Forced repetitions over alphabet lists

Neerja Mhaskar and Michael Soltys

PSC 2016

August 31, 2016
Thank you Franya Franek for presenting for us at PSC2015
Logical foundations of String Algorithms — a formalization
Thue’s 1906 problem: construct word of arbitrary length over a ternary alphabet without repetitions.

Grytczuk in 2010 asked if the same holds true over an alphabet list. Showed the answer is “yes” over alphabet lists where each alphabet has at least 4 symbols.

Problem still open for alphabet lists with 3 symbols per alphabet. We show lots of partial results.
Thue’s problem and solution

$\Sigma_3 = \{1, 2, 3\}$

morphism:

$$S = \begin{cases} 
1 \mapsto 12312 \\
2 \mapsto 131232 \\
3 \mapsto 1323132 
\end{cases}$$

(1)

Given a string $w \in \Sigma_3^*$, we let $S(w)$ denote $w$ with every symbol replaced by its corresponding substitution:

$$S(w) = S(w_1 w_2 \ldots w_n) = S(w_1)S(w_2)\ldots S(w_n).$$
Grytczuk’s problem

Let:

\[ L = L_1, L_2, \ldots, L_n, \]

be an ordered list of (finite) alphabets.

We say that \( w \) is a string over the list \( L \) if \( w = w_1 w_2 \ldots w_n \) where for all \( i, w_i \in L_i \).

We impose no conditions on the \( L_i \)’s: they may be equal, disjoint, or have elements in common.

The only condition on \( w \) is that the \( i \)-th symbol of \( w \) must be selected from the \( i \)-th alphabet, i.e., \( w_i \in L_i \).
If $|L_i| \geq 4$, then no matter what the list is, there is always a square-free string over such a list.

This can be shown with a *probabilistic algorithm*:

*in its $i$-th iteration, it selects randomly a symbol from $L_i$, and continues if the string created thus far is square-free, and otherwise deletes the suffix consisting of the right side of the square it just created, and re-starts from the appropriate position.*
Grytczuk proves by a simple counting argument that with positive probability the length of a constructed sequence exceeds any finite bound, provided the number of symbols is at least 4.

⇒ This implies the existence of arbitrarily long square-free strings.

Note that this argument is non-constructive (or rather, it yields an exponential time procedure).

Thus it is a weaker result than the method employed in Thue’s original problem.

This approach relies on Moser's algorithmic proof of Lovász Local Lemma: the entropy compression argument.
Open Problem 1:

Is there a polytime procedure for computing a square-free string from a given list of length $n$?
List of 3 symbol alphabets

$L = L_1, L_2, \ldots, L_n$, where $|L_i| = 3$.

We can prove it for restricted types of such lists:

- $L$ has an SDR (System of Distinct Representatives)
- $L$ has the union property
- $L$ has a consistent mapping (testing for one is NP-hard)
- $L$ is a partition
Let $C(n)$, an \textit{offending suffix}, be a pattern defined recursively:

\[ C(1) = X_1 a_1 X_1, \text{ and for } n > 1, \]
\[ C(n) = X_n C(n - 1) a_n X_n C(n - 1). \]

Zimin words ... !
We are interested in:

\[ C(3) = x_3x_2x_1a_1x_1a_2x_2x_1a_1x_1a_3x_3x_2x_1a_1x_1a_2x_2x_1a_1x_1, \]
\[ C_s(3) = a_1a_2a_1a_3a_1a_2a_1, \]

and observe that \( C_s(3)a_i \), for \( i = 1, 2, 3 \), all map to strings with squares.
If \( w \) is a square-free string over \( L = L_1, L_2, \ldots, L_{n-1} \), then \( L_n = \{a, b, c\} \) forces a square iff \( w \) has suffix \( C(3) \).

The proof of this result is most of the PSC2016 paper.

\( \iff \) direction is trivial
As all three squares \(tat\), \(ubub\), \(vcvc\) are suffixes of the string \(w\), it follows that \(t, u, v\) must be of different sizes, and so we can order them without loss of generality as follows:
\[
|tat| < |ubu| < |vcv|.
\]

We now consider different cases of the overlap of \(tat, ubu, vcv\), showing in each case that the resulting string has a suffix conforming to the pattern \(C(3)\).
For instance:

\( v = pubu \), where \( p \) is a proper non-empty prefix of \( v \).

Since \( w \) is square-free, we assume that \( pubu \) has no square, and therefore \( p \neq u \) and \( p \neq b \).

From this, we get \( vcv = pubucpubu \). Therefore, this case is possible.
Open Problem 2:

Show that for any $L = L_1, L_2, \ldots, L_n$, where $|L_i| = 3$, there always exists a square-free $w$ over $L$.

Problem: very difficult to run meaningful simulations, as the number of cases jumps up very quickly.

Hope: can we use our offending suffix characterization to do a counting argument?
Open Problem 3:

Map $L = \bigcup_i L_i \mapsto [n]$ where $n = |\bigcup_i L_i|$.

Let $M_L = (m_{ij})$ where $m_{ij} = 1 \iff j \in L_i$.

So each row if $M_L$ will have 3 1s.

Problem: given any such matrix, can we select a single 1 from each row in such a way that there are no $2k$ consecutive rows, where the initial $k$ rows equal the next $k$ rows.

How to run these simulations for $n > 10$?
A clever technique in complexity to show:

- Palindromes require $\Theta(n^2)$ steps on a single tape Turing machine.
- If a language requires less than $o(\log \log n)$ memory to be decided it is in fact regular: “miniscule memory = no memory.”

We use this to show that we can always avoid very large repetitions ($1/c$, $c \in \mathbb{N}$) over any list.
Perl

Truly sophisticated text manipulation libraries.

CGI.pm is a Perl module for CGI web applications — the workhorse of the Internet.

The text processing is done with state of the art algorithms from ... 1970s

The advances in String Algorithms should be reflected in popular implementations.
Please visit:

http://soltys.cs.csuci.edu

and/or email us to discuss this further:

michael.soltys@csuci.edu

pophlin@mcmaster.ca