Simulations of the square-free words random process

Naomi D. Feldheim*

December 18, 2014

1 Overview

The "ABC process" is easy to describe: given a finite alphabet, each step we choose randomly (uniformly) a letter to add to our existing word. If it causes a repetition of any length, we erase this repetition (e.g.: 'aba'+'b' $\to$ 'ab'). Despite its simplicity, this process does not fall into any well-studied catagory, and therefore we lack tools to analyze it. A basic property of interest is the distribution of the length after N steps, and in particular the growth to infinity of the mean length for small alphabet size (3 letters). Another basic question, which is intimately related, is to bound the probability of erasing k letters in one step.

With these questions in mind, this report presents some simulation results. Improving on an algorithm suggested by Ohad Feldheim, I was able to simulate the process efficiently for a large input. Currently, my laptop is able to run a sample of 10^7 steps in 70 minutes for 3 letters, and a few hours for 5 letters (and, in fact, probably 10^9 steps can be generated applying Ohad's ideas, see end of Section 5). For presenting these preliminary results, I did not always run the simulations to the furthest possible extent.

Highlights. Here are some highlights of the simulation results. **Notice:** In this report, one step of time is one random input to the process. Thus, erasing a repetition of k letters is just one tick of the clock. The results (and the program) may be easily modified if one wishes to count this as k steps of time (as in Joel's talk and in Humberto's report).

- The length after N input steps has sampled mean $0.682N$ for 5 letters, and $0.056N$ for 3 letters. Counting one tick for erasing a single letter, the mean length after N ticks is $0.047N$ for 3 letters. This seems larger then Humberto's prediction of roughly $0.02N$.
- Roughly, the distribution of length after N input steps looks normal.
- With 5 letters, the length climbs almost linealry, and there are hardly any long erasings (10 letters at a time at most, running up to 10^7 inputs).
- With 3 letters, there are frequent long erasings (up to almost 50 letters at a time, running 10^7 input steps).
- With 3 letters, it is impossible to erase certain lengths in one tick (i.e., by adding one letter). The forbidden length increments (in one tick) are -4, -6, -8, -9, -13 and -16. In particular, the probability to erase a certain length in one tick is not monotone in the length (at least, not for "small" lengths).

*trinomi@gmail.com

Further investigation. The simulations confirm that the length grows for three letters - which will be of course very interesting to prove. Obviously, the process terminates with two letters. Another version of the process may help in finding a (fractional) thershold for growth is the following: run the same algorithm, with a non-uniform measure on the three letters. That is, pick (independently at every step) one of the letters a, b, c with probabilities $1-2p, p, p$ respectively. Similarly one may consider non-uniform measures on more letters, perhaps in order to approach three letters as a limit.

I would be interested to hear of any other basic questions of interest which may be simulated, or if you would like more data on the current simulations. I am ready to share my software or run simple modifications of it. I am also happy to think about the theoritical proofs to the phenomena observed in this report.

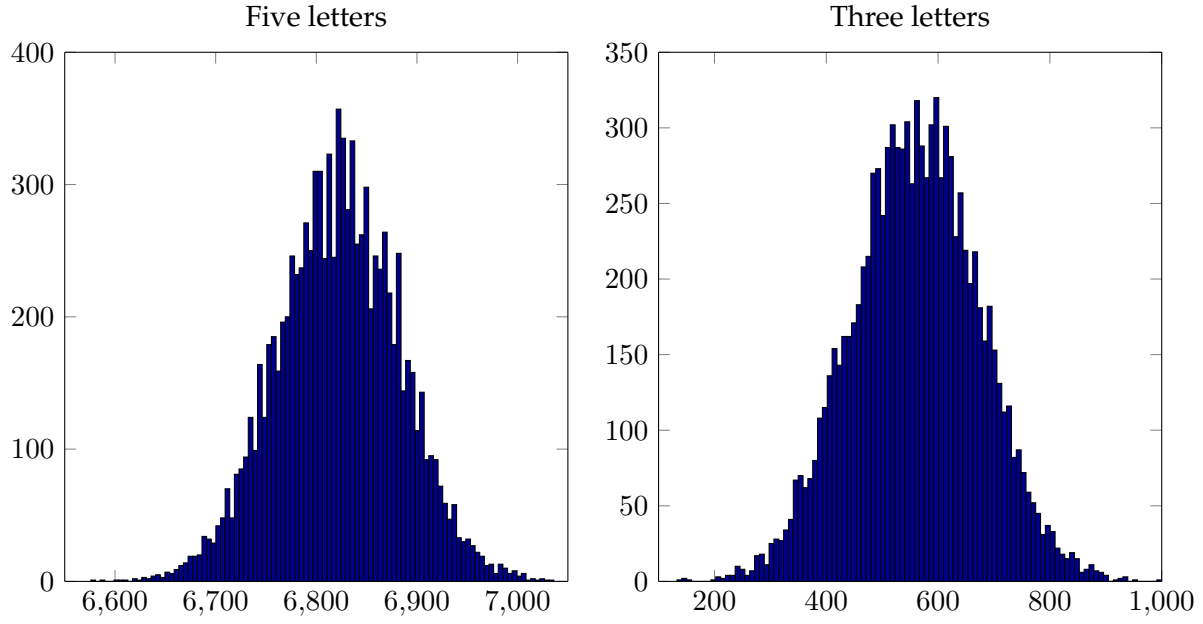
Organization of the report. Section 2 contains histograms of the lengths, Section 3 contains plots of the evolution of length with time, and Section 4 contains emperical probabilities to erase long sequences. All simulations were run twice: for 3 and for 5 letters. In Section 5, we describe the algorithm, and some directions for further efficiency. Lastly Appendix A contains a sample square-free string on three letters, produced by the algorithm.

2 Histograms of lengths

Denote by L_N^q the length of the output after N random input steps with q letters. It is clear that $\mathbb{E}L_N^q = c_q N$, but the constant c_q is unknown. For three letters it is still open to prove that $c_3 > 0$ (as far as I know).

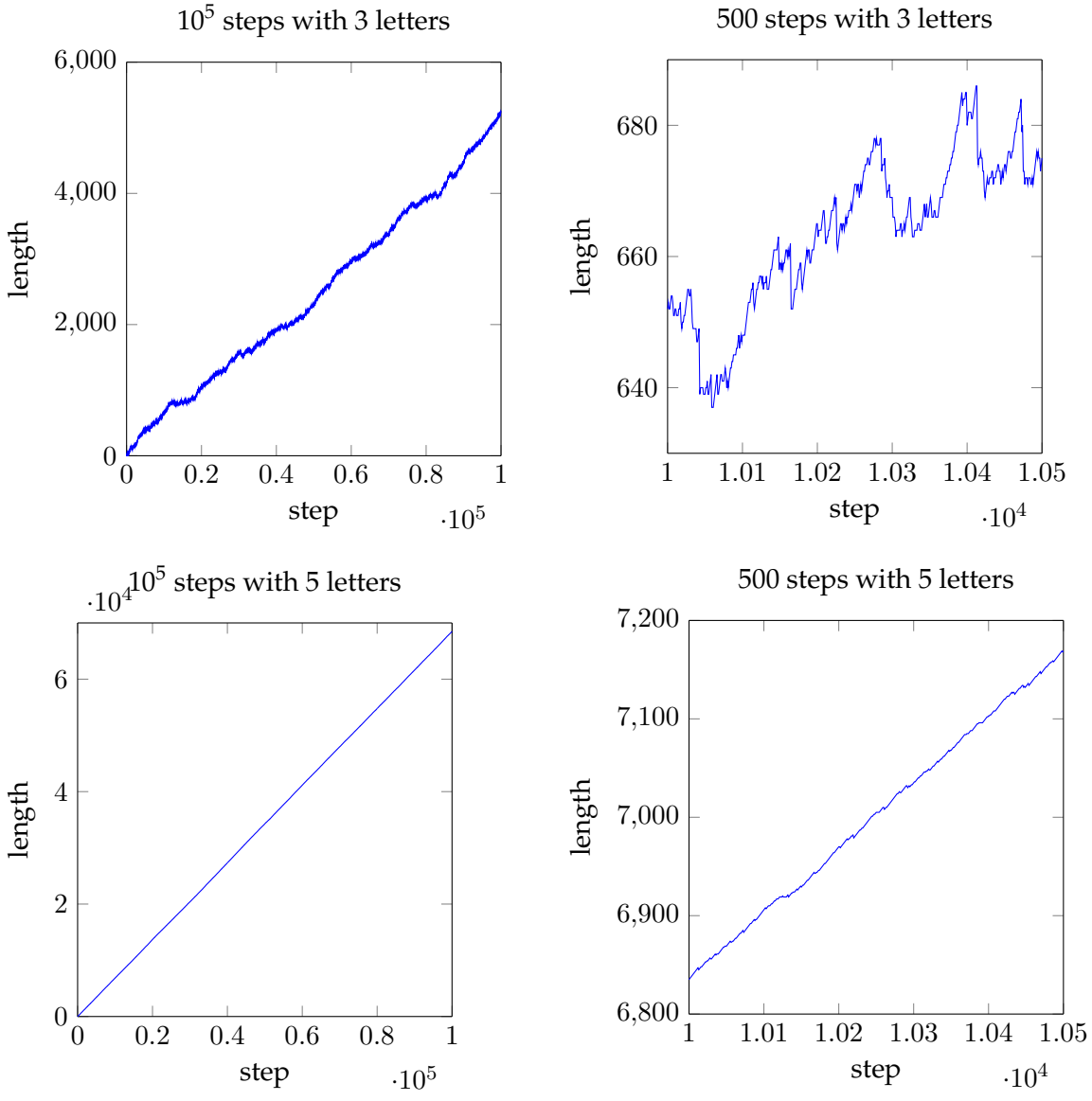
In order to estimate c_q, I ran 10^4 runs of 10^4 steps each. Below are histograms of the final output lengths. The empirical mean suggests that $c_5 \equiv 0.682$, and $c_3 \equiv 0.056$. (Notice this is not a contradiction to Humberto's predictions of less than 0.02, as he counts time differently - erasing a letter is a tick of his clock.)

The empirical distribution of length after $N = 10^4$ steps



3 The evolution of length

Below are plots of the evolution of the length in a single trajectory of the process. I ran a sample of the process with 3 and with 5 letters. Looking from far (10^5 steps), the growth looks steady and approximately linear. Looking up closer (500 steps), the trajectory for 3 letters looks quite rough, but with 5 letters it is still stable and close to linear.



4 Empirical distribution of the increment of length

A more local property of interest, which directly effects the phenomena seen above, is the (asymptotic) probability to erase a long sequence of letters at once. The following tables present the emprical distribution of the increment of length in one step, taken from a single trajectory of 10^7 steps. For instance, the value '-1', which corresponds to erasing 2 letters, has emperical probability 0.0399 for five letters. The value '0' corresponds to the probability of picking the last letter again, which is indeed very close to $\frac{1}{q}$ where q is the number of letters.

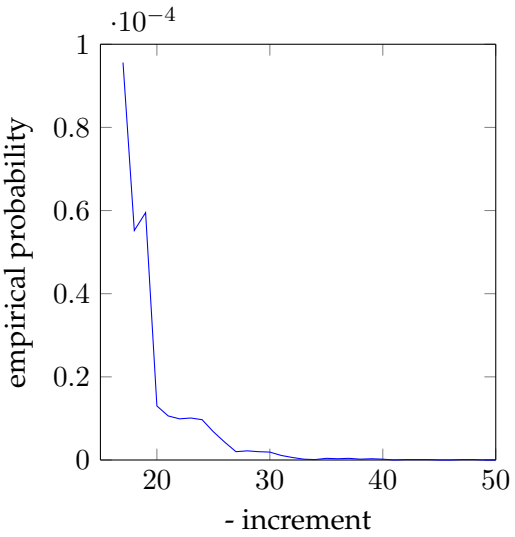
5 letters, 10^7 steps

$1:$	0.7487
$0:$	0.2001
$-1:$	0.0399
$-2:$	0.0085
$-3:$	0.0022
$-4:$	4.8810×10^{-4}
$-5:$	1.419×10^{-4}
$-6:$	3.47×10^{-5}
$-7:$	9.6×10^{-6}
$-8:$	3.2×10^{-6}
$-9:$	7×10^{-7}
$-10:$	1×10^{-7}

3 letters, 10^7 steps

$1:$	0.4602
$0:$	0.3333
$-1:$	0.1111
$-2:$	0.0519
$-3:$	0.0268
$-4:$	0
$-5:$	0.0097
$-6:$	0
$-7:$	0.0042
$-8:$	0
$-9:$	0
$-10:$	10.07×10^{-4}
$-11:$	7.76×10^{-4}
$-12:$	6.88×10^{-4}
$-13:$	0
$-14:$	2.8×10^{-5}
$-15:$	1.34×10^{-5}
$-16:$	0
$-17:$	9.98×10^{-5}
$-18:$	5.1×10^{-5}
$-19:$	5.66×10^{-5}
$-20:$	1.52×10^{-5}
$-21:$	1.08×10^{-5}
$-22:$	1.08×10^{-5}
$-23:$	1.06×10^{-5}
$-24:$	0.9×10^{-5}
$-25:$	0.76×10^{-5}
$-26:$	0.42×10^{-5}
$-27:$	0.16×10^{-5}
$-28:$	0.18×10^{-5}
$-29:$	0.14×10^{-5}
$-30:$	0.26×10^{-5}
$-31:$	11×10^{-7}
$-32:$	6×10^{-7}
$-33:$	2×10^{-7}
$-34:$	1×10^{-7}
$-35:$	4×10^{-7}
$-36:$	3×10^{-7}
$-37:$	4×10^{-7}
$-38:$	2×10^{-7}
$-39:$	3×10^{-7}
$-40:$	1×10^{-7}
$-41 \to -46:$	0
$-47:$	1×10^{-7}
$-48:$	1×10^{-7}
$\leq -49:$	0

It is interesting to note that with five letters, the longest sequence that was erased in one step is of 10 letters, and this happened only once in our run of 10^7. With three letters much longer sequences were erased (almost up to 50), but it seems some length-increments are impossible (-4, -6, -8, -9, -13 and -16). Probably, these are the only impossible increments (similar phenomena was noticed by Humberto). After that, the distribution of increments decays fast:



5 How the program works

The algorithm is as follows. We want to run N random input steps with alphabet size s. We choose a parameter L (recommended to be a bit less than $log(N)/log(s)$) to be our *hash size*. Following the evolution of the string, we will update a hash-table that will refer us from words of length L to a list of their appearances. Here is an example:

Hash-table for 'abcbacb'

abc:	0, 4
bcb:	1
cba:	2
bab:	3

Each time we add a letter, we check manually if it caused a repetition of length $\leq L$ (i.e., in a loop). If it did, we erase and continue. If it didn't, we check the last L letters of the new string in our hashtable, to get the list of places where it appeared. For each apperance (starting from the latest), we check if there is a long repetition that matches our last L letters to this previous appearance. If there is, we erase it and continue. We remark that because of these long erasings the hash may not be updated, so everytime we read a list from the hash we check it is updated and ordered. If no erasing applied, we add a letter and an entery to the approperiate list in the hash and continue. If L is chosen properly, the hash will not be full and lists will not be too long, so checking long repetition will be balanced with checking the short ones.

Computational Remark. The following improvement in running time is due to Ohad. Since long erasings are extremly rare (with 5 or more letters at least), one may be even more efficient and not save all the list of apperances, just the very last few. In the end we will check if the final string is square free. Most likely it will be, and we keep the output. If it is not, we run the simulation from the start (but this rarely, or practically never, happens). For more improvements in time and memory usage, please ask Ohad.

A Appendix: Sampled sequences

A sample with 3 letters:

```
 cbabcabacbcacbacabacbabcacbacabacbcabcbabcabacabcacbacabacbabcacbcabcb
acabcacbacabacbcacbacabcacbabcbacbcabacabcbabcabacbabcbacbcabcbabcacba
cabacbabcabacbcacbabcbacbcabcbabcabacbcabcbacbcabacbabcbacabcbabcacbac
abcbacbcacbabcabacbcacbacabcbabcacbacabacbabcacbcabcbacbcacbacabcbabca
bacbcabcbabcacbcabacbcacbacabcacbcabacabcbabcacbcabacbcacbabcabacbcabc
babcabacabcacbabcbacbcabcbacabcacbabcabacbabcbacabcacbcabacbcacbacabcb
acbcabacbabcabacbcacbacabcacbabcabacbcacbabcacbcabcbacbcabacabcacbcabcb
acabacbabcacbacabcbabcabacbabcbacbcacbacabacbcabcbabcacbacabacbcacbabcb
acabcacbcabacbabcacbcabcbabcacbacabacbabcbacbcacbacabacbabcacbcabacabcb...
```

A sample with 5 letters:

```
 bceaedceabcbacbecdcbdacbdeceaedbcdebdcedbaeabdebdcebdaedeacbdbaecdcb
acaebcabebacbcdaedbdcbadabcaecdabcabeabdebedadcbacdebaebdecadcedabdb
edecdeacdbadbcecadbcbadecebdcbecbceacadebdbeadcadbcebedcaebcadebcecbe
dacbadcaebdadcadbcbebcebadbecedeaebdecdcadbcacedacdadcecadedcbdcadabc
beacadeadbabcaeabedcbdbedecabacabcdecbacedacdbcbaeacbabcdaedabceadaca ...
```