Improved Bounds on the Sum of Exponents of Runs in a String

Arkadiusz Czarkowski
Institute of Informatics, University of Warsaw, Poland

Abstract

A substring of a word is a *run* if it is at least twice as long as its minimum period and cannot be extended to either side with the same period. The *exponent* of a run is the quotient of its length and its minimum period. $\rho(n)$ is the maximum number of runs in a string of length n, while $\sigma(n)$ is the maximum sum of exponents of runs in a string of length n. While quite tight bounds on $\rho(n)$ are known ($0.944575712n \leq \rho(n) \leq n$), the best upper bound on $\sigma(n)$ is $3n$ whereas the best lower bound on $\sigma(n)$ is $2.035n$. In this paper, we improve the upper bound on $\sigma(n)$ to $2.3n$ and the lower bound on $\sigma(n)$ to $2.04448n$. We also provide an improved upper bound on $\sigma(n)$ of $2.2n$ in the case of a binary alphabet. Our results are achieved using a combination of theoretical and computer-based approaches.

2012 ACM Subject Classification Theory of computation → Pattern matching; Mathematics of computing → Combinatorics on words

Keywords and phrases strings, runs, sum of exponents of runs, Lyndon words, L-roots, maximal repetitions, combinatorics on words

Digital Object Identifier 10.4230/LIPIcs.CPM.2026.23

Supplementary Material *Software (Source Code)*: `https://github.com/Markadiusz/runs` archived at `swh:1:dir:4c7ea1dea83b8d0be4e187b11862bd5de304a7e8`

Funding Supported by the Polish National Science Center, grant no. 2022/46/E/ST6/00463.

1 Introduction

For a string $s = s_1 s_2 \dots s_{|s|} \in \Sigma^*$ over some alphabet Σ, we call a triple $r = (i, j, p)$ a *run* if the substring $s_i s_{i+1} \dots s_j$ has a minimum period of p, has length at least $2p$ and cannot be extended to either left or right with the same period p within s. Moreover, for a run $r = (i, j, p)$, we define $e_r = \frac{j-i+1}{p}$ to be its *exponent*.

For example, the string $s = ababcabcab$ contains two runs – $(1, 4, 2)$, corresponding to the substring $abab$, with an exponent of $\frac{4-1+1}{2} = 2$, and $(3, 10, 3)$, corresponding to the substring $abcabcab$, with an exponent of $\frac{10-3+1}{3} = \frac{8}{3}$.

Runs, in a sense, capture all information about repetitions in a string, and their study is important in the design of string algorithms [5] [2]. Two quantities of particular interest are:

- $\rho(n)$ – the maximum possible number of runs in a string of length n
- $\sigma(n)$ – the maximum possible sum of exponents of runs in a string of length n

Two related quantities are:

- $\rho_k(n)$ – the maximum possible number of runs in a string of length n over an alphabet of size k
- $\sigma_k(n)$ – the maximum possible sum of exponents of runs in a string of length n over an alphabet of size k

▶ Note 1. Clearly, $\rho(n) \geq \rho_k(n)$ and $\sigma(n) \geq \sigma_k(n)$ for any n, k.

Due to the exponential number of distinct strings of length n, the exact values of $\rho(n)$ and $\sigma(n)$ are not known for even moderately big n. Much of the previous work was hence focused on obtaining bounds on those quantities.

From the definition, one can see that both $\rho(n)$ and $\sigma(n)$ are at most polynomial in n. Kolpakov and Kucherov [11] proved an $O(n)$ upper bound on both $\rho(n)$ and $\sigma(n)$. The upper bound on $\rho(n)$ has been further refined as follows:

- $5n$ – Rytter [15]
- $3.48n$ – Puglisi et al[14]
- $3.44n$ – Rytter [16]
- $1.6n$ – Crochemore and Ilie [3]
- $1.52n$ – Giraud [9]
- $1.029n$ – Crochemore et al. [4]
- n – Bannai et al. [1]

While many of the above approaches were very technical and often were computer-assisted, the bound $\rho(n) \leq n$ proven by Bannai et al. [1] was based on a very elegant argument that related the runs with the occurrences of so-called Lyndon words in strings.

Other noteworthy results are the papers of Fischer et al. [7] and S. Holub [10], who showed upper bounds on $\rho_2(n)$ of $0.957n$ and $0.9482n$, respectively.

On the other hand, researchers have been trying to come up with run-rich words to obtain a lower bound on $\rho(n)$. Here, the most relevant results are:

- $0.927n$ – Franek and Yang [8]
- $0.94457567n$ – Matsubara et al. [13]
- $0.944575712n$ – Simpson [17]

All of the above lower bounds are achieved by binary strings – strings over an alphabet of size 2.

With the upper bound on $\rho(n)$ being so close to the lower bound, especially in the case of binary strings, the problem of bounding $\rho(n)$ seems to be practically solved. However, the problem of bounding $\sigma(n)$ is quite far from being solved.

The history of improving the upper bound on $\sigma(n)$ includes the results:

- $O(n)$ – Kolpakov and Kucherov [11]
- $25n$ – Rytter [16]
- $5.6n$ – Crochemore and Ilie [3]
- $4.1n$ – Crochemore, Kubica, Radoszewski, Rytter, Waleń [6]
- $3n$ – Bannai et al. [1]

The result $\sigma(n) \leq 3n$ by Bannai et al. [1] was achieved by showing that $\sigma(n) \leq 2\rho(n) + n$. By a similar argument, one can show an analogous inequality for binary strings. This means that the $0.9482n$ upper bound on $\rho_2(n)$ established by S. Holub in [10] in turn gives an upper bound of $2.8964n$ on $\sigma_2(n)$.

On the other hand, the best lower bound on $\sigma(n)$ is $2.035n$ by Crochemore, Kubica, Radoszewski, Rytter, Waleń [6].

This paper addresses the gap between the upper bound and the lower bound on $\sigma(n)$. In Section 3 we show an upper bound of $2.3n$ on $\sigma(n)$, and for $\sigma_2(n)$ we show an even tighter upper bound of $2.2n$. In Section 4 we show a lower bound on $\sigma(n)$ of $2.04448n$ achieved by a binary string.

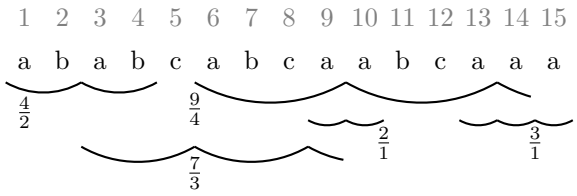
2 Preliminaries

This section is heavily based on Sections 2 and 3 of the Bannai et al. paper [1], which contains the omitted proofs of some of the lemmas.

Let Σ be a finite set called the *alphabet*. An element of Σ^* is called a *string*. The length of a string s is denoted by $|s|$. The empty string ϵ is a string of length 0. For a string $s = xyz$, x, y and z are called a *prefix*, *substring*, and *suffix* of s, respectively. A prefix (resp. suffix) x of s is called a *proper prefix* (resp. suffix) of s if $x \neq s$. The i-th character of a string s is denoted by $s[i]$, where $1 \leq i \leq |s|$. For a string s and two integers $1 \leq i \leq j \leq |s|$, let $s[i..j]$ denote the substring of s that begins at position i and ends at position j. For convenience, let $s[i..j] = \epsilon$ when $i > j$. An integer $p \geq 1$ is said to be a *period* of a string s if $s[i] = s[i+p]$ for all $1 \leq i \leq |s| - p$.

▶ **Definition 2** ([11, maximal repetition]). *A triple $r = (i, j, p)$ is a run of string w, if p is the smallest period of $w[i..j]$, $|w[i..j]| \geq 2p$, the periodicity cannot be extended to the left or right, i.e., $i = 1$ or $w[i-1] \neq w[i-1+p]$, and $j = n$ or $w[j+1] \neq w[j+1-p]$. The rational number $\frac{j-i+1}{p}$ is called the exponent of r.*

▶ **Example 3.** The string *ababcabcaabcaaa* contains runs $(1, 4, 2), (3, 9, 3), (6, 14, 4), (9, 10, 1), (13, 15, 1)$.



Let $Runs(w)$ denote the set of runs of string w. Let $Exp(w)$ denote the sum of exponents of runs of string w. Let $\rho(n)$ denote the maximum possible number of runs in a string of length n. Let $\sigma(n)$ denote the maximum possible sum of exponents of runs in a string of length n. Similarly, let $\rho_2(n)$ denote the maximum possible number of runs in a string of length n over the binary alphabet and $\sigma_2(n)$ denote the maximum possible sum of exponents of runs in a string of length n over the binary alphabet.

Let us now quickly prove two lemmas about exponents of runs that will be useful later.

▶ **Lemma 4.** *Let $s, t \in \Sigma^*$ be two arbitrary strings. We have $Exp(s) + Exp(t) \leq Exp(st)$.*

Proof. Let $u \in \Sigma^*$ and let $r = (i, j, p)$ be a run of u. Consider the string $w = ux$ where $x \in \Sigma$. If $j < |u|$ or $u[|u|+1-p] \neq x$, then clearly r is still a run in ux. Otherwise, $(i, j+1, p)$ is a run in ux (here $j+1 = |u|+1$). An analogous argument holds for prepending a character to u.

Let $s, t \in \Sigma^*$. For each position k of s let

$$L_k = \{(i, p) \mid (i, j, p) \text{ is a run of } s \text{ for some } j \text{ with } i \leq k \leq j\}$$

Similarly, for each position k of t let

$$R_k = \{(j, p) \mid (i, j, p) \text{ is a run of } t \text{ for some } i \text{ with } i \leq k \leq j\}$$

From the period-maximality of runs, we know that there can be at most one run for a given (i, p) pair or (j, p) pair. Sets L_k and R_k describe all runs from s and t, respectively, that include position k. From the previous argument, we see that those sets can only expand when calculated for st. It follows that the contribution of each position to $Exp(st)$ is no smaller than it was to either $Exp(s)$ or $Exp(t)$. ◀

▶ Note 5. A similar inequality does not hold for runs. For example, the string $w = aa$ has one run – $(1, 2, 1)$, while its square $ww = aaaa$ has only one run – $(1, 4, 1)$.

From the above lemma, we can conclude that:

▶ **Corollary 6.** *For $n, m \in \mathbb{Z}_{\geq 0}$, we have $\sigma(n) + \sigma(m) \leq \sigma(n+m)$.*

▶ **Corollary 7.** *For $n, k \in \mathbb{Z}_{\geq 0}$, we have $k\sigma(n) \leq \sigma(nk)$, by induction on k.*

▶ **Lemma 8.** *Let $A, B \geq 0$. If for all $n \in \mathbb{Z}_{\geq 0}$ the inequality $\sigma(n) \leq An + B$ holds, then $\sigma(n) \leq An$ is true as well.*

Proof. Let $A, B \geq 0$, $n \in \mathbb{Z}_{\geq 0}$ and $k \in \mathbb{Z}_{>0}$. From Corollary 7 we have

$$k\sigma(n) \leq \sigma(nk) \leq Ank + B$$

$$\sigma(n) \leq \frac{Ank + B}{k} = An + \frac{B}{k}$$

This means that

$$\sigma(n) \leq \inf\left\{An + \frac{B}{k} \,:\, k \in \mathbb{Z}_{>0}\right\} = An$$ ◀

▶ Note 9. It is important to note that Lemma 8 holds also for $\sigma_2(n)$, since we did not assume anything about the alphabet.

While the definition of a run requires only an equivalence relation on Σ, introducing an order on Σ lets us study runs in more detail.

Let $\prec$ denote some total order on Σ, as well as the lexicographic order induced on Σ^*.

▶ **Definition 10** ([12, Lyndon Word])**.** *A non-empty string $w \in \Sigma^+$ is said to be a Lyndon word with respect to $\prec$, if $w \prec u$ for any non-empty proper suffix u of w.*

▶ Note 11. A Lyndon word w cannot have any period $p < |w|$, since its existence would imply $w = xyx$ for some non-empty x and some y, and $x \prec w$.

▶ **Definition 12** ([1, Definition 4])**.** *Let $r = (i, j, p)$ be a run in a string $w \in \Sigma^*$. An interval $\lambda = [i_\lambda..j_\lambda]$ of length $j_\lambda - i_\lambda + 1 = p$ is an L-root of r with respect to $\prec$ if $i \leq i_\lambda \leq j_\lambda \leq j$ and $w[i_\lambda..j_\lambda]$ is a Lyndon word with respect to $\prec$.*

It is easy to see that for any run and lexicographic order $\prec$, there exists at least one L-root with respect to $\prec$.

Let $\prec_0, \prec_1$ denote some pair of total orders on Σ such that for any pair of characters $a, b \in \Sigma$, we have $a \prec_0 b \Leftrightarrow b \prec_1 a$. For $\ell \in \{0, 1\}$, let $\bar{\ell} = 1 - \ell$.

For any string $w \in \Sigma^*$, let $\bar{w} = w\$$, where $\$ \notin \Sigma$ is a special character that satisfies $\$ \prec_0 a$ (and thus $a \prec_1 \$$) for any $a \in \Sigma$.

▶ Remark 13. The \$ character is introduced mainly to simplify the following lemmas and definitions. Notice that appending such a character to a string does not affect the structure of its runs in any way.

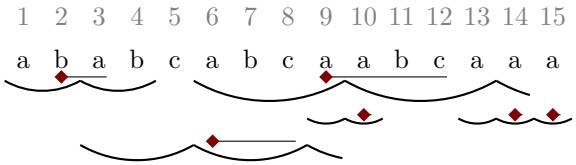
▶ **Definition 14** ([1, Definition 5])**.** *For any string w and position i ($1 \leq i \leq |w|$), let $Lyn_\ell(i) = [i..j]$ where $j = \max\{j' \mid \bar{w}[i..j']$ is a Lyndon word with respect to $\prec_\ell\}$. Notice that $Lyn_\ell(i)$ is well defined since one-character strings are Lyndon words.*

This definition is illustrated in Example A.1 in the Appendix.

▶ **Lemma 15** ([1, Lemma 7])**.** *Let $r = (i, j, p)$ be an arbitrary run in a string w of length n. Then, for a unique $\ell \in \{0, 1\}$ such that $\bar{w}[j+1] \prec_\ell \bar{w}[j+1-p]$, any L-root $\lambda = [i_\lambda..j_\lambda]$ of r with respect to $\prec_\ell$ is equal to $Lyn_\ell(i_\lambda)$.*

For any run $r = (i, j, p)$ of w, let $B_r = \{\lambda = [i_\lambda..j_\lambda] \mid \lambda$ is an L-root of r with respect to $\prec_\ell$, $i_\lambda \neq i\}$, where $\ell \in \{0, 1\}$ is such that $\bar{w}[j+1] \prec_\ell \bar{w}[j+1-p]$, i.e. B_r is the set of all L-roots $[i_\lambda..j_\lambda]$ of r with respect to $\prec_\ell$ such that $[i_\lambda..j_\lambda] = Lyn_\ell(i_\lambda)$, except for the one that starts from i if it exists. For any set I of intervals, let $Beg(I)$ denote the set of beginning positions of intervals in I. Additionally, let S_r denote $Beg(B_r)$ for any run r. The set S_r is called the set of *special positions* of the run r.

▶ **Example 16.** Consider the string *ababcabcaabcaaa*. The special positions of its runs, along with related L-roots, are as follows:



▶ **Lemma 17** ([1, Lemma 8]). *For any two distinct runs r and r' of string w, $S_r \cap S_{r'}$ is empty.*

From Lemma 17 it follows that each position of a string is a special position for at most one run and since each run contains at least one special position, this gives the $\rho(n) \leq n$ upper bound proven by Bannai et al. [1, Theorem 9].

This relation between runs and special positions is not only theoretical, but it can often also be efficiently computed locally without having to consider the whole string.

First, consider some run $r = (i, j, p)$ of a string w. One can find its set of special positions by

- determining an appropriate order $\prec_\ell$, by comparing $\bar{w}[j+1]$ and $\bar{w}[j+1-p]$
- checking which substrings of $w[i..j]$ of length p are Lyndon words with respect to $\prec_\ell$

Second, consider some position $k \in [2..|w|]$ of a string w. One can find a run for which k is special (or determine that there is no such run) by the following method adapted from [7, Lemma 2] to work for general alphabets (the original lemma considered a binary alphabet):

- Assume that k is a special position of $r = (i, j, p)$.
- If $w[k] = w[k-1]$, we know that k is a special position of the unique run with period 1 that contains position k.
- If $w[k] \neq w[k-1]$, we would like to know with respect to which order $\prec_\ell$ the interval $Lyn_\ell(k) = [k..k']$ is an L-root of r. Since $k > i$, we must have $w[k] \prec_\ell w[k-1] = w[k']$, because otherwise $w[k..k']$ would not be a Lyndon word with respect to $\prec_\ell$. Now that we know the only possible order $\prec_\ell$, it is sufficient to check whether the unique period-maximal extension of $Lyn_\ell(k)$ is a run with period $|Lyn_\ell(k)|$, i.e. whether its length is at least twice the period.

3 Upper bound

3.1 Main idea

The main idea of the Bannai et al. paper [1] was to establish a mapping from runs to disjoint sets of positions of the string. The strategy for improving the upper bound on $\sigma(n)$, which we present here, will be based on designing a similar mapping from exponents of runs to rational values assigned to positions of the string.

Consider some run $r = (i, j, p)$ of a string $w \in \Sigma^*$ along with its set of special positions S_r. This run contributes $\frac{j-i+1}{p}$ to $Exp(w)$. From another perspective, each character of $w[i..j]$ is contributing $\frac{1}{p}$ to $Exp(w)$ through r. The main idea will be to partition the interval $[i..j]$ into $|S_r|$ intervals and have each of the special positions of r account for the contributions from its respective part.

Let $s_1, \ldots, s_{|S_r|}$ be the special positions of r in increasing order. Notice that $s_k + p = s_{k+1}$ for $1 \leq k < |S_r|$. We want to partition the interval $[i..j]$ into the following $|S_r|$ intervals:

$$[i..(s_1 + p - 1)], [s_2..(s_2 + p - 1)], [s_3..(s_3 + p - 1)], \ldots, [s_{|S_r|-1}..(s_{|S_r|-1} + p - 1)], [s_{|S_r|}..j]$$

If there is only one special position for r, this position alone will be assigned the full exponent of r.

▶ **Definition 18.** *Let $w \in \Sigma^*$ be a string. Let $k \in [1..|w|]$ be a special position of some run $r = (i, j, p)$ of w. Let $L_w : [1..|w|] \rightharpoonup [1..|w|]$ be a partial function defined on special positions of w as follows:*

$$L_w(k) = \begin{cases} k & \text{if } k - p > i \\ i & \text{if } k - p \leq i \end{cases}$$

Let $R_w : [1..|w|] \rightharpoonup [1..|w|]$ be a partial function defined on special positions of w as follows:

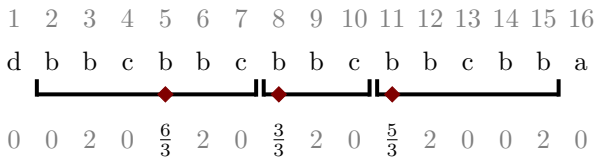
$$R_w(k) = \begin{cases} k + p - 1 & \text{if } k + 2p - 2 < j \\ j & \text{if } k + 2p - 2 \geq j \end{cases}$$

Let $\phi_w : [1..|w|] \to \mathbb{Q}_{\geq 0}$ be a function defined on all positions of w as follows:

$$\phi_w(k) = \begin{cases} \dfrac{R_w(k) - L_w(k) + 1}{p} & \text{if } k \in S_r \text{ for some } r = (i, j, p) \in Runs(w) \\ 0 & \text{otherwise} \end{cases}$$

▶ Note 19. Notice that $\sum_{k=1}^{|w|} \phi_w(k) = Exp(w)$.

▶ **Example 20.** Consider the following string w, which contains, among others, the run $(2, 15, 3)$.



Since $w[16] = a$ must be less than $w[13] = c$, we consider the L-roots of r with respect to $\prec_0$. This means the set of special positions S_r of the run r is $\{5, 8, 11\}$. Notice that 2 is not a special position of r since we cannot have a special position at the start of a run. The interval $[2..15]$ is then partitioned into three intervals: $[2..7], [8..10]$ and $[11..15]$. This means that $\phi_w(5) = \frac{7-2+1}{3} = \frac{6}{3}$, $\phi_w(8) = \frac{3}{3}$, and $\phi_w(11) = \frac{5}{3}$. For positions $x \in \{3, 6, 9, 12, 15\}$ we have $\phi_w(x) = 2$ as they are the only special positions in their two-character runs bb with period 1. All other positions will be assigned the value of 0 since there are no more runs in w.

Notice that in the above example, all values of ϕ_w are quite small. Indeed, it turns out not to be a coincidence.

▶ **Lemma 21.** *For each $k \in [1..|w|]$, we have $\phi_w(k) < 3$.*

Proof. Let (i, j, p) be a run and $k \in S_r$ be its special position. By definition, $k - L_w(k) \leq p$. By definition, $R_w(k) - k \leq \max(p-1, 2p-2) = 2p-2$, since $p \geq 1$. In total, we get that $R_w(k) - L_w(k) + 1 \leq p + (2p-2) + 1 = 3p - 1$. It follows that $\phi_w(k) \leq \frac{3p-1}{p} = 3 - \frac{1}{p} < 3$. ◀

▶ Remark 22. Notice that since the first position of a string can never be special, from the above lemma we can conclude that $\sigma(n) < 3n - 3$, which is the upper bound established in [1, Theorem 10].

An interesting property of the above mapping is that the values assigned to positions are local and often can be deduced from the neighbourhood of the position of size proportional to the period of the corresponding run.

This leads us to the following strategy of improving the upper bound on $\sigma(n)$. Consider some string w. We want to split string w into a sequence of smaller strings $w = pe_1e_2 \dots e_k s$ and obtain a bound for each of the smaller strings separately. Here, p and s are some prefix and suffix of w, respectively, that we want to be of constant size. Assume that for each e_i we are able to bound the sum of the values assigned to positions of e_i by $c|e_i|$, where $c \geq 0$. Then by Lemma 21 we would get the following upper bound on $Exp(w)$:

$$Exp(w) \leq 3(|p| + |s|) + \sum_i c|e_i| \leq 3(|p| + |s|) + c|w| = c|w| + O(1)$$

If we are able to get such a bound for each possible string, then by Lemma 8 it will follow that $\sigma(n) \leq cn$.

Our task is now reduced to efficiently splitting any string w into strings e_i, and then bounding the sum of values assigned to positions of each e_i. Let us cover the bounding methods first.

3.2 Basic bounding method

Consider some string u that is part of a larger string w. Let $f(i)$ be the value assigned to the i-th position of u when mapping from exponents of runs of w to positions of w. More specifically, assuming that u corresponds to $w[k..(k + |u| - 1)]$ for some k, we have $f(i) = \phi_w(i + k - 1)$ for $1 \leq i \leq |u|$. Our goal is to bound $\sum_{i=1}^{|u|} f(i)$.

While some runs of w might not be runs when restricted to u (since they would be shorter than twice their period), sometimes a run of w will also be a run of u. Let $r = (i, j, p)$ be such a run in u. Let us consider a few cases.

1. $1 < i$ and $j < |u|$
   This is the simplest and most pleasant case. Notice that since r is contained strictly within u, the influence of r on f will always be the same no matter what w is. This means that we can determine r's respective $\prec_\ell$ order, find all its L-roots, and assign appropriate values to its special positions.
   This case is illustrated in Example A.2.1 in the Appendix.
2. $1 = i$ and $j < |u|$
   In this case, since $j < |u|$, we can still determine the respective order $\prec_\ell$ for r. However, since $1 = i$, we do not know how far to the left the run r goes within w. This is not that much of a problem as it might seem. For all but the leftmost special position contained in u we can easily compute their assigned values. For the leftmost position, we know that the left end of its interval is at most p to its left, because otherwise there would be another special position.
   This case is illustrated in Example A.2.2 in the Appendix.

3. $j = |u|$
   This case could be further split based on the relation between 1 and i, but we omit it here since the method for the leftmost special position is the same as in the previous two cases.
   Now, we do not know how far to the right the run goes in w. What is worse, we do not know which order $\prec_\ell$ we should use. Again, this turns out not to be much of an issue. We can just consider both cases and take the worst of the two bounds we get for both potential sets of special positions. While considering an order $\prec_\ell$, we will assume that the special positions for the other order $\prec_{\bar{\ell}}$ are all bounded using the naive bound of 3. This case is illustrated in Example A.2.3 in the Appendix.

3.3 Efficient splitting for binary strings

Let us focus on binary strings for now. They are easier to work with and allow for a simpler introduction of some new ideas.

▶ **Theorem 23.** *We have* $\sigma_2(n) \leq 2.875n$.

Proof. Consider all possible 4-character words over the binary alphabet $\{a, b\}$.

Except for *abab* and *baba*, every such string contains a run with period 1, which allows us to bound the value assigned to at least one of its positions by $3 - \frac{1}{1} = 2$. This gives us a total bound of $3 \cdot 3 + 2 = 11$.

In the case of *abab* and *baba*, either the second or the third position is a special position of a run with period 2, so its value can be bounded by $3 - \frac{1}{2} = 2.5$. This gives us a total bound of $3 \cdot 3 + 2.5 = 11.5$.

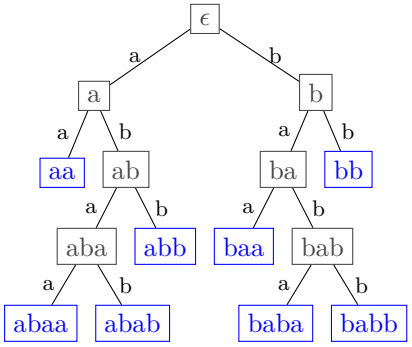
It follows that we get an $\frac{11.5}{4} = 2.875$ average bound per character. ◀

▶ Remark 24. This already improves on the previously best upper bound of $\sigma_2(n) \leq 2.8964n$ established in [10, Theorem 1].

The above proof can be extended by considering strings of greater length. However, it is very inefficient to consider all strings of a given length, since there is an exponential number of them. This approach gets even worse when we consider strings over a general alphabet.

To remedy this issue, we prune unnecessary cases by building a trie of the considered strings as follows:

▶ **Example 25.** Consider the following pruned trie corresponding to the strings considered in the proof of Theorem 23.



All leaves of the trie contain strings that can be bounded by 2.875 per character. Consider the string

bbabaababaaababaab

We split it by each time cutting off some leaf of the above tree as follows:

$(bb)(abaa)(baba)(aa)(baba)(ab)$

Here, the unmatched string (ab) has constant size, so it does not matter by Lemma 8.

In general, our goal is to build a full binary tree and calculate the per-character bounds for each of its leaves. Then, such a tree can be used to split any string by following the edges with appropriate labels downwards from the root until we reach a leaf, cutting off the travelled path, and continuing again from the root until we get to the end of the string.

To save some space, let us focus on the left part of the tree for now. The other part is symmetric and has the same bounds in the case of binary strings.

Algorithm for constructing the trie

To build this tree efficiently, we start with a single node containing a one-character string a. Now, each time we choose a leaf of the tree with the worst bound, and expand from this node by adding its two children and computing their bounds. The bound of a node is the minimum of the bounds of each of its prefixes, since while splitting the string, we can choose to split at any node in the tree with a satisfactory bound.

This approach is illustrated in Example A.3 in the Appendix.

This approach of always expanding the worst bounded leaves allows us to most efficiently reduce the upper bound on $\sigma_2(n)$ while considering a much smaller number of cases than the naive exponential approach.

This method can be quite easily implemented to run on a computer. While the computer assistance greatly improves the bounds we get, the results for binary strings seem to plateau around $2.589n$. To go further, we will need to develop new bounding methods.

3.4 Other bounding methods

3.4.1 Detection of non-special positions

The previously considered basic bounding method works great when the strings we consider have plenty of runs. With increased string lengths, we are able to consider even more runs and further improve our bounds. Problems begin when the strings we analyse have very few runs.

As we can see in the Example A.4 in the Appendix, determining that some position is not a special position improves the bound drastically since we are changing the naive 3 to a 0. Let us describe in detail when we can perform such a deduction.

Let u be a substring of w, and $k \in [2..|u|]$ be a position of u that we want to examine. First, we compare $u[k]$ with $u[k-1]$ to determine which one of $\prec_0$ and $\prec_1$ to use (see final remarks of Section 2). Notice that $u[k]$ cannot be equal to $u[k-1]$ since otherwise k would be a special position for a run with period 1. Second, we need to find some $h \in [k+1..|u|]$ such that $u[h..|u|] \prec_\ell u[k..|u|]$ and $u[h..|u|]$ is not a prefix of $u[k..|u|]$. This allows us to bound the length of the longest Lyndon word with respect to $\prec_\ell$ starting at k no matter what w is. After that, we find the longest Lyndon word with respect to $\prec_\ell$ starting at k and find its period-maximal extension $[l..r]$. If $l > 1$ and $r < |u|$ and $r - l + 1 < 2p$, then we can conclude that k never is a special position irrespective of w.

▶ **Example 26.** Consider the following string u that is part of some other string w.

1	2	3
a	b	c
3	0	3

Let us focus on the second position of u. We choose $\prec_1$, since we want $u[2] = b \prec_1 a = u[1]$. However, because $u[3] = c \prec_1 b = u[2]$, we see that the longest Lyndon word with respect to $\prec_1$ starting at the second position has length 1 and cannot be extended to either left or right. It follows that this is not a special position, so it is assigned a value of 0. This gives a per-character bound of $\frac{3+0+3}{3} = 2$ for u.

This example provides some intuition for why non-binary strings often have small sums of exponents of runs as compared to binary strings.

3.4.2 Expanding the context window

▶ **Example 27.** Let us have a closer look at the string $u = aba$, which is part of some binary string w. Previously in Example 36, we used the naive per-character bound of 3 for it. There are no runs in u, but at the same time, we have too little information to determine that some of its positions are not special. To provide a better bound for u, we need to expand our context window.

Let us consider two cases based on the character of w that is just before u. Let us denote this character $u[0]$. Notice that we can assume that there is a sufficient number of characters in w before u, as long as they are of constant length, by making sure the naively bounded prefix of w is long enough.

- $u[0] = a$

0	1	2	3
a	a	b	a
–	2	3	3

  Since $u[0] = a$, we have a run $(0, 1, 1)$, which means 1 is a special position of a run with period 1 and can be bounded by 2.

- $u[0] = b$

0	1	2	3
b	a	b	a
	◆		
–	$\frac{5}{2}$	3	3

0	1	2	3
b	a	b	a
		◆	
–	3	$\frac{5}{2}$	3

  Since $u[0] = b$, we have a run $(0, 3, 2)$, which means either 1 or 2 is a special position of a run with period 2 and can be bounded by $\frac{5}{2}$.

In total, we get u can be bounded by $\max(2+3+3, \frac{5}{2}+3+3) = 8.5$, which is an $\frac{8.5}{3} \approx 2.83$ per-character bound.

The general idea of this method is to look ahead a couple of characters to the left and right of u and branch on their values. We can assume that there is a sufficient number of characters in w before and after u, as long as they are of constant length, by making sure the naively bounded prefix and suffix of w are long enough. Then, we can refine the bound for u to be the maximum over the bounds for all those cases, which often turns out to be lower than the basic bound, since all those cases have access to more context.

Since there is an exponential number of cases to consider for a given extension length, the actual method used resembles the trie construction described in Section 3.3 and is covered in Section 3.6.

In Section B of the Appendix, we prove by hand the following theorem, which improves the result from Theorem 23:

▶ **Theorem 28.** *We have* $\sigma_2(n) \leq 2.7n$.

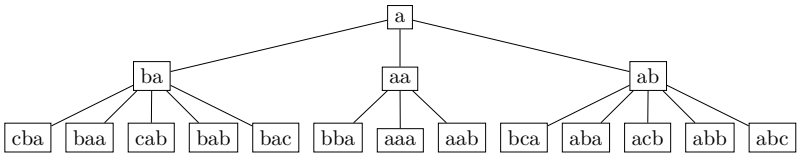
All of the above methods quite easily translate to a computer program that allows us to expand our search to millions of cases. Efficiently implementing the covered approaches gives us the claimed $\sigma_2(n) \leq 2.2n$ bound. The details of the implementation are covered in Section 3.6.

3.5 General alphabet

A natural next step is to adapt the techniques developed for bounding binary strings to strings over a general alphabet.

The first issue we need to tackle is how to iterate over strings with arbitrary characters. The main idea is that when analysing a small substring of some string w, we do not care about the exact values of the characters, but only their relative order in $\prec_0$ and $\prec_1$. This allows us to renumber the considered values, which makes all of them bounded by the length of the considered strings.

The first three levels of a splitting trie for general strings could then look as follows:



It is important to note that in addition to appending an already present character, we also need to consider putting the value of the new character between the values of already present characters. This might cause us to renumber some of the existing values. For example, this is how we get from a to ba – we append a value smaller than a, and this causes us to renumber the resulting string to ba. It is also important to remember that we need to use the same expansion scheme when considering longer context windows to make sure we consider all possible cases.

Unfortunately, after implementing the above approach, we encounter a much bigger issue – there are strings for which the best per-character bound we get is 3.

Consider the string $u = afbecd$ that is part of some string w. There are no runs in u. Moreover, for each position of u, its longest Lyndon word goes all the way to the end of u. This means we cannot definitely say that they are not special positions, because that depends on the rest of w. We are left with the naive per-character bound of 3.

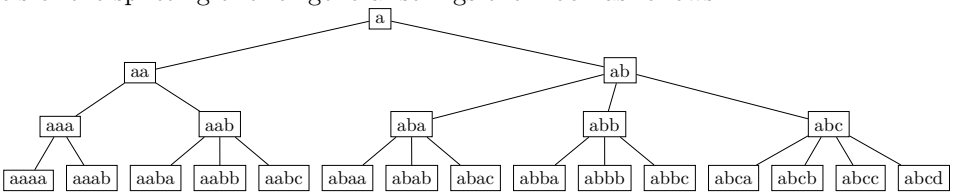
All hope is not lost yet. Recall that in Example 26 we showed a per-character bound of 2 for the string $u = abc$. Similarly, for $u = abcdef$, its 4 middle characters are not special positions. This gives us a per-character bound of $\frac{3+3}{6} = 1$ for u.

While in the previous sections we chose $\prec_0$ to be the alphabetical order, for $u = afbecd$ it would be much better for $\prec_0$ to be such an order that $a \prec_0 f \prec_0 b \prec_0 e \prec_0 c \prec_0 d$. Unfortunately, we cannot just choose a different order for each string u. What we can do instead is to choose some order $\prec_0$ for the whole w. It would be nice to choose the order that allows us to best bound $Exp(w)$. However, this is quite tricky since we only see small sections of w and they can occur in w in arbitrary proportions.

The approach that turns out to work well for this problem is to choose the order at random. More specifically, we want to calculate the expected bound on $Exp(w)$ when $\prec_0$ is chosen uniformly at random from the set of all possible total orders on Σ. It follows that there exists an order which achieves a bound on $Exp(w)$ no worse than the expected bound, and this is the one that we use. By the linearity of the expected value, the expected bound on $Exp(w)$ is the sum of the expected bounds on the small segments of w.

This leads us to the actual way we generalize the bounding method for binary strings to arbitrary strings:

1. We build the splitting trie based on equivalence classes over the string u. The first four levels of the splitting trie for general strings then look as follows:



▶ Remark 29. Actually, we would never expand from the *aaa* node, since it already has an expected per-character bound of 2, which is lower than the per-character lower bound on $\sigma(n)$.

2. To bound some node u, we first consider possible context expansions to gain more information about the neighbourhood of u. In this expansion, we again expand only based on the equivalence between characters of u. We do not care about their relative order yet.
3. Next, for a string u which has k different characters, we consider all $k!$ possible total orders on those k characters and separately bound each of those orders. Because each of those orders on k characters is equally likely among all total orders on characters of w, we just need to calculate their arithmetic mean to get the expected bound.
4. Finally, to get a bound for string u along with some order $\prec_0$, we use the previously developed methods of assigning values to special positions of runs and determining that some positions are not special.

Efficiently implementing the above approach gives us the claimed $\sigma(n) \leq 2.3n$ bound. The details of the implementation are covered in Section 3.6.

3.6 Implementation details

Our final results about the upper bound are the following two theorems:

▶ **Theorem 30.** *We have* $\sigma_2(n) \leq 2.2n$.

▶ **Theorem 31.** *We have* $\sigma(n) \leq 2.3n$.

The proofs of both of them relied on heavy computer assistance and required a few days of CPU time to finish.

A notable idea here is that instead of always expanding the worst bounded node, it's much more efficient to aim to prove some target upper bound, because then we can expand from any node of the tree. Because of that we do not have to keep the whole tree in memory, and the whole procedure can be implemented easily using recursion. This approach allows also for easy parallelization of computations.

4 Lower bound

4.1 Result

Consider the following context-free grammar with productions:

$$
\begin{aligned}
X_0 &\to ababa & X_9 &\to X_7X_6X_5X_6\\
X_1 &\to ababbaba & X_{10} &\to X_7X_6X_5X_7X_6\\
X_2 &\to X_0X_1X_0X_0X_1X_0X_1 & X_{11} &\to X_8X_9X_{10}\\
X_3 &\to X_0X_1X_0X_0X_1X_0X_1X_1 & X_{12} &\to X_8X_9X_{10}X_9\\
X_4 &\to X_0X_1X_0X_0X_1X_0X_1X_1X_0X_1 & X_{13} &\to X_8X_9X_{10}X_9X_{10}\\
X_5 &\to X_4X_3X_2X_4X_2 & X_{14} &\to X_{12}X_{11}\\
X_6 &\to X_4X_3X_2X_4X_2X_3X_2 & X_{15} &\to X_{12}X_{11}X_{13}\\
X_7 &\to X_4X_3X_2X_4X_2X_3X_2X_4X_2 & X_{16} &\to X_{14}X_{15}X_{15}X_{15}\\
X_8 &\to X_7X_6X_5 & X_{17} &\to X_{14}X_{15}X_{15}X_{15}X_{15}\\
& & X_{18} &\to X_{16}X_{17}\\
& & X_{19} &\to X_{16}X_{17}X_{17}
\end{aligned}
$$

$$X_{20} \to X_{18}X_{19}X_{19}X_{18}X_{19}X_{19}X_{19}X_{18}X_{19}X_{19}X_{18}X_{19}X_{19}X_{19}X_{18}X_{19}X_{19}X_{18}X_{19}X_{19}$$

in which a, b are the terminals, X_i are the non-terminals and X_{20} is the start variable. Each X_i uniquely produces some binary string of as and bs. The start symbol X_{20} produces a binary string w of length $n = 4\,201\,174$, whose sum of exponents of runs has been verified by a computer program to be greater than $2.04448n$. This has been done using an approach similar to the linear time method of finding all runs described in [1, Section 4]. This result improves upon the previously best lower bound on $\sigma(n)$ of $2.035n$ from [6, Theorem 4.1].

References

1 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM J. Comput.*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.

2 Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Tomasz Walen, and Wiktor Zuba. Efficient enumeration of distinct factors using package representations. In Christina Boucher and Sharma V. Thankachan, editors, *String Processing and Information Retrieval - 27th International Symposium, SPIRE 2020, Orlando, FL, USA, October 13-15, 2020, Proceedings*, volume 12303 of *Lecture Notes in Computer Science*, pages 247–261. Springer, 2020. doi:10.1007/978-3-030-59212-7_18.

3 Maxime Crochemore and Lucian Ilie. Maximal repetitions in strings. *J. Comput. Syst. Sci.*, 74(5):796–807, 2008. doi:10.1016/j.jcss.2007.09.003.

4 Maxime Crochemore, Lucian Ilie, and Liviu Tinta. The "runs" conjecture. *Theor. Comput. Sci.*, 412(27):2931–2941, 2011. doi:10.1016/J.TCS.2010.06.019.

5 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Extracting powers and periods in a word from its runs structure. *Theor. Comput. Sci.*, 521:29–41, 2014. doi:10.1016/J.TCS.2013.11.018.

6 Maxime Crochemore, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. On the maximal sum of exponents of runs in a string. *J. Discrete Algorithms*, 14:29–36, 2012. doi:10.1016/J.JDA.2011.12.016.

7 Johannes Fischer, Stepan Holub, Tomohiro I, and Moshe Lewenstein. Beyond the runs theorem. In Costas S. Iliopoulos, Simon J. Puglisi, and Emine Yilmaz, editors, *String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015, London, UK, September 1-4, 2015, Proceedings*, volume 9309 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 2015. doi:10.1007/978-3-319-23826-5_27.

8 Frantisek Franek and Qian Yang. An asymptotic lower bound for the maximal number of runs in a string. *Int. J. Found. Comput. Sci.*, 19(1):195–203, 2008. doi:10.1142/S0129054108005620.

9 Mathieu Giraud. Not so many runs in strings. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, volume 5196 of *Lecture Notes in Computer Science*, pages 232–239. Springer, 2008. doi:10.1007/978-3-540-88282-4_22.

10 Stepan Holub. Prefix frequency of lost positions. *Theor. Comput. Sci.*, 684:43–52, 2017. doi:10.1016/J.TCS.2017.01.026.

11 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.

12 Roger Lyndon. On Burnside's problem. *Transactions of the American Mathematical Society*, 77:202–215, 1954. URL: https://api.semanticscholar.org/CorpusID:51747091.

13 Wataru Matsubara, Kazuhiko Kusano, Hideo Bannai, and Ayumi Shinohara. A series of run-rich strings. In Adrian-Horia Dediu, Armand-Mihai Ionescu, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications, Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009. Proceedings*, volume 5457 of *Lecture Notes in Computer Science*, pages 578–587. Springer, 2009. doi:10.1007/978-3-642-00982-2_49.

14 Simon J. Puglisi, Jamie Simpson, and William F. Smyth. How many runs can a string contain? *Theor. Comput. Sci.*, 401(1-3):165–171, 2008. doi:10.1016/J.TCS.2008.04.020.

15 Wojciech Rytter. The number of runs in a string: Improved analysis of the linear upper bound. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 184–195. Springer, 2006. doi:10.1007/11672142_14.

16 Wojciech Rytter. The number of runs in a string. *Inf. Comput.*, 205(9):1459–1469, 2007. doi:10.1016/J.IC.2007.01.007.

17 Jamie Simpson. Modified Padovan words and the maximum number of runs in a word. *Australas. J Comb.*, 46:129–146, 2010. URL: http://ajc.maths.uq.edu.au/pdf/46/ajc_v46_p129.pdf.

A Examples

A.1 Definition of $Lyn_\ell(i)$

▶ **Example 32.** Consider the string *ababcabcaabcaaa*. Its values $Lyn_0(i)$ are as follows:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a	b	a	b	c	a	b	c	a	a	b	c	a	a	a
8	1	3	2	1	3	2	1	4	3	2	1	1	1	1

A.2 Basic bounding method

A.2.1 Case $1 < i$ and $j < |u|$

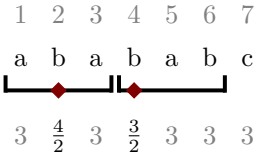
▶ **Example 33.** Consider the following string $u = dababab c$, which contains the run $(2, 7, 2)$.

1	2	3	4	5	6	7	8
d	a	b	a	b	a	b	c
3	3	$\frac{3}{2}$	3	$\frac{3}{2}$	3	3	3

Since $u[8] = c \prec_1 a = u[6]$, the special positions of r are $\{3, 5\}$ and we have $f(3) = f(5) = \frac{3}{2}$. Since we do not know much about other positions (yet), let us bound their f values by 3. This gives us a $\frac{3}{2} + \frac{3}{2} + 6 \cdot 3 = 21$ bound on $\sum_{i=1}^{|u|} f(i)$, which is a $\frac{21}{8} = 2.625$ average bound per a character of u.

A.2.2 Case $1 = i$ and $j < |u|$

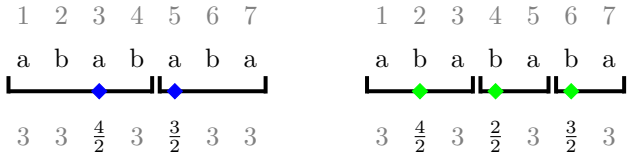
▶ **Example 34.** Consider the following string $u = abababc$, which contains the run $(1, 6, 2)$.



Since $u[7] = c \prec_1 a = u[5]$, the special positions of r are $\{2, 4\}$ and we have $f(2) = \frac{4}{2}$ and $f(4) = \frac{3}{2}$. This gives us a $\frac{4}{2} + \frac{3}{2} + 5 \cdot 3 = 18.5$ bound on $\sum_{i=1}^{|u|} f(i)$, which is an $\frac{18.5}{7} \approx 2.643$ average bound per a character of u.

A.2.3 Case $j = |u|$

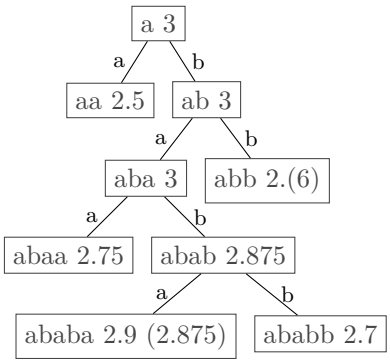
▶ **Example 35.** Consider the following string $u = abababa$, which contains the run $(1, 7, 2)$.



In the figure on the left, we use the $\prec_0$ order, and in the right figure, we use the $\prec_1$ order. In the first case, we have $S_r = \{3, 5\}$, while in the second case we have $S_r = \{2, 4, 6\}$. The bound for the left case is $3 + \frac{4}{2} + 3 + \frac{3}{2} + 3 = 12.5$ for the positions contained in those sets, while the right case gives a bound of $\frac{4}{2} + 3 + \frac{2}{2} + 3 + \frac{3}{2} = 10.5$. We pick the worse of those bounds and along with the bounds for the first and seventh position we get a bound of $3 + 12.5 + 3 = 18.5$ on $\sum_{i=1}^{|u|} f(i)$, which is an $\frac{18.5}{7} \approx 2.643$ average bound per a character of u.

A.3 Algorithm for contructing the trie

▶ **Example 36.** Consider the following left side of the tree that is in the process of being built.



After starting at node a with a bound of 3, we expanded to two leaves aa and ab. Since aa has a great per-character bound of 2.5, there is no need to touch it right now. Instead, we focus on gaining new context for the ab case. After that, we expand from the aba node and then from the $abab$ node. Notice that while the $ababa$ string has a per-character bound of 2.9, we assign it the value from its parent 2.875, since we could have made the cut earlier.

A.4 Detection of non-special positions

▶ **Example 37.** Consider the following string u that is part of some other string w.

1	2	3	4	5	6	7
a	b	a	b	b	a	a
3	$\frac{5}{2}$	~~3~~	3	2	3	2
		0				

String u has runs $(1, 4, 2)$, $(4, 5, 1)$ and $(6, 7, 1)$ which allow us to assign values of $\frac{5}{2}$, 2 and 2 to positions 2, 5 and 7, respectively. This gives us a bound of $3+\frac{5}{2}+3+3+2+3+2 = 18.5$, which is a per-character bound of $\frac{18.5}{7} \approx 2.643$.

However, let us focus on the third position and consider its assigned value. For $u[3]$ to be a special position we consider the order $\prec_0$ in which $w[3] = a \prec_0 b = w[2]$. We can see that the longest Lyndon word with respect to $\prec_0$ that starts at position 3 is of length 3. It cannot be longer, because the suffix starting three positions later will always be smaller. However, the period-maximal extension of the [3..5] interval is [2..6], which is one character short of being a run. We can thus conclude that $u[3]$ is never a special position and is assigned a value of 0. This improves our bound for u to $3 + \frac{5}{2} + 0 + 3 + 2 + 3 + 2 = 15.5$, which is a per-character bound of $\frac{15.5}{7} \approx 2.214$.

B Proof of Theorem 28

Proof. To prove this result, we give improved bounds for those leaves of the trie from Example 36 whose basic per-character bound is worse than 2.7. Those are $abaa$ and $ababa$. The negated strings $babb$ and $babab$ have the same bounds by symmetry.

Let us start with $u = abaa$, which is part of some string w. Let us consider two cases based on the character of w that is just before u. Let us denote this character $u[0]$.

- $u[0] = a$

0	1	2	3	4
a	a	b	a	a
–	2	3	3	2

  Since $u[0] = a$, we have a run $(0, 1, 1)$, which means 1 is a special position of a run with period 1 and can be bounded by 2. Position 4 can also be bounded by 2 as a special position for the run $(3, 4, 1)$.
- $u[0] = b$

0	1	2	3	4
b	a	b	a	a
–	$\frac{5}{2}$	3	3	2

Since $u[0] = b$, we have a run $(0, 3, 2)$, which means 1 is a special position of a run with period 2 and can be bounded by $\frac{5}{2}$. Position 4 can also be bounded by 2 as a special position for the run $(3, 4, 1)$.

It total, we get that u is bounded by $\max(2 + 3 + 3 + 2, \frac{5}{2} + 3 + 3 + 2) = 10.5$, which gives a per-character bound of $\frac{10.5}{4} = 2.625$.

Now, we consider $u = ababa$, which is part of some string w. Let us consider two cases based on the character of w that is just before u. Let us denote this character $u[0]$.

- $u[0] = a$

0	1	2	3	4	5
a	a	b	a	b	a
			◆		
–	2	3	$\frac{5}{2}$	3	3

0	1	2	3	4	5
a	a	b	a	b	a
		◆		◆	
–	2	$\frac{3}{2}$	3	$\frac{3}{2}$	3

Since $u[0] = a$, we have a run $(0, 1, 1)$, which means 1 is a special position of a run with period 1 and can be bounded by 2. We consider two cases based on which set out of $\{3\}$ and $\{2, 4\}$ is the set of special positions of the run $(1, 5, 2)$. We get a bound of $\max(2 + 3 + \frac{5}{2} + 3 + 3, 2 + \frac{3}{2} + 3 + \frac{3}{2} + 3) = 13.5$.

- $u[0] = b$

0	1	2	3	4	5
b	a	b	a	b	a
	◆		◆		
–	$\frac{4}{2}$	3	$\frac{3}{2}$	3	3

0	1	2	3	4	5
b	a	b	a	b	a
		◆		◆	
–	3	$\frac{4}{2}$	3	$\frac{3}{2}$	3

Since $u[0] = b$, we have just the run $(0, 5, 2)$. We consider two cases based on which set out of $\{1, 3\}$ and $\{2, 4\}$ is the set of special positions of the run $(0, 5, 2)$. We get a bound of $\max(\frac{4}{2} + 3 + \frac{3}{2} + 3 + 3, 3 + \frac{4}{2} + 3 + \frac{3}{2} + 3) = 12.5$.

It total, we get that u is bounded by $\max(13.5, 12.5) = 13.5$, which gives a per-character bound of $\frac{13.5}{5} = 2.7$. ◀

C Lower bound

C.1 Discovery

Let us outline the approach used in the discovery of the X_i sequence, which was based on heavy experimentation.

While analysing the best strings up to some length, one of the first observations one can make is that there are some patterns that are not present in them. One such pattern is aaa. This has a good theoretical justification – the value assigned to each a starting from the third one can be bounded by just 1. In general, having runs with more than one special position is a highly inefficient use of the positions of the string. Assuming that we prohibit such runs, we can see that the remaining sensible binary strings can mostly be made of $ababa$ and $ababbaba$ – non-terminals X_0 and X_1.

Now, we can repeat this procedure but with X_0 and X_1. We consider the best strings up to some length over the alphabet $\{X_0, X_1\}$ and look for patterns in them. This allows us to create new building blocks X_2, X_3, X_4, which we use in the next iteration.

After a few iterations, we reach a point where the improvements to the lower bound become negligible. This process generates the described grammar. It is also easy to verify the result since the cost of generating X_{20} is proportional to its length due to the way the sequence X_i is defined.

C.2 Comparison with the previously best result

The previously best lower bound was achieved by Crochemore et al. [6] by a sequence of strings, the best of which had a sum of exponents greater than $2.035n$. We present it below.

Let us define two morphisms $\phi : \{a, b, c\}^* \to \{a, b, c\}^*$ and $\psi : \{a, b, c\}^* \to \{0, 1\}^*$ as follows:

$$\phi(a) = baaba, \phi(b) = ca, \phi(c) = bca,$$

$$\psi(a) = 01011, \psi(b) = \psi(c) = 01001011.$$

Let $w_i = \psi(\phi^i(a))$. A comparison of the sequence X_i and the sequence w_i is presented in Figure 1.

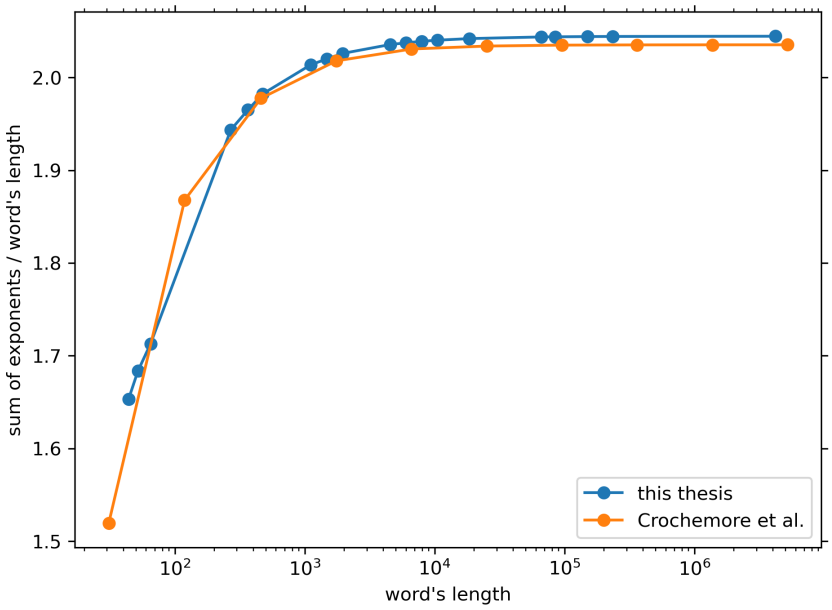


Figure 1 Comparison of the X_i sequence and the result of Crochemore et al. [6].