of \( V_{w,q,q} \).

Similarly, we have

\[
V_{w,q,q} = (V_{w,q,q})^{c+1} \sigma_{w+1} (V_{w,q,q})^q \]

which is a suffix of \( V_{w+1,q,q} = (V_{w,q,q})^q \sigma_{w+1} (V_{w,q,q})^q \).

Now let a \( w \) be given, such that the string \( L_{1,c,q}L_{2,c,q} \ldots L_{w,c,q} \) is a proper suffix of \( V_{w,q,q} \). Then, it follows that \( L_{1,c,q}L_{2,c,q} \ldots L_{w+1,c,q} \) is a suffix of \( V_{w+1,q,q} \). Thus, by induction, the lemma holds for each \( w \geq 1 \).

By construction of the strings of the form \( C_{v,c,q} \), it follows directly from this lemma that the following corollary holds.

Corollary 7.3.3. For \( v \geq 1 \) and \( c \leq q - 1 \), the string \( C_{v,c,q} \) is a proper substring of \( V_{v,q,q} \).

With this corollary, we can now prove an upper bound for the highest power in the strings of the form \( S_{v,d,q} \).

Lemma 7.3.4. For \( v \geq 1 \) and \( d \leq q \), the string \( S_{v,d,q} \) does not contain \((2q + 1)\)-th powers.

Proof. By contradiction:

Assume that there is a \((2q + 1)\)-th power \( P \) in \( S_{v,d,q} \).

Since each \( S_i \) has only one occurrence in \( S \), the \((2q + 1)\)-th power \( P \) cannot contain any \( S_i \). Thus, the substring \( P \) is a substring of \( V_{v,q,q} \) or of any of the substrings of the form \( C_{v,i,q} \). Corollary 7.3.4 proves that the latter are substrings of \( V_{v,q,q} \). Thus, the power \( P \) is a substring of \( V_{v,q,q} \).

Let \( i \) be the highest integer such that \( P \) is a substring of \((V_{i,q,q})^{2q}\).

By construction, the string \( V_{i,q,q} \) contains the character \( \sigma_i \) exactly once. Thus, there exactly \( 2q \) occurrences of the character \( \sigma_i \) in \((V_{i,q,q})^{2q}\). This implies that the character \( \sigma_i \) occurs at most \( 2q \) times in \( P \). Hence, since \( P \) is supposed to be a \((2q + 1)\)-th power, it cannot contain the character \( \sigma_i \).

However, by construction of the strings \( V_{i,q,q} \), this implies that \( P \) is a substring of \((V_{i-1,q,q})^{2q}\). This however, contradicts the maximality of \( i \) and
thereby proves that \( P \) cannot be a \((2q + 1)\)-th power. Therefore, the string \( S_{v,d,q} \) does not contain \((2q + 1)\)-th powers. \( \square \)

Also, we can give an upper bound for the number of LZ77-factors of the strings of the form \( S_{v,d,q} \).

**Lemma 7.3.5.** For \( v \geq 1 \) and \( d \leq q \), the string \( S_{v,d,q} \) can be written with at most \( 1 + 3v + 2d \) LZ77-factors.

**Proof.** The string \( V_{0,q,q} \) consists of a single letter. Each string of the form \( V_{v,q,q} \) extends the prefix \( V_{v-1,q,q} \) by an exponentiation, a new character and a second copy of \((V_{v-1,q,q})^q\). Therefore, the string \( V_{v,q,q} \) can be written with at most 3 more LZ77-factors than \( V_{v-1,q,q} \). By induction, this proves that \( V_{v,q,q} \) can be written with at most \( 1 + 3v \) LZ77-factors.

Corollary 7.3.4 proves that each \( C_{v,q} - i,q \) is a substring of \( V_{v,q,q} \). Therefore, each factor of the form \( S_iC_{v,q} - i,q \) needs at most 2 additional LZ77-factors.

Therefore, the string \( S_{v,d,q} \) needs at most \( 1 + 3v + 2d \) LZ77-factors. \( \square \)

Next, we will prove for construct some maximal repeats in the strings of the form \( S_{v,d,q} \).

**Lemma 7.3.6.** For \( w \leq v - 1 \) and \( 1 \leq m + 1 \leq q - d \leq l, r \leq q - 1 \) the string

\[
M_{w,l,m,r,q} := L_{1,l,q} L_{2,l,q} \ldots L_{w,l,q} (V_{w,q,q})^m R_{w,r,q} \ldots R_{2,r,q} R_{1,r,q}
\]

is a maximal repeat of \( S_{v,d,q} \).

**Proof.** Firstly, we will prove that each of the strings \( M_{w,l,m,r,q} \) is both a proper prefix of \( C_{v,l,q} \) and a proper suffix of \( C_{v,r,q} \).

Lemma 7.3.2 shows that the string \( R_{w,r,q} \ldots R_{2,r,q} R_{1,r,q} \) is proper prefix of \( V_{w,q,q} \). Thus, the string \((V_{w,q,q})^m R_{w,r,q} \ldots R_{2,r,q} R_{1,r,q}\) is a proper prefix of \((V_{w,q,q})^{m+1}\) which is in turn a proper prefix of \( L_{w+1,l,q} \). Hence, the string \( M_{w,l,m,r,q} \) is a proper prefix of \( C_{v,l,q} \). Similarly, the string \( M_{w,l,m,r,q} \) is a proper suffix of \( C_{v,r,q} \).

By construction of \( S_{v,d,q} \), there is an occurrence of \( C_{v,l,q} \) which is preceded by \( S_{q-1} \). Since \( M_{w,l,m,r,q} \) is a proper prefix of \( C_{v,l,q} \), the corresponding occurrence of \( M_{w,l,m,r,q} \) is preceded by \( S_{q-1} \) and succeeded by \( \sigma_i \) for some \( i \).

Conversely, there is an occurrence of \( C_{v,r,q} \) which is succeeded by \( S_{q-r+1} \). Hence, there is a corresponding occurrence of \( M_{w,l,m,r,q} \) which is preceded by \( \sigma_i \) for some \( i \) and succeeded by \( S_{q-r+1} \).

Thus, these two occurrences of \( M_{w,l,m,r,q} \) form a maximal pair. Therefore, the string \( M_{w,l,m,r,q} \) is a maximal repeat of \( S_{v,d,q} \). \( \square \)
The combination of the possible values for \( w, l, m \) and \( r \) now leads to a lower bound for the number of distinct maximal repeats.

**Corollary 7.3.7.** For \( v \geq 1 \) and \( d \leq q \), the string \( S_{v,d,q} \) has at least

\[
(v - 1)(q - d)d^2
\]

distinct maximal repeats.

**Proof.** The previous lemma shows that for each \( w, l, m \) and \( r \) with \( w \leq v - 1 \) and \( 1 \leq m + 1 \leq q - d \leq l, r \leq q - 1 \), the substring \( M_{w,l,m,r,q} \) is a maximal repeat of \( S_{v,d,q} \). Also, by construction, for different tuples \( (w,l,m,r) \), the corresponding substrings \( M_{w,l,m,r,q} \) are distinct.

We have exactly \( v - 1 \) possible values for \( w \), exactly \( q - d \) possible values for \( m \) and exactly \( d \) for both \( l \) and \( r \). Therefore, we have at least \( (v - 1)(q - d)d^2 \) possible tuples that generate at least \( (v - 1)(q - d)d^2 \) distinct maximal repeats.

Combining the upper bounds for the highest Power and the number of LZ77-factors with the lower bound for the number of maximal repeats yields the following lower bound.

**Theorem 7.3.8.** Let \( z \) and \( q \) with \( 2000 \leq z \leq q \) be two natural numbers. Then, there is a string \( S \) with at most \( z \) LZ77-factors and without \( q \)-th powers which has at least \( \frac{1}{500}q^3z^3 \) maximal repeats.

**Proof.** We consider the string \( S = S_{\{\frac{z}{9}\},\{\frac{z}{3}\} - 1,\{\frac{z}{2}\}} \).

Lemma 7.3.4 proves that \( S \) does not contain \( 2\{\frac{z}{2}\} + 1 \)-th powers. The inequality \( 2\{\frac{z}{2}\} + 1 \leq q \) therefore shows that \( S \) does not contain \( q \)-th powers.

Lemma 7.3.5 proves that \( S \) has at most \( 1 + 3\{\frac{z}{9}\} + 2\{\frac{z}{3}\} - 1 \) \( \leq z \) LZ77-factors.

Lastly, Corollary 7.3.7 proves that \( S \) has at least

\[
\left(\left\lfloor \frac{z}{9} \right\rfloor - 1\right) \left(\left\lfloor \frac{q - 1}{2} \right\rfloor - \left(\left\lfloor \frac{z}{3} \right\rfloor - 1\right)\right) \left(\left\lfloor \frac{z}{3} \right\rfloor - 1\right)^2
\]

maximal repeats.

By prerequisite, we have the inequalities

\[
\left\lfloor \frac{z}{9} \right\rfloor - 1 \geq \frac{z}{9} - 2 \geq \frac{z}{9} - 2 \frac{z}{2000} = \left(\frac{1}{9} - \frac{1}{1000}\right)z = \frac{991}{9000}z,
\]

\[
\left\lfloor \frac{z}{3} \right\rfloor - 1 \geq \frac{z}{3} - 2 \geq \frac{z}{3} - 2 \frac{z}{2000} \geq \left(\frac{1}{3} - \frac{1}{1000}\right)z = \frac{997}{3000}z
\]

and
7.3. TIGHTNESS

\[
\left\lfloor \frac{q - 1}{2} \right\rfloor - \left( \left\lfloor \frac{z}{3} \right\rfloor - 1 \right) \geq \frac{q - 1}{2} - 1 - \frac{z}{3} + 1 = \frac{q}{2} - \frac{1}{2} - \frac{z}{3} \\
\geq \frac{q}{2} - \frac{1}{2} \cdot \frac{q}{2000} - \frac{z}{3} \cdot \frac{q}{z} = \left( \frac{1}{2} - \frac{1}{4000} - \frac{1}{3} \right) q \\
= \frac{1997}{12000} q.
\]

Therefore, the number of maximal repeats in \( S \) is at least

\[
\left( \left\lfloor \frac{z}{9} \right\rfloor - 1 \right) \left( \left\lfloor \frac{q - 1}{2} \right\rfloor - \left( \left\lfloor \frac{z}{3} \right\rfloor - 1 \right) \right) \left( \left\lfloor \frac{z}{3} \right\rfloor - 1 \right)^2 \\
\geq \left( \frac{991}{9000} z \right) \left( \frac{1997}{12000} q \right) \left( \frac{997}{3000} z \right)^2 \\
\geq \frac{1}{500} q z^3.
\]

\[\square\]

Similarly to Lemma 5.3.3, we can prove that strongly invertible, uniform morphisms do not decrease the number of extended maximal repeats. The proof follows the same idea as the proof for the corresponding lemma for maximal \( \delta \)-repetitions.

**Lemma 7.3.9.** Let \( S \) be a string over the alphabet \( \Sigma \) and let \( \varphi \) be strongly invertible, uniform morphism from \( \Sigma \) to an alphabet \( \Sigma' \).

Then distinct maximal repeats in \( S \) correspond to distinct maximal repeats in the image of \( \varphi(S) \).

**Proof.** Since \( \varphi \) is uniform, each maximal pair in \( S \) in a pair of identical strings. This pair can be extended to obtain a maximal pair in the image of \( S \).

Let \( \sigma_i P \sigma_r \) and \( \sigma'_i P \sigma'_{r'} \) be the corresponding extended maximal repeats of a maximal pair. Then we have \( \varphi(\sigma_i P) \neq \varphi(\sigma'_i P) \) and \( \varphi(P \sigma_r) \neq \varphi(P \sigma'_{r'}) \). Hence, the corresponding maximal pair in the image of \( S \) is a proper substring of \( \varphi(\sigma_i P \sigma_r) \).

Since \( \varphi \) is strongly invertible, we can reobtain the maximal repeat \( P \) in \( S \) from this substring of \( \varphi(\sigma_i P \sigma_r) \).

Since we can retrieve maximal repeats from the corresponding maximal pairs of their image, distinct maximal repeats in \( S \) naturally correspond to distinct maximal repeats in \( \varphi(S) \). \[\square\]

Theorem 4.0.4 then translates from strings over an unbounded alphabet to strings over a binary alphabet.
Theorem 7.3.10. For all positive integers \( z \) and \( q \) with \( z \leq q \), there is a binary string \( S \) with

- with \( \mathcal{O}(z) \) LZ77-factors,
- with \( \Omega(qz^3) \) distinct maximal repeats and
- without \( q \)-th powers.

7.4 Conclusion

In this chapter, we introduced and proved upper bounds for the number of both non-highly-periodical maximal pairs and non-extendable highly-periodical maximal pairs.

We have shown that the number of distinct non-\( \frac{1}{2q} \)-highly-periodical maximal pairs in a string \( S \) with string attractor \( \Gamma \) is in \( \mathcal{O}\left( |\Gamma|^2 \frac{q}{\log q} \log_2 |S| \right) \) and that this upper bound is tight. We have also proven that the number of distinct non-extendable \( \frac{1}{2q} \)-highly-periodical maximal pairs in a string \( S \) with string attractor \( \Gamma \) is in \( \mathcal{O}\left( |\Gamma|^2 \log_2 |S| \right) \).

We concluded that the number of arcs of the CDAWG of a string \( S \) without \( q \)-th powers and with string attractor \( \Gamma \) is also in \( \mathcal{O}\left( |\Gamma|^2 \frac{q}{\log q} \log_2 |S| \right) \). Also, the number of runs of the RLBWT of a string \( S \) with string attractor \( \Gamma \) is in \( \mathcal{O}\left( |\Gamma|^2 \log_2 |S| \right) \).

The analysis of non-extendable maximal pairs might be used to merge nodes of the CDAWG which correspond to maximal repeats with equal bases. We therefore conjecture that this can lead to a compressed version of CDAWG which only contains \( (|\Gamma|^2 \log_2 |S|) \) arcs but inherits most of the useful properties of the original CDAWG.

If the number of LZ77-factors of the underlying string \( S \) is sublogarithmic, the proof for the upper bound of non-\( \frac{z'}{2q} \)-highly-periodical maximal pairs does not seem to use the structure of \( S \) efficiently. We conjecture that if the substrings of \( S \) and the reversed string \( S_{rev} \) have less than \( z' \) LZ77-factors each, it should be possible to prove that the number of runs in the RLBWT is bounded from above by \( \mathcal{O}(z'|\Gamma|^2) \).

Lastly, the strings in Section 7.3 that were used to prove the tightness of Theorem 7.2.1 use that the highest power in the underlying string is at least as large as the number of LZ77-factors. Further research is needed to find a tight upper bound for strings without \( q \)-th powers if \( q \) is small.
Bibliography


Appendix A

Propositions

In this thesis, I proved the following propositions.

Propositions A.0.1. In strings with a small compressed size, the number of distinct periodic patterns is small as well.

For maximal $\delta$-repetitions, non-$\frac{1}{2q}$-highly-periodic maximal pairs and non-extendable $\frac{1}{2\Sigma}$-highly-periodic maximal pairs, this was proven in Subsections 5.2.1, 7.2.1 and 7.2.2, respectively.

For maximal $\delta$-subrepetitions, this was proven for strings without $q$-th powers in Subsection 5.2.2.

In a string over a given alphabet $\Sigma$, there are only $\Sigma$ distinct subcadence with a given length possible. Therefore, the concept of distinctness is not meaningful for 3-subcadences and their variants. For different 3-subcadences and their variants, Section 6.2 proves that there a no non-trivial bounds.

Propositions A.0.2. While the previous proposition already limits the number of runs in the run-length encoded Burrows-Wheeler transform (RLBWT) without $q$-th powers, the RLBWT should not suffer from high powers.

This proposition was proven in Subsection 7.2.3. An alternative proof with a comparable upper bound were published a few month prior by Kempa and Kociumaka in [57] (arXiv).

Propositions A.0.3. The discrete acyclic convolution can be extended from a rectangle as underlying geometry to arbitrary polygons.

This proposition was proven in Chapter 3.

Propositions A.0.4. The convolution with arbitrary polygons can be used to find 3-cadences with a given character efficiently. Also, on binary strings, only very structured strings can avoid having 3-cadences.
In Subsection 6.4.2, we used the convolution to count the 3-cadences in quasi-linear time.

In Subsection 6.4.3, we have proven that strings without 3-cadences are very structured and use this necessary structure to find a 3-cadence or prove that there are no 3-cadences in the given string.

For longer cadences or larger alphabets, we have proven in Section 6.3, that the cadence detection becomes $\mathcal{NP}$-complete.

**Propositions A.0.5.** For each alphabet $\Sigma$, there is a strongly invertible, uniform and cube-free morphism from $\Sigma$ to $\{a, b\}$.

One such morphism was presented in Chapter 4.

This morphism was used in the Chapters 5 and 7 to prove that the upper bounds for the maximal $\delta$-repetitions and maximal repeats are even on binary strings up to a constant factor tight.