Alexander S. Kulikov
Sofya Raskhodnikova (Eds.)

LNCS 13296

Computer Science – Theory and Applications

17th International Computer Science Symposium in Russia, CSR 2022
Virtual Event, June 29 – July 1, 2022
Proceedings

Springer

Abelian Repetition Threshold Revisited

Elena A. Petrova and Arseny M. Shur(✉)

Ural Federal University, Ekaterinburg, Russia
{elena.petrova,arseny.shur}@urfu.ru

Abstract. In combinatorics on words, *repetition thresholds* are the numbers separating avoidable and unavoidable repetitions of a given type in a given class of words. For example, the meaning of the "classical" repetition threshold $\mathsf{RT}(k)$ is "every infinite k-ary word contains an α-power of a nonempty word for some $\alpha \geq \mathsf{RT}(k)$ but some infinite k-ary words contain no such α-powers with $\alpha > \mathsf{RT}(k)$". It is well known that $\mathsf{RT}(k) = \frac{k}{k-1}$ with the exceptions for $k = 3, 4$.

For *Abelian* repetition threshold $\mathsf{ART}(k)$, avoidance of fractional Abelian powers of words is considered. The exact values of $\mathsf{ART}(k)$ are unknown; the lower bound $\mathsf{ART}(2) \geq \frac{11}{3}$, $\mathsf{ART}(3) \geq 2$, $\mathsf{ART}(4) \geq \frac{9}{5}$, $\mathsf{ART}(k) \geq \frac{k-2}{k-3}$ for all $k \geq 5$ was proved by Samsonov and Shur in 2012 and conjectured to be tight. We present a method of study of Abelian power-free languages using random walks in prefix trees and some experimental results obtained by this method. On the base of these results, we suggest that the lower bounds for $\mathsf{ART}(k)$ by Samsonov and Shur are not tight for all k except $k = 5$. We prove $\mathsf{ART}(k) > \frac{k-2}{k-3}$ for $k = 6, 7, 8, 9, 10$ and state a new conjecture on the Abelian repetition threshold.

Keywords: Abelian-power-free language · repetition threshold · prefix tree · random walk

1 Introduction

Two words are *Abelian equivalent* (A-equivalent) if they have the same multiset of letters; in other terms, if they are anagrams of each other, like English words *knee* and *keen*. *Abelian repetition* (A-repetition) is a pair of A-equivalent factors in a word. The study of A-repetitions originated from the question of Erdős [11]: does there exist an infinite finitary word having no consecutive pair of A-equivalent factors? The factors of the form uu', where u and u' are A-equivalent, are now called *Abelian squares* (A-squares). In modern terms, Erdős's question can be phrased as "are A-squares avoidable over some finite alphabet?" This question was answered in the affirmative by Evdokimov [12]; the smallest possible alphabet has cardinality 4, as was proved by Keränen [14]. In a similar way, kth A-powers are defined for arbitrary $k \geq 2$. Dekking [9] constructed infinite

E. A. Petrova—Supported by the Ministry of Science and Higher Education of the Russian Federation, project FEUZ-2020-0016.

A. M. Shur—Supported by Ural Mathematical Center, project 075-02-2022-877.

A. S. Kulikov and S. Raskhodnikova (Eds.): CSR 2022, LNCS 13296, pp. 302–319, 2022.
https://doi.org/10.1007/978-3-031-09574-0_19

ternary words without A-cubes and infinite binary words without 4th A-powers. The results by Dekking and Keränen form an Abelian analog of the seminal result by Thue [23]: there exist an infinite ternary word containing no squares (factors of the form uu) and an infinite binary word containing no cubes (factors of the form uuu).

Thue's result is considered as the origin of combinatorics on words and, in particular, of avoidability theory. Later this result was strengthened using *fractional* powers of a word: given a word u of length n, take a length-m prefix v of the infinite word $uuu\cdots$; then v is the $(\frac{m}{n})$th power of u ($m > n$ is assumed). Usually, v is referred to as an $(\frac{m}{n})$-power. A word is said to be α-*free* if it contains no $(\frac{m}{n})$-powers with $\frac{m}{n} \geq \alpha$. Fractional powers gave rise to the notion of *repetition threshold* which is the function

$$\mathsf{RT}(k) = \inf\{\alpha : \text{there exists an infinite } k\text{-ary } \alpha\text{-free word}\}.$$

The value $\mathsf{RT}(2) = 2$ is due to Thue [24]. Dejean [8] showed that $\mathsf{RT}(3) = 7/4$ and conjectured the remaining values $\mathsf{RT}(4) = 7/5$ (proved by Pansiot [16]) and $\mathsf{RT}(k) = \frac{k}{k-1}$ for $k \geq 5$ (proved by efforts of many authors [1,3,15,20]). Since the proof of Dejean's conjecture, a number of related results appeared, conjecturing and establishing similar thresholds for stronger or weaker restrictions on repetitions [4,6,25], for restricted classes of words [7,10,19,22], and for different models of words [5,13].

An extension of the notions of fractional power and repetition threshold to the case of A-powers was proposed by Cassaigne and Currie [2] for the case $m < 2n$ and by Samsonov and Shur [21] for the general case. Integral A-powers can be generalized to fractional ones in several ways; however, for the case $m < 2n$, one definition of an $(\frac{m}{n})$-A-power is preferable due to its symmetric nature. According to this definition, a word vuv' is an $(\frac{m}{n})$th A-power of the word vu if $|vu| = n$, $|vuv'| = m$, and v' is A-equivalent to v (in [21], such A-powers were called *strong*). Note that the reversal of an $(\frac{m}{n})$-A-power is also an $(\frac{m}{n})$-A-power. In this paper, we consider *only* strong A-powers; see Sect. 2 for the definition in the case $m > 2n$. Given the definition of fractional A-powers, one naturally defines α-A-free words and *Abelian repetition threshold*

$$\mathsf{ART}(k) = \inf\{\alpha : \text{there exists an infinite } k\text{-ary } \alpha\text{-A-free word}\}.$$

In [2], it was shown that for any $\varepsilon > 0$ there exists a $(1+\varepsilon)$-A-free word over an alphabet of size $2^{\mathrm{poly}(\varepsilon^{-1})}$. This bound is very loose but proves that $\lim_{k\to\infty} \mathsf{ART}(k) = 1$. In [21], the lower bound $\mathsf{ART}(k) \geq \frac{k-2}{k-3}$ for $k \geq 5$ was proved and conjectured to be tight; in full, this conjecture is as follows.

Conjecture 1 ([21]). $\mathsf{ART}(2) = 11/3$; $\mathsf{ART}(3) = 2$; $\mathsf{ART}(4) = 9/5$; $\mathsf{ART}(k) = \frac{k-2}{k-3}$ for $k \geq 5$.

No exact values of $\mathsf{ART}(k)$ are known, and no new bounds have appeared since [21]. One reason for the lack of progress in estimating $\mathsf{ART}(k)$ is the fact that the language $\mathsf{AF}(k,\alpha)$ of all k-ary α-A-free words can be finite but so huge that it cannot be enumerated by exhaustive search.

In the present study, we propose the following approach. The language $\mathsf{AF}(k, \alpha)$ is viewed as the *prefix tree* $\mathcal{T}_{k,\alpha}$: the elements of $\mathsf{AF}(k, \alpha)$ are nodes of the tree and u is an ancestor of v in the tree iff u is a prefix of w. In the same way we treat the languages $\mathsf{AF}(k, \alpha^+) = \bigcap_{\beta>\alpha} \mathsf{AF}(k, \beta)$. Hence it suffices to decide the (in)finiteness of $\mathcal{T}_{k,\alpha}$ and $\mathcal{T}_{k,\alpha^+}$ to compare α to $\mathsf{ART}(k)$. We construct random walks in such trees, using random depth-first search from the root (empty word). Analysing the statistics of such walks, we conjecture (in)finiteness of the studied trees. Our contribution can be summarized as follows:

- to speed up the search in prefix trees, we developed several algorithms to decide whether a word ua is α-A-free for a given α-A-free word u and a letter a; for all algorithms, we proved correctness and complexity guarantees;
- we ran multiple experiments for each of about 20 languages with unknown finiteness status and gathered the statistics;
- depending on statistics of constructed random walks, we classified most of the studied languages either as "finite-like" or as "infinite-like";
- we proved by an optimized exhaustive search that the finite-like languages $\mathsf{AF}(k, \frac{k-2}{k-3}^+)$ for $k = 6, 7, 8, 9, 10$ are finite, thus disproving Conjecture 1;
- we replaced Conjecture 1 with the following conjecture:

Conjecture 2. $\mathsf{ART}(2) > 11/3$; $2 < \mathsf{ART}(3) \le 5/2$; $\mathsf{ART}(4) > 9/5$; $\mathsf{ART}(5) = 3/2$; $4/3 < \mathsf{ART}(6) < 3/2$; $\mathsf{ART}(k) = \frac{k-3}{k-4}$ for $k \ge 7$.

2 Definitions and Notation

We study finite words over finite alphabets, using the standard notation Σ for a (linearly ordered) alphabet, σ for its size, Σ^* for the set of all finite words over Σ, including the empty word λ. For a length-n word $u \in \Sigma^*$ we write $u = u[1..n]$; the elements of the range $[1..n]$ are *positions* in u, and the length of u is denoted by $|u|$. A word w is a *factor* of u if $u = vwz$ for some (possibly empty) words v and z; the condition $v = \lambda$ (resp., $z = \lambda$) means that w is a *prefix* (resp., *suffix*) of u. Any factor w of u can be represented as $w = u[i..j]$ for some i and j ($j < i$ means $w = \lambda$). A factor w of u can have several such representations; we say that $u[i..j]$ specifies the *occurrence of w at position i*.

A *k-power* of a word u is the concatenation of k copies of u, denoted by u^k. This notion can be extended to *α-powers* for an arbitrary rational $\alpha > 1$. The α-power of u is the word $u^\alpha = u \cdots uu'$ such that $|u^\alpha| = \alpha|u|$ and u' is a prefix of u. A word is *α-free* (resp., *α^+-free*) if no one of its factors is a β-power with $\beta \ge \alpha$ (resp., $\beta > \alpha$).

The *Parikh vector* $\Psi(u)$ of a word $u \in \Sigma^*$ is an integer vector of length σ whose coordinates are the numbers of occurrences of the letters from Σ in u. Thus, the word *acabac* over the alphabet $\Sigma = \{a < b < c < d\}$ has the Parikh vector $(3, 1, 2, 0)$. Two words u and v are A-equivalent (denoted by $u \sim v$) iff $\Psi(u) = \Psi(v)$. The *reversal* of a word $u = u[1..n]$ is the word $u^R = u[n]u[n-1] \cdots u[1]$. Clearly, $u \sim u^R$. A nonempty word u is a (*k-A-power*) if $u = w_1 \cdots w_k$, where $w_i \sim w_j$ for all indices i, j. A 2-A-power is an *A-square*,

and a 3-A-power is an *A-cube*. Thus, k-A-powers generalize k-powers by relaxing the equality of factors to their A-equivalence. However, there are many ways to generalize the notion of an α-power to the Abelian case, and all of them have certain drawbacks. The reason is that $u \sim v$ implies $u[i..j] \sim v[i..j]$ for *no* pair of proper factors of u and v. If $1 < \alpha \leq 2$, we define an α-A-power as a word vuv' such that $\frac{|vuv'|}{|vu|} = \alpha$ and $v \sim v'$. The advantage of this definition is that the reversal of an α-A-power is an α-A-power as well. For $\alpha > 2$ the situation is worse: no natural definition compatible with the definition of k-A-power is symmetric with respect to reversals (see [21] for more details). So we give a definition which is compatible with the case $\alpha \leq 2$: an α-A-power is a word $u_1 \cdots u_k u'$ such that $\frac{|u_1 \cdots u_k u'|}{|u_1|} = \alpha$, $k = \lfloor \alpha \rfloor$, $u_1 \sim \cdots \sim u_k$, and u' is A-equivalent to a prefix of u_1. In [21], such words are called *strong* α-A-powers. For a given α, *α-A-free* and *α^+-A-free* words are defined in the same way as α-free (α^+-free) words. It is convenient to extend the set of rationals with "numbers" of the form α^+, postulating the equivalence of the inequalities $\beta > \alpha$ and $\beta \geq \alpha^+$.

A *language* is any subset of Σ^*. The *reversal* L^R of a language L consists of the reversals of all words in L. The α-A-free language over Σ (where α belongs to extended rationals) consists of all α-A-free words over Σ and is denoted by $\mathsf{AF}(\sigma, \alpha)$. These languages are the main objects of the studies aimed at finding the *Abelian repetition threshold* $\mathsf{ART}(k) = \inf\{\alpha : \mathsf{AF}(k, \alpha) \text{ is infinite}\}$. The languages $\mathsf{AF}(k, \alpha)$ are closed under permutations of the alphabet: if π is a permutation of Σ, then the words $u, \pi(u) \in \Sigma^*$ are α-A-free for exactly the same values of α. Hence the words in a language $\mathsf{AF}(\sigma, \alpha)$ can be enumerated by considering only lexicographically minimal (*lexmin*) words: a word $u \in \Sigma^*$ is lexmin if $u < \pi(u)$ for any permutation π of Σ.

Suppose that a language L is *factorial* (i.e., closed under taking factors of its words); for example, all languages $\mathsf{AF}(k, \alpha)$ are factorial. Then L can be represented by its *prefix tree* $\mathcal{T}_L$, which is a rooted labeled tree whose nodes are elements of L and edges have the form $u \xrightarrow{a} ua$, where a is a letter. Thus u is an ancestor of v iff u is a prefix of v. We study the languages $\mathsf{AF}(\sigma, \alpha)$ through different types of search in their prefix trees.

3 Algorithms

In this section we present the algorithms we develop for use in experiments. First we describe the random depth-first search in the prefix tree $\mathcal{T} = \mathcal{T}_L$ of an arbitrary factorial language L. Given a number N, the algorithm visits N distinct nodes of $\mathcal{T}$ following the depth-first order and returns the maximum level of a visited node. The search can be easily augmented to return the word corresponding to the node of maximum level, or to log the sequence of levels of visited nodes. Algorithm 1 below describes one iteration of the search. In the algorithm, $u = u[1..n]$ is the word corresponding to the current node; $\mathsf{Set}[u]$ is the set of all letters a such that the search has not tried the node ua yet; ml is the maximum level reached so far; count is the number of visited nodes; $\mathcal{L}(u)$ is the predicate returning true if $u \in L$ and false otherwise. The lines 3 and 8 refer

to the updates of data structures used to compute $\mathcal{L}(u)$. The search starts with $u = \lambda$, $\mathsf{ml} = 0$, $\mathsf{count} = 1$. A variant of this search algorithm was used in [17] to numerically estimate the entropy of some α-free and α-A-free languages.

Algorithm 1. Random depth-first search in $\mathcal{T}_L$: one iteration

```
1: if count = N then break                          ▷ search finished
2: if Set[u] = ∅ then                               ▷ all children of u were visited
3:     [update data structures]
4:     u ← u[1..|u|−1]                              ▷ return to the parent of u
5: else
6:     a ← random(Set[u]); Set[u] ← Set[u] − a       ▷ take random unused letter
7:     if L(ua) then                                ▷ the node ua is in T(L)
8:         [update data structures]
9:         u ← ua; Set[u] ← Σ; count ← count + 1     ▷ visit ua next
10:        if |u| > ml then ml ← |u|                 ▷ update the maximum level
```

The key to an efficient search is a fast algorithm computing the predicate $\mathcal{L}(ua)$ (line 7); note that $\mathcal{L}(u) = \mathsf{true}$ as the search reached u. In the rest of the section we present such algorithms for $L = \mathsf{AF}(\sigma, \alpha)$. The summary of the algorithms is given in Table 1. As the search calls $\mathcal{L}()$ at least N times, the expected query time (computed for the uniformly random word u) is more informative than the worst-case time. Throughout the paper, we treat σ as a constant.

Table 1. Time and space usage of the algorithms detecting A-powers. Update time refers to a single data structure update (line 3 or 8 of Algorithm 1), query time refers to the computation of $\mathcal{L}$ (line 7 of Algorithm 1); n is the length of the processed word.

Algorithm	Powers	Update time	Expected query time	Space
Algorithm 2	$\alpha < 2$	$O(1)$	$O(n^{3/2})$	$O(n)$
Algorithm 3	$\alpha \le 3/2$	$O(n)$	$O(n)$	$O(n^2)$
Algorithm 4	$\alpha > 2$	$O(1)$	$O(n^{3/2})$	$O(n)$
Algorithm 5	$\alpha > 2$, dual	$O(1)$	$O(n^{1/2})$	$O(n)$

3.1 Avoiding Small Powers

Let $\alpha < 2$ and u be a word of length n such that all proper prefixes of u are α-A-free. To prove u to be α-A-free, it is necessary and sufficient to show that

($\star$) no suffix of u can be written as xyz such that $|z| > 0$, $x \sim z$, and $\frac{|xyz|}{|xy|} \ge \alpha$.

Remark 1. Since A-equivalence is not closed under taking any sort of factors of words, the ratio $\frac{|xyz|}{|xy|}$ in ($\star$) can significantly exceed α. For example, all proper prefixes, and even all proper factors, of the word $u = abcde\,bdaec$ are $\frac{3}{2}$-A-free, while u is an A-square. Hence for each suffix z of u one should check multiple

candidates for the factor x in $(\star)$. The number of such candidates can be as big as $\Theta(n)$; in total, $\Theta(n^2)$ candidates for the pair (x, z) should be analysed.

Remark 1 suggests a simple algorithm with $O(n^2)$ query time, which stores Parikh vectors of all prefixes of u and checks each candidate pair (x, z) directly. Let us describe a more efficient procedure. We store two length-n arrays for each letter $a \in \Sigma$: $c_a[i] = \Psi(u[1..i])(a)$ is the number of occurrences of a in $u[1..i]$ and $d_a[i]$ is the position of ith from the left letter a in the word u. The arrays are updated (lines 3,8 of Algorithm 1) as follows: in line 3, we delete $\Psi(u)$ from c-arrays and delete the last element of $d_{u[|u|]}$; in line 8, we add $\Psi(ua)$ to c-arrays and add a new element $|ua|$ to d_a. Hence each update takes $O(1)$ time. We use two auxiliary functions: $\mathsf{Parikh}(i, j)$ returns $\Psi(u[i..j])$; $\mathsf{cover}(\vec{P}, j)$ returns the biggest number i such that $\Psi(u[i..j]) \geq \vec{P}$ or zero if no such number exists.

Lemma 1. *Functions* $\mathsf{Parikh}(i, j)$ *and* $\mathsf{cover}(\vec{P}, j)$ *are computable in* $O(1)$ *time.*

Proof. Coordinates of $\mathsf{Parikh}(i, j)$ are the differences $c_a[j] - c_a[i-1]$. Further, if $c_a[j] < \vec{P}[a]$ holds for some letter a, then $\Psi(u[1..j]) \not\geq \vec{P}$ and $\mathsf{cover}(\vec{P}, j) = 0$; if all these inequalities fail, then $i = \min_{a \in \Sigma}\{d_a[c_a[j] - \vec{P}[a] + 1]\}$. □

Algorithm 2. A-powers detection (case $\alpha < 2$)

```
1: function alphafree(u)                                   ▷ u=word; n = |u|
2: free ← true                                             ▷ α-A-freeness flag
3: for i = n downto 1+⌈n/2⌉ do                             ▷ z = u[i..n]
4:     right ← i − 1
5:     len ← n − i + 1; P⃗ ← Parikh(i, n)                   ▷ length and Parikh vector of z
6:     left ← cover(P⃗, right)                              ▷ v = u[left..right], Ψ(v) ≥ Ψ(z)
7:     if left = 0 then break                              ▷ Ψ(u[1..right]) ≱ Ψ(z)
8:     while left ≥ max{1, ⌈(αi−1−n)/(α−1)⌉} do            ▷ guarantees |xyz|/|xy| ≥ α
9:         if right − left + 1 = len then                  ▷ x = u[left..right] ∼ z
10:            free ← false; break
11:        else                ▷ shift right leftwards, skip redundant comparisons
12:            right ← left + len − 1
13:        left ← cover(P⃗, right)
14:    if free = false then break
15: return free                                            ▷ the answer to "is u α-A-free?"
```

Proposition 1. *Let* α *be a number such that* $1 < \alpha < 2$ *and* u *be a word all proper prefixes of which are* α*-A-free. Then Algorithm 2 correctly detects whether* u *is* α*-A-free.*

Proof. Let us show that Algorithm 2 verifies the condition $(\star)$. The outer cycle of the algorithm fixes the first position i of the suffix z of u; the suffixes are analysed in the order of increased length $len = |z|$. If a forbidden suffix xyz is detected

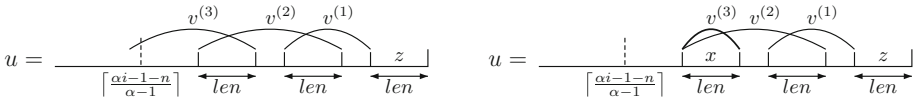

Fig. 1. Illustrating the proof of Proposition 1. Processing the suffix z, Algorithm 2 successively finds three words v satisfying $\Psi(v) \geq \Psi(z)$. On the left picture, the position *left* of $v^{(3)}$ is smaller than the bound in line 8, so the verification of $(\star)$ for z is finished. On the right picture, $|v^{(3)}| = |z|$, so a forbidden suffix, starting with $v^{(3)}$, is detected.

during the iteration, then the algorithm breaks the outer cycle in line 14 and returns false. Thus at the current iteration of the outer cycle the condition $(\star)$ is already verified for all shorter suffixes. The iteration uses a simple observation: if $x \sim z$, then every word v, containing x, satisfies $\Psi(v) \geq \Psi(z)$. Respectively, we fix the rightmost position *right* where a factor x satisfying $x \sim z$ can end. Initially $right = i - 1$ as x can immediately precede z (see Remark 1). Then we compute the shortest factor $v = u[left..right]$ such that $\Psi(v) \geq \Psi(z)$. If $v = x$, the suffix xz of u violates $(\star)$. Otherwise x cannot begin later than at the position *left* by the construction of v. Hence we decrease *right* by setting $right = left + |z| - 1$ and repeat the above procedure in a loop. The verification of $(\star)$ for z ends successfully either if v does not exist (i.e., $\Psi(u[1..right]) \not\geq \Psi(z)$ for the current value of *right*) or *left* is too small (i.e., $xyz = u[left..n]$ with $|x| = |z|$ means $\frac{|xyz|}{|xy|} < \alpha$). The described process is illustrated by Fig. 1.

Details are as follows. In lines 4–6 the algorithm calls Parikh to compute $\Psi(z)$ and cover to find $v = u[left..right]$ for $right = i - 1$. If $left = 0$, then $\Psi(u[1..i-1]) \not\geq \Psi(u[i..n])$ and hence no suffix xyz of u satisfies $x \sim z$. Moreover, one has $\Psi(u[1..j-1]) \not\geq \Psi(u[j..n])$ for each $j < i$ which immediately verifies $(\star)$ for all longer suffixes of u. Hence in this case the verification of $(\star)$ is finished; respectively, the algorithm breaks the outer cycle in line 7 and returns true. If no break happened, the algorithm enters the inner cycle, checks whether $v = x$ (line 9) and breaks with the output false if this condition holds. If it does not, the algorithm decreases *right* as described above (line 12) and computes the new factor v (line 13). If v does not exist, *left* gets 0, which results in the immediate exit from the inner cycle. If v is computed but its position is too small, then the cycle is also exited. The exit means the end of the ith iteration.

Thus, Algorithm 2 returns false only if it finds a suffix xyz of u which violates $(\star)$. For the other direction, let $xyz = u[j..n]$ violate $(\star)$ such that $|z|$ is minimal over all suffixes violating it. Then Algorithm 2 cannot stop before the iteration which checks z. During this iteration, *left* cannot become smaller than j by the definition of the factor v. As *right* decreases at each iteration of the inner cycle, eventually x will be found. Thus the algorithm indeed verifies $(\star)$ and thus detects the α-A-freeness of u. □

Remark 2. By Lemma 1, Algorithm 2 processes a length-n word u in $O(K + n)$ time, where K is the number of the inner cycle iterations during the course of the algorithm. Clearly, $K = O(n^2)$. If u is random, the expected length of a word

v built by Algorithm 2 when processing a suffix z is $|z| + \Omega(\sqrt{|z|})$, as follows from a technical Lemma 2 below. This means $K = O(n^{3/2})$ on expectation, and this is exactly what we have seen in the experiments.

Lemma 2. *Suppose that finite word z and an infinite word $\mathbf{w}$ are chosen uniformly at random over a finite alphabet Σ, and v is the shortest prefix of $\mathbf{w}$ such that $\Psi(v) \geq \Psi(z)$. Then the expected length of v is $|z| + \Omega(\sqrt{|z|})$.*

Proof. Let $\ell = |z|$, $\delta = |v| - |z|$. First consider $\Sigma = \{0, 1\}$. Then the process is as follows: z is generated by ℓ tosses of a fair coin; another ℓ tosses generate some prefix x of v; then tosses are made one by one until the result $\Psi(v) \geq \Psi(z)$ is reached after δ tosses. The Parikh vector of a binary word is determined by its length and the number of 1's. Hence $\Psi(z)$ and $\Psi(x)$ are random variables ξ, η with the binomial distribution $\mathsf{bin}(\ell, \frac{1}{2})$. The vector $\Psi(z) - \Psi(x)$ has the form $(-m, m)$ for some integer m. To obtain v, we should make $|m|$ "successful" tosses with the probability of success being $1/2$; hence the expectation of δ equals $2|m|$. Thus it remains to find the expectation of $|m| = |\xi - \eta|$. Since $E(\xi - \eta) = 0$, we see that $E(|\xi - \eta|)$ is the standard deviation of $\xi - \eta$ by definition.

By symmetry, η and $\ell - \eta$ have the same distribution. Hence we can replace $\xi - \eta$ by $\xi + \eta - \ell$. The random variable $\xi + \eta$ has the distribution $\mathsf{bin}(2\ell, \frac{1}{2})$, so its standard deviation is $\sqrt{\ell/2}$. Thus $E(\delta) = 2E(|\xi - \eta|) = \sqrt{2\ell} = \Omega(\sqrt{\ell})$.

Over larger alphabets the expectation of δ can only increase. The easiest way to see this is to split Σ arbitrarily into two subsets K_1 and K_2 of equal size. Then x with respect to z has a deficiency of letters from one of these subsets, say, K_1. By the argument for the binary alphabet, $\sqrt{2\ell}$ additional letters is needed, on expectation, to cover this deficiency. This is a necessary (but not sufficient) condition to obtain the word v. Hence $E(\delta) = \Omega(\sqrt{\ell})$. □

For the case $\alpha \leq 3/2$ we present a much faster dictionary-based Algorithm 3. Recall that a dictionary contains a set of pairs (key, value), where all keys are unique, and supports fast lookup, addition and deletion by key ($O(1)$ expected time per operation with the help of hash tables). For the dictionary *dict* used in Algorithm 3, the keys are Parikh vectors and the values are lists of positions, in the increasing order, of the factors having this Parikh vector. The algorithm accesses only the last (maximal) element of the list. The updates of the dictionary (lines 3,8 of Algorithm 1) are as follows. At line 3, we delete all suffixes of u from the dictionary. For a suffix z, this means the deletion of the last element from the list $dict[\Psi(z)]$; if the list becomes empty, the entry for $\Psi(z)$ is also deleted. At line 8, all suffixes of ua are added to the dictionary. For a suffix z, if $\Psi(z)$ was not in the dictionary, an entry is created; then the position $|ua| - |z| + 1$ is added to the end of the list $dict[\Psi(z)]$. Thus when answering the query for $u = u[1..n]$, Algorithm 3 has in the dictionary all factors of $u[1..n-1]$ up to the A-equivalence, at the expense of $O(n)$ update time and $O(n^2)$ space.

Proposition 2. *Let α be a number such that $1 < \alpha \leq 3/2$ and u be a word all proper prefixes of which are α-A-free. Then Algorithm 3 correctly detects whether u is α-A-free.*

Algorithm 3. Dictionary-based A-powers detection ($\alpha \le 3/2$)

```
1: function alphafreedict(u)                          ▷ u=word; n = |u|
2: free ← true                                        ▷ α-A-freeness flag
3: P⃗ ← 0⃗
4: for i = n downto 1+⌈n/2⌉ do                        ▷ z = u[i..n]
5:     len ← n − i + 1                                ▷ length of z
6:     P⃗[u[i]] ← P[u[i]] + 1                         ▷ get Ψ(z) from Ψ(u[i+1..n])
7:     pos ← dict[P⃗].last      ▷ position of last occurrence of some x ∼ z, if exists
8:     if pos ≤ i − len and pos ≥ ⌈(αi−1−n)/(α−1)⌉ then   ▷ xyz is forbidden
9:         free ← false; break
10: return free                                       ▷ the answer to "is u α-A-free?"
```

Proof. Let us show that Algorithm 3 verifies ($\star$). First suppose that the algorithm returned false. Then it broke from the **for** cycle (line 9); let z be the last suffix processed. The lookup by the key $\Psi(z)$ returned the position pos of a factor $x \sim z$, and the condition in line 8 was true. The first inequality in line means that x and z do not overlap in u; the second inequality is equivalent to $\frac{|xyz|}{|xy|} \ge \alpha$. Therefore, the suffix $xyz = u[pos..n]$ violates ($\star$).

Now suppose that the algorithm returned true. Aiming at a contradiction, assume that u has a suffix violating ($\star$). Let xyz ($x \sim z$) be the shortest such suffix. Consider the iteration of the **for** cycle where z was processed. The key $\Psi(z)$ was present in the dictionary because $x \sim z$. If pos (line 7) corresponded to the x from our "bad" suffix, i.e., $xyz = u[pos..n]$, then both inequalities in line 8 held because x and z do not overlap in u and $\frac{|xyz|}{|xy|} \ge \alpha$. But then the algorithm would have returned false, contradicting our assumption. Hence pos was the position of some other $x' \sim z$ which occurs in u later than x. By the choice of the suffix xyz, u cannot have shorter suffix $x'y'z$ with $x' \sim z$. This means that the occurrences of x' and z overlap (see Fig. 2).

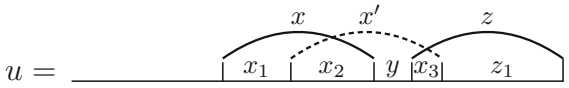

Fig. 2. Location of Abelian equivalent factors (Proposition 2).

Note that x' and x also overlap. Otherwise, xyz has a prefix of the form $x\hat{y}x'$ and $\frac{|x\hat{y}x'|}{|x'\hat{y}|} \ge \frac{|xyz|}{|xy|} \ge \alpha$, contradicting the condition that all proper prefixes of u are α-A-free. Then $x = x_1x_2$, $x' = x_2yx_3$, $z = x_3z_1$, as shown in Fig. 2, and x_1, x_2, x_3, z_1 are nonempty. We observe that $x \sim x' \sim z$ imply $x_1 \sim yx_3$ and $x_2y \sim z_1$. By the condition on the prefixes of u, $\frac{|x_1x_2yx_3|}{|x_1x_2|} < \alpha \le 3/2$. Hence $|yx_3| < |x_2|$ and then $|x_3| < |x_2y| = |z_1|$. Therefore $\frac{|x_2yx_3z_1|}{|x_2yx_3|} > 3/2 \ge \alpha$, so the suffix $x'z_1 = x_2yx_3z_1$ of u violates ($\star$). But $|x'z_1| < |xyz|$, contradicting the choice of xyz. This contradiction proves that u satisfies ($\star$). □

As Algorithm 3 consists of a single cycle, the next statement is immediate.

Proposition 3. *For a word of length n, Algorithm 3 performs $O(n)$ operations, including dictionary operations.*

Remark 3. A slight modification of Algorithm 3 allows one to process the important case $\alpha = (3/2)^+$ within the same complexity bound. The argument from the proof of Proposition 2 remains valid for $\alpha = (3/2)^+$ except for one specific situation: in Fig. 2 it is possible that y is empty and $|x_1| = |x_2| = |x_3| = |x_4|$. Here Algorithm 3 misses the A-square xz. To fix this, we add a patch after line 7:

7.5: **if** $pos = i - len/2$ **then** $pos \leftarrow pos.next$

As an example, consider $u = abcdbadc$. Processing the suffix $z = badc$, Algorithm 3 retrieves $pos = 3$ from the dictionary by the key $\Psi(z)$. The corresponding factor $x' = cdba$ overlaps z and the condition in line 8 would fail for pos. However, pos satisfies the condition in the inserted line 7.5 and thus the factor $x = abcd$ at $pos = 1$ will be reached. The condition in line 8 holds for $pos = 1$ and the A-square is detected.

Remark 4. Algorithm 3 can be further modified to work for all $\alpha < 2$. If we replace the patch from Remark 3 with the following one:

7.5: **while** $pos > i - len$ **do** $pos \leftarrow pos.next$

the algorithm will find the closest factor $x \sim z$ which does not overlap with z. This new patch introduces an inner cycle and thus affects the time complexity but the algorithm remains faster in practice than Algorithm 2.

3.2 Avoiding Big Powers

Let $\alpha > 2$. The case $2 < \alpha < 3$ for ternary words and the case $3 < \alpha < 4$ for binary words are relevant to the studies of Abelian repetition threshold. We provide here the algorithms for the first case; the algorithms for the second case are very similar (the only difference is that one should check for A-cubes instead of A-squares). A β-A-power with $\alpha \le \beta \le 3$ has the form $ZZ'z$, where $Z \sim Z'$, z is equivalent to a prefix of Z, and $\frac{|ZZ'z|}{|Z|} \ge \alpha$. We write the A-square ZZ' as xy where $|x| \le |y|$ and $x \sim z$. Consequently, if all proper prefixes of a word u are α-A-free, then u is α-A-free iff the following analog of $(\star)$ holds:

$(*)$ no suffix of u can be written as xyz such that $|y| \ge |x| > 0$, $x \sim z$, $\frac{2|xyz|}{|xy|} \ge \alpha$, and xy is an Abelian square.

Verifying $(*)$ for u, we process its suffix z as follows. Within the range determined by α, we search for all factors $x = u[left..right]$ such that $x \sim z$ (see Fig. 1). For each x we consider the corresponding suffix xyz of u and check whether xy is an Abelian square. If yes, xyz violates $(*)$. Algorithm 4 below, as well as the next Algorithm 5, uses the same data structure and auxiliary functions as Algorithm 2 and thus has $O(1)$ update time and use $O(n)$ space.

Proposition 4. *Let α be a number such that $2 < \alpha < 3$ and u be a word all proper prefixes of which are α-A-free. Then Algorithm 4 correctly detects whether u is α-A-free.*

Algorithm 4. A-powers detection (case $2 < \alpha < 3$)

```
1: function ALPHAfree(u)                                   ▷ u=word; n = |u|
2: free ← true                                             ▷ α-A-freeness flag
3: for i = n downto 1+⌈2n/3⌉ do                            ▷ z = u[i..n]
4:     right ← i − 1
5:     len ← n − i + 1; P⃗ ← Parikh(i, n)                   ▷ length and Parikh vector of z
6:     left ← cover(P⃗, right)
7:     if left = 0 then break                              ▷ Ψ(u[1..right]) ≱ Ψ(z)
8:     while left ≥ max{1, ⌈(αi−1−2n)/(α−2)⌉} do            ▷ guarantees 2|xyz|/|xy| ≥ α
9:         if left + len − 1 = right then                  ▷ x = u[left..right] ∼ z
10:            if 2 | (i − left) and Σ_{a∈Σ} |c_a[i−1] + c_a[left−1] − 2c_a[(i+left)/2 − 1]| = 0 then
11:                free ← false; break                     ▷ xy = u[left..i−1] is an Abelian square
12:            else
13:                right ← right − 1                       ▷ right bound for the next search
14:        else
15:            right ← left + len − 1                      ▷ right bound for the next search
16:        left ← cover(P⃗, right)
17:    if free = false then break
18: return free                                            ▷ the answer to "is u α-A-free?"
```

Proof. Algorithm 4 is similar to Algorithm 2, so we focus on their difference. If some suffix xyz violates $(*)$, then $|z| \le |xyz|/3 \le n/3$; hence the range for the outer cycle in line 3. For a fixed z we repeatedly seek for the shortest factor $v = u[left..right]$ with the given right bound and the property $\Psi(v) \ge \Psi(z)$. If $|v| = |z|$ (condition in line 9 holds), then v is a candidate for x in the suffix xyz violating $(*)$. The initial value for *right* (line 4) is set to ensure $|x| \le |y|$. The candidate found in line 9 is checked in line 10 for the remaining condition: xy is an Abelian square. Namely, we check that $|xy|$ is even and its left and right halves have the same Parikh vector. If this condition holds, the algorithm breaks both inner and outer cycles and returns false. If the condition fails, we decrease *right* by 1 and compute the factor v for this new right bound. The rest is the same as in Algorithm 2. So we can conclude that Algorithm 4 verifies $(*)$. □

Remark 5. As Algorithm 4 shares the structure with Algorithm 2, it has the same time complexity $O(K + n)$; see Remark 2 for details.

Algorithm 4 is rather slow. But it appears that the reversals of A-powers can be detected by a much faster Algorithm 5 below. We call u a *dual α-A-power* if u^R is an α-A-power; then dual α-A-free words are exactly the reversals of α-A-free words. As a language and its reversal have equal numbers of words of each length, we use Algorithm 5 in the studies of Abelian repetition threshold instead of Algorithm 4.

Assume that all proper prefixes of a word u are dual α-A-free, where $2 < \alpha < 3$. Then u is dual α-A-free iff the following analog of $(*)$ holds:

(†) no suffix of u can be written as xyz such that $y \sim z$, $\frac{|xyz|}{|z|} \ge \alpha$, and x is equivalent to a suffix of z.

Algorithm 5. Dual A-powers detection ($2 < \alpha < 3$)

```
1: function dualALPHAfree(u)                          ▷ u=word; n = |u|
2: free ← true                                        ▷ α-A-freeness flag
3: i ← n
4: while i ≥ 1 + ⌈((α−1)/α) n⌉ do                     ▷ z = u[i..n]
5:     len ← n − i + 1; P⃗ ← Parikh(i, n)              ▷ length and Parikh vector of z
6:     left ← cover(P⃗, i − 1)                         ▷ computing v
7:     if left + len = i then                         ▷ |v| = |z| ⇒ v = y ∼ z
8:         j = ⌈(α − 2) · len⌉                        ▷ minimal length of x
9:         while j ≤ len do                           ▷ possible lengths of x
10:            P⃗_1 ← Parikh(n−j+1, n)                ▷ Parikh vector of the length-j suffix of z
11:            left_1 ← cover(P⃗_1, left − 1)          ▷ computing v_1 for x
12:            if left_1 + j = left then              ▷ x is found
13:                free ← false; break
14:            else
15:                j ← left − left_1
16:        i ← i − 1
17:    else
18:        i ← ⌈(n + left)/2⌉
19:    if free = false then break
20: return free                                       ▷ the answer to "is u dual α-A-free?"
```

Proposition 5. *Let α be a number such that $2 < \alpha < 3$ and u be a word all proper prefixes of which are dual α-A-free. Then Algorithm 5 correctly detects whether u is dual α-A-free.*

Proof. If some suffix xyz violates (†), then $|z| \le |xyz|/\alpha \le n/\alpha$; hence the range for the outer cycle in line 3. The general scheme is as follows. For each processed suffix z, the algorithm first checks if u ends with an Abelian square yz ($y \sim z$); if yes, it checks whether yz is preceded by some x which is equivalent to a suffix of z. If such an x is found, the algorithm detects a violation of (†) and stops. If either x or y is not found, the algorithm moves to the next appropriate suffix.

Let us consider the details. In line 6, the shortest $v = u[left..i-1]$ such that vz is a suffix of u and $\Psi(v) \ge \Psi(z)$ is computed. If $|v| = |z|$ (the condition in line 7), then $y = v$ is found and we enter the inner cycle to find x. If $|v| > |x|$, the suffixes of u of lengths between $2|z|$ and $|vz| - 1$ cannot be A-squares; then the next suffix to be considered has the length $\lceil \frac{|vz|}{2} \rceil$, as is set in line 18. In the inner cycle, a similar idea is implemented: for each processed suffix z_1 of z the algorithm finds the shortest word $v_1 = u[left_1..left-1]$ satisfying $\Psi(v_1) \ge \Psi(z_1)$ (line 11). If $|v_1| = |z_1|$ (line 12), then x is found; consequently, the algorithm returns false. Otherwise, the next suffix of z to be checked is of length $|v_1|$ (line 15). The inner cycle breaks if this length exceeds the length of z. If the algorithm finishes the check of the suffix z without breaking or skips this suffix at all, then u has no suffix xyz, violating (†). Therefore, the algorithm verifies (†). □

Algorithm 5 is extremely fast compared to other algorithms of this section.

Proposition 6. *For a word u picked up uniformly at random from the set Σ^n, Algorithm 5 works in $\Theta(\sqrt{n})$ expected time.*

Proof. By Lemma 2, the expected length of the word v found in line 6 is $|z| + \Omega(\sqrt{|z|})$ and thus, on expectation, the assignment in line 18 leads to skipping $\Omega(\sqrt{|z|})$ suffixes of u. Hence the expected total number of processed suffixes of u is $O(\sqrt{n})$. By the same argument, the inner cycle for a suffix z runs, on expectation, $O(\sqrt{|z|})$ iterations, so its expected time complexity is $O(\sqrt{|z|})$. Thus, processing the suffix of length ℓ, Algorithm 5 performs $O(1) + p_\ell \cdot O(\sqrt{\ell})$ operations, where p_ℓ is the probability to enter the inner cycle, i.e., the probability that two random ternary words of length ℓ are Abelian equivalent. One has

$$p_\ell \le \max_{k_1,k_2,\ldots,k_\sigma} \binom{\ell}{k_1, k_2, \ldots, k_\sigma} \Big/ \sigma^\ell,$$

dividing the maximum number of A-equivalent words of length ℓ over Σ by the total number of such words. This maximum, is $\Theta(\sigma^\ell/\ell^{(\sigma-1)/2})$ by the Stirling formula. Thus $p_\ell = O(1/\ell^{(\sigma-1)/2})$. Then Algorithm 5 performs, on expectation, $O(1)$ operations per iteration of the outer cycle. The result now follows. □

4 Experimental Results

We ran a big series of experiments for α-A-free languages over the alphabets of size $2, 3, \ldots, 10$. Each of the experiments is a set of random walks in the prefix tree of a given language. Each walk follows the random depth-first search (Algorithm 1), with the number N of visited nodes being of order 10^5 to 10^7. The ultimate aim of every experiment was to make a well-grounded conjecture about the (in)finiteness of the studied language. Our initial expectation was that random walks will demonstrate two easily distinguishable types of behaviour:

- *infinite-like*: the level of the current node is (almost) proportional to the number of nodes visited, or
- *finite-like*: from some point, the level of the current node oscillates near the maximum reached earlier.

However, the situation is more tricky: very long oscillations of level were detected during random walks even in some languages which are *known* to be infinite; for example, in the binary 4-A-free language. To overcome such an unwanted behaviour, we endowed Algorithm 1 with a "forced backtrack" rule:

- let $\mathsf{ml} = k$ be the maximum level of a node reached so far; if $f(k)$ nodes were visited since the last update of ml or since the last forced backtrack, then make a forced backtrack: from the current node, move $g(k)$ edges up the tree and continue the search from the node reached.

Here $f(k)$ and $g(k)$ are some heuristically chosen monotone functions; we used $f(k) = \lceil k^{3/2} \rceil$ and $g(k) = \lceil k^{1/2} \rceil$. Forced backtracking deletes the last $g(k)$

letters of the current word in order to get out of a "trap": a very big finite subtree the search was supposed to traverse. The use of forced backtrack allowed us to classify the walks in almost all studied languages either as infinite-like or as finite-like. The results presented below are grouped by the alphabets.

4.1 Alphabets with 6, 7, 8, 9, and 10 Letters

In [21], it was proved (Theorem 3.1) that $\mathsf{ART}(k) \geq \frac{k-2}{k-3}$ for all $k \geq 5$ and conjectured that the equality holds in all cases. However, the random search reveals a different picture. For each of the languages $\mathsf{AF}(k, \frac{k-2}{k-3}^+)$, $k = 6, 7, 8, 9, 10$, we ran random search with forced backtrack, using Algorithm 3 to decide the membership in the language; the search terminated when N nodes were visited. We repeated the search 100 times with with $N = 10^6$ and another 100 times with $N = 2 \cdot 10^6$. The results, presented in columns 3–8 of Table 2, clearly demonstrate finite-like behaviour of random walks. Moreover, the results suggest that neither of these languages contains a word much longer than 100 symbols.

Theorem 1. *One has* $\mathsf{ART}(k) > \frac{k-2}{k-3}$ *for* $k = 6, 7, 8, 9, 10$.

A length-n word is called *n-permutation* if all its letters are pairwise distinct. Observing that a $\frac{k-2}{k-3}^+$-A-free word of length $k-1$ contains a $(k-2)$-permutation, one can easily prove Lemma 3 below. Reducing the search space with the help of Lemma 3, we were able to prove Theorem 1 by exhaustive search.

Lemma 3. *Let* $k \geq 6$, $\alpha = \frac{k-2}{k-3}^+$, *and let* L_1, L_2, *and* L_3 *be subsets of* $L = \mathsf{AF}(k, \alpha)$ *defined as follows:*

- $L_1 = \{w \in L \mid w$ *has the prefix* $01 \cdots (k-3)$ *and no* $(k-1)$*-permutations*$\}$;
- $L_2 = \{w \in L \mid w$ *has the prefix* $01 \cdots (k-2)$ *and no* k*-permutations*$\}$;
- $L_3 = \{w \in L \mid w$ *has the prefix* $01 \cdots (k-1)\}$.

Then L *is finite iff each of* L_1, L_2, *and* L_3 *is finite.*

Table 2. Maximum levels ml reached by random walks in some Abelian power-free languages. Columns 3–5 (resp. 6–8) show the maximum, average, and median values of ml among 100 random walks visiting $N = 10^6$ (resp., $N = 2 \cdot 10^6$) nodes each. Column 9 shows the length of a longest word in the language, found by exhaustive search.

Alphabet size	Avoided power	$N = 10^6$			$N = 2 \cdot 10^6$			Maximum length
		$\mathsf{ml}_{\max}$	ml_{av}	ml_{med}	$\mathsf{ml}_{\max}$	ml_{av}	ml_{med}	
6	$(4/3)^+$	112	98.9	98	114	101.1	101	116
7	$(5/4)^+$	116	100.3	100	124	103.9	102	125
8	$(6/5)^+$	103	94.8	95	102	96.2	96	105
9	$(7/6)^+$	108	95.6	96	107	98.8	99	117
10	$(8/7)^+$	121	107.7	108	128	111.6	111	148*

Remark 6. The number of nodes visited during the optimized exhaustive search proving Theorem 1 ranged from 0.43 billions for $k = 8$ to 615 billions for $k = 10$; the search required over 2500 h of single-core processing time by an ordinary laptop (Algorithm 3 was used to detect A-powers). For each $k = 6, 7, 8, 9$ we have also run a single search enumerating all lexmin words in the language $\mathsf{AF}(k, \frac{k-2}{k-3}^+)$. Thus we found the maximum length of a word in each language (the last column of Table 2) and the distribution of words by their length. For $k = 10$, such a single search would require too much resources; here the value in Table 2 is the length of the longest word found by the search based on Lemma 3.

As the next step after Theorem 1, we ran experiments for the languages $\mathsf{AF}(k, \frac{k-3}{k-4})$. The results for $k = 7, 8, 9, 10$ are presented in Table 3; random walks in these languages clearly demonstrate finite-type behaviour, while proving finiteness by exhaustive search looks impossible. On the contrary, the walks in the 6-ary language $\mathsf{AF}(6, \frac{3}{2})$ demonstrate an infinite-like behaviour: the average value of ml for our experiments with $N = 10^5$ is greater than $5 \cdot 10^4$. We note that the obtained words are too long for Algorithm 3, so we had to use slower Algorithm 2. Finally, we constructed random walks for the languages $\mathsf{AF}(k, \frac{k-3}{k-4}^+)$ $(k = 7, 8, 9, 10)$. They also demonstrate infinite-like behaviour. The obtained experimental results allow us to state the part of Conjecture 2 for the alphabets with 6 and more letters.

Table 3. Maximum levels ml reached by random walks in some Abelian power-free languages. Columns 3–5 (resp. 6–8) show the maximum, average, and median values of ml among 100 random walks visiting $N = 10^6$ (resp., $N = 2 \cdot 10^6$) nodes each.

Alphabet size	Avoided power	$N = 10^6$			$N = 2 \cdot 10^6$		
		$\mathsf{ml}_{\max}$	$\mathsf{ml}_{\mathrm{av}}$	$\mathsf{ml}_{\mathrm{med}}$	$\mathsf{ml}_{\max}$	$\mathsf{ml}_{\mathrm{av}}$	$\mathsf{ml}_{\mathrm{med}}$
7	4/3	510	374.5	371	510	397.5	394
8	5/4	211	179.7	179	223	185.0	184
9	6/5	192	157.2	156	191	162.3	161
10	7/6	175	154.0	154	187	159.7	158

4.2 Alphabets with 2, 3, 4, and 5 Letters

Random walks in the prefix tree of the language $\mathsf{AF}(5, \frac{3}{2}^+)$ demonstrate the infinite-like behaviour; Fig. 3 shows an example of dependence of the level of the current node on the number of nodes visited. The obtained results give us sufficient evidence to support Conjecture 1 for $k = 5$.

Remark 7. As the language $\mathsf{AF}(5, \frac{3}{2}^+)$ is probably infinite, it is interesting to estimate its growth. Based on the technique described in [17], we estimate the number of words of length n in $\mathsf{AF}(5, \frac{3}{2}^+)$ as growing exponentially with n at the rate close to 1.5. The upper bound 2.335 [21] on this rate is thus very loose.

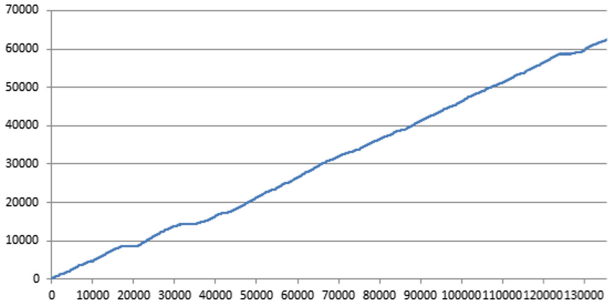

Fig. 3. An infinite-like random walk in the language $\mathsf{AF}(5, \frac{3}{2}^+)$: a point (n, m) of the graph means that the nth node visited by the walk has depth m.

Further, we studied the languages $\mathsf{AF}(4, \frac{9}{5}^+)$, $\mathsf{AF}(3, 2^+)$, and $\mathsf{AF}(2, \frac{11}{3}^+)$, indicated by Conjecture 1 as infinite; we replaced the last two languages with their reversals to benefit from the fast detection of A-powers by Algorithm 5 and its version for $\alpha > 3$. Random walks in each of three languages show the finite-like behaviour; see Table 4 and the example in Fig. 4 (multiple forced backtracks provide slightly better average results for longer searches). So the experimental results justify lower bounds from Conjecture 2 for $k = 2, 3, 4$. To get the upper bound for the ternary alphabet, we ran random walks for the language $\mathsf{AF}(3, \frac{5}{2}^+)$ with the results similar to those obtained for $\mathsf{AF}(5, \frac{3}{2}^+)$: all walks demonstrate the infinite-like behaviour; the level $\mathsf{ml} = 10^5$ is reached within minutes.

Overall, the conducted experiments justify the formulation of Conjecture 2.

Table 4. Maximum levels ml reached by random walks in some A-power-free languages. Columns 3–5 (resp. 6–8, 9–11) show the maximum, average, and median values of ml among 100 random walks visiting $N = 10^6$ (resp., $N = 2 \cdot 10^6$, $N = 10^7$) nodes each.

Alphabet size	Avoided power	$N = 10^6$			$N = 2 \cdot 10^6$			$N = 10^7$		
		$\mathsf{ml}_{\max}$	ml_{av}	ml_{med}	$\mathsf{ml}_{\max}$	ml_{av}	ml_{med}	$\mathsf{ml}_{\max}$	ml_{av}	ml_{med}
2	$(11/3)^+$	775	435.8	416	706	477.0	453	759	589.7	588
3	2^+	3344	1700.0	1671	5363	2228.8	2140	5449	3078.1	3148
4	$(9/5)^+$	1367	861.2	835	1734	986.8	956	2453	1414.7	1369

5 Future Work

Clearly, the main challenge in the topic is to find the exact values of the Abelian repetition threshold. Even finding one such value would be a great progress. Choosing the case to start with, we would suggest proving $\mathsf{ART}(5) = 3/2$ because in this case the lower bound is already checked by exhaustive search in [21]. For all other alphabets, the proof of lower bounds suggested in Conjecture 2 is already a challenging task which cannot be solved by brute force.

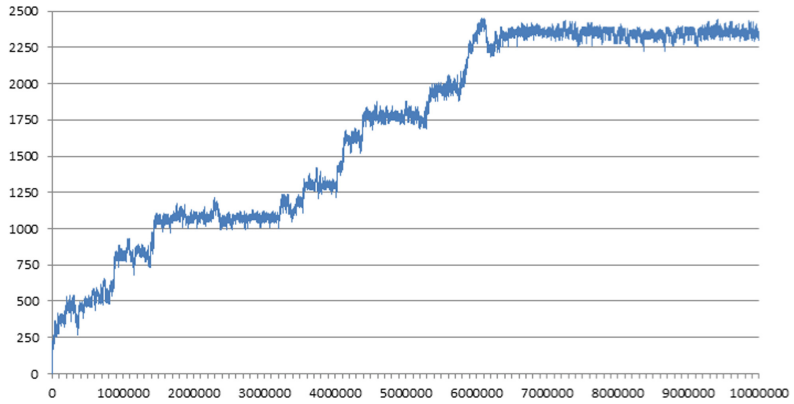

Fig. 4. A finite-like random walk in the language $\mathsf{AF}(4, \frac{9}{5}^+)$: a point (n, m) of the graph means that the nth node visited by the walk has depth m.

Another piece of work is to refine Conjecture 2 by suggesting the precise values of $\mathsf{ART}(2)$, $\mathsf{ART}(3)$, $\mathsf{ART}(4)$, and $\mathsf{ART}(6)$. For bigger k, random walks demonstrate an obvious "phase transition" at the point $\frac{k-3}{k-4}$: the behaviour of a walk switches from finite-like for $\mathsf{AF}(k, \frac{k-3}{k-4})$ to infinite-like for $\mathsf{AF}(k, \frac{k-3}{k-4}^+)$. However, the situation with small alphabets can be trickier. We tried $11/6$ as the next natural candidate for $\mathsf{ART}(4)$. For the random walks in $\mathsf{AF}(4, \frac{11}{6}^+)$, with $N = 10^6$ and forced backtracks, the range of obtained maximum levels in our experiments varied from 3000 to 20000; such big lengths show that there is no hope to see a clear-cut phase transition in the experiments with random walks.

Third, we want to draw attention to the following fact. The quaternary 2-A-free word constructed by Keränen [14] contains arbitrarily long factors of the form xax', where a is a letter and $x \sim x'$; thus it is not α-A-free for any $\alpha < 2$. Similarly, the word constructed by Dekking [9] for the ternary (resp., binary) alphabet is not α-A-free for any $\alpha < 3$ (resp., $\alpha < 4$). Hence some new constructions are necessary to improve upper bounds for ART.

The algorithmic part of this work also rises some questions. As established by Radoszewski et al. [18], the commonly believed 3-SUM conjecture implies that it is impossible to decide in $O(n^{2-\varepsilon})$ time whether a word contains an A-square (and thus decide in $O(n^{1-\varepsilon})$ time whether a word contains an A-square as a suffix). So, what are the lower bounds for detecting fractional A-powers?

References

1. Carpi, A.: On Dejean's conjecture over large alphabets. Theoret. Comput. Sci. **385**, 137–151 (1999)
2. Cassaigne, J., Currie, J.D.: Words strongly avoiding fractional powers. Eur. J. Comb. **20**(8), 725–737 (1999)
3. Currie, J.D., Rampersad, N.: A proof of Dejean's conjecture. Math. Comp. **80**, 1063–1070 (2011)

4. Currie, J.D., Mol, L.: The undirected repetition threshold and undirected pattern avoidance. Theor. Comput. Sci. **866**, 56–69 (2021)
5. Currie, J.D., Mol, L., Rampersad, N.: Circular repetition thresholds on some small alphabets: last cases of Gorbunova's conjecture. Electron. J. Comb. **26**(2), P2.31 (2019)
6. Currie, J.D., Mol, L., Rampersad, N.: The number of threshold words on n letters grows exponentially for every $n \geq 27$. J. Integer Seq. **23**(3), 20.3.1 (2020)
7. Currie, J.D., Mol, L., Rampersad, N.: The repetition threshold for binary rich words. Discret. Math. Theor. Comput. Sci. **22**(1) (2020)
8. Dejean, F.: Sur un théorème de Thue. J. Combin. Theory. Ser. A **13**, 90–99 (1972)
9. Dekking, F.M.: Strongly non-repetitive sequences and progression-free sets. J. Combin. Theory. Ser. A **27**, 181–185 (1979)
10. Dolce, F., Dvořáková, L., Pelantová, E.: Computation of critical exponent in balanced sequences. In: Lecroq, T., Puzynina, S. (eds.) WORDS 2021. LNCS, vol. 12847, pp. 78–90. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85088-3_7
11. Erdős, P.: Some unsolved problems. Magyar Tud. Akad. Mat. Kutató Int. Közl. **6**, 221–264 (1961)
12. Evdokimov, A.A.: Strongly asymmetric sequences generated by a finite number of symbols. Soviet Math. Dokl. **9**, 536–539 (1968)
13. Gorbunova, I.A.: Repetition threshold for circular words. Electron. J. Comb. **19**(4), P11 (2012)
14. Keränen, V.: Abelian squares are avoidable on 4 letters. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 41–52. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55719-9_62
15. Moulin-Ollagnier, J.: Proof of Dejean's conjecture for alphabets with $5, 6, 7, 8, 9, 10$ and 11 letters. Theoret. Comput. Sci. **95**, 187–205 (1992)
16. Pansiot, J.J.: A propos d'une conjecture de F. Dejean sur les répétitions dans les mots. Discr. Appl. Math. **7**, 297–311 (1984)
17. Petrova, E.A., Shur, A.M.: Branching frequency and Markov entropy of repetition-free languages. In: Moreira, N., Reis, R. (eds.) DLT 2021. LNCS, vol. 12811, pp. 328–341. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81508-0_27
18. Radoszewski, J., Rytter, W., Straszynski, J., Walen, T., Zuba, W.: Hardness of detecting abelian and additive square factors in strings. CoRR abs/2107.09206 (2021)
19. Rampersad, N., Shallit, J.O., Vandomme, É.: Critical exponents of infinite balanced words. Theoret. Comput. Sci. **777**, 454–463 (2019)
20. Rao, M.: Last cases of Dejean's conjecture. Theoret. Comput. Sci. **412**, 3010–3018 (2011)
21. Samsonov, A.V., Shur, A.M.: On Abelian repetition threshold. RAIRO Theor. Inf. Appl. **46**, 147–163 (2012)
22. Shallit, J.O., Shur, A.M.: Subword complexity and power avoidance. Theoret. Comput. Sci. **792**, 96–116 (2019)
23. Thue, A.: Über unendliche Zeichenreihen. Norske vid. Selsk. Skr. Mat. Nat. Kl. **7**, 1–22 (1906)
24. Thue, A.: Über die gegenseitige Lage gleicher Teile gewisser Zeichenreihen. Norske vid. Selsk. Skr. Mat. Nat. Kl. **1**, 1–67 (1912)
25. Tunev, I.N., Shur, A.M.: On two stronger versions of Dejean's conjecture. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 800–812. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32589-2_69