Theoretical Computer Science 601 (2015) 67–72

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Generating square-free words efficiently

Arseny M. Shur [1]

Ural Federal University, Ekaterinburg, Russia

ARTICLE INFO

Article history:
Received 15 January 2014
Accepted 21 October 2014
Available online 14 July 2015

Keywords:
Square-free word
Random word
String algorithms

ABSTRACT

We study a simple algorithm generating square-free words from a random source. The source produces uniformly distributed random letters from a k-ary alphabet, and the algorithm outputs a $(k+1)$-ary square-free word. We are interested in the "conversion ratio" between the lengths of the input random word and the output square-free word. For any $k \geq 3$ we prove the expected value of this ratio to be a constant and calculate it up to an $O(1/k^5)$ term. For the extremal case of ternary square-free words, we suggest this ratio to have a constant expectation as well and conjecture its actual value from computer experiments.

1. Introduction

Square-free words are one of quite popular objects of study in combinatorics. Their first appearance dates back to the Thue paper [8], who proved the infiniteness of the set of ternary square-free words. Most of the time, square-free words are considered in a broader context of repetition-free words; nevertheless, they usually serve as a "firing ground" for new concepts and types of problems.

The idea of repetition-free words generated from some random source is rather new. Grytczuk, Kozik, and Witkowski used this idea to prove an avoidability result about some sort of "strong" square-freeness [2], while Camungol and Rampersad applied the same technique to get a similar result about "approximate" square-freeness [1]. The method used in these papers is basically as follows: one builds a word avoiding the desired repetitions letter by letter, choosing the letters uniformly at random from the given alphabet. At the moment when the obtained word w encounters a forbidden repetition, one dismisses some suffix of w and continues the random process. If for each n it can be proved that the word under construction reaches the length n with a nonzero probability, then the considered repetition is avoidable over the alphabet. As the authors mention explicitly, this method is inspired by the constructive proof of the Lovász local lemma, given by Moser and Tardos [3]. We should note that the techniques based on this proof already lead to several nice results in different branches of combinatorics. Speaking about combinatorics on words, we point out the solution given by Ochem and Pinlou [4] to the well-known open problem on pattern avoidability.

In this paper, we study the efficiency of this method applied to "usual" square-free words. That is, given an alphabet, we want to find the expected number of rounds of the random process needed to build a square-free word of length n. We know that the number of $(k+1)$-ary square-free words grows exponentially, and the growth rate is given by the asymptotic formula $k - 1/k - 1/k^3 + O(1/k^5)$ [6]. This formula approximates the actual growth rate very well for $k \geq 3$ and is quite reasonable even for $k = 2$ (for more information on the growth of repetition-free languages see [7]). From this formula,

E-mail address: Arseny.Shur@urfu.ru.

[1] Supported by Ministry of Education and Science of the Russian Federation under the Agreement 02.A03.21.0006 of 27.08.2013 with Ural Federal University, and by the grant 13-01-00852 of Russian Foundation for Basic Research.

http://dx.doi.org/10.1016/j.tcs.2015.07.027
0304-3975/

it seems quite probable that the expected number of rounds of the random process is only slightly over n. We introduce Algorithm R2F (Random-t(w)o-free) which implements a modification of the general method: we use random letters from a k-letter alphabet to generate a $(k+1)$-ary square-free word. Our main result is the following

Theorem 1. *The expected number of random k-ary letters used by Algorithm R2F to construct a $(k+1)$-ary square-free word of length n is*

$$N = n(1 + 2/k^2 + 1/k^3 + 4/k^4 + O(1/k^5)) + O(1). \tag{1}$$

The text is organized as follows. After short preliminaries, we introduce Algorithm R2F in Section 2. Then in Section 3 the performance of this algorithm is analyzed and Theorem 1 is proved. Section 4 contains results of computer experiments and a short discussion.

2. Preliminaries and the algorithm

We study words over the finite alphabets $\Sigma_k = \{1, \dots, k\}$, $k \geq 2$. For words, we use the array notation: $w = w[1..n]$, $w[i]$ is the ith letter of w, $w[i..j]$ is the factor of w occupying the indicated range of positions. We write $|w|$ for the length of w and λ for the empty word. A word of the form ww is a *square*; it is an *r-square* if $|w| = r$. A word is *square-free* if it contains no squares as factors.

As usual, we use the notation X^* [respectively, X^+] for the [positive] iteration of the word or language X. Studying the structure of a word, we write "$w = xy^*z$" rather than "$w \in xy^*z$".

The *growth rate* of a language $L \in \Sigma^*$ is given by the limit $\limsup_{n\to\infty} |L \cap \Sigma^n|^{1/n}$. For the languages closed under factors, lim sup can be replaced by lim.

In his WORDS'2013 lecture, Rampersad described the following algorithm to construct k-ary square-free words (the same algorithm was used in [1] to build words avoiding approximate squares). Starting with an empty word, one appends to its end one letter per round; the letter is given by a uniform random source. If the current word ends with an r-square, then one dismisses the right half of this square. The algorithm works until the constructed word reaches the required length n. In this paper, we use a modification of this algorithm; this modification generates square-free words a bit faster and is easier to analyze.

For a word $w \in \Sigma_k$, its *hash* χ_w is the permutation of Σ_k defined by "recency" of letters. Namely, a precedes b in χ_w if the rightmost position of a in w is to the right of the rightmost position of b in w. The letters that do not occur in w stay in the end of the hash in increasing order:

$$w = 136263163 \in \Sigma_6 \text{ has the hash } \chi_w = 361245.$$

Now let w have no factors aa for $a \in \Sigma_k$. Then $w[i+1] \neq w[i] = \chi_{w[1..i]}[1]$ for any $i \geq 1$. Hence w can be encoded by its first letter and a word $u \in \Sigma_{k-1}^{|w|-1}$ by the rule $u[i] = j$ if and only if $w[i+1] = \chi_{w[1..i]}[j+1]$:

$$w = 136263163 \in \Sigma_6 \text{ is encoded by } w[1] = 1 \text{ and } u = 25312322 \in \Sigma_5.$$

We use this encoding to generate a square-free word by the following

Algorithm R2F. *Input*: integers $k, n > 1$.
Output: a $(k+1)$-ary square-free word w of length n.

1. Initialization:
   - choose $w[1] \in \Sigma_{k+1}$ uniformly at random;
   - set χ_w to $w[1]$ followed by all other letters of Σ_{k+1} in increasing order;
   - set the number N of iterations to 0.
2. Append:
   - choose $j \in \Sigma_k$ uniformly at random;
   - append $a = \chi_w[j+1]$ to the end of w;
   - update χ_w shifting the first j elements to the right and setting $\chi_w[1] = a$;
   - increment N by 1.
3. Cut:
   - if w ends with an r-square, delete the last r letters of w.
4. Check for termination:
   - if $|w| < n$ then goto step 2 else return w.

Remark 1.

(1) On termination, Algorithm R2F returns a square-free word (if w ends with a square, we proceed to a shorter word which is square-free).