COMP. & STRUCT. APPROACHES TO PERIODICITIES IN STRINGS

COMPUTATIONAL AND STRUCTURAL APPROACHES TO PERIODICITIES IN STRINGS

By ANDREW BAKER, B.Comp(H) M.Sc.

A Thesis Submitted to the School of Graduate Studies in Partial Fulfilment of the Requirements for the Degree Doctor of Philosophy

DOCTOR OF PHILOSOPHY (Computer Science) 2012

McMaster University, Hamilton, Ontario

TITLE: Computational and Structural Approaches to Periodicities in Strings

AUTHOR: Andrew Baker, B.Comp(H) (University of Guelph) M.Sc. (University of Guelph)

SUPERVISORS: Professors Antoine Deza and Frantisek Franek

NUMBER OF PAGES: viii, 95

Abstract

We investigate the function $\rho_d(n) = \max\{\ r(\boldsymbol{x}) \mid \boldsymbol{x}$ is a (d,n)-string $\}$ where $r(\boldsymbol{x})$ is the number of runs in the string $\boldsymbol{x}$, and a (d,n)-string is a string with length n and exactly d distinct symbols. Our investigation is motivated by the conjecture that $\rho_d(n) \leq n - d$. We present and discuss fundamental properties of the $\rho_d(n)$ function. The values of $\rho_d(n)$ are presented in the $(d, n-d)$-table with rows indexed by d and columns indexed by $n - d$ which reveals the regularities of the function. We introduce the concepts of the r-cover and core vector of a string, yielding a novel computational framework for determining $\rho_d(n)$ values. The computation of the previously intractable instances is achieved via first computing a lower bound, and then using the structural properties to limit our exhaustive search only to strings that can possibly exceed this number of runs. Using this approach, we extended the known maximum number of runs in binary string from 60 to 74. In doing so, we find the first examples of run-maximal strings containing four consecutive identical symbols. Our framework is also applied for an arbitrary number of distinct symbols, d. For example, we are able to determine that the maximum number of runs in a string with 23 distinct symbols and length 46 is 23. Further, we discuss the structural properties of a shortest (d,n)-string $\boldsymbol{x}$ such that $r(\boldsymbol{x}) > n - d$, should such a string exist.

Acknowledgements

My sincere thanks to my supervisors, Antoine Deza and Frantisek Franek, who provided invaluable support and encouragement to me in my research. Thanks also to my colleagues who assisted me in my work and were excellent moral support.

Thanks are also given to my friends and family for their ongoing love and support. To my parents, who have stood by me and encouraged me through 24 straight years of school. To my wife Stephanie, who's daily support and unfailing belief in me keeps me going. Finally, to my sons Benjamin and Thomas, who drive me to do my best.

This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET:`www.sharcnet.ca`).

Contents

List of Figures

List of Tables

Chapter 1

Introduction and string basics

Ubiquitous in the world of computer science are strings–linear sequences of symbols. Strings have particular importance in the fields of computational biology and bioinformatics, where they represent, among other things, DNA and protein sequences. Repetitive structures in biological sequences are of particular importance, as they are often related to the function of the sequence, and can be used to deduce evolutionary history. The vast amounts of data being created daily have lead to an increased need to both analyze and compress that data efficiently. Strings often need to be transmitted securely, and therefore string algorithms are important in the field of cryptography.

Besides the practical implications, repeats in strings are one of the most fundamental aspects of combinatorics on words [9, 35, 41]. In this thesis we will explore a particular repetitive structure on strings, namely runs. Specifically, we are interested in strings which have the maximum number of runs possible given certain parameters. We will develop several structural insights, and use them to compute run-maximal strings.

1.1 Introduction to string terminology

A string $\boldsymbol{x}$ is a linear sequence of symbols and is indexed from 1 through n, $\boldsymbol{x}[1] \ldots \boldsymbol{x}[n]$. The symbols of a string are drawn from a given alphabet, $\mathcal{A}$. The alphabet of a string $\boldsymbol{x}$ is the set of symbols occurring in $\boldsymbol{x}$ and is denoted by $\mathcal{A}(\boldsymbol{x})$.

A (d, n)-string refers to a string of length n with exactly d distinct symbols. The function $d(\boldsymbol{x})$ denotes the number of distinct symbols of the string $\boldsymbol{x}$, while the function $n(\boldsymbol{x})$ gives the length of $\boldsymbol{x}$. As a shorthand, when unambiguous it will be assumed that a string has its length given by n and its number of distinct symbols given by d. It follows that $|\mathcal{A}(\boldsymbol{x})| = d(\boldsymbol{x})$. A distinction must be drawn between $\mathcal{A}$, the alphabet from which $\boldsymbol{x}$ takes its symbols, and the alphabet of $\boldsymbol{x}$, $\mathcal{A}(\boldsymbol{x})$. The former case refers to the symbols which *may* appear in $\boldsymbol{x}$, whereas $\mathcal{A}(\boldsymbol{x})$ is exactly the set of symbols which *do* appear in $\boldsymbol{x}$. This distinction is important when discussing strings generated on a certain sized alphabet, corresponding to $\mathcal{A}$, compared to the strings with some number of distinct symbols, corresponding to $\mathcal{A}(\boldsymbol{x})$.

A symbol which occurs exactly one, two, or k times in a string will be termed respectively a ***singleton***, ***pair***, or k***-tuple***.

A substring $\boldsymbol{x}[i..j]$ is the concatenation of the symbols $\boldsymbol{x}[i]$, $\boldsymbol{x}[i+1]$, ..., $\boldsymbol{x}[j]$. If $j < i$, then the substring $\boldsymbol{x}[i..j]$ will be taken to be the empty string ε.

The ***join***, $\boldsymbol{x}[i_1..i_k] \cup \boldsymbol{x}[j_1..j_m]$, of two substrings of a string $\boldsymbol{x}$ is defined when:

- $i_1 \leq j_1 \leq i_k + 1$, in which case $\boldsymbol{x}[i_1..i_k] \cup \boldsymbol{x}[j_1..j_m] = \boldsymbol{x}[i_1..max\{i_k, j_m\}]$, or
- $j_1 \leq i_1 \leq j_m + 1$, in which case $\boldsymbol{x}[i_1..i_k] \cup \boldsymbol{x}[j_1..j_m] = \boldsymbol{x}[j_1..max\{i_k, j_m\}]$.

Thus, the join is defined when the two substrings are either adjacent or overlap,

in which case it is the substring of elements which are in one of the substrings or both. Otherwise, the join is said to be undefined [2].

The ***intersection***, $\boldsymbol{x}[i_1..i_k] \cap \boldsymbol{x}[j_1..j_m]$, of two substrings of a string $\boldsymbol{x}$ is defined by one of three cases:

- $i_1 \leq j_1 \leq i_k$, in which case $\boldsymbol{x}[i_1..i_k] \cap \boldsymbol{x}[j_1..j_m] = \boldsymbol{x}[j_1..min\{i_k, j_m\}]$, or
- $j_1 \leq i_1 \leq j_m$, in which case $\boldsymbol{x}[i_1..i_k] \cap \boldsymbol{x}[j_1..j_m] = \boldsymbol{x}[i_1..min\{i_k, j_m\}]$, or
- $i_k < j_1$ or $j_m < i_1$ in which case $\boldsymbol{x}[i_1..i_k] \cap \boldsymbol{x}[j_1..j_m] = \varepsilon$.

Thus, the intersection is non-empty when the two substrings overlap, and is the substring consisting of the elements which are in both of the substrings. Otherwise, the intersection is empty.

We illustrate the concept of a join and intersection of two substrings in Figure 1.1.

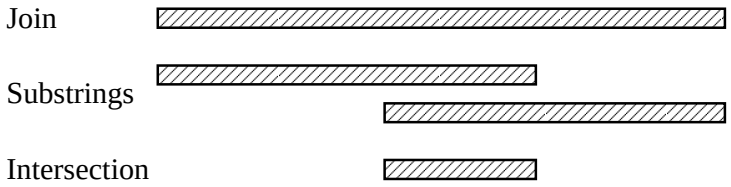

Figure 1.1: A graphical illustration of the join and the intersection of two substrings.

1.2 Notations and definitions

In the papers which are generally considered the foundation of the field of combinatorics on words Thue [47] investigated strings which lack repetitions. A repetition consists of an integer number of copies of a substring adjacent to each other.

Definition 1.1: *The triple* (s, p, e) *represents a* ***repetition*** *in* $\boldsymbol{x}$ *at the substring* $\boldsymbol{x}[s..s + ep - 1]$ *if:*

- $\boldsymbol{x}[s..s + e - 1] = \boldsymbol{x}[s + e..s + 2e - 1] = \ldots = \boldsymbol{x}[s + (e - 1)p..s + ep - 1]$*, and*
- $1 \leq s < n$*,* $1 \leq 1 \leq \frac{n}{2}$*, and* $e \geq 2$*.*

The variable s is the ***start*** of the repetition, e is the ***exponent***, and p is the ***period*** of the repetition–the length of the substring which is repeated. The repeated substring $\boldsymbol{x}[s..s + p - 1]$ is termed the ***generator***. The period has also been called the *period length* [37]. The generator of a repetition has also been called the *root* [36] and the *periodic part* [42, 43]. Other names for repetitions in the literature include *integer repetitions* [36] and *integer powers* [36],

Thue was interested in building infinitely long strings with α distinct symbols which do not contain a repetition of exponent at least r, but do have repetitions of exponent $r - 1$ [47]. Since that time, counting repetitions and finding strings which are repetition-dense rather than repetition-free has become a focus of research attention.

As the initial work on repetitions was on the prevention of repetitions, the concept of a repetition is quite general. For example, under the most general definition of a repetition, the string *aaaa* contains 6 repetitions: $(1, 1, 2)$, $(1, 1, 3)$, $(1, 1, 4)$, $(2, 1, 2)$, $(2, 1, 3)$, $(3, 1, 2)$, and $(1, 2, 2)$. After attention turned to counting repetitions, it became desirable to find more efficient ways to talk about repetitions.

The term ***left-maximal repetition*** refers to a repetition which cannot be extended by another copy of the generator to the left, while a ***right-maximal repetition*** corresponds to the same property to the right. In other words, (s, p, e)

is left-maximal when $(s-p, p, e+1)$ is not a repetition, and is right-maximal when $(s, p, e+1)$ is not a repetition. A ***maximal repetition*** is a repetition which cannot be extended with another copy of the generator to either the left or the right [6, 36]. In the literature, when the usage is unambiguous, the phrase *maximal repetition* has also been used as a shorthand for *maximal fractional repetition*, another name for runs which are defined below [36].

Primitively rooted repetitions are those repetitions which have a generator which is primitive–that is, not itself a repetition [11, 31, 35, 36, 38, 42, 43].

A repetition (s, p, e) is *left-shiftable* when $\boldsymbol{x}[s-1]$ is defined, that is $s > 1$, and $\boldsymbol{x}[s-1] = \boldsymbol{x}[s+p-1]$. Similarly, a repetition is *right-shiftable* when $\boldsymbol{x}[s+ep]$ is defined, that is $s + ep \leq n$ and $\boldsymbol{x}[s] = \boldsymbol{x}[s+ep]$. In other words, a repetition (s, p, e) is left-shiftable exactly when $(s-1, p, e)$ is defined and also a repetition, and is right-shiftable exactly when $(s+1, p, e)$ is defined and also a repetition. Note that if a repetition is left-maximal it does not imply that it is not left-shiftable. Left-maximality requires a whole additional copy of the generator to the left, while left-shiftablity only requires that there is a repetition of the same period starting immediately to the left. It follows, however, that if a repetition is not left-maximal, then it must be left-shiftable. The same properties hold for right-maximality and right-shiftability.

In Figure 1.2 we illustrate a small string with many repetitions. Table 1.1 identifies the properties of all the repetitions labelled in Figure 1.2.

A repetition with exponent 2 is termed a ***square***. When the exponent is 3, the repetition is termed a ***cube***. Repetitions with a larger exponent will be referred to as ***a repetition with exponent*** e.

Crochemore [6] showed that the maximum number of primitively rooted

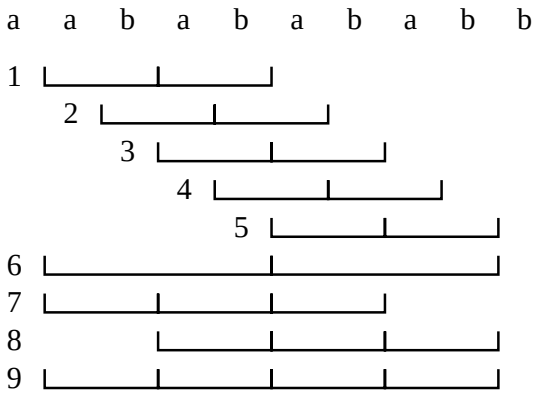

Figure 1.2: The nine repetitions which include both an a and b in the string *aababababb*. Table 1.1 classifies these repetitions.

maximal repetitions in a string is $O(n \log n)$, and this bound is the best possible due to results on Fibonacci strings. This provides a limit on the run-time of repetition-finding algorithms as in order to report all repetitions the algorithm requires at least $O(n \log n)$ steps. In an effort to get past this bound, the search turned to a more efficient way to encode repetitions. This led to the development of the notion of runs.

Definition 1.2: *The quadruple* (s, p, e, t) *represents a* ***run*** *in* $\boldsymbol{x}$ *if:*

- $(s + i, e, p)$ *is a repetition for all* $0 \leq i \leq t$*, and*
- *there is no repetition* $(s - 1, e, p)$*, that is the run is not left-shiftable, and*
- *there is no repetition* $(s + t + 1, e, p)$*, that is the run is not right-shiftable, and*
- $\boldsymbol{x}[s..s + p - 1]$ *is primitive.*

As in the case of repetitions, s is the ***start*** of the run, e is the ***exponent***, p is the ***period***, and $\boldsymbol{x}[s..s + p - 1]$ is the ***generator***. The term t refers to the

Table 1.1: The classification of nine of the repetitions in the string *aababababb*, as identified in Figure 1.2.

Repetition	Left-maximal	Right-maximal	Maximal	Primitively rooted	Left-shiftable	Right-shiftable
1	yes	no	no	yes	no	yes
2	yes	no	no	yes	yes	yes
3	no	no	no	yes	yes	yes
4	no	yes	no	yes	yes	yes
5	no	yes	no	yes	yes	no
6	yes	yes	yes	no	no	no
7	yes	no	no	yes	no	yes
8	no	yes	no	yes	yes	no
9	yes	yes	yes	yes	no	no

tail, and is the length of the proper prefix of the generator which occurs at the end of the run. The starting position s has also been referred to as the *occurrence* of the run [42].

The term ***run*** was coined by Iliopoulos, Moore, and Smyth in [31]. The term *fractional repetitions* appears in the literature, and refers both to runs, and to a part of a run which may be extended to the left or right [36]. A *maximal fractional repetition* cannot be extended by a single letter to either the left or right–exactly corresponding with the definition of a run [36]. As a short-hand, runs have also been referred to as *maximal repetitions* [7, 35, 36]. Other terms used to refer to runs include *maximal periodicities* [37] and *m-repetitions* [34].

Runs encode all repetitions in a string, and do so in the most efficient way possible; they encode all the repetitions in a string in a linear amount of space [36]. A given run represents $t + 1$ maximal repetitions, starting at each position s through $s + t$. Specifically, if $\boldsymbol{x}$ has the run (s, p, e, t), it must also have maximal

repetitions (s, p, e), $(s+1, p, e)$, ..., $(s+t, p, e)$. This correspondence is illustrated in Figure 1.2.

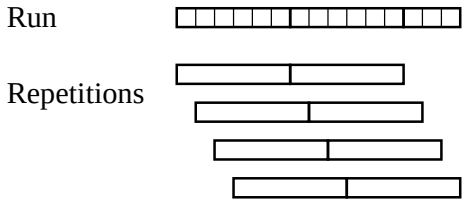

Figure 1.3: An illustration of how a run represents $t+1$ maximal repetitions. In this case, a run with $t = 3$ corresponds to 4 maximal repetitions.

We introduce some terminology related to runs.

Definition 1.3: *Given a run (s, p, e, t) in $\boldsymbol{x}$, the **leading square** of the run is the square $(s, p, 2)$, given by the substring $\boldsymbol{x}[s..s + 2p - 1]$ – the first two copies of the generator.*

The leading square has also been termed the *square part* [43, 42] and the *leftmost square* [23].

Definition 1.4: *Given a run (s, p, e, t) in $\boldsymbol{x}$, the **trailing square** of the run is the square $(s+(e-2)p+t, p, 2)$, given by the substring $\boldsymbol{x}[s+(e-2)p+t..s+ep+t-1]$ – the last $2p$ positions in the run.*

The trailing square has also been termed the *rightmost square* [23].

Definition 1.5: **[23]** *Given a run (s, p, e, t) in $\boldsymbol{x}$, the **core** of the run is the substring $\boldsymbol{x}[s + (e - 2)p + t..s + 2p - 1]$, that is, the intersection of the leading and trailing squares of the run.*

If the leading and trailing squares do not overlap, by the definition of the intersection, the core is empty. The core is the essential part of the run. If a

symbol within the core is changed or the core is split, then the run is destroyed. Conversely, if a symbol is changed outside of the run or the run is split in such a way that the core is preserved, then the presence of a run is maintained. For example, in the run $ab\underline{ab}ab$, the underlined ab in the middle is the core. If one of these symbols is replaced, say, $ab\underline{cb}ab$, the run is destroyed. Similarly, if a new symbol is inserted in the middle of the core, $ab\underline{a}c\underline{b}ab$, the run is destroyed. Conversely, in the run *abababab* which has an empty core, we can change any symbol, say *abacabab*, and a run still exists. Similarly, no matter where we split the run with a new symbol, a run is maintained: *abacbabab*. When a run has no core, changing a symbol of the run or splitting it may actually increase the number of runs, if both the leading and trailing squares are undamaged, as in the case of *ababcabab*.

We illustrate the concepts of leading and trailing squares in Figure 1.2, along with an existent core.

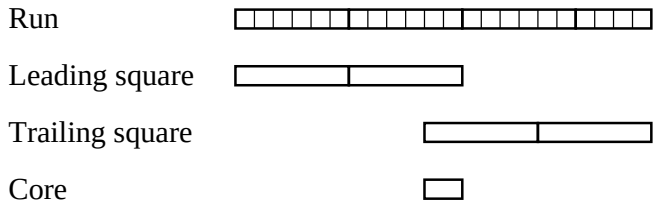

Figure 1.4: An illustration of a run with period 6, exponent 3, and tail 3. The leading square, trailing square, and core are identified.

Let $r(\boldsymbol{x})$ give the number of runs in a string $\boldsymbol{x}$. The function $\rho(n)$ gives the maximum number of runs over all strings of length n, so $\rho(n) = \max\{\, r(\boldsymbol{x}) | n(\boldsymbol{x}) = n \,\}$.

1.3 Background

The number of primitively rooted maximal repetitions in a string of length n is $O(n \log n)$. This is derived from the fact that there can only be $\log_\phi n$ primitively rooted squares which start at a given position, and that there are $n - 1$ possible starting positions for a square, showing there are $O(n \log n)$ primitively rooted maximal repetitions over the length of the whole string [11].

This bound was shown to be tight through the investigation of finite prefixes of the infinite Fibonacci string, a set of repetition-dense strings. See Section 1.7 for the definition. Crochemore [6] shows that the Fibonacci strings contain $\Omega(n \log n)$ squares, and that all squares in a Fibonacci string are primitively rooted.

The notion of a runs was introduced by Main [37] and so named by Iliopoulos, Moore, and Smyth [31]. They are a generalization of repetitions, as one run can represent multiple repetitions. They suggest that there could be a linear algorithm to find all runs, and therefore a linear number of runs. This idea was supported by the fact that Main showed there is a linear number of leftmost runs, and Iliopoulos, Moore and Smyth showed that there were only a linear number of runs in the repetition-dense Fibonacci strings.

Kolpakov and Kucherov [34, 35, 36] show that $\rho(n) = O(n)$. However, their approach does not give an idea of the constant factor involved [36]. The constant factor is conjectured to be 1.

Conjecture 1.6: Maximum Number of Runs Conjecture [35] *The maximum number of runs in a string is bounded by the length of the string. That is, for all* $n \geq 0$, $\rho(n) \leq n$.

Additional conjectures were also put forward regarding the properties of

$\rho(n)$ and run-maximal strings.

Conjecture 1.7: [**28**] *The maximum number of runs increases by at most two when the length of string increase by one:* $\rho(n) \leq \rho(n-1) + 2$.

Conjecture 1.8: [**45**] *The maximum number of runs* $\rho(n)$ *is obtained by a cube-free string on the binary alphabet for all* $n > 2$.

Although it is widely believed that there is always a run-maximal string on the binary alphabet, the cube-free condition does not hold. See Chapter 5 for further details. In fact, though Kolpakov and Kucherov originally followed their maximum number of runs conjecture with "at least for the binary alphabet", the binary-maximality conjecture is so widely accepted that this latter part has generally been left out of the conjecture.

Franek and Yang [29] consider the property of the function $\frac{\rho(n)}{n}$, and point out that it may not be monotonic. Indeed, it follows that $\frac{\rho(n)}{n} - \frac{\rho(n-1)}{n-1} < 0$ exactly when $\rho(n) = \rho(n-1)$.

Based on an analogy with the largest possible diameter $\Delta(d, n)$ of a d-dimensional bounded polytope of n facets, in addition to considering the length n of the string we began to take into account the number of distinct symbols d. Let the function $\rho_d(n)$ refer to the maximum number of runs over all (d, n)-strings rather than all strings of length n, so $\rho_d(n) = \max\{\ r(\boldsymbol{x}) \mid d(\boldsymbol{x}) = d \text{ and } n(\boldsymbol{x}) = n\ \}$. We propose that this second parameter is important to understanding the nature of run-maximal strings.

Conjecture 1.9: [**13**] *For all* $2 \leq d \leq n$, $\rho_d(n) \leq n - d$.

It follows that $\rho_d(n) \leq n - d$ implies $\rho_d(n) \leq n - 2$ for $2 \leq d \leq n$. We will show that the introduction of the second parameter d provides additional

approaches which may be used to explore the conjecture. While this second parameter makes a minor change to the conjectured bound for small d, this may prove fruitful in the search for an inductive proof. This is supported by the fact that there are no known strings of length n with n runs, but there are infinitely many (d, n)-strings with $n - d$ runs. While most work has focused on bounding the values of $\rho(n)$, in this thesis we concentrate on the computation of exact $\rho_d(n)$ values–we want to capture the behaviour of $n - \rho_d(n)$.

Multiple approaches have been explored in an attempt to prove $\rho(n) \leq n$. In the following sections, we first discuss the results of efforts to provide an upper bound for the maximum number of runs in strings of sufficient length. We then consider the search for extremely run-dense strings. Next we describe some of the algorithms developed to find runs in strings. Then we consider some specific classes of strings, and how many runs they may contain. Finally, we complete the overview of the field by looking at some related work concerning other type of repeats. At the end of this chapter, we provide an outline for the remainder of the thesis.

Under the assumption that $\rho_2(n) = \rho(n)$, most computations have focused on the binary strings. Kolpakov and Kucherov [34] give the values for $\rho_2(n)$ for $5 \leq n \leq 31$. These values were confirmed by Franek [21] and extended to $n = 35$. Kolpakov and Kucherov [33] have since found the values for $\rho_2(n)$ for $n \leq 60$.

1.4 Tightening the upper bounds

While proof that $\rho(n) \leq n$ has not yet been established, some close asymptotic bounds have been obtained. We present a brief overview of the results which have

lowered the upper bound to its current value.

As previously mentioned, the initial proof that the maximum number of runs is linear relative to the length of the string by Kolpakov and Kucherov [36] gave no explicit bounds on the constant factor. Giraud [30] showed that the limit $\lim_{n\to\infty} \frac{\rho(n)}{n}$ exists, and that it cannot be reached. The first concrete constant for an asymptotic bound was given by Rytter [42], who showed $\lim_{n\to\infty} \frac{\rho(n)}{n} \leq 5$. To achieve this bound, Rytter considers separately *highly periodic runs* (hp-runs) and *weakly periodic runs*. Highly periodic runs are those runs with a generator which is itself a run with period at most one quarter the length of the large generator. Runs are then grouped as neighbours according to their starting positions.

Puglisi *et al.* [41] tighten the limit to $\lim_{n\to\infty} \frac{\rho(n)}{n} \leq 3.48$. This is achieved through a refinement of the approach used by Rytter. They consider θ *highly periodic runs* (θ-hp-runs), a generalization of the hp-runs of Rytter. Again, the θ-hp-runs and the other runs are handled separately. They note that if $\rho(n) \leq n$, then this proof approach will likely not be able to prove the conjecture, and that the results of Kolpakov and Kucherov suggest there are no θ-hp runs in run-maximal strings.

Rytter [43] later gave a bound of $\lim_{n\to\infty} \frac{\rho(n)}{n} \leq 3.44$ through a further refinement of his previous approach. However, a final note by Puglisi, Simpson, and Smyth [41] indicates that a portion of Rytter's Lemma 6 is incorrect, and thus the final bound should actually be approximately 3.9.

Franek, Simpson, and Smyth [28] suggest an approach which can show that $\lim_{n\to\infty} \frac{\rho(n)}{n} \leq 2$. The approach relies on two unproven conjectures, either of which is sufficient to achieve the bound.

Crochemore and Ilie [7] show that $\lim_{n\to\infty} \frac{\rho(n)}{n} \leq 1.6$, and their presentation

of this proof suggests how the bound could be improved to $\lim_{n\to\infty} \frac{\rho(n)}{n} \leq 1.18$ or further. As with Rytter's approach, they consider different classes of runs separately. In this case, however, they look at runs with short periods, *microruns*, and those with long periods, *macroruns*. The boundary for microruns was taken to be $p \leq 9$. While Rytter's approach grouped runs which started close to each other, Crochemore and Ilie group runs which have centres close to each other, that is, similar values of $s + p$. The bound for the macroruns is based on a discrete mathematics argument, while the microruns are amortized over a region of the string through computation. This shows that for the microruns, $\lim_{n\to\infty} \frac{\rho^{p\leq 9}(n)}{n} \leq 1$ and for the macroruns, $\lim_{n\to\infty} \frac{\rho^{p>9}(n)}{n} \leq 0.6$. Crochemore, Ilie, and Tinta [9] remark that grouping runs by centres is somewhat counter-intuitive, as there may be a linear number of runs with the same centre, compared to a logarithmic number of runs with the same start. The computation-based microruns half of this approach was confirmed and extended by an independent investigation by Franek and Holub [23], who showed that when microruns are bounded by $p \leq 10$, $\lim_{n\to\infty} \frac{\rho^{p\leq 10}(n)}{n} \leq 1$.

Crochemore, Ilie, and Tinta [9] improve upon this approach, showing that $\lim_{n\to\infty} \frac{\rho(n)}{n} \leq 1.048$. In order to do this, they increased the bound for microruns to $p \leq 50$. In addition, instead of amortizing such that at most k runs have centres over k positions, they searched until they found at most k run centres over at least $\frac{k}{b}$ positions. This value was obtained with $b = 0.93$. Crochemore, Ilie, and Tinta [10] have since continued this computation with $p \leq 60$ while keeping $b \leq 0.93$. They have been able to obtain a further improvement of $\lim_{n\to\infty} \frac{\rho(n)}{n} \leq 1.029$.

1.5 Run-dense families of strings

In addition to the efforts to provide an upper bound on $\lim_{n\to\infty} \frac{\rho(n)}{n}$, there has been investigations into extremely run-dense families of strings.

Franek, Simpson, and Smyth [28] develop a method of constructing an infinite family of strings $\boldsymbol{x_i}$ such that $\lim_{i\to\infty} \frac{r(\boldsymbol{x_i})}{n(\boldsymbol{x_i})} = \frac{3}{1+\sqrt{5}} \approx 0.927$. Therefore, $\lim_{n\to\infty} \frac{\rho(n)}{n} > 0.927$. This was further developed by Franek and Yang [29].

This lower bound was improved by Matsubara, Kusano, Ishino, Bannai, and Shinohara [38]. They gave a string of length 184973, with 174697 runs, demonstrating $\rho(n) \geq 0.9444459n$ for $n = 184973$. They then show that multiple copies of this run-dense string can be concatenated together, establishing that $\lim_{n\to\infty} \frac{\rho(n)}{n} \geq \frac{174719}{184973} > 0.94456488$. This approach has since been used by Puglisi and Simpson [39] who give a seed string of length 29196442 which, when manipulated a similar way, establishes $\lim_{n\to\infty} \frac{\rho(n)}{n} \geq 0.944575$.

1.6 Algorithms to find all runs in a string

Main [37] gives an algorithm which finds all leftmost runs. The algorithm runs in linear time with the assumption you have the Lempel-Ziv factorization of the string, also called the *s*-factorization. The Lempel-Ziv factorization can be found in linear time, as shown by Crochemore, Ilie, and Smyth [8]. The runs found by Main's algorithm correspond to the *Class II* runs as described by Smyth [46]. Class II runs are those runs which start and end in different factors of the Lempel-Ziv factorization. Conversely, *Class I* runs start and end in the same factor. The leftmost runs are a subset of the Class II runs in a string.

Kolpakov and Kucherov [35] give a linear time algorithm to compute all

runs in a string, which they describe as a modification of Main's algorithm. They add an additional step to the algorithm in which they find all the Class I runs, and then merge the results. Puglisi, Simpson, and Smyth [41] comment that while it requires linear time, this algorithm "requires significant algorithmic machinery and working memory, and is thus not suitable for very long (for instance, genome-sized) strings."

Crochemore's partitioning algorithm [6], originally designed to find maximal repetitions has been modified to also find runs and primitively rooted distinct squares [24, 25, 26]. While the modified Crochemore's algorithm runs in $O(n \log n)$ time, Crochemore's approach lends itself more easily to parallelization compared to the algorithm of Kolpakov and Kucherov.

The run-times of implementations of various run-finding algorithms were compared by Franek and Fuller [22].

1.7 Special strings

Fibonacci strings are a class of strings known to be extremely repetition-dense. These strings are often an example of the worst case scenario in the testing of various pattern matching algorithms. As such, they have been well-studied with respect to the repetitions and runs they contain.

The finite Fibonacci strings are defined recursively: $F_0 = b$, $F_1 = a$, and $F_n = F_{n-1}F_{n-2}$ $(n \geq 2)$ [6, 34, 36]. The length of finite Fibonacci string F_n is f_n, the nth Fibonacci number [36]. Note that by the definition of the Fibonacci string F_n, every shorter Fibonacci string F_i, $1 \leq i < n$ is a prefix of F_n. Therefore, the infinite Fibonacci string, F, also simply called *the Fibonacci string*, is the infinite

string with every finite Fibonacci string F_n for $n \geq 1$ as a prefix [31].

Fraenkel and Simpson [20] show that the number of primitively rooted maximal repetitions in finite Fibonacci strings is proportional to $n \log n$. Iliopoulos, Moore, and Smyth [31] show the number of runs in finite Fibonacci strings is proportional to n, and Kolpakov and Kucherov [34, 36] give an exact formula for the number of runs in a Fibonacci string: $r(F_n) = 2f_{n-2} - 3$.

The Fibonacci string is a special case of Sturmian string. A Sturmian string is an infinite string defined on a binary alphabet such that it contains exactly $k+1$ distinct substrings of length k. Franek, Karaman, and Smyth [27] show that the maximum number of runs in any finite prefix of a Sturmian string is proportional to its length.

1.8 Related investigations

Repetitions and runs are by no means the only type of repetitive structure which have been investigated. Here we outline some of these different types of repeats and give an overview of some known results.

1.8.1 Sum of exponents of runs

Let $\mu(n) = \max\{$ sum of exponents of runs in $\boldsymbol{x} \mid n(\boldsymbol{x}) = n \}$, that is, the maximum sum of the exponents of the runs in a string, considered over all strings of length n.

Kolpakov and Kucherov [35] show that $\mu(n)$ is linear in order to prove the linearity of the number of runs. Rytter [42] doesn't give a constant for the bound of the sum of exponents, remarking only that it is "not satisfactory". He

does comment that it appears that $\mu(n) \leq 2n$, and suggests that his proof that $\lim_{n\to\infty} \frac{\rho(n)}{n} \leq 5$ could be re-written to give an upper bound on the sum of exponents. Crochemore and Ilie [7] suggest that such an approach gives a bound of around 25. Crochemore and Ilie [7] further show that $\lim_{n\to\infty} \frac{\mu(n)}{n} < 5.6$, and suggest their proof could be improved to 2.9 or further. They also show that when only considering those runs with maximum exponent 4, $\lim_{n\to\infty} \frac{\mu^{e\leq 4}(n)}{n} \leq 2$.

Conjecture 1.10: [**42**] *The sum of exponents is at most* $2n$*. That is, for all* $n > 1$*,* $\mu(n) \leq 2n$*.*

An area for future research involves extending the introduction of the number of distinct symbols parameter to the maximum sum of exponents of runs in a string, and investigating the properties of $\mu_d(n)$.

1.8.2 Distinct squares

Let $s(\boldsymbol{x})$ be the number of primitively rooted distinct squares in string $\boldsymbol{x}$, and $\sigma(n)$ be the maximum $s(\boldsymbol{x})$ over all $\boldsymbol{x}$ of length n

While it is known that the number of distinct squares in a string is at most $2n$ [19], it is conjectured to be at most n [36].

Conjecture 1.11: Maximum Number of Distinct Squares [36] *The maximum number of primitively rooted distinct squares in a string of length* n *is limited by* n*. That is,* $\sigma(n) \leq n$ *for all* $n > 0$*.*

There are many similarities between the maximum number of runs and the maximum number of distinct squares in strings, beyond just the conjectured bound. Kolpakov and Kucherov [36] suggest that $\sigma(n) \leq \rho(n)$ for all n, based on

experimental results. However, they found that in Fibonacci strings, the number of runs is exactly one less than the number of distinct squares. They further suggest that there isn't a connection between the values of $\rho(n)$ and $\sigma(n)$, despite them being similar, though they do wonder if the fact that $r(F_n) = \sigma(F_n) - 1$ is a coincidence or "has some combinatorial explanation".

The similarities between distinct symbols and runs is reinforced by the work of Deza, Franek, and Jiang [15]. Just as we introduce the parameter d and consider $\rho_d(n)$, they have looked at the properties of the function $\sigma_d(n)$, the maximum number of distinct squares over all strings of length n with exactly d distinct symbols. They suggest, much as we do for runs, that the d value is integral to understanding the nature of the maximum number of distinct squares, and propose that $\sigma_d(n) \leq n - d$. We elaborate on some further interrelations between run-maximality and square-maximality in Chapter 7.1.

Conjecture 1.12: **[15]** *The maximum number of distinct primitively rooted squares in a (d, n)-string is limited by $n - d$. That is, $\sigma_d(n) \leq n - d$ for all $2 \leq d \leq n$.*

1.8.3 Permuted repetitions

An *Abelian square*, introduced by Erdős [16], is a substring $\boldsymbol{uv}$ such that the symbols of $\boldsymbol{u}$ can be permuted in such a way as to obtain $\boldsymbol{v}$. That is, each symbol in $\boldsymbol{u}$ must occur the same number of times in $\boldsymbol{v}$, and vice versa. Thus, *aababa* is an Abelian square, as both *aab* and *aba* have two *a*'s and one *b*. This concept is extended by Cummings and Smyth [12] to *weak repetitions* $\boldsymbol{u_1 u_2 \ldots u_k}$ such that each $\boldsymbol{u_i u_{i+1}}$ for $1 \leq i < k$ is an Abelian square. Fici *et al.* [17] extend this concept to the idea of an *Abelian period*, which adds a head and tail to the beginning and

end of a weak repetition. The head is a substring such that each symbol which appears in the head appears at most as often as it does in $\boldsymbol{u_1}$. The tail is defined analogously. Constantinescu and Ilie [5] extend Fine and Wilf's periodicity results [18] from repetitions to Abelian periods.

1.9 Outline of thesis

Having presented an overview of the history of the problem and some related work, we now proceed with the main body of the thesis. First we introduce the $(d, n-d)$-table which we use to present the values of $\rho_d(n)$. Here we include some basic properties of the $\rho_d(n)$ function, and show how these properties propagate values through the table. Then we then proceed to discuss some structural properties of run-maximal strings in the form of core vectors and r-coverings, and introduce the concept of $\rho_d^-(n)$-density. Then we describe our computational approach to finding run-maximal strings which we used to compute new $\rho_d(n)$ values. Next we describe additional structural insights which can be used to show $\rho_d(n) \leq n - d$ holds for larger values of n and d, but cannot provide actual $\rho_d(n)$ values. Further work may be able to extend these restrictions such that it can be shown a counter-example to the maximum number of runs conjecture can never exist. We conclude with some opportunities for future work.

Chapter 2

The $(d, n-d)$-table

We present the values of $\rho_d(n)$ in a tabular form, which we term the $(d, n-d)$-table, so named because the rows of the table are indexed by d, and the columns are indexed by $n-d$. Each cell contains the corresponding $\rho_d(n)$ value. This presentation gives a way to see how properties of the $\rho_d(n)$ function propagate values through the table and to visualize recursions. We present the upper-left corner of the $(d, n-d)$-table in Table 2.1.

The $(d, n-d)$-table for $\rho_d(n)$ values was originally inspired by work on the largest possible diameter $\Delta(d, n)$ of a d-dimensional bounded polytope with n facets. The common presentation of $\Delta(d, n)$ values is in a table where again the rows are indexed by d, and the columns are indexed by $n-d$. The Hirsch conjecture states that $\Delta(d, n) \leq n-d$, though this has recently been disproved by Santos [44] who found a counter-example with a 43-dimensional polytope with 86 facets. The use of the $(d, n-d)$-table for the presentation of $\rho_d(n)$ values first appeared in [13].

The benefit of the table comes from the easy visualization it gives of prop-

Table 2.1: Values for $\rho_d(n)$ with $1 \leq d \leq 15$ and $1 \leq n-d \leq 15$. For more values, see [1].

		$n-d$															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	1	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	.
	2	1	**2**	2	3	4	5	5	6	7	8	8	10	10	11	12	.
	3	1	2	**3**	3	4	5	6	6	7	8	9	10	11	11	12	.
	4	1	2	3	**4**	4	5	6	7	7	8	9	10	11	12	12	.
	5	1	2	3	4	**5**	5	6	7	8	8	9	10	11	12	13	.
	6	1	2	3	4	5	**6**	6	7	8	9	9	10	11	12	13	.
	7	1	2	3	4	5	6	**7**	7	8	9	10	10	11	12	13	.
d	8	1	2	3	4	5	6	7	**8**	8	9	10	11	11	12	13	.
	9	1	2	3	4	5	6	7	8	**9**	9	10	11	12	12	13	.
	10	1	2	3	4	5	6	7	8	9	**10**	10	11	12	13	13	.
	11	1	2	3	4	5	6	7	8	9	10	**11**	11	12	13	14	.
	12	1	2	3	4	5	6	7	8	9	10	11	**12**	12	13	14	.
	13	1	2	3	4	5	6	7	8	9	10	11	12	**13**	13	14	.
	14	1	2	3	4	5	6	7	8	9	10	11	12	13	**14**	14	.
	15	1	2	3	4	5	6	7	8	9	10	11	12	13	14	**15**	.
	16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

erties of $\rho_d(n)$. While these properties exist independently of the presentation style, the $(d, n-d)$-table provides a useful tool for discussion. We now proceed to define some terminology of the $(d, n-d)$-table and present some properties it exhibits.

When we move to the right or down, the values of the table are non-decreasing.

Property 2.1: [13] *For $2 \leq d \leq n$, $\rho_d(n) \leq \rho_d(n+1)$. That is, values are non-decreasing along a row from left to right.*

Proof. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string. Then $\boldsymbol{x}[1]\boldsymbol{x}$ is a $(d, n+1)$-string and $\rho_d(n) = r(\boldsymbol{x}) \leq r(\boldsymbol{x}[1]\boldsymbol{x}) \leq \rho_d(n+1)$. □

Property 2.2: **[13]** *For* $1 \leq d \leq n$, $\rho_d(n) \leq \rho_{d+1}(n+1)$. *That is, values are non-decreasing down a column.*

Proof. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string, and let z be a symbol such that $z \notin \mathcal{A}(\boldsymbol{x})$. Then $\boldsymbol{x}z$ is a $(d+1, n+1)$-string and $\rho_d(n) = r(\boldsymbol{x}) = r(\boldsymbol{x}z) \leq \rho_{d+1}(n+1)$. □

While the value may remain constant in a single step to the right, it is guaranteed to increase whenever the length is increased by at least two.

Property 2.3: **[29]** *For* $2 \leq d \leq n$, $\rho_d(n) < \rho_d(n+2)$. *That is, values are increasing with a step of two to the right.*

Proof. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string, and let $\boldsymbol{x}[n] = a$. Then $\boldsymbol{x}bb$ is a $(d, n+2)$-string, and $\rho_d(n) = r(\boldsymbol{x}) < r(\boldsymbol{x}bb) \leq \rho_d(n+2)$. □

Moving on a diagonal from the upper-left to the lower-right, the values are increasing.

Property 2.4: **[13]** *For* $1 \leq d \leq n$, $\rho_d(n) < \rho_{d+1}(n+2)$. *That is, values are increasing down a diagonal to the lower right.*

Proof. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string, and let z be a symbol such that $z \notin \mathcal{A}(\boldsymbol{x})$. Then $\boldsymbol{x}zz$ is a $(d+1, n+2)$-string and $\rho_d(n) = r(\boldsymbol{x}) < r(\boldsymbol{x}zz) \leq \rho_{d+1}(n+2)$. □

The ***main diagonal*** is the diagonal of the $(d, n-d)$-table where $n = 2d$. This is a critical region of the $(d, n-d)$-table. We can show that every value below the main diagonal in the $(d, n-d)$-table is equal to the value on the main diagonal directly above it. In other words, the values on and below the main diagonal within a column are constant.

Before we do this though, we must first introduce the concept of a *safe position*.

Definition 2.5: *A **safe** position in a string $\boldsymbol{x}$ is one which, when removed from $\boldsymbol{x}$, does not result in two runs being merged into one in the resulting new string.*

A safe position does not ensure that the number of runs will not change when that position is removed, only that no runs will be lost through being merged; runs may still be destroyed by having a position in the core removed. Safe positions are important in that they may be removed from a string while only affecting the runs which contain them. When the position of a symbol is unambiguous, we may thus refer to a *safe symbol* rather than to its position – for instance we can talk about a safe singleton, or about the first member of a pair being safe, etc. Conversely, an ***unsafe position*** is one which, when removed from its string, causes at least two runs to merge. A result of this is that an unsafe singleton must have a pair of identical squares immediately to the left and right of the unsafe position, namely, $\boldsymbol{x} = \boldsymbol{x_1 uucuux_2}$, where c is the unsafe position and $\boldsymbol{u}$ is a non-empty string. A consequence of this is given in Lemma 2.6.

Lemma 2.6: [4] *If a string $\boldsymbol{x}$ consists only of singletons, pairs, and 3-tuples, then every position is safe.*

Proof. By definition, a position is unsafe when a run is lost through two runs merging when that position is removed. Therefore, if a position is unsafe, there is a trailing square of some run immediately to its left with the same generator as the leading square of a run immediately to its right. Any symbol in a square must appear twice, and as there are two squares with the same generator, some symbol must appear four times, a contradiction. □

In the following lemmas we will need to relabel all occurrences of a symbol in a string or substring. Let $\boldsymbol{x}_b^a$ denote the string $\boldsymbol{x}$, in which all occurrences of a are replaced by b, and vice versa. Note that permuting the symbols of a string does not change the number of runs in the string, that is, $r(\boldsymbol{x}) = r(\boldsymbol{x}_b^a)$, for any a and b, no matter if a and b appear in $\boldsymbol{x}$ or not.

Lemma 2.7: [4] *There exists a run-maximal (d, n)-string with no unsafe singletons for all $2 \leq d \leq n$.*

Proof. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string. We will show that one of the following four conditions must hold:

(i) $\boldsymbol{x}$ has no singletons, or

(ii) $\boldsymbol{x}$ has exactly one singleton which is safe, or

(iii) $\boldsymbol{x}$ has exactly one singleton which is unsafe, and there exists another run-maximal (d, n)-string $\boldsymbol{x'}$ such that $\boldsymbol{x'}$ has no singletons, or

(iv) $\boldsymbol{x}$ has more than one singleton, all of which are safe.

In case (i), there are no singletons, and therefore no unsafe singletons. Similarly, in case (ii), if there is only one singleton which is safe, there are no unsafe singletons. Assume then that $\boldsymbol{x}$ has some unsafe singletons.

Consider case (iii) in which $\boldsymbol{x}$ has exactly one singleton, c, which is unsafe. We show that we can perform an operation on $\boldsymbol{x}$ yielding a new string with no singletons but the same number of runs. Due to the unsafe singleton, the string must have the form $\boldsymbol{x} = \boldsymbol{u}[a\boldsymbol{v}a\boldsymbol{v}]c[a\boldsymbol{v}a\boldsymbol{v}]\boldsymbol{w}$, where $a \in \mathcal{A}(\boldsymbol{x}) - \{c\}$. We have denoted the squares to either side of the singleton with square brackets for clarity.

Let $\boldsymbol{x'} = \boldsymbol{u}a\boldsymbol{v}a\boldsymbol{v}(ca\boldsymbol{v}a\boldsymbol{v}\boldsymbol{w})_c^a = \boldsymbol{u}a\boldsymbol{v}a\boldsymbol{v}ac\boldsymbol{v}_c^a c\boldsymbol{v}_c^a\boldsymbol{w}_c^a$. Then, $\boldsymbol{x'}$ is still a (d, n)-string, and $r(\boldsymbol{x'}) \geq r(\boldsymbol{x})$, so $\boldsymbol{x'}$ is run-maximal. There are now two instances of the symbol c, so it is no longer a singleton. The only symbol which occurs fewer times in $\boldsymbol{x'}$ than in $\boldsymbol{x}$ is a, and it still appears at least twice in $\boldsymbol{x'}$.

Finally, consider case (iv), that $\boldsymbol{x}$ has at least 2 singletons. We will prove that all the singletons in the string must be safe by contradiction. Assume $\boldsymbol{x}$ has singletons c_1 and c_2, of which at least one is unsafe. Without loss of generality, we can assume c_1 is unsafe and occurs before c_2. Let us break the string into three parts to simplify the discussion: $\boldsymbol{x} = \boldsymbol{x_1}c_1\boldsymbol{x_2}c_2\boldsymbol{x_3}$. As c_1 is unsafe, we have $\boldsymbol{x_1} = \boldsymbol{u}[a\boldsymbol{v}a\boldsymbol{v}]$, and $\boldsymbol{x_2} = [a\boldsymbol{v}a\boldsymbol{v}]w$. We create a new string $\boldsymbol{x'} = \boldsymbol{x_1}\boldsymbol{x_2}_{c_1}^a c_2 c_2 \boldsymbol{x_3}$. No runs were lost through merging in the concatenation of $\boldsymbol{x_1}$ and $\boldsymbol{x_2}_{c_1}^a = c_1\boldsymbol{v}_{c_1}^a c_1 \boldsymbol{v}_{c_1}^a \boldsymbol{w}_{c_1}^a$ as c_1 does not appear in $\boldsymbol{x_1}$. As no runs cross the original occurrence of c_2, no runs are damaged by the insertion of a second copy of c_2. No new symbols were introduced, nor were any symbols lost, so $d(\boldsymbol{x}) = d(\boldsymbol{x'})$. However, $r(\boldsymbol{x}) = r(\boldsymbol{x'}) - 1$, as a new run c_2c_2 was introduced, contradicting the choice of $\boldsymbol{x}$ as a run-maximal string. Therefore, if $\boldsymbol{x}$ has more than one singleton, they must all be safe. □

We are now in a position to show that values are constant in a column under the main diagonal.

Property 2.8: [13] *We have* $\rho_d(n) = \rho_{n-d}(2n - 2d)$ *for* $2 \leq d \leq n < 2d$.

Proof. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string, where $2 \leq d \leq n < 2d$. Since $n < 2d$, $\boldsymbol{x}$ must have a singleton. By Lemma 2.7 we can assume that singleton is safe. Therefore, we can remove this safe singleton, yielding a new $(d-1, n-1)$-string $\boldsymbol{y}$ and so $\rho_d(n) = r(\boldsymbol{x}) = r(\boldsymbol{y}) \leq \rho_{d-1}(n-1)$. By Property 2.2, $\rho_d(n) \leq \rho_{d+1}(n+1)$ for $2 \leq d \leq n$, so $\rho_{d-1}(n-1) = \rho_d(n)$. □

We can combine the fact that columns are constant below the diagonal, Property 2.8, with the fact that columns are non-decreasing as you go down, Property 2.2, to show that the maximum value in a column can be found on the main diagonal.

Property 2.9: **[13]** *We have,* $\rho_d(2d) = \max_{\epsilon > -d}\{\ \rho_{d+\epsilon}(2d+\epsilon)\ \}$. *That is, the maximum value in a column is found on the main diagonal.*

Proof. The proof follows directly from Property 2.8 and Property 2.2. □

A corollary is that if there is a counter-example to the conjecture that $\rho_d(n) \leq n - d$, then there is a counter-example on the main diagonal.

Corollary 2.10: *If* $\rho_d(n) > n - d$ *for some* $2 \leq d \leq n$*, then there exists some* d_0 *such that* $\rho_{d_0}(2d_0) > d_0$.

We refer to the propagation of a counter-example to the main diagonal as being "pushed up" or "pushed down". Let a column i of the $(d, n-d)$-table be the column containing the values $\rho_{i+\epsilon}(2i+\epsilon)$ for $\epsilon > -i$. Therefore we can consider a column i to be a counter-example to the conjecture if $\rho_i(2i) > i$. To say that column i satisfies the conjecture that $\rho_d(n) \leq n - d$ means that $\rho_i(2i) = i$, and so by Property 2.9, every $\rho_{i+\epsilon}(2i+\epsilon) \leq i$ for $\epsilon > -i$.

If a given column of the $(d, n-d)$ table were to prove to violate the conjecture that $\rho_d(n) \leq n - d$, so would all subsequent columns.

Property 2.11: *If column* i *of the* $(d, n-d)$*-table has* $\rho_i(2i) > i$*, every column* $i' > i$ *also has* $\rho_{i'}(2i') > i'$.

Proof. If column i is a counter-example, then $\rho_i(2i) > i$. By Property 2.4, $\rho_{i+1}(2(i+1)) \geq \rho_i(2i) + 1 > i + 1$. By induction, this holds for every $i' > i$. □

We extend Property 2.8 to bound the behaviour of the entries in the immediate neighbourhood above the main diagonal in the $(d, n-d)$ table. Proposition 2.12 establishes that the difference between $\rho_d(2d)$ and $\rho_{d-1}(2d-1)$ is at most 1. In addition, the difference is 1 if and only if every run-maximal $(d, 2d)$-string consists entirely of pairs; otherwise, the difference is 0.

Note that in Propositions 2.12 and 2.13, the symbol z is taken to represent the "last" symbol in the string, and is not meant to imply that the strings have $d = 26$.

Proposition 2.12: **[4]** *We have* $\rho_d(2d) \leq \rho_{d-1}(2d-1) + 1$ *for* $d \geq 3$.

Proof. Let $\boldsymbol{x}$ be a run-maximal $(d, 2d)$-string, and assume by Lemma 2.7 that it has no unsafe singletons. Consider two cases: $\boldsymbol{x}$ has a singleton and $\boldsymbol{x}$ does not have a singleton.

(i) Assume $\boldsymbol{x}$ has a singleton. Since it is safe, we can remove it and form a new $(d-1, 2d-1)$-string $\boldsymbol{y}$ without destroying any runs. Therefore, $\rho_d(2d) = r(\boldsymbol{x}) \leq r(\boldsymbol{y}) \leq \rho_{d-1}(2d-1)$, and as by Property 2.2 $\rho_{d-1}(2d-1) \leq \rho_d(2d)$, we have $\rho_{d-1}(2d-1) = \rho_d(2d)$ when there exists a run-maximal $(d, 2d)$-string with a singleton.

(ii) If $\boldsymbol{x}$ does not have a singleton, then it consists entirely of pairs. As each pair can be in at most one run, the total runs are maximized by adjacent pairs, and so $r(\boldsymbol{x}) = d$ with $\boldsymbol{x} = aabbcc\ldots zz$. We may transmute the string by changing $\boldsymbol{x}[2]$ to a b and removing $\boldsymbol{x}[1]$, giving the $(d-1, 2d-1)$-string $\boldsymbol{y} = bbbcc\ldots zz$ and $\rho_d(2d) = r(\boldsymbol{x}) = r(\boldsymbol{y}) + 1 \leq \rho_{d-1}(2d-1) + 1$. □

We have seen that the difference between the value on the main diagonal and the first value above the main diagonal is at most 1. Proposition 2.13 estab-

lishes that the three entries immediately above the main diagonal are identical.

Proposition 2.13: [4] *We have* $\rho_{d-1}(2d-1) = \rho_{d-2}(2d-2) = \rho_{d-3}(2d-3)$ *for* $d \geq 5$.

Proof. We prove this equality in two stages: *(i)* showing $\rho_{d-1}(2d-1) = \rho_{d-2}(2d-2)$, and *(ii)* showing $\rho_{d-2}(2d-2) = \rho_{d-3}(2d-3)$.

(i) Let $\boldsymbol{x}$ be a run-maximal $(d-1, 2d-1)$-string. By Lemma 2.7 we can assume that if $\boldsymbol{x}$ has a singleton, it is safe. Therefore, if $\boldsymbol{x}$ has a singleton, we can remove it obtaining a new $(d-2, 2d-2)$-string $\boldsymbol{y}$ such that $\rho_{d-1}(2d-1) = r(\boldsymbol{x}) \leq r(\boldsymbol{y}) \leq \rho_{d-2}(2d-2)$. As $\rho_{d-2}(2d-2) \leq \rho_{d-1}(2d-1)$ by Property 2.2, we know $\rho_{d-1}(2d-1) = \rho_{d-2}(2d-2)$.

Otherwise, if $\boldsymbol{x}$ has no singletons, it consists of pairs and one 3-tuple, and thus, by Lemma 2.6, all positions are safe. Each pair can be involved in at most one run, and a 3-tuple in two runs. However, there is no way in which to use a pair to break up a 3-tuple without "using up" the run for that square. For example, the substring *aabab* involves the 3-tuple in two runs, but one of those is the same run that involves the pair of *b*'s. Therefore, we can maximize the number of runs in the string with one of three structures: $\boldsymbol{x_1} = aaabbcc\ldots zz$, $\boldsymbol{x_2} = aababcc\ldots zz$, or $\boldsymbol{x_3} = ababbcc\ldots zz$. The string $\boldsymbol{x_1}$ may be transmuted to form the string $ccbbcc\ldots zz$, a $(d-2, 2d-2)$-string, without destroying any runs. It follows that $\rho_{d-1}(2d-1) = r(\boldsymbol{x}) \leq r(\boldsymbol{y}) \leq \rho_{d-2}(2d-2)$, and so $\rho_{d-1}(2d-1) = \rho_{d-2}(2d-2)$.

(ii) Let $\boldsymbol{x}$ be a run-maximal $(d-2, 2d-2)$-string. Again, if $\boldsymbol{x}$ has a singleton, we can assume by Lemma 2.7 that it is safe. As it is safe, we can then form

a $(d-3, 2d-3)$-string $\boldsymbol{y}$ by removing the singleton, and $\rho_{d-2}(2d-2) = r(\boldsymbol{x}) \leq r(\boldsymbol{y}) \leq \rho_{d-3}(2d-3)$, so $\rho_{d-2}(2d-2) = \rho_{d-3}(2d-3)$.

Therefore we consider the case that $\boldsymbol{x}$ does not have a singleton. We show then that $r(\boldsymbol{x}) = d-1$. Consider the two cases:

(a) The string $\boldsymbol{x}$ consists of two 3-tuples and several pairs. The most runs which may be obtained in such a string, after grouping the pairs at the end of the string, is through the arrangement $aababbccdd \ldots zz$. In this case, there are $d-4$ runs from the pairs, and 3 runs from the 3-tuples, giving a total of $d-1$ runs.

(b) The string $\boldsymbol{x}$ consists of a quadruple and several pairs. There are three structures which yield a run-maximal string,assuming all pairs but the b's have been grouped at the end: $\boldsymbol{x_1} = aabbaaccdd \ldots zz$, $\boldsymbol{x_2} = aabaabccdd \ldots zz$, and $\boldsymbol{x_3} = abbabbccdd \ldots zz$. In each case, there are $d-4$ runs involving symbols c through z, and 3 runs involving the symbols a and b, again giving a total of $d-1$ runs.

□

Chapter 3

Structural properties of run-maximal strings

Before we develop the structural properties of run-maximal strings, we will first explain why structural properties are desirable and outline their eventual use in order to provide some context. Our aim is to calculate $\rho_d(n)$ values for as large values of d and n as possible. If we were to consider every (d, n)-string in order to find the maximum, we are faced with an exponentially-growing pool to search. Therefore, as much as possible, we aim to avoid generating strings that cannot be run-maximal. Furthermore, once we have found a (d, n)-string with r runs in it, we no longer need to search for strings with r runs, but only those that could have $r + 1$ runs.

Run-maximal strings are difficult to generate for two reasons. First, there is little to be gained from iterative efforts to find run-maximal strings, at least on the traditional binary case. That is, knowing all the run-maximal $(2, n')$-strings for $n' < n$ typically provides little insight into the run-maximal $(2, n)$-strings.

Second, run-maximal strings do not demonstrate readily apparent structures. In this chapter, we present some structural properties of run-maximal strings. In the Chapter 4, we illustrate how these structures may be exploited in order to speed the generation of run-maximal strings.

Definition 3.1: [2] *Let $k_i(\boldsymbol{x})$ be the number of cores in $\boldsymbol{x}$ containing the position i. Given a string $\boldsymbol{x}$, the vector $k(\boldsymbol{x}) = [k_1(\boldsymbol{x}), \ldots, k_n(\boldsymbol{x})]$ is referred to as the* ***core vector*** *of $\boldsymbol{x}$.*

If we want to build strings that are run-dense, it makes sense that we would want to create strings in which every position is in the core of some run. Such a core vector, in which every element $k_i(\boldsymbol{x}) > 0$ will be termed a ***strictly positive core vector***. However, building such a string is not a straight-forward matter. Assume that we have partially constructed a string such that every position is in the core of a run. As we extend the string, we may extend one of the runs, thus shrinking its core. This can be illustrated by the partial string $\boldsymbol{x_1} = abab$ in which every position is in the core of a run. If the next symbol added is another a, then $\boldsymbol{x_2} = ababa$ in which $\boldsymbol{x_2}[1]$ is not in the core of a run. In other cases, extending an existing run may result in a position still being in a core. Consider $\boldsymbol{x_1} = aabaab$ which is again extended by a single a. The resulting string $\boldsymbol{x_2} = aabaaba$ still has position 1 in the core of a run, though not two as it was previously.

We therefore introduce the r-cover as a generalization of a strictly positive core vector. An r-cover is a specific set of leading squares of the runs in a string.

Definition 3.2: R-cover [23] *An* ***r-cover*** *of a string $\boldsymbol{x}$ is a sequence of primitively rooted squares $\{\ S_i = (s_i, p_i, 2) \mid 1 \leq i \leq m\ \}$ so that*

(1) *no S_i, $1 \leq i \leq m$, is left-shiftable;*

(2) $s_i < s_{i+1} \leq s_i + 2p_i < s_{i+1} + 2p_{i+1}$ *for any* $1 \leq i < m$*, that is, two consecutive squares are either adjacent or overlap without one containing the other;*

(3) $\bigcup_{1 \leq i \leq m} S_i = \boldsymbol{x}$*;*

(4) *for any run* $R = (s, p, e, t)$ *of* $\boldsymbol{x}$ *there is an* S_i *with* $1 \leq i \leq m$ *containing the leading square of the run* R*, that is,* $s_i \leq s < s + 2p \leq s_i + 2p_i$*.*

A string that has an r-cover is referred to as ***r-covered***. It follows that a string with a strictly positive core vector has an r-cover.

Lemma 3.3: *If a run-maximal* (d, n)*-string has a strictly positive core vector, then the string has an r-cover [2].*

Proof. We build an r-cover by induction:

Since $k_1(x) \neq 0$, position 1 is in at least one core, and so there must be at least one run starting at position 1. Among all runs starting at position 1, set S_1 to the leading square of the run with the largest period.

Let the inductive hypothesis be that we have built the r-cover up to element t: $\{S_i = (s_i, p_i, 2) \mid 1 \leq i \leq t\}$. If $\bigcup_{1 \leq i \leq t} S_i = \boldsymbol{x}$, we are done. Otherwise $\bigcup_{1 \leq i \leq t} S_i = \boldsymbol{x}[1..s_t + 2p_t - 1]$. Since $k_{s_t + 2p_t}(\boldsymbol{x}) \neq 0$, there is at least one run $(s, p, 2)$ in $\boldsymbol{x}$ such that $s \leq s_t + 2p_t \leq s + 2p - 1$. From all such runs chose the set of runs with the leftmost starting position, and among them choose the one with the largest period. Set S_{t+1} to the leading square of the chosen run.

One can verify that all the conditions of Definition 3.2 are satisfied and that we have built an r-cover of $\boldsymbol{x}$. □

The approach used in this proof can be extended to find an r-cover for any string, if it exists. That is, first choose the run that starts at position 1 with

the largest period. Then iteratively choose the next r-cover element as outlined in the proof of Lemma 3.3. If at any point before the string is covered there is no valid run to choose, then the string does not have an r-cover [3]. We give the pseudocode for this in Algorithm 1.

Algorithm 1: Find the r-cover of $\boldsymbol{x}$, if it exists.

```
begin Find r-cover of x
    s_1 ← 1, p_1 ← 0
    foreach run (s, p, e, t) in x do
        if s = 1 and p > p_1 then
            p_1 ← p
    if p_1 = 0 then
        return failure
    m ← 1, uncovered ← s_1 + 2p_1
    while uncovered ≤ n do
        m ← m + 1, s_m ← n + 1, p_m ← 0
        foreach run (s, p, e, t) in x do
            if s ≤ uncovered and s + 2p > uncovered and s ≤ s_m then
                if p > p_m then
                    s_m ← s, p_m ← p
        if s_m > n then
            return failure
        uncovered ← s_m + 2p_m
    return { (s_1, p_1, 2), ... (s_m, p_m, 2) }
```

A nice property of r-covers is that for a given string, if the string has an r-cover, then that r-cover is unique.

Lemma 3.4: [2] *The r-cover of an r-covered string is unique.*

Proof. Let us assume that we have two different r-covers of $\boldsymbol{x}$, $\{ S_i \mid 1 \leq i \leq m \}$ and $\{ S'_j \mid 1 \leq j \leq k \}$. We shall prove by induction that they are identical.

By Definition 3.2 (4), S_1 is a substring of S'_1 and, by the same argument, S'_1 is a substring of S_1, and thus $S_1 = S'_1$. Let the induction hypothesis be $S_i = S'_i$ for $1 \leq i \leq t$. If $\bigcup_{1 \leq i \leq t} S_i = x$, we have $t = m = k$ and we are done. Otherwise consider S_{t+1}. By Definition 3.2 (4), there is S'_v so that S_{t+1} is a substring of S'_v and $v > t$. We need to show that $v = t+1$. If not, then S'_{t+1} is a substring of $\bigcup_{1 \leq i \leq t+1} S_i$ as otherwise S'_{t+1} would contain S_{t+1}, contradicting $v \neq t+1$. Since S'_{t+1} is not a substring of $\bigcup_{1 \leq i \leq t} S_i$, then S'_{t+1} is a substring of S_{t+1}, which in turn is a substring of S'_v, a contradiction. Therefore, S_{t+1} is a substring of S'_{t+1}. Similarly, S'_{t+1} is a substring of S_{t+1} and so $S_{t+1} = S'_{t+1}$, which completes the induction. □

If a string does not have an r-cover, then it must have at last one ***weak point***.

Definition 3.5: [3] *A **weak point** is a position in a string which is not in the leading square of any run.*

When considering run-maximal strings of length n that are run-maximal over all alphabets, it can be shown that they contain at most one weak point.

Lemma 3.6: [4] *Let $\boldsymbol{x}$ be a run-maximal n-string. Then $\boldsymbol{x}$ has at most one weak point.*

Proof. Assume that the run-maximal n-string $\boldsymbol{x}$ has at least two weak points at positions i and j, and that $i < j$. The string has the form $\boldsymbol{x} = \boldsymbol{x_1}\boldsymbol{x}[i]\boldsymbol{x_2}\boldsymbol{x}[j]\boldsymbol{x_3}$, where $\boldsymbol{x_1} = \boldsymbol{x}[1..i-1]$, $\boldsymbol{x_2} = \boldsymbol{x}[i+1..j-1]$, and $\boldsymbol{x_3} = \boldsymbol{x}[j+1..n]$. Let c_1 and c_2 be two distinct symbols that do not occur in $\boldsymbol{x}$; $c_1 \neq c_2$, $c_1 \notin \mathcal{A}(\boldsymbol{x})$, $c_2 \notin \mathcal{A}(\boldsymbol{x})$. If $\boldsymbol{x_2} = \varepsilon$, then we have $r(\boldsymbol{x_1}c_1c_1\boldsymbol{x_3}) = r(\boldsymbol{x}) + 1$, contradicting the run-maximality

of $\boldsymbol{x}$. Otherwise, $\boldsymbol{x_2}$ is non-empty. In that case, let $\widetilde{\boldsymbol{x_2}}$ be $\boldsymbol{x_2}$ with every instance of $\boldsymbol{x}[i+1]$ replaced with c_1 and every instance of $\boldsymbol{x}[j-1]$ replaced with c_2. If $\boldsymbol{x}[i+1] = \boldsymbol{x}[j-1]$, then every instance of the symbol will be replaced with c_2. Then $r(\boldsymbol{x_1}\widetilde{\boldsymbol{x_2}}\boldsymbol{x_3}c_2c_2) = r(\boldsymbol{x}) + 1$, again contradicting the run-maximality of $\boldsymbol{x}$. Therefore, every string of length n which is run-maximal over all alphabets has at most one weak point. □

Lemma 3.6 is the best bound possible, as there are multiple run-maximal strings which contain a weak point. The only known case of a run-maximal string with a weak point occurring at a position other than 1 or n is when $n = 5$. The run-maximal strings *aabaa* and *aaabb* both have a weak point at position 3. No run-maximal $(2, n)$-strings for $6 \leq n \leq 66$ contain an interior weak point.

As the proof of Lemma 3.6 relies on changing the alphabet, it cannot be directly applied to run-maximal (d, n)-strings. However, we can bound the value of $\rho_d(n)$ if some run-maximal (d, n)-string has a weak point.

Lemma 3.7: [2] *If a run-maximal (d, n)-string $\boldsymbol{x}$ does not have an r-cover, then $\rho_d(n) \leq \rho_d(n_1) + \rho_d(n_2)$ for some $n_1, n_2 \geq 0$ such that $n_1 + n_2 = n-1$.*

Proof. Since $\boldsymbol{x}$ does not have an r-cover, there is a position i that is not in the core of any run. Let $\boldsymbol{x_1} = \boldsymbol{x}[1..i-1]$ and $\boldsymbol{x_2} = \boldsymbol{x}[i+1..n]$, with lengths n_1 and n_2 respectively. We consider two cases:

(a) If $\mathcal{A}(\boldsymbol{x}) = \mathcal{A}(\boldsymbol{x_1}) = \mathcal{A}(\boldsymbol{x_2})$, then $\rho_d(n) = r(\boldsymbol{x}) \leq r(\boldsymbol{x_1}) + r(\boldsymbol{x_2}) \leq \rho_d(n_1) + \rho_d(n_2)$.

(b) If $\mathcal{A}(\boldsymbol{x_1}) \neq \mathcal{A}(\boldsymbol{x_2})$, then without loss of generality, assume there is $\boldsymbol{c} \in \mathcal{A}(\boldsymbol{x_1}) \setminus \mathcal{A}(\boldsymbol{x_2})$. Permute the alphabet of $\boldsymbol{x_1}$ creating a new string $\widetilde{\boldsymbol{x_1}}$, so

that $\widetilde{\boldsymbol{x_1}}[i{-}1] = c$. Then $\widetilde{\boldsymbol{x_1}}$ and $\boldsymbol{x_2}$ can be concatenated into a string of length $n-1$ without merging any runs. Therefore, $\rho_d(n) = r(\boldsymbol{x}) \leq r(\widetilde{\boldsymbol{x_1}}) + r(\boldsymbol{x_2}) = r(\boldsymbol{x_1}) + r(\boldsymbol{x_2}) \leq \rho_d(n{-}1) \leq \rho_d(n)$, and so $\rho_d(n) = \rho_d(n{-}1)$.

Therefore, with the convention that $\rho_d(0) = 0$, $\rho_d(n) \leq \rho_d(n_1) + \rho_d(n_2)$ for some $n_1, n_2 \geq 0$ such that $n_1 + n_2 = n{-}1$. □

A basic property of r-covered strings is that they cannot contain a singleton.

Lemma 3.8: [2] *Every r-covered string is singleton-free.*

Proof. Let $\{\ S_j \mid 1 \leq j \leq m\ \}$ be the r-cover of $\boldsymbol{x}$. For any $1 \leq i \leq n$, $\boldsymbol{x}[i] \in S_t$ for some $1 \leq t \leq m$ by Definition 3.2 (3). Since S_t is a square, the symbol $\boldsymbol{x}[i]$ occurs in $\boldsymbol{x}$ at least twice. □

When considering a string that has a singleton though, we can provide a bound on the maximum number of runs it can contain.

Lemma 3.9: [2] *If a run-maximal (d, n)-string has a singleton, then either $\rho_d(n) = \rho_{d-1}(n{-}1)$ or $\rho_d(n) = \rho_d(n{-}1)$.*

Proof. For a given run-maximal (d, n)-string $\boldsymbol{x}$ there are three cases:

(a) The string $\boldsymbol{x}$ has a singleton at the end or the beginning. If it is at the end, then $\rho_d(n) = r(\boldsymbol{x}) = r(\boldsymbol{x}[1..n{-}1]) \leq \rho_{d-1}(n{-}1)$, as $\boldsymbol{x}[1..n{-}1]$ is a $(d{-}1, n{-}1)$-string. It follows that $\rho_d(n) = \rho_{d-1}(n{-}1)$. If the singleton is at the beginning, the proof is identical except for taking the substring $\boldsymbol{x}[2..n]$.

(b) The string $\boldsymbol{x}$ has a singleton in the middle at a position j and the alphabets of the two parts are different; that is, there is c so that either $c \in \mathcal{A}(\boldsymbol{x}[1..j{-}1]) \setminus$

$\mathcal{A}(\boldsymbol{x}[j{+}1..n])$ or $c \in \mathcal{A}(\boldsymbol{x}[j{+}1..n]) \setminus \mathcal{A}(\boldsymbol{x}[1..j{-}1])$. If $c \in \mathcal{A}(\boldsymbol{x}[j{+}1..n]) \setminus \mathcal{A}(\boldsymbol{x}[1..j{-}1])$, then create $\boldsymbol{x_1}$ by permuting the alphabet of $\boldsymbol{x}[j{+}1..n]$ so that c moves to the position $j{+}1$. This will not affect the number of runs so $r(\boldsymbol{x}) = r(\boldsymbol{x_1})$. Create $\boldsymbol{x_2}$ by moving the singleton $\boldsymbol{x}[j]$ to the end of the string. Again, the number of runs will not be affected so $r(\boldsymbol{x_1}) = r(\boldsymbol{x_2})$. Then $\boldsymbol{y} = \boldsymbol{x_2}[1..n{-}1]$ is a $(d{-}1, n{-}1)$-string and $\rho_d(n) = r(\boldsymbol{x}) = r(\boldsymbol{x_2}) = r(\boldsymbol{y}) \leq \rho_{d-1}(n{-}1) \leq \rho_d(n)$. The argument is similar if $c \in \mathcal{A}(\boldsymbol{x}[1..j{-}1]) \setminus \mathcal{A}(\boldsymbol{x}[j{+}1..n])$.

(c) The string $\boldsymbol{x}$ has a singleton c in the middle at a position j and the alphabets of the two parts are the same; that is, $\mathcal{A}(\boldsymbol{x}[j{+}1..n]) = \mathcal{A}(\boldsymbol{x}[1..j{-}1]) = \mathcal{A}(\boldsymbol{x}){-}\{c\}$. Replace all occurrences of $\boldsymbol{x}[j{+}1]$ in $\boldsymbol{x}[j{+}1..n]$ with the singleton c, producing $\boldsymbol{x_1}$. This will not affect any runs so $r(\boldsymbol{x}) = r(\boldsymbol{x_1})$. Moreover, $\mathcal{A}(\boldsymbol{x}) = \mathcal{A}(\boldsymbol{x_1})$. Create a string $\boldsymbol{x_2}$ by removing $\boldsymbol{x_1}[j]$. Since no runs are merged, $\boldsymbol{r}(x_1) = \boldsymbol{r}(x_2)$. Since $\mathcal{A}(\boldsymbol{x}) = \mathcal{A}(\boldsymbol{x_2})$, $\boldsymbol{x_2}$ is a $(d, n{-}1)$-string and thus $\rho_d(n) = r(\boldsymbol{x}) = r(\boldsymbol{x_2}) \leq \rho_d(n{-}1) \leq \rho_d(n)$.

Therefore, $\rho_d(n) = \rho_{d-1}(n{-}1)$ or $\rho_d(n) = \rho_d(n{-}1)$ when a run-maximal string has a singleton. □

A useful result related to Lemma 3.9 is that when a run-maximal (d, n)-string contains singletons, we can assume that every singleton is grouped together at the end of the string.

Lemma 3.10: *If there is a run-maximal (d, n)-string with singletons, then there is a run-maximal (d, n)-string with all the singletons at the end of the string.*

Proof. Let $\boldsymbol{x}$ have a singleton at position i and a symbol at position j, $i < j \leq n$, that is not a singleton. We show that we can transform this string either to remove

the singleton, or to move it to the end of the string without affecting the number of runs. Let $\boldsymbol{x} = \boldsymbol{x_1}\boldsymbol{x}[i]\boldsymbol{x_2}\boldsymbol{x_3}$, where $\boldsymbol{x_3}$ is composed entirely of singletons and $\forall c \in \mathcal{A}(\boldsymbol{x_3})$, $c \notin \mathcal{A}(\boldsymbol{x_1}\boldsymbol{x_2})$. That is, $\boldsymbol{x_3}$ is the substring composed of singletons that are already grouped at the end of the string. There are three cases to consider:

(a) If $i = 1$, so the singleton is at the beginning of the string, then $\boldsymbol{x_1} = \varepsilon$. The singleton cannot be involved in any runs and is safe, so it can be moved to the end without affecting the number of runs.

(b) If $\boldsymbol{x}$ has a singleton in the middle and $\mathcal{A}(\boldsymbol{x_1}) \neq \mathcal{A}(\boldsymbol{x_2})$, then as in case (b) of the proof of Lemma 3.9 let us assume without loss of generality that there exists a $c \in \mathcal{A}(\boldsymbol{x_2}) \setminus \mathcal{A}(\boldsymbol{x_1})$. We permute the symbols of $\boldsymbol{x_2}$ such that $\boldsymbol{x}[i+1] = c$ yielding $\widetilde{\boldsymbol{x_2}}$. We can then obtain $\boldsymbol{y} = \boldsymbol{x_1}\widetilde{\boldsymbol{x_2}}\boldsymbol{x_3}\boldsymbol{x}[i]$, and $r(\boldsymbol{y}) = r(\boldsymbol{x})$.

(c) If $\boldsymbol{x}$ has a singleton in the middle and $\mathcal{A}(\boldsymbol{x_1}) = \mathcal{A}(\boldsymbol{x_2})$, then we are able to remove the singleton from the string. Create $\widetilde{\boldsymbol{x_2}}$ from $\boldsymbol{x_2}$ such that every instance of $\boldsymbol{x}[i+1]$ is replaced with $\boldsymbol{x}[i]$. Then we have $\boldsymbol{y} = \boldsymbol{x_1}\boldsymbol{x}[i]\widetilde{\boldsymbol{x_2}}\boldsymbol{x_3}$. If $r(\boldsymbol{y}) > r(\boldsymbol{x})$, we contradict the choice of $\boldsymbol{x}$ as a run-maximal (d, n)-string. Otherwise, $r(\boldsymbol{y}) = r(\boldsymbol{x})$.

□

Even if a string does have an r-cover, the structure of that r-cover may impose limits on the maximum number of runs in that string. We divide r-covers into two distinct sets. In the first set, termed ***adjacent r-covers***, we have the r-covers in which a pair of subsequent squares are touching each other, but not overlapping. That is, for some $1 \leq i < m$, $s_{i+1} = s_i + 2p_i$. The squares S_i and S_{i+1}

are the ***adjacent squares***. The remaining r-covers will be termed ***overlapping r-covers***. They are the r-covers that for every $1 \leq i < m$, $s_{i+1} < s_i + 2p_i$.

Lemma 3.11: [2] *If there is a run-maximal (d, n)-string $\boldsymbol{x}$ with an adjacent r-cover, then $\rho_d(n) \leq \rho_{d_1}(n_1) + \rho_{d_2}(n_2)$ for some $1 \leq d_1, d_2 \leq d$ and some $n_1, n_2 \geq 0$ such that $n_1 + n_2 = n$.*

Proof. Let $\{S_i \ : \ 1 \leq i \leq m\}$ be the r-cover of $\boldsymbol{x}$. Let $S_j = (s_j, p_j, 2)$ and $S_{j+1} = (s_{j+1}, p_{j+1}, 2)$ be two adjacent squares of the r-cover, such that $s_{j+1} = s_j + 2p_j$. Let $\boldsymbol{x_1} = \bigcup_{1 \leq i \leq j} S_i$ and $\boldsymbol{x_2} = \bigcup_{j < i \leq m} S_i$. Then $\rho_d(n) = r(\boldsymbol{x}) \leq r(\boldsymbol{x_1}) + r(\boldsymbol{x_2}) \leq \rho_{d(\boldsymbol{x_1})}(n(\boldsymbol{x_1})) + \rho_{d(\boldsymbol{x_2})}(n(\boldsymbol{x_2}))$. □

Strictly positive core vectors provide one approach to obtaining strings with a certain run density, and now we introduce another. Let us assume that we have some lower bound for $\rho_d(n)$, given by $\rho_d^-(n)$. We are attempting to search for (d, n)-strings which contain more than $\rho_d^-(n)$ runs.

Definition 3.12: [2] *A (d, n)-string $\boldsymbol{x}$ is $\rho_d^-(n)$-**dense**, if its core vector $k(\boldsymbol{x})$ satisfies $k_i(\boldsymbol{x}) > \rho_d^-(n) - r(\boldsymbol{x}[1..i-1]) - m_i$ for $1 \leq i \leq n$, where $m_i = \max \{ \rho_{d_2}(n-i) \mid d - d_1 \leq d_2 \leq \min(n-i, d) \}$ and $d_1 = d(\boldsymbol{x}[1..i-1])$.*

Lemma 3.13: *If a (d, n)-string x is not $\rho_d^-(n)$-dense, then $\boldsymbol{r}(x) \leq \rho_d^-(n)$.*

Proof. For any string $\boldsymbol{x}$, $r(\boldsymbol{x}) \leq r(\boldsymbol{x}[1..i-1]) + r(\boldsymbol{x}[i+1..n]) + k_i(\boldsymbol{x})$ for all $1 \leq i \leq n$. Note that in most situations $r(\boldsymbol{x}) = r(\boldsymbol{x}[1..i-1]) + r(\boldsymbol{x}[i+1..n]) + k_i(\boldsymbol{x})$, except when the core of some run containing i is empty. Such a run is split into two runs: one in $\boldsymbol{x}[1..i-1]$ and the other in $\boldsymbol{x}[i+1..n]$. If $\boldsymbol{x}$ is not $\rho_d^-(n)$-dense, then for some i_0, $k_{i_0}(\boldsymbol{x}) \leq \rho_d^-(n) - r(\boldsymbol{x}[1..i_0-1]) - m_{i_0}$. Since $r(\boldsymbol{x}) \leq r(\boldsymbol{x}[1..i_0-1]) + r(\boldsymbol{x}[i_0+1..n]) + k_{i_0}(\boldsymbol{x}) \leq r(\boldsymbol{x}[1..i_0-1]) + m_{i_0} + k_{i_0}(\boldsymbol{x})$, then $r(\boldsymbol{x}) \leq r(\boldsymbol{x}[1..i_0-1]) + m_{i_0} + \rho_d^-(n) - r(\boldsymbol{x}[1..i_0-1]) - m_{i_0} = \rho_d^-(n)$. □

We combine the concepts of $\rho_d^-(n)$-density and r-covers. If we have a string, and we know its r-cover, if we extend that string in such a way that the r-cover is maintained, most of the values of the core vector cannot increase.

Lemma 3.14: *Let $\boldsymbol{x}$ be a string with the r-cover $\{ S_i \mid 1 \leq i \leq m \}$. Let $\boldsymbol{x'} = \boldsymbol{xy}$ be a string with the r-cover $\{ S'_i \mid 1 \leq i \leq m' \}$. If $S_i = S'_i$ for $1 \leq i \leq m$, then $k_i(\boldsymbol{x}) \leq k_i(\boldsymbol{x'})$ for $1 \leq i \leq s_m$.*

Proof. Assume that $k_i(\boldsymbol{x}) > k_i(\boldsymbol{x'})$ for some $1 \leq i \leq s_m$. This implies that there exists a run $R = (s, p, e, t)$ in $\boldsymbol{x'}$ such that i is in the core of R, $s < s_m$, and $s + 2p > s_m + 2p_m$. Therefore, $S_m \subset (s, p, 2)$, which violates the definition of an r-cover. If we add $(s, p, 2)$ to the r-cover $\{ S'_i \}$, it contradicts our condition that $S_i = S'_i$ for $1 \leq i \leq m$. Therefore, the lemma holds. □

Let us assume that we can extend a string without modifying the existing r-cover. Then most of the values of the core vector are non-increasing as the string is extended. Therefore, if a string is not $\rho_d^-(n)$-dense in a position before the start of the last element in its r-cover, then no extension to the string will allow it to exceed $\rho_d^-(n)$ runs.

We now consider how these structural insights can be used to speed the computation of $\rho_d(n)$ values, and the search for run-maximal strings.

Chapter 4

Computational approaches

Generating all (d, n)-strings in order to find $\rho_d(n)$ quickly becomes computationally infeasible. Considering only the binary strings, each time the length is increased by a single position, there are twice as many strings to consider. The computational time more than doubles, as the run-counting algorithm takes longer as the strings increase in length. The symbols of the string are immaterial; only the pattern of their appearance matters. We would need to generate every pattern-distinct string, as described by Moore, Smyth, and Miller [40] in order to test every string. Moore, Smyth, and Miller show that there are $\left\{ {n \atop d} \right\}$ pattern-distinct strings of length n with d distinct symbols, a Stirling number of the second kind. We exploit the structural properties of run-maximal strings in order to drastically reduce the search space.

We determine the values of $\rho_d(n)$ by an iterative process. In Figure 4, we show how all strings may be divided into three partitions. The maximum number of runs in strings which are not r-covered and those with adjacent r-covers can be bounded based on the properties of those strings and previously computed values

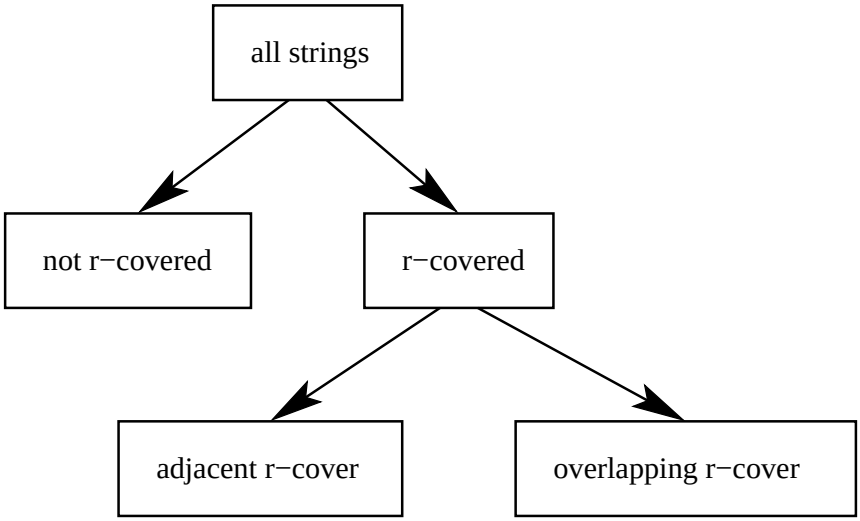

Figure 4.1: An illustration of how all strings may be divided into three partitions.

of $\rho_d(n)$. We have an initial lower bound from Property 2.1 of $\rho_d^-(n) = \rho_d(n-1)$, but we attempt to improve upon this bound through a search of a restricted class of (d, n)-strings in order to establish a lower bound $\rho_d^-(n)$. Finally, we perform an exhaustive search for any (d, n)-strings which may have more than $\rho_d^-(n)$ runs. If no such string is found, we have established that $\rho_d(n) = \rho_d^-(n)$. Otherwise, we update our lower bound and continue the search.

First we will discuss how the maximum number of runs in strings without r-covers or with adjacent r-covers can be computed using previously known $\rho_d(n)$ values. Next, we skip ahead to the algorithm used to search for (d, n)-strings with more than $\rho_d^-(n)$ runs. Then we explain the various heuristics which were added to the search algorithm in order to quickly determine a sufficient lower bound.

4.1 Strings without r-covers or with adjacent r-covers

In computing the value of $\rho_d(n)$, we will assume that for every $d' \leq d$ and $n' < n$, we already know $\rho_{d'}(n')$.

Based on Lemma 3.7, we can establish an upper bound on the maximum number of runs in any string without an r-cover from previously known values. We iterate through every pair of values n_1 and n_2 such that $n_1 + n_2 = n - 1$, and find the maximum values of $\rho_d(n_1) + \rho_d(n_2)$. This algorithm is given in Algorithm 2.

Algorithm 2: The algorithm for finding the maximum number of runs in a string of length n with d distinct symbols which does not have an r-cover.

```
begin Find the maximum number of runs in a string without a r-cover
    maximum ← 0, n1 = 0, n2 ← n − 1 − n1
    while n1 ≤ n2 do
        if ρd(n1) + ρd(n2) > maximum then
            maximum ← ρd(n1) + ρd(n2)
        n1 ← n1 + 1, n2 ← n − 1 − n1
    return maximum
```

For the strings which have an adjacent r-cover, there are additional cases to consider, but once again we can find the maximum possible number of runs based on the previously calculated values. Algorithm 3 is based on Lemma 3.11.

In practice, these algorithms typically do not yield run-maximal strings except for some cases where $n_1 = 1$ or 2. These two cases are guaranteed to yield actual strings which achieve this value (corresponding to Properties 2.1 and 2.3), but if the maximum is reached for some $n_1 > 3$, then it must be confirmed that the corresponding candidate strings can actually be concatenated without merging runs.

Algorithm 3: The algorithm for finding the maximum number of runs in a string of length n with d distinct symbols which has an adjacent r-cover.

```
begin Find the maximum number of runs in a string with an adjacent r-cover
    maximum ← 0, n1 = 0, n2 ← n − n1
    while n1 ≤ n2 do
        for d1 ← 1 . . . d do
            for d2 ← 1 . . . d do
                if ρ_{d1}(n1) + ρ_{d2}(n2) > maximum then
                    maximum ← ρ_{d1}(n1) + ρ_{d2}(n2)
        n1 ← n1 + 1, n2 ← n − n1
    return maximum
```

4.2 Searching for strings with more than $\rho_d^-(n)$ runs

As previously mentioned, we now assume that we have obtained a lower bound $\rho_d^-(n)$, and must show that there is no (d, n)-string with more than $\rho_d^-(n)$ runs. While Property 2.1 provides a lower bound of $\rho_d^-(n) = \rho_d(n-1)$, we will outline in Section 4.3 how we can drastically reduce the search space while searching for a string with more runs than this initial lower bound. The higher the lower bound we are able to obtain, the faster the search for strings with $\rho_d^-(n) + 1$ runs.

Having considered the strings without r-covers and those with adjacent r-covers in Section 4.1, we only need to consider strings with overlapping r-covers. Furthermore, as we have already found a string with $\rho_d^-(n)$ runs, we need only search for those strings with at least $\rho_d^-(n) + 1$ runs. If no such strings are found, we have established that the current lower bound gives the correct value for $\rho_d(n)$.

In order to generate only strings with r-covers, we generate those strings

directly from their r-covers. The first square must start at position 1, and must have period at most $\frac{n}{2}$. We must consider every pattern-distinct generator of each length. We then iteratively add another square to the r-cover, until the desired length is reached.

Each time a new square is added, we must consider every starting position within the previous square. We must also consider each period such that the square extends past the end of the previous square. For each such pair of starting position and period, one of two cases exists: either the generator is completely determined by the portion of the string which was already set, or we must consider every possible way to complete the generator.

After each square is added, there are some additional checks which must be performed. We must ensure that the string is the leading square of a run; that is, that it is not left-shiftable. The generator of the new string must be primitive. Finally, we must ensure that no additional leading square was introduced which violates the definition of the r-cover.

When the string is complete, we check the number of distinct symbols the string contains and count the number of runs. If we have found a string with the desired number of runs, we output it. We give the pseudocode for this approach in Algorithm 4.

While generating only r-covered strings results in a significant reduction in the search space, we can make further improvements by requiring that the strings be $\rho_d^-(n)$-dense. Recall the interaction between r-covers and core vectors described in Lemma 3.14. We use $\rho_d^-(n)$-density to limit the starting positions of squares S_t for $t > 1$, and to check for the presence of shrinking cores as the string is extended.

When adding the square S_t we originally consider every starting value s_t

Algorithm 4: Algorithm to generate all strings with overlapping r-covers.

```
begin Set the first square, S_1:
    s_1 ← 1
    for p_1 ← 1 ... ⌊n/2⌋ do
        foreach primitive string g of length p_1 do
            x[1..2p_1 − 1] ← gg
            addSquare(2)

begin addSquare(m)
    for s_m ← s_{m−1}+1 ... s_{m−1} + 2p_{m−1} − 1 do
        for p_m ← 1 ... (n−s_m+1)/2 do
            if s_m + p_m ≤ s_{m−1} + 2p_{m−1} then
                if the squaring of the generator does not conflict with the
                existing string then
                    square the generator
                    finishSquare(m)
                    remove the square
            else
                foreach completion of the generator do
                    square the generator
                    finishSquare(m)
                    remove the square

begin finishSquare(m)
    if x[s_m − 1] ≠ x[s_m + p_m − 1] then (the square is not left-shiftable)
        if x[s_m..s_m + 2p_m − 1] is primitive then
            if no run (s, p, e, t) so that s_{m−1} ≤ s ≤ s_m ≤ e_{m−1} ≤ e ≤ e_m and
            (s, p, 2) ∉ {S_i | 1 ≤ i ≤ m} then
                if s_m + 2p_m − 1 = n then
                    if number of distinct symbols = d then
                        output x
                else
                    { S_i } ← { S_i } ∪ { (s_m, p_m, 2) }
                    addSquare(m + 1)
                    { S_i } ← { S_i } ∖ { (s_m, p_m, 2) }
```

such that $s_{t-1} < s_t < s_{t-1} + 2p_{t-1}$. We know though, from Lemma 3.14, that the core vector for any position $i < s_t$ will not increase as the string is extended. Therefore, if the string is not $\rho_d^-(n)$-dense at some position $i < s_t$, then no extension of the string will ever make it $\rho_d^-(n)$-dense. We can call this least position i a ***cut***. We can then modify a portion of Algorithm 4 to take this property into account, as shown in Algorithm 5. Recall the definition of $\rho_d^-(n)$-dense, Definition 3.12, for the value of m_i.

Algorithm 5: Restricting the starting position of S_t by requiring $\rho_d^-(n)$-density.

```
begin addSquare(m)
    cut ← the least i such that k_i(x) ≤ ρ_d^-(n) − r(x[1..i−1]) − m_i
    for s_m ← s_{m−1}+1 ... min(s_{m−1} + 2p_{m−1} − 1, cut) do
        for p_m ← 1 ... (n−s_m+1)/2 do
            if s_m + p_m ≤ s_{m−1} + 2p_{m−1} then
                if the squaring of the generator does not conflict with the
                existing string then
                    square the generator
                    finishSquare(m)
                    remove the square
            else
                foreach completion of the generator do
                    square the generator
                    finishSquare(m)
                    remove the square
```

Although the combination of Algorithm 4 and 5 will ensure that the string is $\rho_d^-(n)$-dense as it is built, if a run is extended beyond just the leading square its core will shrink and some $k_i(\boldsymbol{x})$ value will decrease. Each time a square is finalized, we check to ensure that every position up to the start of that square continues to be $\rho_d^-(n)$-dense. We add this additional step into the pseudocode

and present it in Algorithm 6. In order to avoid checking every position up to s_m for $\rho_d^-(n)$-density one could do additional processing in order to determine which runs could have been extended by the addition of the new square. However, the additional overhead and complexity is not worth it, given the speed at which every position may be checked naïvely.

Algorithm 6: Checking for $\rho_d^-(n)$-density violations due to shrinking cores.

```
begin finishSquare(m)
    if x[s_m − 1] ≠ x[s_m + p_m − 1] then (the square is not left-shiftable)
        if x[s_m..s_m + 2p_m − 1] is primitive then
            if no run (s, p, e, t) so that s_{m−1} ≤ s ≤ s_m ≤ e_{m−1} ≤ e ≤ e_m and
            (s, p, 2) ∉ {S_i | 1 ≤ i ≤ m} then
                if every position 1 ≤ i < s_t is ρ_d^-(n)-dense then
                    if s_m + 2p_m − 1 = n then
                        if number of distinct symbols = d then
                            output x
                    else
                        { S_i } ← { S_i } ∪ { (s_m, p_m, 2) }
                        addSquare(m + 1)
                        { S_i } ← { S_i } ∖ { (s_m, p_m, 2) }
```

This code is quite efficient at establishing the value of $\rho_d(n)$. To naïvely generate all pattern-distinct $(2, 33)$-strings and count the number of runs each contains took approximately 85 hours of computation time on an Orca node of the SHARCNET computer cluster. Conversely, having already obtained a $(2, 33)$-string with 27 runs, it took only an average of 1.8 seconds over 5 executions to confirm that this is indeed the maximal value.

In order to parallelize Algorithm 4, first we divide the required execution into separate cases based on the value assigned to p_1. While this parallelization certainly does not result in equal the run-time across the different processes, it

is an adequate approach for strings of moderate length. When the worst cases begin to once again take longer than we would like, we divide them up again by seeding different processors with different prefixes. Every possible pattern-distinct prefix of a sufficient length is generated, and one given to each processor. The sufficient length is experimentally determined based on the run-time of the previous computations.

The bottleneck for this algorithm is exponentially growing number of generators which must be considered for the long first periods. This illustrates how effective the $\rho_d^-(n)$-density condition is; most partially generated strings are eliminated early on in the computation. Table 4.1 gives the runtimes required for each starting period for establishing that $\rho_2(62) = 53$. This is a typical case in that $\rho_2(62) = \rho_2(61) + 1$. The cases of $p_1 = 1$ and $p_1 = 2$ both complete immediately, while $p_1 = 31$ takes the longest.

Table 4.1: Run-time in seconds required to establish that $\rho_2(62) = 53$, given that we have a $(2, 62)$-string with 53 runs. Note that $\rho_2(62) = \rho_2(61) + 1$.

p_1	Time (s)	p_1	Time (s)	p_1	Time (s)
1, 2	0	12	11	22	81
3	1860	13	6	23	170
4	942	14	3	24	356
5	919	15	1	25	745
6	317	16	2	26	1554
7	362	17	3	27	3277
8	152	18	4	28	6808
9	80	19	9	29	14073
10	43	20	18	30	29574
11	24	21	38	31	61501

When we have a case where $\rho_2(n) = \rho_2(n-1)$, the computation slows down significantly. In Table 4.2, we illustrate this difference by presenting the run-time

to establish that $\rho_2(66) = 56$. In this case, $\rho_2(66) = \rho_2(65) = 56$.

Table 4.2: Run-time in seconds required to establish that $\rho_2(66) = 56$, given that we have a $(2, 66)$-string with 56 runs. Note that $\rho_2(66) = \rho_2(65)$.

p_1	Time (s)	p_1	Time (s)	p_1	Time (s)
1	194884	12	245	23	180
2	43259	13	154	24	374
3	69691	14	62	25	778
4	36270	15	32	26	1618
5	29216	16	16	27	3362
6	9443	17	10	28	6973
7	8470	18	9	29	14465
8	3712	19	11	30	30083
9	1912	20	20	31	62230
10	963	21	42	32	129837
11	536	22	86	33	268789

We now turn to the heuristic search for run-maximal strings.

4.3 Heuristic search for $\rho_2^-(n)$

For strings with $d > 2$, the properties outlined in Chapter 2 provide a very good lower bound for $\rho_d(n)$. In fact, for all known cases, this bound is tight. Thus we are able to provide a good estimate by taking the maximum over these three constructions:

- Let $\boldsymbol{y_1}$ be a run-maximal $(d, n-1)$-string with $a \in \mathcal{A}(\boldsymbol{y_1})$, then $\boldsymbol{x_1} = \boldsymbol{y_1}a$.
- Let $\boldsymbol{y_2}$ be a run-maximal $(d-1, n-1)$-string with $z \notin \mathcal{A}(\boldsymbol{y_2})$, then $\boldsymbol{x_2} = \boldsymbol{y_2}z$.
- let $\boldsymbol{y_3}$ be a run-maximal $(d-1, n-2)$-string with $z \notin \mathcal{A}(\boldsymbol{y_3})$, then $\boldsymbol{x_3} = \boldsymbol{y_3}zz$

These constructions correspond to basic properties of $\rho_d(n)$ visible in the $(d, n-d)$-table, namely Properties 2.1, 2.2, and 2.4 respectively.

We therefore focus on the binary strings, $d = 2$. First we make use of Properties 2.1 and 2.3 in order to obtain a basic lower bound. We use the search algorithm described in Section 4.2 together with some additional heuristics to further reduce the search space. In most cases, $\rho_2(n) = \rho_2(n-1)+1$, so if we find a $(2, n)$-string with $\rho_2(n-1)+1$ runs, we end our heuristic search and continue on to the elimination phase to attempt to establish that the found string is actually run-maximal. The only known binary cases where we have an increase of more than one are $\rho_2(14) = \rho_2(13) + 2$, and $\rho_2(42) = \rho_2(41) + 2$.

Our search is based on three restrictions in order to reduce the search pool: no triples, balanced over every prefix, and maximum period. We will the decisions to include these heuristic rules and the associated parameters in Sections 4.3.1, 4.3.2, and 4.3.3 respectively.

4.3.1 No triples

For most n, a run-maximal binary, triple-free string exists. Therefore we remove from consideration all strings which contain *aaa* or *bbb*. This is accomplished through a simple nested conditional check. Let us assume we are extending a string by adding a symbol at position i, due to either completing or squaring a generator. If $i > 2$, we check to see if $\boldsymbol{x}[i] = \boldsymbol{x}[i-1] = \boldsymbol{x}[i-2]$. If so, we indicate that adding that symbol has failed, and the algorithm moves on to the next step.

This heuristic fails to find a run-maximal string when $n = 25$ or 65. See Chapter 5 for a further elaboration of the presence of triples in run-maximal strings.

4.3.2 Balanced prefixes

By ***balanced prefix*** we mean that for any given prefix of the string, the difference between the number of a's and the number of b's is bounded by a constant. Conceptually, if a string has too many of one symbol grouped together, then it would limit the maximum number of runs. We analyzed our exhaustive list of run-maximal strings for small n in order to establish an acceptable parameter for the maximum prefix difference. As additional run-maximal strings were found, we updated our analysis.

We present in Table 4.3 the minimum and maximum prefix differences found in run-maximal binary strings. We are typically interested in the minimum prefix difference, as we only need to find a single run-dense string.

While there is a general trend toward larger prefix differences as the length increases, a maximum prefix difference of 6 is sufficient to find a run-maximal $(2, n)$-string for every $n \leq 64$.

The following is the only run-maximal string of length 65, and it is weighted heavily toward a's:

$$aababaababbabaababaabababbabaababaaababaabababbabaababaabababbabaababa$$

This string has 37 a's but only 28 b's. This is the length with the largest minimum prefix difference on a binary string up to $n = 66$.

The binary run-maximal strings with the largest prefix difference (for $3 \leq n \leq 66$) are of length 41 and 43. In both of these strings the symbols are weighted toward a's by a difference of 11.

$$aabaabbaabaaabaabbaabaabbaabaaabaabbaabaa$$

$$aabaabbaabaaabaabbaabaabbaabaaabaabbaabaabb$$

Table 4.3: The minimum and maximum prefix difference over all run-maximal binary strings for $3 \leq n \leq 66$.

n	Min.	Max.	n	Min.	Max.	n	Min.	Max.
3	1	2	25	5	7	47	2	4
4	2	2	26	2	4	48	2	4
5	1	3	27	2	5	49	3	5
6	2	3	28	2	4	50	3	4
7	2	3	29	2	4	51	2	5
8	2	2	30	2	2	52	2	5
9	1	3	31	2	4	53	3	5
10	2	3	32	2	4	54	6	6
11	2	3	33	5	5	55	2	9
12	4	4	34	2	6	56	2	7
13	1	5	35	2	5	57	2	7
14	2	4	36	2	6	58	2	8
15	2	5	37	3	9	59	3	9
16	1	4	38	2	10	60	4	8
17	1	4	39	2	7	61	5	7
18	2	4	40	2	6	62	6	7
19	2	5	41	2	11	63	5	7
20	4	4	42	4	4	64	4	8
21	1	5	43	2	11	65	9	9
22	2	4	44	2	6	66	2	10
23	2	7	45	3	9			
24	2	4	46	2	6			

4.3.3 Maximum period

The maximum period in a string is $\lfloor \frac{n}{2} \rfloor$. Indeed, there are multiple run-maximal strings which contain a run with $p = \lfloor \frac{n}{2} \rfloor$, as described in Chapter 5. In Table 4.4 we present the minimum and maximum values of the maximum period size over all binary strings of length n for $3 \leq n \leq 66$. That is, for each string of length n, we find the largest period it contains. Then we find the minimum values over all strings of that length.

Table 4.4: The minimum and maximum values of the largest period in a string over all run-maximal binary strings for $3 \leq n \leq 66$.

n	Min.	Max.	n	Min.	Max.	n	Min.	Max.
3	1	1	25	11	11	47	13	21
4	1	1	26	13	13	48	13	21
5	1	2	27	11	13	49	13	21
6	1	3	28	13	13	50	13	21
7	3	3	29	13	13	51	18	21
8	4	4	30	13	13	52	21	21
9	3	4	31	13	13	53	21	21
10	3	5	32	13	13	54	21	21
11	4	5	33	13	13	55	13	24
12	4	4	34	13	13	56	21	27
13	3	6	35	13	13	57	21	28
14	7	7	36	13	13	58	21	28
15	5	7	37	13	18	59	22	28
16	5	8	38	13	19	60	24	28
17	7	8	39	13	19	61	24	28
18	7	8	40	13	18	62	31	31
19	7	8	41	13	18	63	31	31
20	8	8	42	13	13	64	31	32
21	7	10	43	13	21	65	32	32
22	8	11	44	13	21	66	13	33
23	8	11	45	13	21			
24	7	8	46	13	18			

As with the balanced prefix, we are typically interested in the minimum value, as we only need to find a single run-dense string. We introduced this heuristic based on the minimum values from 3 through 50. To this point, the minimum value behaves very nicely, with there always existing a run-maximal string with a maximum period 13. While the minimum largest period does grow after this point, in many cases we are able to find a run-maximal string with a maximum period less than half the length.

This heuristic integrates well with the generation of strings via their r-

cover. Recall that by the definition of the r-cover, the leading square of every run in the string must be a substring of a square of the r-cover. Therefore, the maximum period of any run is bounded by the length of the largest period of the r-cover. Instead of iterating through every possible period size, we can terminate the loop when the bound is reached.

4.4 Implementing the heuristic search for $\rho_2^-(n)$

We implement these three heuristic rules into the search algorithm from Section 4.2 in order to find run-dense binary strings quickly. The values for the maximum prefix difference and maximum period size are passed to the program via command-line argument.

In Table 4.5 we present the run-times of the lower bound search algorithm for $(2, 58)$-strings, based on various combinations of parameters for the heuristic rules. The more restrictive parameters result in the program running faster overall, but may also eliminate some strings which would be returned by less restrictive parameters. Eliminating too many strings may increase the time required to find a single $(2, n)$-string with more than $\rho_2(n-1)$ runs. As only a single such string is required to establish the improved lower bound, choosing appropriate values requires balancing these two conflicting influences.

Table 4.5: The run-time of various combinations of the heuristic search algorithm looking for a $(2, 58)$-string with at least 49 runs.

Triples	Prefix Diff.	Max. Period	First (s)	Total (s)
allowed	no limit	no limit	16086	56486
allowed	no limit	21	287	503
allowed	no limit	26	6113	16039
allowed	6	no limit	7387	16947
allowed	6	21	112	199
allowed	6	26	3172	5290
allowed	10	no limit	9156	31717
allowed	10	21	186	329
allowed	10	26	3588	9428
not allowed	no limit	no limit	72	233
not allowed	no limit	21	72	234
not allowed	no limit	26	73	236
not allowed	6	no limit	89	183
not allowed	6	21	5	10
not allowed	6	26	55	98
not allowed	10	no limit	26	220
not allowed	10	21	6	12
not allowed	10	26	41	116

Chapter 5

Results on run-maximal strings

In this chapter, we will discuss some of the interesting results based on our search for run-maximal strings. The binary results, which make up the majority of this section, are based on the run-maximal strings for $3 \leq n \leq 66$, the lengths for which we are sure to have found every run-maximal string.

Run-maximal strings which are squares themselves are not particularly uncommon. This is a disappointing finding, as large periods in the r-cover are a computational bottleneck in the elimination algorithm. Had long periods not existed, and this could be proven to hold in general, it could have dramatically improved the runtime. In fact, there are run-maximal $(2, n)$-strings which are squares for $n = 6$, 8, 10, 14, 16, 22, 26, 38, 62, 64, and 66. For $n = 8$, 14, 26, and 62, every run-maximal $(2, n)$-string is a square.

Triples are also fairly common in run-maximal strings, though there is typically a run-maximal string without one. Triples occur in the run-maximal $(2, n)$-strings for $n = 5$, 9, 13, 15, 21, 23, 25, 27, 34, 37, 38, 39, 41, 43, 45, 55, 57, 58, 59, 65, and 66. Every run-maximal $(2, 25)$- and $(2, 65)$-string contains a triple.

Therefore, our heuristic search which eliminates triples will fail in these cases.

Other cubes are not uncommon either. Though rare in run-maximal $(2, n)$-strings for $3 \leq n \leq 37$, they are quite common in run-maximal strings beyond this length.

Kolpakov and Kucherov suggest that the maximum exponent in a run-maximal string may be 3 based on experimental results and the structure of the run-dense Fibonacci strings [34]. However, we find two run-maximal strings that contain quadruples. The run-maximal $(2, 38)$-string

$$aabaabbaabaabbaabaaaabaabbaabaabbaabaa$$

and the run-maximal $(2, 66)$-string

$$aababaababbabaababaabababbabaababaaaababaababbabaababaababbabaababaa$$

both contain the quadruple $aaaa$. In our data set, there are no strings with runs with $e \geq 4$ and $p > 1$. However, we would not be surprised to find such runs in longer strings.

We mentioned in Section 4.3 that in every case we have computed, the value of $\rho_d(n)$ is given by one of Properties 2.1, 2.2, and 2.4 for $d > 2$. That is, $\rho_d(n) = \max\{\rho_d(n-1), \rho_{d-1}(n-1), \rho_{d-1}(n-2)+1\}$. However, not every run-maximal (d, n)-string for $d > 2$ can be created by one of the three constructions.

In particular, we are interested in run-maximal (d, n)-strings for $d > 2$ with overlapping r-covers. The exist for $d = 3$ for some lengths, but not all. We have run-maximal $(3, n)$-strings for $11 \leq n \leq 14$, $17 \leq n \leq 20$, $23 \leq n \leq 26$, $36 \leq 42$, and $n = 45$. Note that the lengths where $\rho_3(n) = \rho_3(n-1)$ are $n = 7$, 11, 17, 23, 36, 40, and 45. The overlap suggests that it is the flexibility afforded by the tie $\rho_3(n) = \rho_3(n-1)$ that allows for run-maximal strings with overlapping r-covers.

For the $(4, n)$-strings, we have fewer strings with overlapping r-covers. In this case, we have overlapping r-covers for $19 \leq n \leq 20$ and $25 \leq n \leq 26$. Again, there is a correspondence between overlapping r-covers and $\rho_4(n) = \rho_4(n-1)$. We have $\rho_4(n) = \rho_4(n-1)$ when $n = 9$, 13, 19, 25, and 38.

We have been unable to find any run-maximal (d, n)-string for $d > 4$ with an overlapping r-cover.

In most cases, $\rho_d(n) - 1 \leq \rho_d(n-1) \leq \rho_d(n)$. That is, increasing the length by 1 leaves the maximum number of runs the same, or increases it by one. However, there are two known points where $\rho_d(n) = \rho_d(n-1) + 2$: $n = 14$ and $n = 42$. This makes for a unique region of the $(d, n-d)$-table. In the first case, $n = 14$, the skip by 2 disappears on the $d = 3$ line, as it is cancelled by merging with the tie $\rho_2(12) = \rho_2(13)$. This structure is shown in Table 5.1. However, in the case of $n = 42$, the $+2$ skip propagates straight down the $(d, n-d)$-table. We expect that it will eventually be cancelled by intersecting with the tie $\rho_2(35) = \rho_2(36)$ which propagates down the diagonal.

Table 5.1: Values for $\rho_d(n)$ with $2 \leq d \leq 3$ and $10 \leq n - d \leq 13$. Note how the skip of $+2$ on row $d = 2$ disappears on row $d = 3$.

		$n-d$					
		9	10	11	12	13	14
	2	.	8	8	10	10	.
d	3	.	8	9	10	11	.
	4	.	.	.	.	.	.

Chapter 6

Necessary conditions for a (d, n)-string to have more than $n - d$ runs

Computational evidence supports the conjecture that $\rho_d(n) \leq n-d$, which implies $\rho(n) \leq n$. We consider the structural properties of a (d, n)-string with strictly more than $n - d$ runs. We hope that future work may be able to restrict the structure of such a string to such a point that it can never exist, thus proving that $\rho_d(n) \leq n - d$ for all $2 \leq d \leq n$.

Let us assume that $\rho_{i-1}(2i - 2) \leq i - 1$. Therefore by Property 2.11, for every $1 \leq i' < i$, $\rho_{i'}(2i') \leq i'$. If there exists a (d, n)-string $\boldsymbol{x}$ with $r(\boldsymbol{x}) > n-d = i$, we can restrict the structure of that string.

Proposition 6.1: [4] *Let $\rho_{i'}(2i') = i'$ for $1 \leq i' < i$. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string for some $n - d = i$. Either $r(\boldsymbol{x}) = \rho_d(n) \leq n - d$ or no symbol occurs exactly 2, 3, ..., 8 times in $\boldsymbol{x}$.*

The proof of Proposition 6.1 relies on a series of lemmas, all of which deal with the same basic scenario. We consider a run-maximal (d, n)-string $\boldsymbol{x}$ containing a k-tuple of c's such that $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c \ldots \boldsymbol{u_{k-1}}c\boldsymbol{u_k}$. We show that either $\boldsymbol{x}$ satisfies the conjectured upper bound, or can be used to generate a new string a new string $\boldsymbol{y}$ with more distinct symbols and the same length. We ensure that the manipulation process does not destroy more runs than the number by which the alphabet is increased. This allows us to place a limit on the number of runs in $\boldsymbol{y}$ based on the values in the $(d, n-d)$ table for $i' < i$. In essence, we manipulate a string from column i to form a string from some column $i' < i$ while monitoring how the number of runs changes. In the manipulation process, we put an upper limit on the number of runs that are destroyed, which will be denoted by π, and a lower limit on how many additional symbols are introduced, denoted by δ.

Following the definition of a mapping and an outline of the approaches to be used, we present seven lemmas which taken together prove Proposition 6.1. The proof for each lemma requires multiple cases, and they get increasingly complex. It is for this reason that we concluded the investigation at 8-tuples, but we expect that a similar argument would extend the proof to larger tuples. After the proofs of the lemmas, we will return to a discussion of the implications of Proposition 6.1.

Definition 6.2: Map *A run (s, p, e, t) in a string $\boldsymbol{x}$ **maps** position i to position j if $s \leq i < j < s + ep + t$, $\boldsymbol{x}[i] = \boldsymbol{x}[j]$, and $j - i = p$. Let $i \to j$ denote a mapping from i to j, called a **single-mapping**. Let the **double-mapping** $(i_1, i_2) \to (j_1, j_2)$ indicate that $s \leq i_1 < i_2 < j_1 < j_2 < s + ep + t$, $\boldsymbol{x}[i_1] = \boldsymbol{x}[i_2] = \boldsymbol{x}[j_1] = \boldsymbol{x}[j_2]$, $i_2 - i_1 < p$ and $j_1 - i_1 = j_2 - i_2 = p$. The **triple-mappings** and higher order mappings are defined analogously.*

A ***multi-mapping*** is any mapping which is not a single-mapping. The presence of a multi-mapping imposes equality on the substrings bounded on each side. For example, in the double-mapping $(i_j, i_{j+1}) \rightarrow (i_{j+2}, i_{j+3})$, the substring between i_j and i_{j+1} is the same as the substring between i_{j+2} and i_{j+3}. That is, $\boldsymbol{x}[i_j..i_{j+1}] = \boldsymbol{x}[i_{j+2}..i_{j+3}]$

We present a mapping as a consequence of having a run in a string. However, given positions $1 \leq i < j \leq n$, with $\boldsymbol{x}[i] = \boldsymbol{x}[j]$, we can consider a candidate mapping between these positions corresponding to a run with period $j - i$. For simplicity's sake, we will refer to such a candidate mapping simply as a mapping. Further investigation of the structure of the string may reveal that no such run can occur.

When searching for a run from a mapping we will assume that all copies of a symbol from the generator are included in the mapping. That is, if we consider a triple-mapping, the generator of the corresponding run must have exactly three copies of that symbol. We will also typically assume that the mapping applies to the leading square.

To ensure that there are sufficiently more distinct symbols in $\boldsymbol{y}$ than in $\boldsymbol{x}$, we use multiple strategies. The two most common approaches are as follows. First, we can change all but one of the c's to new characters c_2, c_3, $\ldots c_k$, thus introducing $k - 1$ new symbols. This destroys, at most, the runs which contain a c. The other primary method involves the substrings between the c's. When multiple disjoint copies of a substring occur in $\boldsymbol{x}$, we can replace all copies of a symbol within one copy of the substring with a new symbol which does not occur elsewhere in $\boldsymbol{x}$. Given $\boldsymbol{x} = \boldsymbol{uvu}$, we can increase the number of distinct symbols with $\boldsymbol{y} = \boldsymbol{u}c\widehat{\boldsymbol{u}}$. We will assume that $\boldsymbol{u}c\widehat{\boldsymbol{u}}c\widehat{\boldsymbol{u}}$ has two more distinct symbols than

$\boldsymbol{u}c\boldsymbol{u}c\boldsymbol{u}$ does, and so on. That is, while we continue to use the same notation, each copy of $\widehat{\boldsymbol{u}}$ introduces a new distinct symbol. As all copies of the symbol within the substring are replaced, the runs contained entirely within the substring remain intact and only those runs which extend outside of the substring may be destroyed. This approach can only be used when the substring having a symbol changed is non-empty.

Another strategy used is to show that the candidate run derived from a mapping cannot actually exist in the string. This is typically the case when some substring is assumed to be empty, resulting in a structure of the form $\ldots cc\boldsymbol{u}cc\ldots$. As we always assume the presence of every single mapping, we assume that there is a mapping between the second and third c's, which bracket $\boldsymbol{u}$. However, assuming that $\boldsymbol{u}$ is non-empty, there must be a symbol between the first and second c's, or the third and fourth. Since no such symbol can exist in either place, the run referred to by the single-mapping cannot exist.

Another way we can decrease the number of possible runs is by having runs merged together. When the substring between elements of the k-tuple are the same due to multi-mappings, occasionally this collapses multiple runs down to one with a larger exponent. For example, $\ldots c\boldsymbol{u}c\boldsymbol{u}c\ldots$ can only have one run corresponding to a single mapping. Runs can also be merged together when substrings are assume to be empty.

The manipulations we perform keep the length of the string constant while the number of distinct symbols increases. Therefore, $\boldsymbol{y}$ is a $(d+\delta, n)$-string and since $n-(d+\delta) < n-d$, it must satisfy the maximum number of runs conjecture. If π is the upper bound to the number of runs which are destroyed through the modification of the string, $r(\boldsymbol{x}) - \pi \leq r(\boldsymbol{y}) \leq i - \delta$. Therefore, $\rho_d(n) = r(\boldsymbol{x}) \leq$

$n - d - \delta + \pi$. Thus, whenever $\pi \leq \delta$, $\rho_d(n) \leq n - d$.

The proofs of the lemmas consist of several cases and subcases. The cases typically involve considering that different subsets of mappings apply to the string. The subcases deal with substrings being empty or non-empty.

Lemma 6.3: *Let $\rho_{i'}(2i') = i'$ for $1 \leq i' < i$. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string for some $n - d = i$. Either $r(\boldsymbol{x}) = \rho_d(n) \leq n - d$ or $\boldsymbol{x}$ does not contain a pair.*

Proof. As noted in [13], a pair of c's can be involved in at most one run. This corresponds to one single-mapping. We change the second c to a new symbol c_2 creating $\boldsymbol{y} = \boldsymbol{u_0} c \boldsymbol{u_1} c_2 \boldsymbol{u_2}$. We destroy at most the single run which contains the pair ($\pi \leq 1$), and gain 1 symbol ($\delta = 1$). As $\pi \leq \delta$, either $r(\boldsymbol{x}) \leq i$ or $\boldsymbol{x}$ does not contain a pair. □

Lemma 6.4: *Let $\rho_{i'}(2i') = i'$ for $1 \leq i' < i$. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string for some $n - d = i$. Either $r(\boldsymbol{x}) = \rho_d(n) \leq n - d$ or $\boldsymbol{x}$ does not contain a 3-tuple.*

Proof. A 3-tuple of c's at positions $i_1 < i_2 < i_3$ can be involved in at most two runs, corresponding to the single-mappings $i_1 \to i_2$ and $i_2 \to i_3$. If a 3-tuple of c's is involved in less than two runs, we can proceed as in the proof of Lemma 6.3. Therefore, let us therefore assume that the c's are involved in two runs.

The string then has the form $\boldsymbol{x} = \boldsymbol{u_0} c \boldsymbol{u_1} c \boldsymbol{u_2} c \boldsymbol{u_3}$. In this case, we replace two of the c's with new symbols c_2 and c_3 creating $\boldsymbol{y} = \boldsymbol{u_0} c \boldsymbol{u_1} c_2 \boldsymbol{u_2} c_3 \boldsymbol{u_3}$. This destroys, at most, only the two possible runs which contain a c ($\pi \leq 2$), while we gain two symbols ($\delta = 2$). Again, δ is sufficiently large so that either $r(\boldsymbol{x}) \leq i$, or $\boldsymbol{x}$ does not have a 3-tuple. □

Lemmas 6.3 and 6.4 imply that if only single-mappings are involved, then we can obtain a new string with sufficiently more distinct symbols while limiting

the number of runs destroyed. In Lemmas 6.5 through 6.9, we will only consider the cases which include a multi-mapping.

Lemma 6.5: *Let $\rho_{i'}(2i') = i'$ for $1 \leq i' < i$. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string for some $n - d = i$. Either $r(\boldsymbol{x}) = \rho_d(n) \leq n - d$ or $\boldsymbol{x}$ does not contain a 4-tuple.*

Proof. A 4-tuple of c's at positions $i_1 < i_2 < i_3 < i_4$ can be involved in at most four runs, corresponding to a double-mapping $(i_1, i_2) \rightarrow (i_3, i_4)$ and single-mappings $i_1 \rightarrow i_2$, $i_2 \rightarrow i_3$, and $i_3 \rightarrow i_4$. If the c's are involved in only three or fewer runs, replacing three occurrences of c by three new symbols will give $\delta = 3$ and $\pi \leq 3$, so $\pi \leq \delta$, proving the lemma.

We therefore assume that the c's are involved in exactly four runs. In this case replacing three of the c's by new symbols is no longer enough, as π would be greater than δ. However, from the double-mapping $(i_1, i_2) \rightarrow (i_3, i_4)$, we know that $\boldsymbol{x}[i_1..i_2] = \boldsymbol{x}[i_3..i_4]$. Therefore, if $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_3}c\boldsymbol{u_4}$, then $\boldsymbol{u_1} = \boldsymbol{u_3}$, so $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_4}$. We must consider the possibility of $\boldsymbol{u_1}$ being either non-empty or empty.

Case 1: Assume $\boldsymbol{u_1} \neq \varepsilon$. We replace the last three copies of c with new symbols c_2, c_3, and c_4, and all instances of some symbol in the second occurrence of $\boldsymbol{u_1}$ with a new symbol producing: $\boldsymbol{y} = \boldsymbol{u_0}c\boldsymbol{u_1}c_2\boldsymbol{u_2}c_3\widehat{\boldsymbol{u_1}}c_4\boldsymbol{u_4}$. This gives $\pi \leq 4$, but now $\delta = 4$, satisfying the lemma.

Case 2: Assume $\boldsymbol{u_1} = \varepsilon$. Now the string is $\boldsymbol{x} = \boldsymbol{u_0}cc\boldsymbol{u_2}cc\boldsymbol{u_4}$. We must consider the cases of $\boldsymbol{u_2}$ being empty or non-empty

Case 2.1: Assume $\boldsymbol{u_2} \neq \varepsilon$, giving $\boldsymbol{x}$ the form: $\boldsymbol{x} = \boldsymbol{u_0}cc\boldsymbol{u_2}cc\boldsymbol{u_4}$. Since $\boldsymbol{u_2}$ is non-empty, if the mapping $i_2 \rightarrow i_3$ corresponds to a run, then a symbol in $\boldsymbol{u_2}$ must also appear between the first and second c's, or the third and fourth c's. However, this requires $\boldsymbol{u_1}$ to be non-empty, a contradiction. Therefore, the mapping $i_2 \rightarrow i_3$

cannot refer to a run in the string, and so the c's are not involved in four different runs.

Case 2.2: Assume $\boldsymbol{u_2} = \varepsilon$, so $\boldsymbol{x} = \boldsymbol{u_0}cccc\boldsymbol{u_4}$. This merges all 4 possible runs into a single run, so there are not 4 runs covering the c's. □

Lemma 6.6: *Let $\rho_{i'}(2i') = i'$ for $1 \leq i' < i$. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string for some $n - d = i$. Either $r(\boldsymbol{x}) = \rho_d(n) \leq n - d$ or $\boldsymbol{x}$ does not contain a 5-tuple.*

Proof. A 5-tuple of c's at positions $i_1 < i_2 < i_3 < i_4 < i_5$ can be involved in at most 5 runs despite there being 6 possible mappings: double-mappings $(i_1, i_2) \to (i_3, i_4)$ and $(i_2 \to i_3) \to (i_4, i_5)$, and single-mappings $i_1 \to i_2$, $i_2 \to i_3$, $i_3 \to i_4$, and $i_4 \to i_5$. If both double-mappings exist, they correspond to the same run, as they have the same period p and overlap by at least p.

In the case of a 5-tuple, we can always introduce 1 new symbol while only destroying at most a single run. There are 3 cases to consider:

Case 1: All mappings correspond to runs. Then $\boldsymbol{x}[i_5]$ is involved in two runs, one corresponding to both double-mappings $(i_1, i_2) \to (i_3, i_4)$ and $(i_2, i_3) \to (i_4, i_5)$, and one corresponding to $i_4 \to i_5$. If we replace $\boldsymbol{x}[i_5]$ by a new symbol c_5, we destroy the run corresponding to $i_4 \to i_5$, but not the core of the run corresponding to $(i_1, i_2) \to (i_3, i_4)$ and $(i_2 \to i_3) \to (i_4, i_5)$. This gives us $\pi \leq 1 = \delta$.

Case 2: The mapping $(i_1, i_2) \to (i_3, i_4)$ exists, but $(i_2, i_3) \to (i_4, i_5)$ does not, while all single-mappings exist. As $\boldsymbol{x}[i_5]$ is only involved in the single mapping, we replace it with c_5, destroying at most one run and introducing one new symbol $(\pi \leq 1 = \delta)$.

Case 3: The mapping $(i_1, i_2) \to (i_3, i_4)$ does not exist, but $(i_2, i_3) \to (i_4, i_5)$ does, and all possible single-mappings exist. We proceed as in the second case, but replace the symbol at i_1 with c_1.

□

We remarked previously that if only single-mappings exist, then it is easy to perform our string manipulation in order to satisfy the lemma. From Lemma 6.6, we can see that, in fact, every c must be covered by some multi-mapping. In Lemmas 6.7 through 6.9, we will only consider the cases where every c is involved in at least one multi-mapping.

Lemma 6.7: *Let $\rho_{i'}(2i') = i'$ for $1 \leq i' < i$. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string for some $n - d = i$. Either $r(\boldsymbol{x}) = \rho_d(n) \leq n - d$ or $\boldsymbol{x}$ does not contain a 6-tuple.*

Proof. A 6-tuple at positions $i_1 < \ldots < i_6$ can be involved in at most 8 runs, despite there being 9 available mappings:

- triple-mapping: $(i_1, i_2, i_3) \rightarrow (i_4, i_5, i_6)$
- double-mappings: $(i_1, i_2) \rightarrow (i_3, i_4)$, $(i_2 \rightarrow i_3) \rightarrow (i_4, i_5)$, and $(i_3, i_4) \rightarrow (i_5, i_6)$
- single-mappings: $i_1 \rightarrow i_2$, $i_2 \rightarrow i_3$, $i_3 \rightarrow i_4$, $i_4 \rightarrow i_5$, and $i_5 \rightarrow i_6$

As in Lemma 6.6, if either both $(i_1, i_2) \rightarrow (i_3, i_4)$ and $(i_2 \rightarrow i_3) \rightarrow (i_4, i_5)$, or $(i_2 \rightarrow i_3) \rightarrow (i_4, i_5)$ and $(i_3, i_4) \rightarrow (i_5, i_6)$ exist, the two runs they correspond to are actually the same run due to their overlap.

Let $\boldsymbol{x} = \boldsymbol{u_0} c \boldsymbol{u_1} c \boldsymbol{u_2} c \boldsymbol{u_3} c \boldsymbol{u_4} c \boldsymbol{u_5} c \boldsymbol{u_6}$. We consider each configuration of multi-mappings separately:

Case 1: $(i_1, i_2) \rightarrow (i_3, i_4)$, $(i_3, i_4) \rightarrow (i_5, i_6)$, and all single-mappings: By the double-mappings, $\boldsymbol{u_1} = \boldsymbol{u_3} = \boldsymbol{u_6}$, and therefore the string $\boldsymbol{x}$ has the form: $\boldsymbol{x} = \boldsymbol{u_0} c \boldsymbol{u_1} c \boldsymbol{u_2} c \boldsymbol{u_1} c \boldsymbol{u_4} c \boldsymbol{u_1} c \boldsymbol{u_6}$. We now consider sub-cases of different combinations of empty and non-empty substrings:

Case 1.1: Assume $\boldsymbol{u_1} \neq \varepsilon$. We replace 5 of the c's with new symbols, and all instances of some symbol in 2 of the 3 copies of $\boldsymbol{u_1}$. This gives $\boldsymbol{y} = \boldsymbol{u_0} c \boldsymbol{u_1} c_2 \boldsymbol{u_2} c_3 \widehat{\boldsymbol{u_1}} c_4 \boldsymbol{u_4} c_5 \widehat{\boldsymbol{u_1}} c_6 \boldsymbol{u_6}$, resulting in $\pi \leq 7 = \delta$.

Case 1.2: Otherwise, assume $\boldsymbol{u_1} = \varepsilon$. The string now has the form: $\boldsymbol{x} = \boldsymbol{u_0} cc \boldsymbol{u_2} cc \boldsymbol{u_4} cc \boldsymbol{u_6}$. Much like in Lemma 6.5, when both $\boldsymbol{u_2}$ and $\boldsymbol{u_4}$ are non-empty this eliminates the possibility of runs from the single-mappings $i_2 \to i_3$ and $i_4 \to i_5$. We can then replace 5 of the c's with new symbols. We have $\pi \leq 5 = \delta$. Otherwise, when either $\boldsymbol{u_2}$ or $\boldsymbol{u_4}$ is empty, three single-mappings and a double-mapping reduce down to a single run, allowing us to introduce enough new symbols from replacing the c's alone.

Case 2: $(i_1, i_2, i_3) \to (i_4, i_5, i_6)$ and all single-mappings: By the triple-mapping, $\boldsymbol{u_1} = \boldsymbol{u_4}$ and $\boldsymbol{u_2} = \boldsymbol{u_5}$, so the string is $\boldsymbol{x} = \boldsymbol{u_0} c \boldsymbol{u_1} c \boldsymbol{u_2} c \boldsymbol{u_3} c \boldsymbol{u_1} c \boldsymbol{u_2} c \boldsymbol{u_6}$

Case 2.1: Assume $\boldsymbol{u_1} = \boldsymbol{u_2} = \varepsilon$, then the possible run from the single mapping $i_1 \to i_2$ is merged with the one from $i_2 \to i_3$, and $i_4 \to i_5$ is merged with $i_5 \to i_6$. By replacing 5 of the c's with new symbols, we have $\pi \leq 4 < \delta = 5$.

Case 2.2: Assume at least one of $\boldsymbol{u_1}$ and $\boldsymbol{u_2}$ are non-empty. Without loss of generality, assume $\boldsymbol{u_1} \neq \varepsilon$, giving $\boldsymbol{y} = \boldsymbol{u_0} c_1 \boldsymbol{u_1} c_2 \boldsymbol{u_2} c_3 \boldsymbol{u_3} c_4 \widehat{\boldsymbol{u_1}} c_5 \boldsymbol{u_2} c_6 \boldsymbol{u_6}$. This transformation destroys at most 6 runs and introduces 6 new symbols, so $\pi \leq 6 = \delta$. If $\boldsymbol{u_1} = \varepsilon$, then simply replace a symbol in $\boldsymbol{u_2}$ to obtain the same result.

Case 3: $(i_1, i_2, i_3) \to (i_4, i_5, i_6)$, one of $(i_1, i_2) \to (i_3, i_4)$ or $(i_3, i_4) \to (i_5, i_6)$ but not both, and all the single-mappings: Without loss of generality, we will assume that $(i_1, i_2) \to (i_3, i_4)$ exists. By the double- and triple-mappings, $\boldsymbol{u_1} = \boldsymbol{u_3} = \boldsymbol{u_4}$ and $\boldsymbol{u_2} = \boldsymbol{u_5}$, making the string $\boldsymbol{x} = \boldsymbol{u_0} c \boldsymbol{u_1} c \boldsymbol{u_2} c \boldsymbol{u_1} c \boldsymbol{u_1} c \boldsymbol{u_2} c \boldsymbol{u_6}$.

Case 3.1: Assume $\boldsymbol{u_1} \neq \varepsilon$. Replace each instance of a symbol in 2 copies of $\boldsymbol{u_1}$ and 5 of the c's with new symbols, giving $\boldsymbol{y} = \boldsymbol{u_0} c \boldsymbol{u_1} c_2 \boldsymbol{u_2} c_3 \widehat{\boldsymbol{u_1}} c_4 \widehat{\boldsymbol{u_1}} c_5 \boldsymbol{u_2} c_6 \boldsymbol{u_6}$.

This increases the number of distinct symbols by 7 while destroying at most 7 runs from the mappings ($\pi \leq 7 = \delta$).

Case 3.2: Otherwise, assume $\boldsymbol{u_1} = \varepsilon$, giving $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_2}ccc\boldsymbol{u_2}c\boldsymbol{u_6}$. This arrangement loses 1 possible run due to merging of mappings $i_3 \to i_4$ and $i_4 \to i_5$. If $\boldsymbol{u_2} = \varepsilon$, all the mappings collapse down to a single run. Assume then that $\boldsymbol{u_2} \neq \varepsilon$. This eliminates the possibility of the mapping $i_2 \to i_3$ reducing π from 7 down to 5. By replacing 5 of the c's with new symbols, we obtain the inequality $\pi \leq 5 = \delta$.

Case 4: The multi-mappings $(i_1, i_2, i_3) \to (i_4, i_5, i_6)$ and $(i_2, i_3) \to (i_4, i_5)$ exist, and so do all the single-mappings. By the double- and triple-mappings, $\boldsymbol{u_1} = \boldsymbol{u_2} = \boldsymbol{u_4} = \boldsymbol{u_5}$, giving $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_3}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_6}$.

Case 4.1: Assume $\boldsymbol{u_1} \neq \varepsilon$. We relabel each instance of a symbol in 3 copies of $\boldsymbol{u_1}$: $\boldsymbol{y} = \boldsymbol{u_0}c_1\boldsymbol{u_1}c_2\widehat{\boldsymbol{u_1}}c_3\boldsymbol{u_3}c_4\widehat{\boldsymbol{u_1}}c_5\widehat{\boldsymbol{u_1}}c_6\boldsymbol{u_6}$. This increases the number of distinct symbols by 8 while destroying at most 7 runs ($\pi \leq 7 < \delta = 8$).

Case 4.2: Otherwise, $\boldsymbol{u_1} = \varepsilon$, so $\boldsymbol{x} = \boldsymbol{u_0}ccc\boldsymbol{u_3}ccc\boldsymbol{u_6}$, and the runs from 2 single-mappings are lost through merging $i_1 \to i_2$ with $i_2 \to i_3$ and $i_4 \to i_5$ with $i_5 \to i_6$. Replacing 5 of the c's with new symbols is sufficient to give $\pi \leq 5 = \delta$.

Case 5: The multi-mappings $(i_1, i_2, i_3) \to (i_4, i_5, i_6)$, $(i_1, i_2) \to (i_3, i_4)$, and $(i_3, i_4) \to (i_5, i_6)$ exist, and so do all the single-mappings. From the double- and triple-mappings, $\boldsymbol{u_1} = \boldsymbol{u_2} = \boldsymbol{u_3} = \boldsymbol{u_4} = \boldsymbol{u_5}$, giving $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_6}$. All the possible runs are actually one long run, so the last c may be replaced with a new symbol without destroying any runs. This gives $\pi = 0 < \delta = 1$. □

From this point on, we will only consider cases which include at least one triple-mapping. Any cases which have only single- and double-mappings follow a structure similar to the cases already considered.

Lemma 6.8: *Let $\rho_{i'}(2i') = i'$ for $1 \leq i' < i$. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string for some $n - d = i$. Either $r(\boldsymbol{x}) = \rho_d(n) \leq i$ or $\boldsymbol{x}$ does not contain a 7-tuple.*

Proof. A 7-tuple of c's at positions $i_1 < \ldots < i_7$ can be involved in 9 runs, despite there being 12 possible mappings:

- triple-mappings: $(i_1, i_2, i_3) \rightarrow (i_4, i_5, i_6)$ and $(i_2, i_3, i_4) \rightarrow (i_5, i_6, i_7)$
- double-mappings: $(i_1, i_2) \rightarrow (i_3, i_4)$, $(i_2, i_3) \rightarrow (i_4, i_5)$, $(i_3, i_4) \rightarrow (i_5, i_6)$, and $(i_4, i_5) \rightarrow (i_6, i_7)$
- single-mappings: $i_1 \rightarrow i_2$, $i_2 \rightarrow i_3$, $i_3 \rightarrow i_4$, $i_4 \rightarrow i_5$, $i_5 \rightarrow i_6$, and $i_6 to i_7$

As with the double-mappings which overlap by more than p positions, if both triple-mappings are present, they correspond to the same run. Since having both triple-mappings cannot increase the possible number of runs above having only one, we assume without loss of generality, that if a triple-mapping is present, it is $(i_1, i_2, i_3) \rightarrow (i_4, i_5, i_6)$.

Let $\boldsymbol{x} = \boldsymbol{u_0} c \boldsymbol{u_1} c \boldsymbol{u_2} c \boldsymbol{u_3} c \boldsymbol{u_4} c \boldsymbol{u_5} c \boldsymbol{u_6} c \boldsymbol{u_7}$. There are 3 cases to consider:

Case 1: $(i_1, i_2) \rightarrow (i_3, i_4)$, $(i_4, i_5) \rightarrow (i_6, i_7)$, and all single-mappings (a total of 8 mappings). Due to the double-mappings, $\boldsymbol{u_1} = \boldsymbol{u_3}$ and $\boldsymbol{u_4} = \boldsymbol{u_6}$, giving $\boldsymbol{x}$ the form $\boldsymbol{x} = \boldsymbol{u_0} c \boldsymbol{u_1} c \boldsymbol{u_2} c \boldsymbol{u_1} c \boldsymbol{u_4} c \boldsymbol{u_5} c \boldsymbol{u_4} c \boldsymbol{u_7}$.

Case 1.1: Assume $\boldsymbol{u_1} \neq \varepsilon$ and $\boldsymbol{u_4} \neq \varepsilon$. Replace all instances of a symbol in 1 copy of each of $\boldsymbol{u_1}$ and $\boldsymbol{u_4}$, along with 6 of the c's with new symbols, yielding $\boldsymbol{y} = \boldsymbol{u_0} c \boldsymbol{u_1} c_2 \boldsymbol{u_2} c_3 \widehat{\boldsymbol{u_1}} c_4 \boldsymbol{u_4} c_5 \boldsymbol{u_5} c_6 \widehat{\boldsymbol{u_4}} c_7 \boldsymbol{u_7}$. This destroys at most 8 runs and introduces 8 new symbols ($\pi \leq 8 = \delta$).

Case 1.2: Assume $\boldsymbol{u_1} \neq \varepsilon$ while $\boldsymbol{u_4} = \varepsilon$. The string $\boldsymbol{x}$ now has the form $\boldsymbol{x} = \boldsymbol{u_0} c \boldsymbol{u_1} c \boldsymbol{u_2} c \boldsymbol{u_1} cc \boldsymbol{u_5} cc \boldsymbol{u_7}$. This eliminates the possibility of a run corresponding

to the mapping $i_5 \to i_6$, unless $\boldsymbol{u_5} = \varepsilon$, in which case 2 possible runs are lost to merging into one. Replacing all instances of a symbol in 1 copy of $\boldsymbol{u_1}$ along with 6 of the c's by new symbols gives $\pi \leq 7 = \delta$.

Case 1.3: Assume $\boldsymbol{u_1} = \varepsilon$ while $\boldsymbol{u_4} \neq \varepsilon$. This is a reversal of the previous case, and is satisfied accordingly.

Case 1.4: Assume $\boldsymbol{u_1} = \boldsymbol{u_4} = \varepsilon$. Therefore, $\boldsymbol{x} = \boldsymbol{u_0}cc\boldsymbol{u_2}ccc\boldsymbol{u_5}cc\boldsymbol{u_7}$. If $\boldsymbol{u_2} = \varepsilon$ or $\boldsymbol{u_5} = \varepsilon$, the possible runs from 5 mappings are lost due to being merged, thus reducing π to at most 3. Therefore assume that $\boldsymbol{u_2}$ and $\boldsymbol{u_5}$ are non-empty. In this case, the possibility of runs corresponding to the mappings $i_2 \to i_3$ and $i_5 \to i_6$ are eliminated, so relabeling 6 of the c's gives $\pi \leq 6 = \delta$.

Case 2: Let there be multi-mappings $(i_1, i_2, i_3) \to (i_4, i_5, i_6)$, $(i_4, i_5) \to (i_6, i_7)$, and all single-mappings (a total of 8 mappings). From the multi-mappings, $\boldsymbol{u_1} = \boldsymbol{u_4} = \boldsymbol{u_6}$ and $\boldsymbol{u_2} = \boldsymbol{u_5}$, so the string $\boldsymbol{x}$ must have the form $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_3}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_7}$.

Case 2.1: Assume $\boldsymbol{u_1} \neq \varepsilon$. We replace all instances of a symbol in 2 copies of $\boldsymbol{u_1}$, along with 6 of the c's with new symbols yielding $\boldsymbol{y} = \boldsymbol{u_0}c\boldsymbol{u_1}c_2\boldsymbol{u_2}c_3\boldsymbol{u_3}c_4\widehat{\boldsymbol{u_1}}c_5\boldsymbol{u_2}c_6\widehat{\boldsymbol{u_1}}c_7\boldsymbol{u_7}$. This gives $\pi \leq 8 = \delta$.

Case 2.2: Otherwise, $\boldsymbol{u_1} = \varepsilon$, so the string must have the form $\boldsymbol{x} = \boldsymbol{u_0}cc\boldsymbol{u_2}c\boldsymbol{u_3}cc\boldsymbol{u_2}cc\boldsymbol{u_7}$.

Case 2.2.1: Let $\boldsymbol{u_2} \neq \varepsilon$. This eliminates the possibility of a run corresponding to the mapping $i_5 \to i_6$, so by replacing all instances of a symbol in a $\boldsymbol{u_2}$ along with 6 of the c's with new symbols, we have $\pi \leq 7 = \delta$.

Case 2.2.2: Otherwise, $\boldsymbol{u_2} = \varepsilon$. This gives the string the form $\boldsymbol{x} = \boldsymbol{u_0}ccc\boldsymbol{u_3}cccc\boldsymbol{u_7}$. The possible runs collapse, leaving at most 3 runs.

Case 3: $(i_1, i_2, i_3) \to (i_4, i_5, i_6)$, $(i_1, i_2) \to (i_3, i_4)$, $(i_4, i_5) \to (i_6, i_7)$, and all

the single-mappings (a total of 9 mappings). From the multi-mappings, $\boldsymbol{u_1} = \boldsymbol{u_4} = \boldsymbol{u_6}$ and $\boldsymbol{u_2} = \boldsymbol{u_3} = \boldsymbol{u_5}$. This makes the string $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_7}$.

Case 3.1: Assume that $\boldsymbol{u_1}$ are $\boldsymbol{u_2}$ are both non-empty. Replacing all instances of a symbol in 2 copies of each of $\boldsymbol{u_1}$ and $\boldsymbol{u_2}$ along with 6 of the c's with new symbols, gives us $\pi \leq 9 < \delta = 10$.

Case 3.2: Assume $\boldsymbol{u_1} = \varepsilon$. The string is then $\boldsymbol{x} = \boldsymbol{u_0}cc\boldsymbol{u_2}c\boldsymbol{u_2}cc\boldsymbol{u_2}cc\boldsymbol{u_7}$. The possible run corresponding to the mapping $i_5 \to i_6$ is eliminated, so replacing all instances of a symbol in 2 copies of $\boldsymbol{u_2}$ along with 6 of the c's with new symbols is sufficient to give $\pi \leq 8 = \delta$.

Case 3.3: Assume $\boldsymbol{u_2} = \varepsilon$. The string is then $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}ccc\boldsymbol{u_1}cc\boldsymbol{u_1}c\boldsymbol{u_7}$. The runs corresponding to the mappings $i_2 \to i_3$ and $i_3 \to i_4$ are merged, and the possible run corresponding to the mapping $i_4 \to i_5$ is eliminated. Therefore, replacing all instances of a symbol in 2 copies of $\boldsymbol{u_1}$ along with 6 of the c's with new symbols is sufficient to give $\pi \leq 7 < \delta = 8$.

□

Lemma 6.9: *Let $\rho_{i'}(2i') = i'$ for $1 \leq i' < i$. Let $\boldsymbol{x}$ be a run-maximal (d, n)-string for some $n - d = i$. Either $r(\boldsymbol{x}) = \rho_d(n) \leq n - d$ or $\boldsymbol{x}$ does not contain an 8-tuple.*

Proof. Case 1: Let us take $(i_1, i_2, i_3) \to (i_4, i_5, i_6)$, $(i_5, i_6) \to (i_7, i_8)$, and all single-mappings (a total of 9 mappings). By the multi-mappings, the string has the form $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_3}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_6}c\boldsymbol{u_2}c\boldsymbol{u_8}$.

Case 1.1: Assume $\boldsymbol{u_2} \neq \varepsilon$, we can replace all instances of a symbol in 2 copies of $\boldsymbol{u_2}$ and 7 of the c's: $\boldsymbol{y} = \boldsymbol{u_0}c\boldsymbol{u_1}c_2\boldsymbol{u_2}c_3\boldsymbol{u_3}c_4\boldsymbol{u_1}c_5\widehat{\boldsymbol{u_2}}c_6\boldsymbol{u_6}c_7\widehat{\boldsymbol{u_2}}c_8\boldsymbol{u_8}$. This gives $\pi \leq 9 = \delta$.

Case 1.2: Otherwise, $\boldsymbol{u_2} = \varepsilon$, giving $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}cc\boldsymbol{u_3}c\boldsymbol{u_1}cc\boldsymbol{u_6}cc\boldsymbol{u_8}$. This

eliminates the possibility of a run from the mapping $i_6 \to i_7$. This means $\pi \leq 8$. However, we must consider further sub-cases.

Case 1.2.1: If $\boldsymbol{u_1} \neq \varepsilon$, we can replace all instances of a symbol in 1 of the copies of $\boldsymbol{u_1}$ along with 7 of the c's, giving $\boldsymbol{y} = \boldsymbol{u_0}c\boldsymbol{u_1}c_2c_3\boldsymbol{u_3}c_4\widehat{\boldsymbol{u_1}}c_5c_6\boldsymbol{u_6}c_7c_8\boldsymbol{u_8}$. This results in $\pi \leq 8 = \delta$.

Case 1.2.2: If $\boldsymbol{u_1} = \varepsilon$, the string has the structure $\boldsymbol{x} = \boldsymbol{u_0}ccc\boldsymbol{u_3}ccc\boldsymbol{u_6}cc\boldsymbol{u_8}$. This structure eliminates the possible run from the mapping $i_6 \to i_7$. Additionally, the runs corresponding to the single mappings $i_1 \to i_2$ and $i_2 \to i_3$ are merged, along with the runs corresponding to the mappings $i_4 \to i_5$ and $i_5 \to i_6$. This reduces the maximum number of runs to $\pi \leq 6$. By relabeling 7 of the c's, we obtain $\pi \leq 6 < \delta = 7$. This assumes that $\boldsymbol{u_3}$ and $\boldsymbol{u_6}$ are non-empty, but if either is empty, sufficient runs are lost through merging.

Case 2: Let the string have the multi-mappings $(i_1, i_2, i_3) \to (i_4, i_5, i_6)$, $(i_1, i_2) \to (i_3, i_4)$, $(i_5, i_6) \to (i_7, i_8)$, and all single-mappings (a total of 10 mappings). By the multi-mappings, the string is $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_6}c\boldsymbol{u_2}c\boldsymbol{u_8}$.

Case 2.1: Assume $\boldsymbol{u_1}$ and $\boldsymbol{u_2}$ are both non-empty. We can then replace all instances of a symbol in 2 copies of each, along with 7 of the c's: $\boldsymbol{y} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\widehat{\boldsymbol{u_1}}c\widehat{\boldsymbol{u_1}}c\widehat{\boldsymbol{u_2}}c\boldsymbol{u_6}c\widehat{\boldsymbol{u_2}}c\boldsymbol{u_8}$. This results in $\pi \leq 10 < \delta = 11$.

Case 2.2: Assume $\boldsymbol{u_1} = \varepsilon$ and $\boldsymbol{u_2} \neq \varepsilon$. So $\boldsymbol{x} = \boldsymbol{u_0}cc\boldsymbol{u_2}ccc\boldsymbol{u_2}c\boldsymbol{u_6}c\boldsymbol{u_2}c\boldsymbol{u_8}$. This eliminates the possibility of a run corresponding to the mapping $i_2 \to i_3$, and merges the runs corresponding to $i_3 \to i_4$ and $i_5 \to i_6$, so $\pi \leq 8$. We replace all instances of a symbol in 2 of the copies of $\boldsymbol{u_2}$, and $\boldsymbol{y} = \boldsymbol{u_0}cc_2\boldsymbol{u_2}c_3c_4c_5\widehat{\boldsymbol{u_2}}c_6\boldsymbol{u_6}c_7\widehat{\boldsymbol{u_2}}c_8\boldsymbol{u_8}$. This results in $\pi \leq 8 < \delta = 9$.

Case 2.3: Assume $\boldsymbol{u_1} \neq \varepsilon$, and $\boldsymbol{u_2} = \varepsilon$. So $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}cc\boldsymbol{u_1}c\boldsymbol{u_1}cc\boldsymbol{u_6}cc\boldsymbol{u_8}$. This eliminates the possibility of a run corresponding to the mapping $i_6 \to$

i_7,unless $\boldsymbol{u_6} = \varepsilon$, which results in 3 possible runs being merged. We replace all instances of a symbol in 2 copies of $\boldsymbol{u_1}$, along with 7 of the c's with new symbols, giving $\boldsymbol{y} = \boldsymbol{u_0}c\boldsymbol{u_1}c_2c_3\widehat{\boldsymbol{u_1}}c_4\widehat{\boldsymbol{u_1}}c_5c_6\boldsymbol{u_6}c_7c_8\boldsymbol{u_8}$. This results in $\pi \leq 9 = \delta$.

Case 2.4: Assume $\boldsymbol{u_1} = \boldsymbol{u_2} = \varepsilon$. We then have $\boldsymbol{x} = \boldsymbol{u_0}cccccc\boldsymbol{u_6}cc\boldsymbol{u_8}$, merging 5 runs corresponding to the single mappings, and preventing the possible run corresponding to $(i_1, i_2) \rightarrow (i_3, i_4)$ because its generator would not be primitive. Therefore, by replacing 7 of the c's with new symbols, we obtain$\pi \leq 5 < \delta = 7$.

Case 3: Let the string have the multi-mappings $(i_1, i_2, i_3) \rightarrow (i_4, i_5, i_6)$, $(i_2, i_3) \rightarrow (i_4, i_5)$, $(i_5, i_6) \rightarrow (i_7, i_8)$, and all single-mappings, a total of 10 mappings. By the multi-mappings, the string is $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_3}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_6}c\boldsymbol{u_1}c\boldsymbol{u_8}$.

Case 3.1: Assume $\boldsymbol{u_1} \neq \varepsilon$. Then we can replace all instances of a symbol in 4 copies of $\boldsymbol{u_1}$, along with 7 of the c's with new symbols, yielding $\boldsymbol{y} = \boldsymbol{u_0}c\boldsymbol{u_1}c_2\widehat{\boldsymbol{u_1}}c_3\boldsymbol{u_3}c_4\widehat{\boldsymbol{u_1}}c_5\widehat{\boldsymbol{u_1}}c_6\boldsymbol{u_6}c_7\widehat{\boldsymbol{u_1}}c_8\boldsymbol{u_8}$. This results in $\pi \leq 10 < \delta = 11$.

Case 3.2: If $\boldsymbol{u_1} = \varepsilon$, the string has the form $\boldsymbol{x} = \boldsymbol{u_0}ccc\boldsymbol{u_3}ccc\boldsymbol{u_6}cc\boldsymbol{u_8}$. This merges the runs corresponding to the mappings $i_1 \rightarrow i_2$ with $i_2 \rightarrow i_3$, and $i_4 \rightarrow i_5$ with $i_5 \rightarrow i_6$, and eliminates the possible run corresponding to the mapping $(i_2, i_3) \rightarrow (i_4, i_5)$, unless $\boldsymbol{u_3} = \varepsilon$, in which case 2 more runs are lost through merging. This gives $\pi \leq 7 = \delta$ by just replacing 7 of the c's with new symbols.

Case 4: Let the string have the multi-mappings $(i_1, i_2, i_3) \rightarrow (i_4, i_5, i_6)$, $(i_3, i_4) \rightarrow (i_5, i_6)$, $(i_5, i_6) \rightarrow (i_7, i_8)$, and all single-mappings, a total of 10 mappings. By the multi-mappings, the string is $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_6}c\boldsymbol{u_2}c\boldsymbol{u_8}$.

Case 4.1: If $\boldsymbol{u_2} \neq \varepsilon$, replace all instances of a symbol in 3 copies of $\boldsymbol{u_2}$ with new symbols, giving $\pi \leq 10 = \delta$.

Case 4.2: If $\boldsymbol{u_2} = \varepsilon$, the runs corresponding to the single mappings $i_2 \rightarrow i_3$ and $i_3 \rightarrow i_4$ are merged, giving 9 possible runs. If $\boldsymbol{u_1} \neq \varepsilon$, the mapping

corresponding to the $i_4 \rightarrow i_5$ is also prevented, giving 8 possible runs. By replacing all instances of some symbol in 1 copy of $\boldsymbol{u_1}$ along with 7 of the c's gives $\pi \leq 8 = \delta$. If $\boldsymbol{u_1} = \varepsilon$, 5 possible runs are lost through merging, making the process trivial.

Case 5: Let the string have the multi-mappings $(i_1, i_2, i_3) \rightarrow (i_4, i_5, i_6)$, $(i_1, i_2) \rightarrow (i_3 \rightarrow i_4)$, $(i_3, i_4) \rightarrow (i_5 \rightarrow i_5)$, $(i_5, i_6) \rightarrow (i_7, i_8)$, and all single-mappings,a total of 11 mappings. By the multi-mappings, the string has the form $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_1}c\boldsymbol{u_6}c\boldsymbol{u_1}c\boldsymbol{u_8}$. If $\boldsymbol{u_1} \neq \varepsilon$, replace all instances of a symbol in 5 copies of $\boldsymbol{u_1}$, along with 7 of the c's with new symbols, giving $\pi \leq 11 < \delta = 12$. Otherwise, $\boldsymbol{u_1} = \varepsilon$, and 4 single runs are lost through being merged, giving $\pi \leq 7 = \delta$.

Case 6: Let the string have the multi-mapping $(i_1, i_2, i_3, i_4) \rightarrow (i_5, i_6, i_7, i_8)$ and all single-mappings, a total of 8 mappings, By the quadruple-mapping, the string has the form $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_3}c\boldsymbol{u_4}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_3}c\boldsymbol{u_8}$. We cannot have $\boldsymbol{u_1}$, $\boldsymbol{u_2}$ and $\boldsymbol{u_3}$ all be empty or several runs are merged, so we replace all instances of a symbol in at least 1 of them, along with 7 of the c's with new symbols. This gives $\pi \leq 8 \leq \delta \leq 10$.

Case 7: Let the string have the multi-mappings $(i_1, i_2, i_3, i_4) \rightarrow (i_5, i_6, i_7, i_8)$, $(i_1, i_2) \rightarrow (i_3, i_4)$, and all single-mappings, initially giving 9 runs. By the multi-mappings, the string has the form $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_4}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_8}$. This gives a total of 10 runs. See Case 9, where this string structure is solved under the stronger assumption of 11 mappings.

Case 8: Let the string have the multi-mappings $(i_1, i_2, i_3, i_4) \rightarrow (i_5, i_6, i_7, i_8)$, $(i_2, i_3) \rightarrow (i_4, i_5)$, and all single-mappings, a total of 9 mappings. By the multi-mappings, the string has the form $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_3}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_3}c\boldsymbol{u_8}$.

Case 8.1: Assume $\boldsymbol{u_2} \neq \varepsilon$. Replace all instances of a symbol in 2 copies of

it, along with 7 of the c's with new symbols, giving $\pi \leq 9 = \delta$.

Case 8.2: Otherwise, $\boldsymbol{u_2} = \varepsilon$, giving $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}cc\boldsymbol{u_3}cc\boldsymbol{u_1}cc\boldsymbol{u_3}c\boldsymbol{u_8}$. This eliminates the possibility of a run corresponding to the single mappings $i_3 \to i_4$ and $i_5 \to i_6$, unless $\boldsymbol{u_1}$ or $\boldsymbol{u_3}$ is empty; in either case, 2 possible runs are lost through merging. This gives $\pi \leq 7$, which is achievable by replacing 7 of the c's.

Case 9: Let the string have the multi-mappings $(i_1, i_2, i_3, i_4) \to (i_5, i_6, i_7, i_8)$, $(i_3, i_4) \to (i_5, i_6)$, and all single-mappings, a total of 9 mappings. By the multi-mappings, the string has the form $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_4}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_8}$. This same configuration was previously created in Case 7 and is solved in Case 10 where it is assumed to have 11 mappings.

Case 10: Let the string have multi-mappings $(i_1, i_2, i_3, i_4) \to (i_5, i_6, i_7, i_8)$, $(i_1, i_2) \to (i_3, i_4)$, $(i_3, i_4) \to (i_5, i_6)$, and all single-mappings, initially giving 10 runs. Having one of the double-mappings completely enclosed within one side of the quadruple-mapping means it exists on the other side of the quadruple-mapping too, so $(i_5, i_6) \to (i_7, i_8)$ also exists. By the multi-mappings, the string has the form $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_4}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_8}$. This gives a total of 11 runs.

Case 10.1: Assume $\boldsymbol{u_1} \neq \varepsilon$.

Case 10.1.1: Assume $\boldsymbol{u_2} \neq \varepsilon$. In this case, replace all instances of a symbol in 3 of the copies of $\boldsymbol{u_1}$ and one copy of $\boldsymbol{u_2}$, along with 7 of the c's, giving $\pi \leq 11 = \delta$.

Case 10.1.2: Otherwise $\boldsymbol{u_2} = \varepsilon$. This gives the string the structure $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}cc\boldsymbol{u_1}c\boldsymbol{u_4}c\boldsymbol{u_1}cc\boldsymbol{u_1}c\boldsymbol{u_8}$. If $\boldsymbol{u_4} = \varepsilon$, the whole string collapses down to 4 runs. Assume then that $\boldsymbol{u_4} \neq \varepsilon$. This eliminates the possibility of a run from mapping $(i_3, i_4) \to (i_5, i_6)$, making $\pi \leq 10$. We can then obtain $\delta = 10$ by replacing 7 of the c's and all copies of a symbol in 3 of the copies of $\boldsymbol{u_1}$.

Case 10.2: Otherwise, $\boldsymbol{u_1} = \varepsilon$, so $\boldsymbol{x} = \boldsymbol{u_0}cc\boldsymbol{u_2}cc\boldsymbol{u_4}cc\boldsymbol{u_2}cc\boldsymbol{u_8}$. This eliminates the possibility of the single-mappings $i_2 \to i_3$, $i_4 \to i_5$, and $i_6 \to i_7$, unless $\boldsymbol{u_2}$ or $\boldsymbol{u_4}$ are empty, in which case 4 or 2 possible runs are lost through merging, respectively. This reduces the number of possible runs to at most 8, and we can achieve $\pi \leq 7 = \delta$ by simply replacing 7 of the c's with new symbols and all copies of a symbol in one of the copies of $\boldsymbol{u_2}$.

Case 11: Let the string have multi-mappings $(i_1, i_2, i_3, i_4) \to (i_5, i_6, i_7, i_8)$, $(i_1, i_2) \to (i_3, i_4)$, $(i_3, i_4) \to (i_5, i_6)$, and all single-mappings, a total of 10 mappings. By the multi-mappings, the string is $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_4}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_8}$. This same configuration was previously discussed in Case 10 when we assumed it had 11 mappings, so it can be satisfied again in this case.

Case 12: Let the string have multi-mappings $(i_1, i_2, i_3, i_4) \to (i_5, i_6, i_7, i_8)$, $(i_2, i_3) \to (i_4, i_5)$, $(i_4, i_5) \to (i_6, i_7)$, and all single-mappings, a total of 10 mappings. By the multi-mappings, $\boldsymbol{u_1} = \boldsymbol{u_5}$, $\boldsymbol{u_2} = \boldsymbol{u_4} = \boldsymbol{u_6}$, and $\boldsymbol{u_3} = \boldsymbol{u_7}$, so the string has the form $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_3}c\boldsymbol{u_2}c\boldsymbol{u_1}c\boldsymbol{u_2}c\boldsymbol{u_3}c\boldsymbol{u_8}$.

Case 12.1: Assume $\boldsymbol{u_2} \neq \varepsilon$ and one of $\boldsymbol{u_1}$ or $\boldsymbol{u_3}$ is non-empty. Replace all instances of a symbol in 1 copy of $\boldsymbol{u_1}$ or $\boldsymbol{u_3}$ and 2 copies of $\boldsymbol{u_2}$, along with 7 of the c's with new symbols, giving $\pi \leq 10 \leq \delta \leq 11$.

Case 12.2: If $\boldsymbol{u_2} \neq \varepsilon$, but both $\boldsymbol{u_1} = \boldsymbol{u_3} = \varepsilon$, the string has the form $\boldsymbol{x} = \boldsymbol{u_0}cc\boldsymbol{u_2}cc\boldsymbol{u_2}cc\boldsymbol{u_2}cc\boldsymbol{u_8}$. The possibilities of runs corresponding to the mappings $i_2 \to i_3$, $i_4 \to i_5$, and $i_6 \to i_7$ are eliminated, so by replacing 7 of the c's we achieve $\pi \leq 7 = \delta$.

Case 12.3: If $\boldsymbol{u_2} = \varepsilon$, the string has the form $\boldsymbol{x} = \boldsymbol{u_0}c\boldsymbol{u_1}cc\boldsymbol{u_3}cc\boldsymbol{u_1}cc\boldsymbol{u_3}c\boldsymbol{u_8}$. The possibility of runs corresponding to the mappings $i_3 \to i_4$ and $i_5 \to i_6$ is eliminated. Since neither $\boldsymbol{u_1}$ nor $\boldsymbol{u_3}$ are empty or many more possible runs are

lost through merging, raising 1 copy of each of these gives $\pi \leq 8 < \delta = 9$.

Case 13: Let the string have multi-mappings $(i_1, i_2, i_3, i_4) \to (i_5, i_6, i_7, i_8)$, $(i_1, i_2) \to (i_3, i_4)$, $(i_4, i_5) \to (i_6, i_7)$, and all single-mappings, a total of 10 mappings. By the multi-mappings, the string is $\boldsymbol{x} = \boldsymbol{u_0 c u_1 c u_2 c u_1 c u_2 c u_1 c u_2 c u_1 c u_8}$. Therefore, $\boldsymbol{x} = \boldsymbol{u_0}(c\boldsymbol{u_1}c\boldsymbol{u_2})^3 c\boldsymbol{u_1}c\boldsymbol{u_8}$, so we can replace the first c only destroying at most a single run ($\pi \leq \delta = 1$).

Case 14: Let the string have multi-mappings $(i_1, i_2, i_3, i_4) \to (i_5, i_6, i_7, i_8)$, $(i_1, i_2, i_3) \to (i_4, i_5, i_6)$, and all single-mappings, a total of 9 mappings. By the multi-mappings, the string has the form $\boldsymbol{x} = \boldsymbol{u_0 c u_1 c u_1 c u_3 c u_1 c u_1 c u_1 c u_3 c u_8}$. This merges the possible runs from $i_1 \to i_2$ and $i_2 \to i_3$, as well as $i_4 \to i_5$, $i_5 \to i_6$, and $i_6 \to i_7$, leaving 6 possible runs. Replacing 7 of the c's with new symbols is sufficient to give $\pi \leq 6 < \delta = 7$.

In addition, we can layer up to 2 double-mappings on top of the triple- and quadruple-mappings, giving a total of 11 mappings. Again, there are at least 3 possible runs lost through merging, giving at most 8 runs. Since $\boldsymbol{u_1}$ and $\boldsymbol{u_3}$ cannot both be empty, we can replace all instances of a symbol in 1 of the copies of $\boldsymbol{u_1}$ or $\boldsymbol{u_3}$. Therefore, $\pi \leq 8 \leq \delta$.

Case 15: Let the string have multi-mappings $(i_1, i_2, i_3, i_4) \to (i_5, i_6, i_7, i_8)$, $(i_2, i_3, i_4) \to (i_5, i_6, i_7)$, and all single-mappings, a total of 9 mappings. By the multi-mappings, the string has the form $\boldsymbol{x} = \boldsymbol{u_0 c u_1 c u_1 c u_1 c u_4 c u_1 c u_1 c u_1 c u_8}$. This merges the possible runs corresponding to $i_1 \to i_2$, $i_2 \to i_3$, and $i_3 \to i_4$, along with $i_5 \to i_6$, $i_6 \to i_7$, and $i_7 \to i_8$, decreasing the maximum number of runs by 4. By replacing 7 of the c's with new symbols, we get $\pi \leq 5 < 7 = \delta$.

Once again, we can also layer up to 2 additional double-mappings on top of the triple- and quadruple-mappings. However, we are still limited to 11 possible

runs. After discounting the 4 possible runs lost to merging, we have $\pi \leq 7 = \delta$ from replacing 7 of the c's with new symbols. □

This completes the last lemma which supports Proposition 6.1. Taken together, they show that a (d, n)-string $\boldsymbol{x}$ with $n - d = i$ in the first such column and $r(\boldsymbol{x}) > i$ cannot have a 2- through 8-tuple.

A corollary of this result is that the first counter-example on the main diagonal must have many singletons, given that it must have either 1 or at least 9 of each symbol.

Corollary 6.10: **[4]** *Let $\rho_{i'}(2i') = i'$ for $2 \leq i' < i$. Let $\boldsymbol{x}$ be a run-maximal $(i, 2i)$-string. Either $r(\boldsymbol{x}) = i$ or $\boldsymbol{x}$ has at least $\lceil \frac{7i}{8} \rceil$ singletons.*

Proof. Let m_1 denote the number of singletons, and m_2 the number of non-singleton symbols of $\boldsymbol{x}$. We have $m_1 + 9m_2 \leq 2i$ and $m_1 + m_2 = i$, which implies that $m_2 \leq i/8$ and so $m_1 \geq \lceil \frac{7i}{8} \rceil$. □

Recall that from Proposition 2.9, if some $\rho_d(n) > n - d$, then there exists a $\rho_{d'}(2d') > d$, on the main diagonal. Using the fact that any such string in the first column with a counter-example must have at least 9 copies of each non-singleton symbol, we can provide another equivalent statement.

Corollary 6.11: *We have $\{\rho_d(n) \leq n - d$ for $2 \leq d \leq n\} \Leftrightarrow \{\rho_d(9d) \leq 8d$ for $d \geq 2\}$.*

The lemmas in this section, while involving many cases, have a certain structural similarity. We suggest that it may be possible to automate the proof of such lemmas in order to eliminate larger k-tuples. Any extension would further improve the speed of computing $\rho_d(2d)$, as outlined in Section 6.1.

6.1 Efficient computation of $\rho_d(2d)$

There are two related ramifications from Proposition 6.1. The first is that this result provides a completely non-computational proof that $\rho_d(n) \leq n - d$ for all $n - d \leq 15$. Coupled with the computation of $\rho_2(n)$ for $18 \leq n \leq 26$, we can show that $\rho_d(n) \leq n - d$ for all $n - d \leq 24$. As we have also computed all values for $\rho_d(n)$ for $2 \leq d \leq 3$ and $n - d \leq 32$, we show that $\rho_d(n) \leq n - d$ for all $n - d \leq 32$.

Corollary 6.12: *For $n - d \leq 23$, $\rho_d(n) \leq n - d$.*

Proof. Let $\boldsymbol{x}$ be a run-maximal $(23, 46)$-string. By Corollary 6.10, either $r(\boldsymbol{x}) = 23$ and so $\rho_{23+\epsilon}(46 + \epsilon) \leq 23$ ($\epsilon > -23$), or $\boldsymbol{x}$ has at least $\lceil \frac{7i}{8} \rceil = 21$ singletons. By Lemma 3.10, we can group all the singletons at the end of the string. As the independent computations by multiple people have shown that $\rho_2(25) = 19$, the maximum number of runs for a $(23, 46)$-string is 19 if it contains any singletons. □

We can use these results to speed the computational confirmation that columns of the $(d, n - d)$-table satisfy the maximum number of runs conjecture. Since by Lemma 3.10 we can move all singletons to the end of the string where they cannot affect the number of runs, we need only consider the repetitive part which requires at least 9 copies of each symbol. That is, we can assume a run-maximal $(d, 2d)$-string $\boldsymbol{x}$ has the form $\boldsymbol{x} = \boldsymbol{yz}$ where $\boldsymbol{z}$ consists of all the singletons in $\boldsymbol{x}$. Therefore, we only need to consider the strings $\boldsymbol{y}$ which are r-covered and in which every symbol occurs at least 9 times.

There is an additional restriction we can put on the strings we need to generate. While the string must be r-covered, that r-cover must also satisfy the ***parity condition***.

Definition 6.13: **[2]** *The r-cover* $\{ S_i = (s_i, p_i, 2) \mid 1 \leq i \leq m \}$ *of* (d, n)*-string* $\boldsymbol{x}$ *satisfies the* ***parity condition*** *if for every* $1 \leq i < m$, $\mathcal{A}(\boldsymbol{x}[1..s_{i+1} - 1]) \cap \mathcal{A}(\boldsymbol{x}[s_i + 2p_i..n]) \subseteq \mathcal{A}(\boldsymbol{x}[s_{i+1}..s_i + 2p_i - 1])$.

To satisfy the parity condition, for every overlap between elements in the r-cover, any symbol which occurs both before and after the overlap must also occur within the overlap of the elements.

Lemma 6.14: **[2]** *Let* $\rho_i(2i) = i$ *for* $i < d$. *The singleton-free portion of a run-maximal* $(d, 2d)$*-string* $\boldsymbol{x}$ *with all its singletons at the end has an r-cover satisfying the parity condition.*

Proof. Assume that $\boldsymbol{x}$ has $v \leq d-2$ singletons, all at the end, by Lemma 3.10. Then $\boldsymbol{x} = \boldsymbol{yz}$ with $\boldsymbol{y} = \boldsymbol{x}[1..2d - v]$ and $\boldsymbol{z} = \boldsymbol{x}[2d - v + 1..2d]$. Then, $\boldsymbol{y}$ is a $(d-v, 2d-v)$-string and $\rho_d(2d) = r(\boldsymbol{x}) = r(\boldsymbol{y}) = \rho_{d-v}(2d-v)$. First we will show that $\boldsymbol{y}$ has an r-cover, and then we will show that the r-cover satisfies the parity condition.

Let us assume that $\boldsymbol{y}$ is not r-covered with $\{ S_i \}$ for $1 \leq i \leq m$. Therefore, for some $1 \leq i \leq 2d - n$, $k_i(\boldsymbol{x}) = 0$. If $i = 1$ or $i = 2d - v$, then $r(\boldsymbol{x}) = \rho_d(2d) = \rho_d(2d-1) = \rho_{d-1}(2d-2)$ which, by Property 2.4, contradicts $\boldsymbol{x}$ being run-maximal. Therefore, consider $1 < i < 2d - v$. If $\mathcal{A}(\boldsymbol{y}[1..i-1]) = \mathcal{A}(\boldsymbol{y}[i+1..2d-v])$, then $r(\boldsymbol{y}) = r(\boldsymbol{y}[1..i - 1]) + r(\boldsymbol{y}[i + 1..2d - v]) \leq \rho_d(i - 1) + \rho_d(2d - v - i) \leq (i - 1 - d) + (2d - v - i - d) = d - v - 1 < d$. Again, a contradiction that $\boldsymbol{x}$ is run-maximal as we can always achieve d runs with d adjacent pairs. Therefore, there must be some symbol c so that either $c \in \mathcal{A}(\boldsymbol{y}[1..i-1]) \setminus \mathcal{A}(\boldsymbol{y}[i+1..2d-v])$ or $c \in \mathcal{A}(\boldsymbol{y}[i+1..2d-v]) \setminus \mathcal{A}(\boldsymbol{y}[1..i-1])$. Without loss of generality, assume $c \in \mathcal{A}(\boldsymbol{y}[1..i-1]) \setminus \mathcal{A}(\boldsymbol{y}[i+1..2d-v])$. We proceed in a manner similar to the proof of

Lemma 3.7. Permute the alphabet of $\boldsymbol{y}[1..i-1]$ to $\widetilde{\boldsymbol{y}}[1..i-1]$ such that $\widetilde{\boldsymbol{y}}[i-1] = c$. Then let $\boldsymbol{y'} = \boldsymbol{y}[\boldsymbol{i}]\widetilde{\boldsymbol{y}}[1..i-1]\boldsymbol{y}[i+1..2d-v]$, and so $r(\boldsymbol{y}) = r(y')$ by the run-maximality of $\boldsymbol{x}$. However, this results in $k_1(\boldsymbol{y'}) = 0$, which was already eliminated by contradiction. Therefore, $\boldsymbol{y}$ has an r-cover.

Now we turn to the proof that the r-cover of $\boldsymbol{y}$ must satisfy the parity condition. Let us assume that the r-cover of $\boldsymbol{y}$ does not satisfy the parity condition. Assume that for some $1 \leq t < m$ there is a symbol c such that $c \in \mathcal{A}(\boldsymbol{y}[1..s_{t+1}-1])$ and $c \in \mathcal{A}(\boldsymbol{y}[s_t + 2p_t..2d - v])$ but $c \notin \mathcal{A}(\boldsymbol{y}[s_{t+1}..s_t + 2p_t - 1]$. This includes the possibility that $s_t + 2p_t = s_{t+1}$ and so $\mathcal{A}(\boldsymbol{y}[s_{t+1}..s_t + 2p_t - 1]) = \emptyset$. Let us replace all copies of c in $\boldsymbol{y}[1..s_t - 1]$ with $\hat{c} \notin \mathcal{A}(\boldsymbol{x})$. This results in a new $(d+1, n)$-string $\hat{\boldsymbol{y}}$ with $r(\hat{\boldsymbol{y}}) = r(\boldsymbol{y})$. This gives $\rho_d(2d) = r(\boldsymbol{y}) = r(\hat{\boldsymbol{y}}) \leq \rho_{d+1}(2d) = \rho_{d-1}(2d-2)$. However, by Property 2.4, $\rho - d(2d) > \rho_{d-1}(2d-2)$, a contradiction. Therefore, there cannot be a symbol c found in both $\mathcal{A}(\boldsymbol{y}[1..s_t+2p_t-1])$ and $\mathcal{A}(\boldsymbol{y}[s_{t+1}..2d-v])$ but not in $\mathcal{A}(\boldsymbol{y}[s_{t+1}..s_t + 2p_t - 1])$.

□

While we required a computational check to eliminate the possibility of adjacent r-cover in the run-maximal strings, we can eliminate them *a priori* in this case.

Corollary 6.15: [2] *Let $\rho_i(2i) = i$ for all $i < d$. Either $\rho_d(2d) = d$ or every run-maximal $(d, 2d)$-string $\boldsymbol{x}$ with $v > 0$ singletons has an overlapping r-cover satisfying the parity condition.*

Proof. Once again, assume that $\boldsymbol{x}$ has $v \leq d-2$ singletons, all at the end, by Lemma 3.10. Then $\boldsymbol{x} = \boldsymbol{yz}$ with $\boldsymbol{y} = \boldsymbol{x}[1..2d-v]$ and $\boldsymbol{z} = \boldsymbol{x}[2d-v+1..2d]$. By Lemma 6.14 if $\boldsymbol{y}$ has an adjacent r-cover $\{ S_i \}$ for $1 \leq i \leq m$ such that $s_t+2p_t = s_{t_1}$

for some $1 \leq t < m$, then $\mathcal{A}(\boldsymbol{y}[1..s_t + 2p_t - 1]) \cap \mathcal{A}(\boldsymbol{y}[s_{t+1}..2d - v]) = \emptyset$. Let $d_1 = d(\boldsymbol{y}[1..s_t + 2p_t - 1])$, $n_1 = s_t + 2p_t - 1$, $d_2 = d - v - d_1$, and $n_2 = 2d - v - n_1$. Then $r(\boldsymbol{y}) = r(\boldsymbol{y}[1..s_t + 2p_t - 1]) + r(\boldsymbol{y}[s_{t+1}..2d - v]) \leq \rho_{d_1}(n_1) + \rho_{d_2}(n_2) \leq n_1 - d_1 + n_2 - d_2 = d$. □

A computer program could be written to generate only the strings with overlapping r-covers which satisfy the parity condition and have at least copies of each symbol. Such a program could then be used to show that the maximum number of runs conjecture holds for columns of the $(d, n - d)$-table even where it is infeasible to compute the actual $\rho_d(n)$ values.

Chapter 7

Conclusion

We have explored the function $\rho_d(n)$, the maximum number of runs over all strings of length n with exactly d distinct symbols. We presented the values of $\rho_d(n)$ in a $(d, n-d)$ table, in order to illustrate properties of the $\rho_d(n)$ function, including avenues for recursion involving both variables d and n. Then we explored structural properties of run-maximal strings in the form of the core vector and r-cover, and introduced the concept of $\rho_d^-(n)$-density. We combined these ideas to develop an algorithm to compute $\rho_d(n)$ values in a two stage process. We exploit the availability of a sufficient lower bound to reduce the search space enabling us to exhaustively search only those runs which could exceed this lower bound. The implementation of this algorithm was used to extend the best known computations from $\rho_2(60)$ to $\rho_2(74)$, and to compute values of $\rho_d(n)$ for $d \geq 3$ which had not previously been considered. Finally, we discuss the structural properties of a shortest (d, n)-string $\boldsymbol{x}$ such that $r(\boldsymbol{x}) > n - d$, should such a string exist. These properties can be used in a computer program to determine if $\rho_d(n) \leq n - d$ holds for larger values of d and n than for which the values of $\rho_d(n)$ can be computed.

7.1 Connection with the maximum number of distinct squares conjecture

Recall from Section 1.8.2 that $s(\boldsymbol{x})$ is the number of primitively rooted distinct squares in $\boldsymbol{x}$ and $\sigma(n)$ is the maximum value of $s(\boldsymbol{x})$ over all strings of length n. Having presented our results on runs, we briefly touch on the recent results by Deza, Franek, and Jiang with respect to square-maximal strings and discuss the similarities with our work.

Deza, Franek, and Jiang [15], inspired by our new results on run-maximal strings, began exploring a similar extension for square-maximal strings. They introduce the function $\sigma_d(n)$, the maximum number of primitively rooted distinct squares over all strings of length n with d distinct symbols. They present the values of $\sigma_d(n)$ in a $(d, n-d)$-table, just as we do for $\rho_d(n)$ values. Many properties of the $\rho_d(n)$ function and therefore the $(d, n-d)$-table for runs correspond to properties of $\sigma_d(n)$ and the $(d, n-d)$-table for distinct squares.

- Values are non-decreasing along a row: $\sigma_d(n) \leq \sigma_d(n+1)$ [15] corresponding to Property 2.1.
- Values are non-decreasing down a column: $\sigma_d(n) \leq \sigma_{d+1}(n+1)$ [15] corresponding to Property 2.2.
- Values are increasing down a diagonal to the right: $\sigma_d(n) < \sigma_{d+1}(n+2)$ [15] corresponding to Property 2.4.
- Every value below the main diagonal in a column is equal to the value on the main diagonal in the same column: $\sigma_d(n) = \sigma_{n-d}(2n-2d)$ for $2 \leq d \leq n \leq 2d$ [15] corresponding to Property 2.8.

- If there is a counter-example, it can be drawn to the main diagonal: $\sigma_d(n) > n - d \leftrightarrow \sigma_{d'}(2d')$ [15] corresponding to Corollary 2.10.

- The two values immediately above the main diagonal are equal: $\sigma_d(2d+1) = \sigma_{d-1}(2d)$ [14] corresponding to part of Proposition 2.13.

One property of run-maximal strings which does not apply to square-maximal strings is Property 2.3–that increasing the length by 2 while holding the number of distinct symbols constant always results in an increase of the maximum number of runs. This property cannot apply to square-maximal strings as appending either an aa or a bb to the end of the string will not increase the number of distinct squares if both squares already exist in the string.

Conversely, there is a limit to how much the maximum number of distinct squares can increase as the string is extended, which is only conjectured to hold in the case of run-maximal strings. That is, $\sigma_d(n+1) - \sigma_d(n) \leq 2$. The increase moving down the main diagonal can also be bounded: $\sigma_{d+1}(2d+2) - \sigma_d(2d) \leq 2$ [14].

In Proposition 6.1, we are able to eliminate the possibility of 2- through 8-tuples in a conceptual first (d, n)-string $\boldsymbol{x}$ with $r(\boldsymbol{x}) > n - d$. Deza, Franek, and Jiang [15] have been able to make some steps in this direction, but the approach has proven to be more complicated for square-maximal strings. They are able to eliminate pairs, and some forms of other k-tuples.

Just as we develop the concept of an r-cover, Deza, Franek, and Jiang [14] introduce the s-cover, an analogous structure dealing with distinct squares. They exploit the s-cover structure and a density argument, combined with a heuristic search, in order to speed the computation of $\sigma_d(n)$ values. They have been able

to compute $\sigma_2(n)$ for all $n \leq 53$ as well as several $\rho_d(n)$ values for $d \geq 3$.

Strings exist that are both run- and square-maximal. Based on a comparison of the run- and square-maximal strings, Jiang [32] suggests that this occurs exactly when $n = 2d + i$ when $i = 1$, 2, 3, and 5 for $d \geq 2$.

Finally, while our computational results have supported the conjecture that for every $n > 2$, $\rho_2(n) = \rho(n)$, Deza, Franek, and Jiang [14] have shown that cases exist where $\sigma_3(n) > \sigma_2(n)$. This occurs when $n = 33$, and could exist for other values.

7.2 Future work

Barring the development of an exact formula, it is desirable to compute additional values of $\rho_d(n)$. Currently we are at the bounds of tractability with this approach. An attempt to compute $\rho_2(75)$ using the approach outlined has required over 110 days worth of computing time on SHARCNET's Orca cluster without establishing the value. The most run-dense $(2, 75)$-string $\boldsymbol{x}$ we currently have found has $r(\boldsymbol{x}) = 64 = \rho_2(74)$ Recall that determining $\rho_d(n)$ when $\rho_d(n) = \rho_d(n-1)$ is computationally more difficult than when $r(\boldsymbol{x}) > \rho_d(n-1)$. We hope that further structural insights will allow additional values of $\rho_d(n)$ to be computed.

Beside being able to compute more values of $\rho_d(n)$, it is possible to apply the approach from Chapter 6 in order to confirm that $\rho_d(n) \leq n - d$ for larger values of d and n.

While the difference between the conjectures $\rho_d(n) \leq n-d$ and $\rho(n) \leq n$ is small for large n, we would like conclude by noting that $\rho_d(n) = n-d$ for infinitely many values of n and d while there are no known instances of where $\rho(n) = n$.

Bibliography

[1] Andrew Baker. Run-maximal strings. website, October 2012. `http://optlab.mcmaster.ca/~bakerar2/research/runmax/index.html`.

[2] Andrew Baker, Antoine Deza, and Frantisek Franek. A computational framework for determining run-maximal strings. Submitted, 2012.

[3] Andrew Baker, Antoine Deza, and Frantisek Franek. On the structure of run-maximal strings. *Journal of Discrete Algorithms*, 10:10–14, 2012.

[4] Andrew Baker, Antoine Deza, and Frantisek Franek. A parameterized formulation for the maximum number of runs problem. In Jan Holub, Bruce Watson, and Jan Žďárek, editors, *Festschrift for Bořivoj Melichar*, pages 102–117, Prague, Czech Republic, 2012. Czech Technical University.

[5] Sorin Constantinescu and Lucian Ilie. Fine and Wilf's theorem for abelian periods. *Bulletin of the European Association for Theoretical Computer Science*, 89:167–170, 2006.

[6] Maxime Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12(5):244–250, 1981.

[7] Maxime Crochemore and Lucian Ilie. Maximal repetitions in strings. *Journal of Computer and System Sciences*, 74(5):796–807, 2008.

[8] Maxime Crochemore, Lucian Ilie, and William. Smyth. A simple algorithm for computing the Lempel-Ziv factorization. In James Storer and Michael Marcellin, editors, *Data Compression Conference*, pages 482–487, Snowbird, Utah, United States of America, March 2008.

[9] Maxime Crochemore, Lucian Ilie, and Liviu Tinta. Towards a solution to the "runs" conjecture. *Lecture Notes in Computer Science*, 5029:290–302, 2008.

[10] Maxime Crochemore, Lucian Ilie, and Liviu Tinta. The "runs" conjecture. website, August 2012. `http://www.csd.uwo.ca/faculty/ilie/runs.html`.

[11] Maxime Crochemore and Wojciech Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13:405–425, 1995.

[12] Larry Cummings and William Smyth. Weak repetitions in strings. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 24:33–48, 1997.

[13] Antoine Deza and Frantisek Franek. A *d*-step analogue for runs on strings. AdvOL-Report 2010/2, McMaster University, 2010.

[14] Antoine Deza, Frantisek Franek, and Mei Jiang. A computational framework for determining square-maximal strings. AdvOL-Report 2011/5, McMaster University, 2011.

[15] Antoine Deza, Frantisek Franek, and Mei Jiang. A *d*-step approach for distinct squares in strings. AdvOL-Report 2011/1, McMaster University, 2011.

[16] Paul Erdős. Some unsolved problems. *Hungarian Academy of Sciences Mat. Kutató Intézet Közl.*, 6:221–254, 1961.

[17] Gabriele Fici, Thierry Lecroq, Arnaud Lefebvre, Élise Prieur-Gaston, and William Smyth. A parameterized formulation for the maximum number of runs problem. In Jan Holub and Jan Žďárek, editors, *Proceedings of PSC 2012*, pages 103–110, Prague, Czech Republic, 2012. Czech Technical University.

[18] Nathan Fine and Herbert Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965.

[19] Aviezri Fraenkel and Jamie Simpson. How many squares can a string contain? *Journal of Combinatorial Theory, Series A*, 82(1):112–120, 1998.

[20] Aviezri Fraenkel and Jamie Simpson. The exact number of squares in fibonacci words. *Theoretical Computer Science*, 218:95–106, 1999.

[21] Frantisek Franek. Research. website, November 2012. `http://www.cas.mcmaster.ca/~franek/research.html`.

[22] Frantisek Franek and Robert Fuller. A note on performance comparisons of various runs programs. AdvOL-Report 2012/1, McMaster University, 2012.

[23] Frantisek Franek and Jan Holub. A different proof of Crochemore-Ilie lemma concerning microruns. In Joseph Chan, Jacqueline Daykin, and M. Sohel Rahman, editors, *London Algorithmics 2008: Theory and Practice*, pages 1–9, London, 2009. King's College.

[24] Frantisek Franek and Mei Jiang. Crochemore repetition algorithm revisited–computing runs. AdvOL-Report 2009/1, McMaster University, 2011.

[25] Frantisek Franek and Mei Jiang. Crochemore's repetitions algorithm revisited: computing runs. *International Journal of Foundations of Computer Science*, 23(2):389–401, 2012.

[26] Frantisek Franek, Mei Jiang, and Chia-Chun Weng. An improved version of the runs algorithm based on crochemore's partitioning algorithm. AdvOL-Report 2011/3, McMaster University, 2011.

[27] Frantisek Franek, Ayşe Karaman, and William Smyth. Repetitions in sturmian strings. *Theoretical Computer Science*, 249(2):289–303, 2000.

[28] Frantisek Franek, Jamie Simpson, and William Smyth. The maximum number of runs in a string. In *Proceedings of 14th Australasian Workshop on Combinatorial Algorithms AWOCA 2003*. Seoul National University, Seoul, Korea, July 2003.

[29] Frantisek Franek and Qian Yang. An asymptotic lower bound for the maximal number of runs in a string. *International Journal of Foundations of Computer Science*, 19(1):195–203, 2008.

[30] Mathieu Giraud. Not so many runs in strings. *Lecture Notes in Computer Science*, 5196:232–239, 2008.

[31] Costas Iliopoulos, Dennis Moore, and William Smyth. A characterization of the squares in a Fibonacci string. *Theoretical Computer Science*, 172:281–291, 1997.

[32] Mei Jiang. personal communication, June 2012.

[33] Roman Kolpakov and Gregory Kucherov. personal communication.

[34] Roman Kolpakov and Gregory Kucherov. Maximal repetitions in words or how to find all squares in linear time. Rapport Interne LORIA 98-R-227, INRIA-Lorraine/LORIA, 1998.

[35] Roman Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 596–604, 1999.

[36] Roman Kolpakov and Gregory Kucherov. On maximal repetitions in words. *Lecture Notes in Computer Science*, 1684:374–385, 1999.

[37] Michael Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25:145–153, 1989.

[38] Wataru Matsubara, Kazuhiko Kusano, Akira Ishino, Hideo Bannai, and Ayumi Shinohara. New lower bounds for the maximum number of runs in a string. In Jan Holub and Jan Žďárek, editors, *Proceedings of the Prague Stringology Conference 2008*, pages 140–145, Czech Technical University in Prague, Czech Republic, 2008. Czech Technical University, Prague, Czech Republic.

[39] Wataru Matsubara, Kazuhiko Kusano, Akira Ishino, Hideo Bannai, and Ayumi Shinohara. Lower bounds for the maximum number of runs in a string. website, January 2010. `http://www.shino.ecei.tohoku.ac.jp/runs/`.

[40] Dennis Moore, William Smyth, and Dianne Miller. Counting distinct strings. *Algorithmica*, 23:1–13, 1999.

[41] Simon Puglisi, Jamie Simpson, and William Smyth. How many runs can a string contain? *Theoretical Computer Science*, 401:165–171, 2008.

[42] Wojciech Rytter. The number of runs in a string: Improved analysis of the linear upper bound. *Lecture Notes in Computer Science*, 3884:184–195, 2006.

[43] Wojciech Rytter. The number of runs in a string. *Information and Computation*, 205:1459–1469, 2007.

[44] Francisco Santos. A counterexample to the Hirsch conjecture. *Annals of Mathematics*, 176(1):383–412, 2012.

[45] William Smyth. Repetitive perhaps, but certainly not boring. *Theoretical Computer Science*, 249:343–355, 2000.

[46] William Smyth. *Computing Patterns in Strings*. Pearson Education Limited, 2003.

[47] Axel Thue. Über unendliche zeichenreihen. *Kra. Vidensk. Selsk. Skrifter. I. Mat.-Nat. Christiana*, 7:1–22, 1906.

Index