Fast algorithms for the Kolakoski sequence

Richard P. Brent

Australian National University
and University of Newcastle

16 November 2016

Joint work with Judy-anne Osborn

Abstract

The *Kolakoski sequence* $1221121221221121122\ldots$ is the unique countable $\{1, 2\}$-sequence $k_1k_2k_3\ldots$ such that $k_1 = 1$, whose j-th block has length k_j. We shall briefly survey what is known and conjectured about this sequence.
Define the *Kolakoski discrepancy function* $\delta(n) := \sum_{1\le j\le n}(-1)^{k_j}$.
It is an open question whether $\delta(n) = o(n)$, i.e. whether the density of 1's in k is $\frac{1}{2}$.
Nilsson (2012) gave an algorithm for computing $k_1 \ldots k_n$ (and hence $\delta(n)$) in time $T = O(n)$ and space $S = O(\log n)$.
We shall describe an algorithm that computes $\delta(n)$ faster, using a space-time tradeoff. Ignoring logarithmic factors, it is conjectured that the algorithm runs in time $T = O((2/3)^d n + 2^d)$ using space $S = O(2^d)$. Taking $d \approx \log(n)/\log(3)$, this gives $T = O(n^\alpha)$ and $S = O(n^\alpha)$ for $\alpha = \log(2)/\log(3) \approx 0.631$.
Using our algorithm, we have computed $\delta(n)$ for various $n \le 5 \times 10^{17}$. The results provide numerical evidence for the conjecture that $\delta(n) = \widetilde{O}(n^{1/2})$.

Notation

$\widetilde{O}(f(n))$ means $O(f(n)(\log n)^c)$ for some $c \geq 0$.

We consider finite or countably infinite sequences over an alphabet Σ of two symbols. In the exposition we take $\Sigma := \{1, 2\}$.
In programs it is often more convenient to use $\Sigma := \{0, 1\}$.

We use *sequence*, *string* and *word* interchangeably
(in the literature some distinctions are made – e.g. strings are sometimes assumed to be finite and sequences can be over an uncountable set such as $\mathbb{R}$).

If $\sigma = \sigma_1 \ldots \sigma_n$ is a finite sequence, the *length* of σ is $\lambda(\sigma) := n$.

More notation

$\mathbb{N}$ is the set of positive integers.

The *empty string* (of length zero) is written as ε.

Σ^n is the set of 2^n strings $\sigma_1 \ldots \sigma_n$ of length exactly $n \geq 0$.

$\Sigma^* = \cup_{n \geq 0} \Sigma^n$ is the set of finite strings over Σ.

$\Sigma^{\mathbb{N}}$ is the set of countably infinite strings $\sigma_1 \sigma_2 \sigma_3 \ldots$ over Σ.

$U = \Sigma^*$, $\Sigma^{\mathbb{N}}$, or $\Sigma^* \cup \Sigma^{\mathbb{N}}$ is the *universe* of strings under consideration.

σ^j is a string of j consecutive σ s ($\sigma^0 = \varepsilon$, $\sigma^{j+1} = \sigma^j \sigma$).

If $\sigma = \sigma_1^{j_1} \sigma_2^{j_2} \sigma_3^{j_3} \ldots$ where $\sigma_i \neq \sigma_{i+1}$ and $j_i > 0$, then $\sigma_n^{j_n}$ is the n-th *run* in σ and its length is j_n.

Run-length decoding

We can define a "run-length decoding" function $T : U \to U$ by

$$T(\sigma_1\sigma_2\ldots) = 1^{\sigma_1}2^{\sigma_2}1^{\sigma_3}2^{\sigma_4}\ldots$$

For example, if $\sigma = 12221121$, then $T(\sigma) = 122112212112$.

σ is called the *mother* of $T(\sigma)$.

Strings can be motherless. In our example, σ contains a run of length three, so has no mother in $\{1, 2\}^*$.

Sometimes I feel like a motherless child (anon.)

The Kolakoski sequence k

The *Kolakoski sequence* k is the (unique) fixed point of the map $\sigma \in \Sigma^{\mathbb{N}} \mapsto T(\sigma)$. In other words, $k = T(k)$, and this defines $k \in \Sigma^{\mathbb{N}}$ uniquely.

Proof of uniqueness (sketch). Suppose $k = k_1 k_2 \ldots$ satisfies $k = T(k)$. Then $k_1 k_2 k_3 \ldots = 1^{k_1} 2^{k_2} 1^{k_3} \ldots$, where $1 \le k_j \le 2$. Comparing the first symbols on left and right shows that $k_1 = 1$. Thus $1 k_2 k_3 \ldots = 12^{k_2} \ldots$. Thus $k_2 = 2$ and $k_3 = 2$. Proceeding in this way, we see that, for $j \ge 3$, k_j on the left is defined by $k_1, k_2, \ldots, k_m$ on the right, for some $m < j$. This shows that the solution is unique (and also gives an algorithm for computing it, thus showing existence). $\square$

A small generalisation

We can define

$$T_1(\sigma_1\sigma_2\ldots) = 1^{\sigma_1}2^{\sigma_2}1^{\sigma_3}2^{\sigma_4}\ldots$$

and

$$T_2(\sigma_1\sigma_2\ldots) = 2^{\sigma_1}1^{\sigma_2}2^{\sigma_3}1^{\sigma_4}\ldots$$

(so T_1 is the same as T, and T_2 differs only in that its output string starts with 2 instead of 1).

We say that $\sigma = \sigma_1\sigma_2\ldots$ is the *mother* of both $T_1(\sigma)$ and $T_2(\sigma)$.

Thus, starting from any $u \in U = \Sigma^{\mathbb{N}} \cup \Sigma^*\backslash\{\varepsilon\}$ as the root, we obtain an (infinite) binary tree where the left child of σ is $T_1(\sigma)$ and the right child is $T_2(\sigma)$.

Remarks on fixed points

The equation $T_2(\widehat{k}) = \widehat{k}$ has a unique nonempty solution

$$\widehat{k} = 22112122122\ldots \in \Sigma^{\mathbb{N}},$$

and this solution is the same as the Kolakoski sequence k with the first symbol deleted, i.e.

$$k = 1\widehat{k}.$$

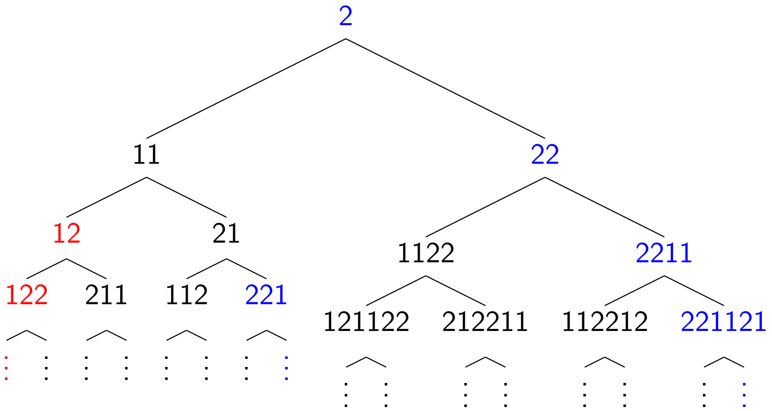
Sometimes the Kolakoski sequence is defined to be $\widehat{k}$ instead of k. We follow the definition in Sloane's OEIS (sequence A000002).

If we allow finite sequences, i.e. $\sigma \in \Sigma^*$, then there are some "trivial" fixed points:

$$T_1(\varepsilon) = \varepsilon, \quad T_2(\varepsilon) = \varepsilon \text{ and } T_1(1) = 1.$$

Example of a tree generated by T_1 and T_2

Starting from $\sigma = 2 \in \Sigma^*$, we obtain the following (infinite) tree whose nodes are finite $\{1,2\}$-sequences. Note that finite segments of the Kolakoski sequences $k = 122112\ldots$ and $\widehat{k} = 221121\ldots$ occur in the tree.



Generalisation

If we take a random (or deterministic) path down the tree from the root, we obtain a set of finite sequences that *could* (probably do) appear in the Kolakoski sequence.

Most of the results and conjectures about the Kolakoski sequence have analogues for such sets. For example, we can conjecture that the density of 1s in such a set is $1/2$.

Some history

The "Kolakowski" sequence was first defined (but not named) in a paper by Rufus Oldenburger (1939). The paper used the terminology of "symbolic dynamics" and referred to earlier work by Morse and Hedlund.

In 1965, William Kolakoski (in the problem section of the *American Mathematical Monthly*) gave the first 41 terms of the sequence and asked

1. What is a simple rule for constructing the sequence?
2. What is [a formula for] the n-th term?
3. Is the sequence periodic?

In 1966, Necdet Üçoluk answered Kolokoski's questions 1 and 3 (answer: the sequence is not periodic). Kolakoski and Üçoluk were apparently unaware of Oldenburger's paper.

History continued

Michael Keane (1991) and others conjectured that the density of 1s in k is $\frac{1}{2}$. This is still open. Even the existence of a density is open. Vašek Chvátal (1993) proved an upper density of 0.501 (and a corresponding lower density of 0.499).

Michaël Rao (2012) improved these bounds to 0.5001 and 0.4999.

Arturo Carpi (1994) surveyed previous results and proved that the Kolakoski sequence is cube-free, and contains only squares of length $2, 4, 6, 18$ and 54. Here a *square* is a nonempty substring of the form xx (with length defined to be $\lambda(xx) = 2\lambda(x)$), and a *cube* is a nonempty substring of the form xxx.

Johan Nilsson (2012) gave an algorithm for computing $k_1 \ldots k_n$ in time $T = O(n)$ and space $S = O(\log n)$.

For other historical references, see the OEIS entry A000002 and the Wikipedia page for "Kolakoski sequence".

Some open questions

There are many open questions/conjectures about the properties of the Kolakoski sequence k. Here is a sample from a report by Dekking (1995).

- What is the *subword complexity* of k, i.e. the cardinality $P(n)$ of the set of words of length n that occur in k? Dekking (1981) showed that $P(n)$ has polynomial growth.
- If a word $x = x_1 \ldots x_n$ occurs in k, does it occur infinitely often? Does it occur with bounded gaps? Does the *reverse* $x_n \ldots x_1$ occur? Does the *complement* $x'_1 \ldots x'_n$ occur? (Here $1' = 2, 2' = 1$.)
- *Equidistribution*: Does the frequency of 1 in k exist, and is it equal to $1/2$?
- *Subword equidistribution*: For each word w in k, does the frequency of w in k exist?
- *Strict transitivity*: For each word w in k, does the frequency of w in $k_n k_{n+1} \ldots$ exist uniformly in n?

Equidistribution

We'll concentrate on the question of equidistribution.

For a string $\sigma = \sigma_1 \ldots \sigma_n \in \Sigma^*$, recall that $\lambda(\sigma) = n$ denotes the length of σ. For $x \in \Sigma$, define $\lambda_x(\sigma)$ to be the number of occurrences of x in σ. For example, $\lambda_1(1121) = 3$, $\lambda_2(1121) = 1$.

By an abuse of notation, we abbreviate $\lambda_x(k_1 \ldots k_n)$ by $\lambda_x(n)$.

k is equidistributed if

$$\lim_{n\to\infty} \frac{\lambda_1(n)}{n} = \lim_{n\to\infty} \frac{\lambda_2(n)}{n} = \frac{1}{2}\,.$$

Since $\lambda_1(n) + \lambda_2(n) = n$, an equivalent condition is

$$\lim_{n\to\infty} \frac{\lambda_1(n)}{n} = \frac{1}{2}\,.$$

Equivalent conditions

Since

$$\sum_{j=1}^{n} k_j = \lambda_1(n) + 2\lambda_2(n),$$

an equivalent condition is

$$\lim_{n\to\infty} \frac{1}{n} \sum_{j=1}^{n} k_j = \frac{3}{2} \cdot$$

In other words, the “expansion factor” in going from $k_1 \ldots k_n$ to its child $1^{k_1} 2^{k_2} 1^{k_3} \cdots (\cdots)^{k_n}$ is asymptotically equal to $3/2$.

Another equivalent condition (stated in the abstract) is $\delta(n) = o(n)$, where

$$\delta(n) := \sum_{j}^{n} (-1)^{k_j} = \lambda_2(n) - \lambda_1(n) = n - 2\lambda_1(n).$$

If the k_j behaved like independent random variables (which they don't), we would expect $\delta(n) = O(n^{1/2} \log\log n)$ almost surely.

Algorithms

Since there is no known theoretical approach to proving equidistribution of k, there is an interest in computing $\delta(n)$ for large n to see if we can obtain numerical evidence for/against equidistribution.

To avoid needless anxiety, I will reveal now that computations for $n \le 5 \times 10^{17}$ support the equidistribution conjecture.

A linear time and space algorithm

It is straightforward to generate $k_1 k_2 \ldots k_n$ in linear fashion, using the fact that k is a fixed point of the "run-length decoding" function $T_1(\sigma_1\sigma_2\ldots) = 1^{\sigma_1}2^{\sigma_2}1^{\sigma_3}2^{\sigma_4}\ldots$

Denote the complement of $x \in \Sigma$ by x'. Thus $1' = 2,\ 2' = 1$, but we can represent 2 by 0 if desired (to save space).

We use an array A of $n+1$ bits $A_1 \ldots A_{n+1}$, and indices i, j.

Algorithm 1:

```
$(A_1, A_2, i, j) \leftarrow (1, 1', 2, 2)$;
while $i \le n$ do
 if $A_j = 1$ then $(A_i, i, j) \leftarrow (A'_{i-1}, i+1, j+1)$
   else     $(A_i, A_{i+1}, i, j) \leftarrow (A'_{i-1}, A'_{i-1}, i+2, j+1)$.
```

Now $A_1 \ldots A_n = k_1 \ldots k_n$.

The algorithm uses time $O(n)$ and space $O(n)$.

Illustration of Algorithm 1

The algorithm can be illustrated by the following table. As the indices i and j increase, the third and fourth columns both represent initial segments of the Kolakoski sequence.

i increases by 2 when $A_j = 2$ (skipped values in parentheses).

i	j	A_i	A_j
1	1	1	1
2	2	2	2
(3)		2	
4	3	1	2
(5)		1	
6	4	2	1
7	5	1	1
8	6	2	2
(9)		2	
10	7	1	1

Saving space via recursion

Nilsson (2012) improved Algorithm 1 by reducing the space required to generate k_n (or $\delta(n)$) from $O(n)$ to $O(\log n)$. The time required is still $O(n)$.

The idea is to generate the Kolakoski sequence by a recursive procedure, where the reference to A_j in Algorithm 1 is replaced by a recursive call to the same procedure, unless $j \leq 2$.

This can be illustrated by the table on the next slide.

Recursive generation of the Kolakoski sequence

Each column $A, B, C, \ldots$ gives the Kolakoski sequence. Column B generates column A, column C generates column B, etc. The depth of recursion increases at each blue row.

index	A	B	C	D	E	F	G	...
1	1	1	1	1	1	1	1	...
2	2	2	2	2	2	2	2	...
3	2							
4	1	2						
5	1							
6	2	1	2					
7	1	1						
8	2	2	1	2				
9	2							
10	1	1	1					
11	2	2	2	1	2			
12	2							
13	1	2						
14	1							
15	2	1	1	1				
16	1	2	2	2	1	2		
17	1							
18	2	2						
19	2							
20	1	1	2					
21	2	1						
22	1	2	1	1	1			
23	1							
24	2	1	2	2	2	1	2	
...								

A sublinear algorithm (Algorithm 2)

Nilsson's algorithm takes time of order n to generate k_n or $\delta(n)$. This is the best that we can do if all of $k_1, \ldots, k_n$ are required.

However, it is possible to generate a single k_n or $\delta(n)$ value (or a sparse sequence of such values) in time $\ll n$.

The first step is to convert Nilsson's recursive algorithm into an iterative algorithm that generates the table on the previous slide row by row. This gives a non-recursive algorithm with essentially the same time and space requirements as Nilsson's algorithm.

Next, we observe that a row in the table determines some of the following rows, as illustrated on the following slide.

One row determines several following rows

index	A	B	C	D	E	F
1–10	$\cdots$					
11	2	2	2	1	2	
12	2					
13	1	2				
14	1					
15	2	1	1	1		
16	1	2	2	2	1	2
$\cdots$						

Row 11 determines rows 12–15 as the number of columns (5) in row 11 is at least the number of columns in rows 12–15.

However, row 11 does not determine row 16, since row 16 has 6 columns, and the column labelled "F" could have an entry 1 or 2.

Another example

index	A	B	C	D
1–3	$\cdots$			
4	1	2		
5	1			
6	2	1	2	
7-12	$\cdots$			
13	1	2		
14	1			
15	2	1	1	1
$\cdots$				

Row 4 determines row 5, but not row 6. Row 13 is identical to row 4, so determines row 14, but not row 15. Note that row 6 and row 15 differ in the entries marked in red.

Using table lookup

For each d, $1 \leq d \leq d_{max}$, where d_{max} is determined by the amount of random-access memory available, we can construct a table of 2^d entries indexed by d-bit binary integers (the *keys*). Each such integer m corresponds to a row (say row r) with d symbols from Σ, which we can map to $\{0, 1\}$.

The table tells us how far we can skip ahead, say ℓ rows, and sufficient information to construct row $r + \ell$ from row r. It should also contain $\sum_{r<j\leq r+\ell} (-1)^{k_j}$ so there is enough information to determine $\delta(n)$ when we reach row n.

In fact, it is sufficient to consider rows ending in 2, since rows ending in 1 have $\ell = 0$.

If a row has length $> d_{max}$, or if the table lookup would skip over the desired index n, we revert to the "slow but sure" iterative version of Nilsson's algorithm.

Space-time tradeoff

The mean value of ℓ over all 2^d d-bit keys is $(3/2)^d$. On the assumption that each possible key is equally likely to occur, we expect a speedup of order $(3/2)^{d_{max}}$, ignoring logarithmic factors. This is confirmed by numerical experiments.

Note: the keys are *not* substrings of the Kolakoski sequence. The latter can not include strings such as 111 or 222, so the number of length-d substrings that occur in the Kolakoski sequence is (much) less than 2^d. (It is polynomial in d.)

We have to initialise the lookup tables. This takes time $\widetilde{O}(3^{d_{max}})$ if done in a lazy manner, but $\widetilde{O}(2^{d_{max}})$ if done recursively, starting from small d.

Thus, the total time to compute $\delta(n)$ is [conjectured to be]

$$\widetilde{O}((2/3)^{d_{max}} n + 2^{d_{max}}).$$

Choosing d_{max}

Choosing $d_{max} = \lfloor \log(n)/\log(3) \rfloor$ gives time $\widetilde{O}(n^{\alpha})$ and space $O(n^{\alpha})$, where $\alpha = \log(2)/\log(3) \approx 0.631$.

For the values of n that we are interested in, say $n \approx 10^{18}$, this would give $d_{max} \approx 37$. Our program uses about $24 \times 2^{d_{max}}$ bytes, so on a machine with say 128 GB of memory available we are limited to $d_{max} \le 32$. Thus, in practice, we are limited by the availability of memory. Choosing the largest possible $d_{max} \le \log(n)/\log(3)$, the runtime is $\widetilde{O}((2/3)^{d_{max}}\, n)$.

One way to interpret this analysis is as follows. Each time that we double the amount of memory available for lookup tables, we obtain a speedup by a factor of about $3/2$. Since $(3/2)^{30} \approx 191{,}000$, the speedup is significant in practice. It is optimal for $n \approx 3^{30} \approx 2 \times 10^{14}$, but sub-optimal for larger n.

Computing other quantities

It is possible to compute other functions related to $\delta(n)$ with only a constant factor slowdown by variants on the algorithm described for $\delta(n)$.

For example, we can compute

$$\max_{a \le n \le b} \delta(n) \quad \text{and} \quad \min_{a \le n \le b} \delta(n), \text{ and hence } \max_{a \le n \le b} |\delta(n)|,$$

at the expense of a factor $5/3$ in memory for lookup tables, and a small constant factor increase in running time.

Fractal nature of the Kolakoski sequence

A table lookup algorithm can speed up computation of $\delta(n)$ because the Kolakoski sequence exhibits some self-similarity, i.e. in some sense it is a *fractal* (I won't try to define this).

A *Wiener process* can be defined as a limit of a random walk. It can be used to model *Brownian motion*. Can we replace the random walk by $\delta(n)$ and obtain something that looks like a typical Wiener process? For example, do the zeros have Hausdorff dimension $1/2$?

Computation of $\delta(n)$

The following table gives some values of $\delta(n)$ and scaled values of $\delta(n)$ and $\Delta(n)$, where $\Delta(n) := \max_{1 \le j \le n} |\delta(j)|$.

n	$\delta(n)$	$\delta(n)/n^{1/2}$	$\Delta(n)/n^{1/2}$
10^3	-4	-0.1265	0.1897
10^6	+28	+0.0280	0.0660
10^9	-2,446	-0.0773	0.1560
10^{12}	-101,402	-0.1014	0.1515
10^{15}	-1,954,842	-0.0618	0.1390
5×10^{17}	+40,997,288	+0.0580	0.0921

Our computation took about 3.5 hours per block of 10^{15} on a 2GHz Intel Xeon running Linux, using 80GB memory ($d_{max} = 31$).

The $\delta(n)$ values are in agreement with those computed by Michaël Rao using a different program (to be discussed later).

Another table

n	$\delta(n)/n^{1/2}$	$\Delta(n)/n^{1/2}$
10^{10}	0.0466	0.1017
10^{11}	-0.0100	0.0945
10^{12}	-0.1014	0.1515
10^{13}	-0.0052	0.0997
10^{14}	-0.0632	0.0987
10^{15}	-0.0618	0.1390
10^{16}	0.1072	0.1202
10^{17}	0.1424	0.1519
2×10^{17}	0.1384	0.1456
3×10^{17}	0.0017	0.1189
4×10^{17}	0.0503	0.1029
5×10^{17}	0.0580	0.0921

Conjecture

From our $\Delta(n)$ computations we can conclude that

$$|\delta(n)| < n^{1/2}/4 \text{ for } 2000 \le n \le 5 \times 10^{17}.$$

Thus, it is natural to conjecture that $\delta(n) = \widetilde{O}(n^{1/2})$.

For a random walk with i.i.d. random variables we would get $\delta(n) \ll \sqrt{n \log\log n}$ almost surely (Khinchin's *law of the iterated logarithm*). More precisely,

$$\limsup \frac{\delta(n)}{\sqrt{2n \log\log n}} = 1 \text{ a.s.}$$

The function $\sqrt{\log\log n}$ grows too slowly to make a significant difference to the numerical results.

Another relevant result for random walks is that

$$\lim_{n\to\infty} \frac{E(|\delta(n)|)}{\sigma\sqrt{n}} = \sqrt{\frac{2}{\pi}} \approx 0.798 > 1/4.$$

A dubious conjecture

John Smith and Ariel Scolnikov (2013)[1] conjectured that

$$\delta(n) = O(\log n).$$

Our numerical results make this conjecture extremely unlikely. For example,

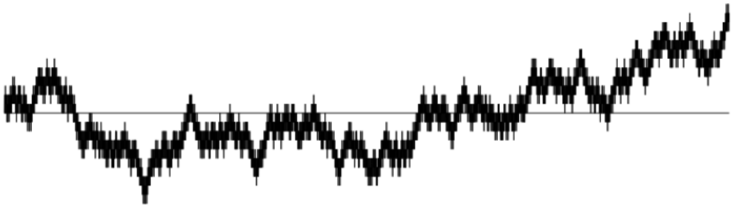
$$\frac{\delta(n)}{\log n} > 10^6 \text{ for } n = 5 \times 10^{17}.$$

[1] http://planetmath.org/kolakoskisequence. The conjecture is repeated at http://codegolf.stackexchange.com/questions/8369/.

$\delta(n)$ for $n \le 8000$

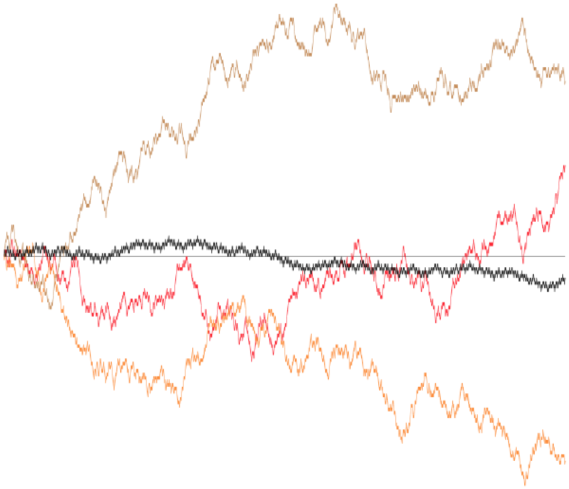
The graph shows $\delta(n)$ for $n \le 8000$.

There is evidence of self-similarity within the graph. This is as expected from the success of our table-lookup algorithm.



Comparison with random walks

The graphs compare the first 1600 steps of the Kolakoski sequence (black) with three random walks (using i.i.d. random variables with unit variance).



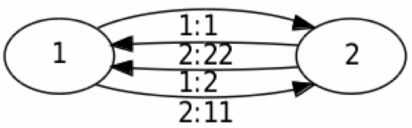
Finite-state transducers

A *finite-state transducer (FST)* is a finite-state machine with an input tape and an output tape.

For example, the Kolakoski sequence k is a fixed point of the FST $\mathcal{K}$ defined by the table

initial state	input	output	final state
1	1	1	2
1	2	11	2
2	1	2	1
2	2	22	1

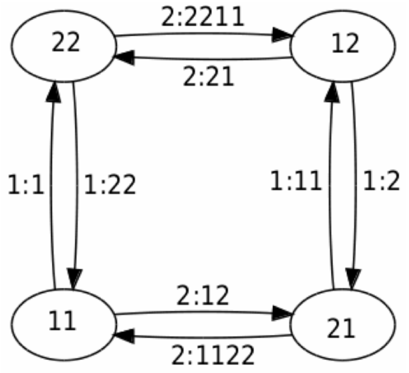
or equivalently by the labelled, directed graph



Composition of FSTs

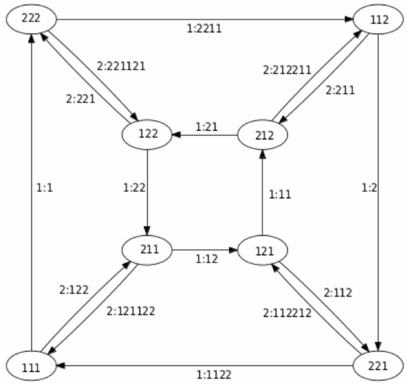
By connecting the output of an FST $\mathcal{K}$ to the input of an FST $\mathcal{L}$ we get an FST that is denoted by $\mathcal{K} \circ \mathcal{L}$. (Note the order!) The set of states of $\mathcal{K} \circ \mathcal{L}$ is the cartesian product of the sets of states of $\mathcal{K}$ and $\mathcal{L}$.

For example, $\mathcal{K}^2 := \mathcal{K} \circ \mathcal{K}$ can be represented by the graph



Composition of FSTs cont.

Similarly, $\mathcal{K}^3 := \mathcal{K}^2 \circ \mathcal{K}$ can be represented by



Acknowledgement: the graphs for $\mathcal{K}$, $\mathcal{K}^2$, $\mathcal{K}^3$ are from Michaël Rao's website `http://www.arthy.org/kola/kola.php`.

Rao's approach

Michaël Rao (2012) uses composition of FSTs to compute the Kolakoski sequence k.

Rao does not describe his approach in detail, but presumably he chooses some d_{max} determined by the computing resources available, computes (some subset of) $\mathcal{K}, \mathcal{K}^2, \ldots, \mathcal{K}^{d_{max}}$, then uses these FSTs to quickly compute (some subset of) $k_1, \ldots, k_n$ and $\delta(1), \ldots, \delta(n)$.

This is analogous to our Algorithm 2 that uses lookup-tables with keys of up to d_{max} bits. We have not implemented the FST approach or analysed it in detail. It is plausible that, if implemented efficiently, the space and time-complexity are the same (up to constant or perhaps logarithmic factors) as for Algorithm 2.

Rao's computation

Michaël Rao computed $\lambda_1(n)^2$ for various $n \le 10^{18}$. We have confirmed the values that he gives up to $n = 5 \times 10^{17}$.

Rao does not give any information about running times or storage requirements, so we can not compare the efficiency of his algorithm with that of our Algorithm 2.

So far as we know, Rao did not compute $\Delta(n)$.

[2]Recall that $\delta(n) = n - 2\lambda_1(n)$, so computing $\lambda_1(n)$ is essentially equivalent to computing $\delta(n)$.

Parallel algorithms

The initialisation phase of Algorithm 2 (and of Rao's algorithm) can easily be parallelised. However, it is not clear how to parallelise the final phase of either algorithm.

If we use $p \ll n^\alpha$ processors and parallelise just the initialisation phase of Algorithm 2, the running time becomes

$$\widetilde{O}\left(n\,(2/3)^{d_{max}} + 2^{d_{max}}/p\right).$$

The optimal choice of d_{max} gives running time $\widetilde{O}(n^\alpha p^{\alpha-1})$, where (as before) $\alpha \approx 0.631$. Thus, the parallel speedup is of order $p^{1-\alpha}$, assuming that sufficient memory is available. In practice it would pay to keep local copies of the smaller lookup tables and only distribute the larger tables.

Acknowledgement: thanks to David Harvey for his comments on an earlier version of these slides, particularly in relation to parallel algorithms.

References

A. Carpi, On repeated factors in C^∞-words, *Information Processing Letters* **52** (1994), 289–294.

V. Chvátal, *Notes on the Kolakoski sequence*, DIMACS Tech. Report 93-84, Dec. 1993. `http://dimacs.rutgers.edu/TechnicalReports/abstracts/1993/93-84.html`

F. M. Dekking, *What is the long range order in the Kolakoski sequence?*, Report 95–100, TU-Delft, 1995. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.28.6839`

W. Kolakoski, Self generating runs, Problem 5304, *Amer. Math. Monthly* **72** (1965), 674. Partial solution: Ü. Necdet, *ibid* **73** (1966), 681–682.

J. Nilsson, A space-efficient algorithm for calculating the digit distribution in the Kolakoski sequence, *J. of Integer Sequences* **15**, article 12.6.7 (2012).

J. Nilsson, Letter frequencies in the Kolakoski sequence, *Acta Physica Polonica A* **126** (2014), 549–552.

References cont.

R. Oldenburger, Exponent trajectories in symbolic dynamics, *Trans. Amer. Math. Soc.* **46** (1939), 453–466.

OEIS, Sequence A000002, Kolakoski sequence, http://oeis.org/A000002.

OEIS, Sequence A088568, partial sums of Oldenburger-Kolakoski sequence A000002, http://oeis.org/A088568.

M. Rao, *Trucs et bidules sur la séquence de Kolakoski*, Oct. 1, 2012. http://www.arthy.org/kola/kola.php.

Wikipedia, *Kolakoski sequence*, https://en.wikipedia.org/wiki/Kolakoski_sequence.

Wikipedia, *L-system*, https://en.wikipedia.org/wiki/L-system.