DIMACS Technical Report 93-84
December 1993

NOTES ON THE KOLAKOSKI SEQUENCE

by

Vašek Chvátal[1]
Dept. of Computer Science
Rutgers University
New Brunswick, New Jersey 08903

[1]Permanent Member

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

The *Kolakoski sequence*, 12211212212211211221211212211211212212211212212112..., is the unique countable 1–2 sequence $a_1 a_2 a_3 \ldots$ with $a_1 = 1$, whose j-th block has length a_j; Keane asked whether the density of 1's (and therefore also the density of 2's) in this sequence is 0.5. The purpose of this note is twofold: (i) to report computations proving that the upper density of 1's as well as the upper density of 2' in the Kolakoski sequence is less than 0.501, and (ii) to speculate about possible ways of answering Keane's question in the affirmative.

0. INTRODUCTION

A *block* in a sequence is any maximal subsequence of consecutive identical terms; each term of a *1–2 sequence* is 1 or 2. The *Kolakoski sequence* ([5]; see also [1, 2, 4, 8, 9])

$$122112122122112112212112122112112122122112122121 12\ldots,$$

is the unique countable 1–2 sequence $a_1a_2a_3\ldots$ with $a_1 = 1$, whose j-th block has length a_j; Keane ([3], p.50; see also [6], p.66, and [1]) asked whether the density of 1's (and therefore also the density of 2's) in this sequence is 0.5. The purpose of this note is twofold: (i) to report computations proving that the upper density of 1's as well as the upper density of 2' in the Kolakoski sequence is less than 0.501, and (ii) to speculate about possible ways of answering Keane's question in the affirmative.

1. THE RESULT

By the *mother* of a sequence s, we shall mean the sequence whose j-th term is the length of the j-th block of s. A 1–2 sequence will be always called *1-feasible*; it will be called *d-feasible* for some integer d greater than 1 if and only if its mother is a $(d-1)$-feasible 1–2 sequence. For example, the periodic sequence obtained by repeating 12 is 2-feasible but not 3-feasible; the periodic sequence obtained by repeating 122 is 3-feasible but not 4-feasible; the periodic sequence obtained by repeating 1221121122 is 4-feasible but not 5-feasible; the Kolakoski sequence is d-feasible for all positive integers d. As we shall see, the upper density of 1's as well as the upper density of 2's in every 22-feasible sequence is less than 0.50084.

Given any countable d-feasible sequence $a_1a_2a_3\ldots$, we shall refer to certain positive integers (interpreted as subscripts i of a_i) as *d-special* with respect to the sequence; to each d-special integer, we shall assign a *type* that is a 1–2 sequence of length d. Every positive integer i is 1-special of type a_i; when $d > 1$, we say that i is d-special of type $a_it_2\ldots t_d$ if and only if, for some positive integer j which is $(d-1)$-special of type $t_2\ldots t_d$ with respect to the mother of $a_1a_2a_3\ldots$, the j-th block of $a_1a_2a_3\ldots$ ends at the i-th term of $a_1a_2a_3\ldots$. To put it differently, let us write the mother of $a_1a_2a_3\ldots$ directly below $a_1a_2a_3\ldots$, aligning the two sequences in such a way that the j-th term of the mother is aligned with the last term in the j-th block of the daughter (and so each 2 in the mother is preceded by a blank); iterating this process (writing the grandmother below the mother, and so on), we eventually obtain an array with d rows such that the first row holds our starting sequence $a_1a_2a_3\ldots$ and each subsequent row holds the mother of the sequence in the row directly preceding it. For example, starting with $d = 4$ and $a_1a_2a_3\ldots = 212221121122122112122121122\ldots$, we obtain the array

2	1	2	2	1	1	2	1	1	2	2	1	2	2	1	1	2	1	2	2	1	2	1	1	2	2	…
1	1		2		2	1		2		2	1		2		2	1	1		2	1	1		2		2	…
	2				2	1				2	1				2		2		1		2				2	…
					2	1				1	1						2		1						2	…

A positive integer i is d-special if and only if the i-th column in this array contains no blanks; this column is the type of i. In our example, 6 is 4-special of type 1222, 7 is 4-special of type 2111, 11 is 4-special of type 2221, 12 is 4-special of type 1111, 18 is 4-special of type 1122, 20 is 4-special of type 2211, 26 is 4-special of type 2222, … .

No matter which countable d-feasible sequence $a_1a_2a_3\ldots$ we start with and no matter which two consecutive d-special integers i and j (consecutive in the sense that $i<j$ and that no integer strictly between i and j is d-special) we choose,

- the type of j
  determines the first $d-1$ terms of the type of i as well as all of $a_{i+1}a_{i+2}\ldots a_j$

and

- the type of i and the last term of the type of j
  determine the remaining $d-1$ terms of the type of j (and therefore all of $a_{i+1}a_{i+2}\ldots a_j$).

These relationships are conveniently described by a certain directed graph, whose vertices are all the 1–2 sequences of length d: there is a directed edge from the type of i to the type of j and this edge is labeled by $a_{i+1}a_{i+2}\ldots a_j$. We shall refer to this graph as G_d; for illustration, G_1,G_2,G_3, and G_4 are shown in Figures 1 and 2.

Each G_d with $d>1$ arises from G_{d-1} by the following construction: with $u \xrightarrow{\alpha} v$ meaning that there is a directed edge from u to v and that this edge is labeled by α,

$$\begin{array}{rcl}
A1 \xrightarrow{\alpha} B2 \text{ in } G_{d-1} & \text{gives rise to} & A11 \xrightarrow{\alpha} B21 \text{ and } A12 \xrightarrow{\alpha} B21 \text{ in } G_d, \\
A2 \xrightarrow{\alpha} B1 \text{ in } G_{d-1} & \text{gives rise to} & A21 \xrightarrow{\alpha} B11 \text{ and } A22 \xrightarrow{\alpha} B11 \text{ in } G_d, \\
A1 \xrightarrow{\alpha} B2 \xrightarrow{\beta} C2 \text{ in } G_{d-1} & \text{gives rise to} & A11 \xrightarrow{\alpha\beta} C22 \text{ and } A12 \xrightarrow{\alpha\beta} C22 \text{ in } G_d, \\
A2 \xrightarrow{\alpha} B1 \xrightarrow{\beta} C1 \text{ in } G_{d-1} & \text{gives rise to} & A21 \xrightarrow{\alpha\beta} C12 \text{ and } A22 \xrightarrow{\alpha\beta} C12 \text{ in } G_d.
\end{array}$$

Hence for each 1–2 sequence X of a positive length d there are a unique 1–2 sequence $p(X)$ of length $d-1$ and a unique finite 1–2 sequence $s(X)$ such that

$$p(X)1 \xrightarrow{s(X)} X \quad \text{and} \quad p(X)2 \xrightarrow{s(X)} X :$$

we have

$$\begin{array}{l}
p(Y21) = p(Y2)1, \\
p(Y11) = p(Y1)2, \\
p(Y22) = p(Z2)1 \text{ with } Z = p(Y2), \\
p(Y12) = p(Z1)2 \text{ with } Z = p(Y1)
\end{array}$$

and

$$\begin{array}{rl}
s(Y1) &= s(Y), \\
s(Y22) &= s(Z2)s(Y2) \text{ with } Z = p(Y2), \\
s(Y12) &= s(Z1)s(Y1) \text{ with } Z = p(Y1).
\end{array}$$

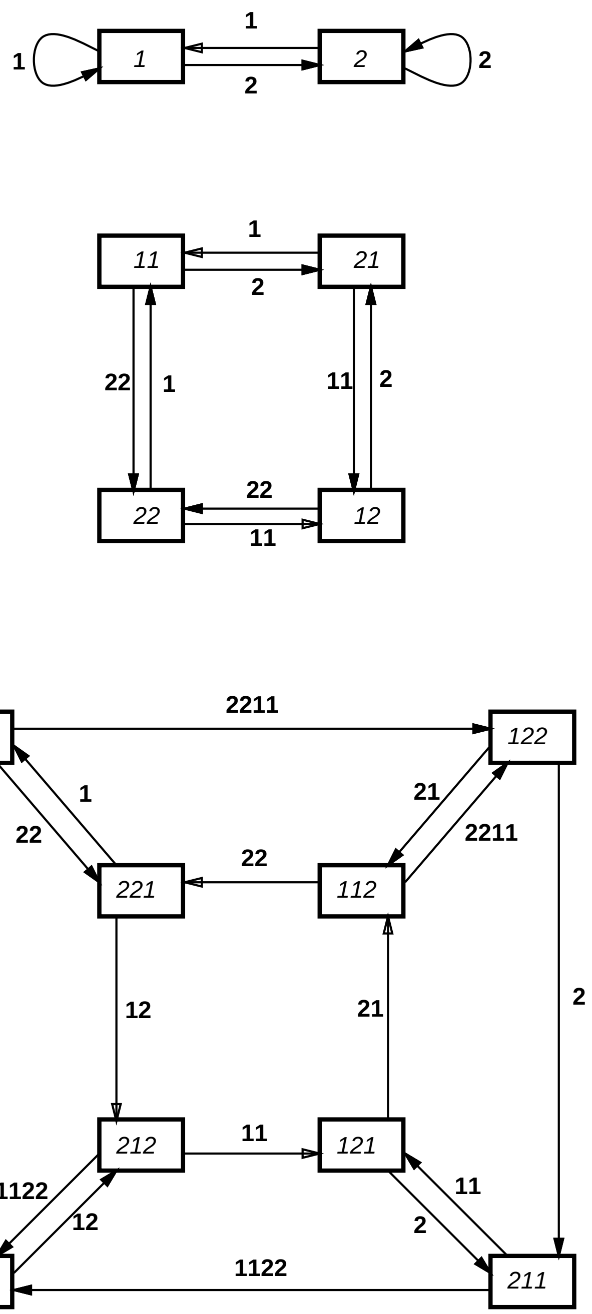

Figure 1: G_1, G_2, G_3.

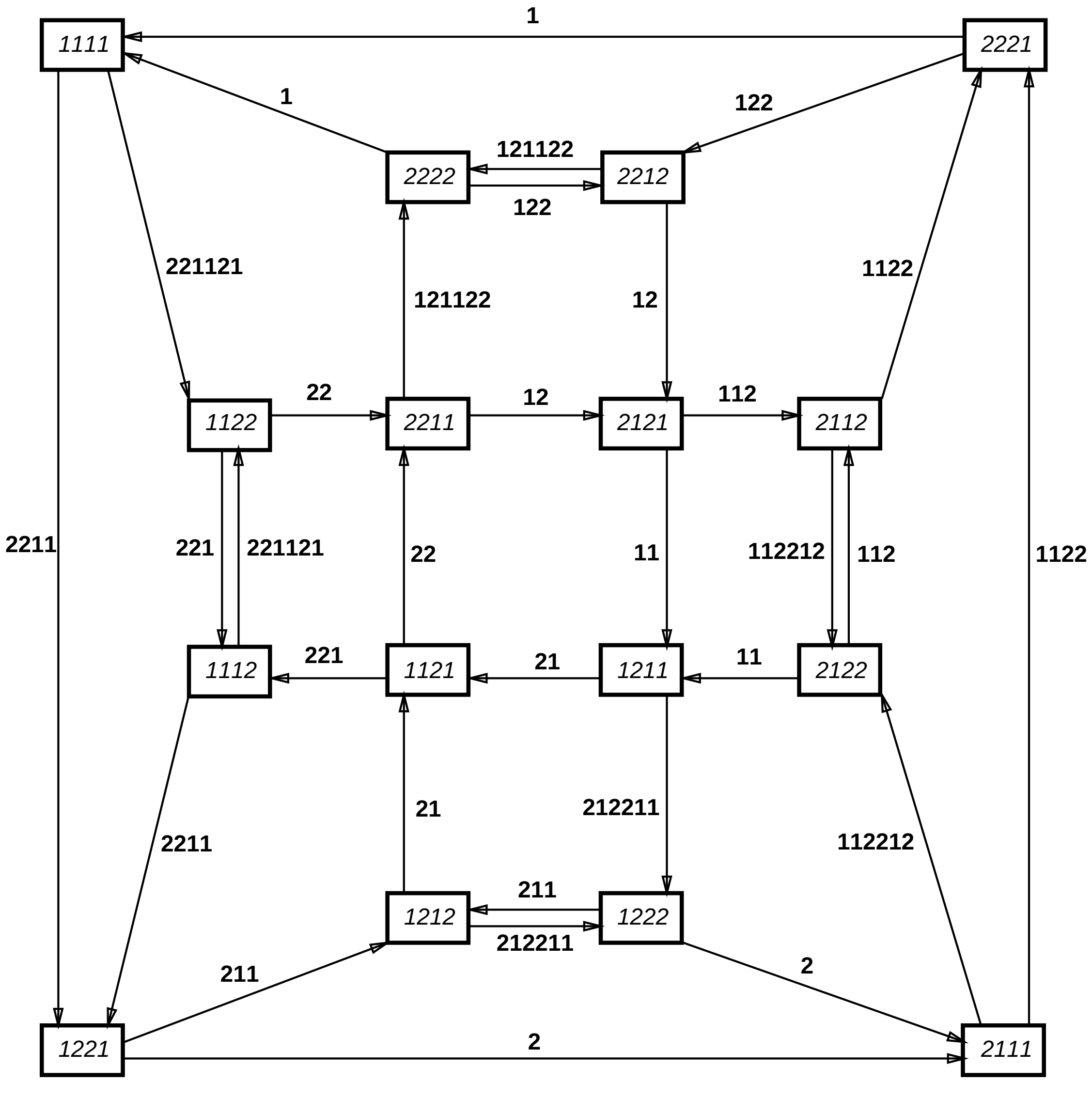
1111
2221
1
1
122
2222
2212
121122
122
221121
121122
12
1122
1122
22
2211
12
2121
112
2112
2211
221
221121
22
11
112212
112
1122
1112
221
1121
21
1211
11
2122
21
212211
2211
112212
1212
211
1222
212211
211
2
1221
2
2111

Figure 2: G_4.

The values of $p(X)$ and $s(X)$ can be computed from these recursive formulas or by iteratively computing 1–2 sequences $s_d, s_{d-1}, \ldots, s_1$ such that s_d consists of the last term of X and each s_j with $j < d$ is the unique daughter of s_{j+1}, whose last term is the j-th term of X: the j-th term of $p(X)$ is distinct from the first term of s_j, and $s(X) = s_1$. For example, if $X = 2212121$ then we compute
$s_7 = 1$,
$s_6 = 2$,
$s_5 = 11$,
$s_4 = 12$,
$s_3 = 211$,
$s_2 = 2212$,
$s_1 = 1122122$,
and conclude that $p(X) = 211221$, $s(X) = 1122122$. This process can be reversed: given a 1–2 sequence Y of length d, we can compute 1–2 sequences $s_d, s_{d-1}, \ldots, s_1$ such that s_d is either 1 or 2 and each s_j with $j < d$ is the unique daughter of s_{j+1}, whose first term is distinct from the j-th term of Y; letting X stand for the 1–2 sequence of length d, whose j-th term is the last term of s_j, we have $Y \stackrel{s}{\to} X$ with $s = s_1$. For example, if $Y = 2212121$ then we compute
$s_7 = 1$,
$s_6 = 1$,
$s_5 = 2$,
$s_4 = 11$,
$s_3 = 21$,
$s_2 = 112$,
$s_1 = 1211$
to conclude that $Y \stackrel{s}{\to} 1211211$ with $s = 1211$ and we compute
$s_7 = 2$,
$s_6 = 11$,
$s_5 = 21$,
$s_4 = 112$,
$s_3 = 2122$,
$s_2 = 1121122$,
$s_1 = 1211212211$
to conclude that $Y \stackrel{s}{\to} 1222112$ with $s = 1211212211$.

Let the *weight* of of a directed cycle in G_d mean the total number of digits in the labels on its edges and let the the *content* of of the directed cycle mean the total number of 1's in the labels on its edges; let u_d denote the largest ratio of content to weight over all directed cycles in G_d. Every d-feasible sequence consists of a finite prefix followed by the concatenation of labels on the edges of some infinite walk through G_d; hence the upper density of 1's in every d-feasible sequence is at most u_d. (Since switching the first character in every type yields an automorphism of G_d that switches all characters in all edge-labels, the upper density of 2's in every d-feasible sequence is also at most u_d.) Machine computations show that

$$
\begin{aligned}
u_1 &= 1/1 &&= 1.000000,\\
u_2 &= 2/3 &&\approx 0.666667,\\
u_3 &= 2/3 &&\approx 0.666667,\\
u_4 &= 5/9 &&\approx 0.555556,\\
u_5 &= 8/15 &&\approx 0.533333,\\
u_6 &= 36/69 &&\approx 0.521739,\\
u_7 &= 36/69 &&\approx 0.521739,\\
u_8 &= 64/123 &&\approx 0.520325,\\
u_9 &= 64/123 &&\approx 0.520325,\\
u_{10} &= 286/561 &&\approx 0.509804,\\
u_{11} &= 286/561 &&\approx 0.509804,\\
u_{12} &= 624/1234 &&\approx 0.505673,\\
u_{13} &= 1158/2295 &&\approx 0.504576,\\
u_{14} &= 1992/3952 &&\approx 0.504049,\\
u_{15} &= 10333/20535 &&\approx 0.503190,\\
u_{16} &= 5310/10557 &&\approx 0.502984,\\
u_{17} &= 9912/19737 &&\approx 0.502204,\\
u_{18} &= 28157/56097 &&\approx 0.501934,\\
u_{19} &= 30975/61761 &&\approx 0.501530,\\
u_{20} &= 30354/60555 &&\approx 0.501263,\\
u_{21} &= 284631/568101 &&\approx 0.501022,\\
u_{22} &= 616904/1231743 &&\approx 0.500838;
\end{aligned}
$$

for details, see the Appendix. In particular, the upper density of 1's (as well as the upper density of 2's) in every 22-feasible sequence is less than 0.50084.

To compare this result with the initial behaviour of the Kolakoski sequence $a_1a_2a_3\dots$, let b_n denote the number of 1's minus the number of 2's in $a_1a_2a_3\dots a_n$. From $b_1 = 1$, the values of b_n eventually swing into the negative region, reaching $b_{12} = -2$. The next wave swings up, reaching its maximum at $b_{32} = 2$,

$$
\begin{aligned}
&\text{followed by a wave with a minimum at} && b_{93} &&= -3,\\
&\text{followed by a wave with a maximum at} && b_{257} &&= 3,\\
&\text{followed by a wave with a minimum at} && b_{471} &&= -5,\\
&\text{followed by a wave with a maximum at} && b_{1533} &&= 11,\\
&\text{followed by a wave with a minimum at} && b_{97502} &&= -66,\\
&\text{followed by a wave with a maximum at} && b_{334915} &&= 63,\\
&\text{followed by a wave with a minimum at} && b_{2222194} &&= -154,\\
&\text{followed by a wave with a maximum at} && b_{526200069} &&= 4933,
\end{aligned}
$$

and so on (?); the first billion values of b_n are between -154 and 4933. In particular, the density of 1's in $a_1a_2\dots a_n$ is about 0.5036 when $n = 1533$, but *seems* to be confined to the band 0.5 ± 0.00026 for all n greater than 97501.

2. SPECULATIONS

Let w_d denote the smallest ratio of weight to length (=number of edges) over all directed cycles in G_d. We claim that

$$w_{d+m} \geq w_d w_{m+1} \quad \text{for all choices of positive integers } d \text{ and } m. \tag{1}$$

To justify this claim, recall that each edge uv in G_{d+m} arises from a path in G_{d+m-1}, which consists either of a single edge (in case v ends with 1) or of two edges (in case v ends with 2); in either case, this path leads from a prefix of u to a prefix of v and its weight is the weight of uv. Iterating m times, we unfold each edge uv in G_{d+m} into a walk from a prefix of u to a prefix of v in G_d; the weight of this walk is the weight of uv; its length is the weight of the edge leading from a suffix of u to a suffix of v in G_{m+1}. In this way, each directed cycle C in G_{d+m} unfolds into a closed walk C' in G_d; the weight of C' is the weight of C; the length of C' is at least w_{m+1} times the length of C. Since C' is in G_d, its weight is at least w_d times its length; hence the weight of C is at least $w_d w_{m+1}$ times the length of C.

From (1), we have

$$w_d = \Omega((w_{m+1}^{1/m})^d) \text{ for all positive integers } m; \tag{2}$$

machine computations show that

$$\begin{array}{rcrcr}
w_1 &=& 1/1 &=& 1.0000,\\
w_2 &=& 2/2 &=& 1.0000,\\
w_3 &=& 3/2 &=& 1.5000,\\
w_4 &=& 8/4 &=& 2.0000,\\
w_5 &=& 12/4 &=& 3.0000,\\
w_6 &=& 18/4 &=& 4.5000,\\
w_7 &=& 27/4 &=& 6.7500,\\
w_8 &=& 80/8 &=& 10.0000,\\
w_9 &=& 120/8 &=& 15.0000,\\
w_{10} &=& 180/8 &=& 22.5000,\\
w_{11} &=& 269/8 &=& 33.6250,\\
w_{12} &=& 804/16 &=& 50.2500,\\
w_{13} &=& 1206/16 &=& 75.3750,\\
w_{14} &=& 1809/16 &=& 113.0625,\\
w_{15} &=& 5420/32 &=& 169.3750,\\
w_{16} &=& 8130/32 &=& 254.0625,\\
w_{17} &=& 12195/32 &\approx& 381.0938,\\
w_{18} &=& 36580/64 &\approx& 571.5625,\\
w_{19} &=& 54870/64 &\approx& 857.3438,\\
w_{20} &=& 164570/128 &\approx& 1285.7031,\\
w_{21} &=& 246855/128 &\approx& 1928.5547,\\
w_{22} &=& 740484/256 &\approx& 2892.5156;
\end{array}$$

setting $m = 21$ in (2), we get

$$w_d = \Omega(1.46158^{\,d}). \tag{3}$$

Incidentally, the duality principle of linear programming (see also the Appendix) guarantees that

$$w_d = \max_{\pi} \min_{uv} \,(\text{length of the label on } uv + \pi(u) - \pi(v))$$

with the minimum over all the edges uv of G_d and the maximum over all the "potential functions" π that assign rational numbers to vertices of G_d; since the *average* length of an edge-label in G_d is $1.5^{\,d-1}$ and G_d is regular, it follows that $w_d \leq 1.5^{\,d-1}$ for all d. It is tempting to conjecture that $w_d = (1.5 + o(1))^{\,d}$.

Next, let the *excess* of a directed cycle in G_d mean the total number of 1's minus the total number of 2's in the labels on its edges; let e_d denote largest ratio of excess to length over all directed cycles in G_d.

CONJECTURE 1: $e_d = O(1.46157^{\,d})$.

Since $u_d \leq 0.5 + e_d/2w_d$, an affirmative answer to Keane's question would follow from Conjecture 1 by virtue of (3). Machine computations show that

$$\begin{aligned}
e_1 &= 1/1 &&= 1.0000,\\
e_2 &= 1/2 &&= 0.5000,\\
e_3 &= 1/2 &&= 0.5000,\\
e_4 &= 1/2 &&= 0.5000,\\
e_5 &= 3/12 &&= 0.2500,\\
e_6 &= 3/8 &&= 0.3750,\\
e_7 &= 5/12 &&\approx 0.4167,\\
e_8 &= 5/8 &&= 0.6250,\\
e_9 &= 5/8 &&= 0.6250,\\
e_{10} &= 11/16 &&= 0.6875,\\
e_{11} &= 43/56 &&\approx 0.7679,\\
e_{12} &= 25/26 &&\approx 0.9615,\\
e_{13} &= 21/16 &&= 1.3125,\\
e_{14} &= 57/36 &&\approx 1.5833,\\
e_{15} &= 131/64 &&\approx 2.0469,\\
e_{16} &= 63/24 &&= 2.6250,\\
e_{17} &= 47/16 &&= 2.9375,\\
e_{18} &= 102/26 &&\approx 3.9231,\\
e_{19} &= 137/30 &&\approx 4.5667,\\
e_{20} &= 207/36 &&= 5.7500,\\
e_{21} &= 367/52 &&\approx 7.0577,\\
e_{22} &= 828/100 &&= 8.2800,
\end{aligned}$$

suggesting that e_d may grow like $c^{\,d}$ with c around 1.2 or 1.3.

Finally, let f_d denote the maximum of the number of 1's minus the number of 2's in an edge-label in G_d.

CONJECTURE 2: $f_d = O(1.46157^d)$.

Since $e_d \le f_d$ for all d, Conjecture 2 is stronger than Conjecture 1. However, the notion of f_d is simpler than that of e_d: recall that edge-labels in G_d are precisely those 2^d sequences s for which the d times iterated mother of s is the sequence 1. Machine computations show that

$$\begin{array}{rclrclrcl}
f_1 &=& 1 & f_{11} &=& 6, & f_{21} &=& 36,\\
f_2 &=& 2, & f_{12} &=& 8, & f_{22} &=& 47,\\
f_3 &=& 2, & f_{13} &=& 8, & f_{23} &=& 57,\\
f_4 &=& 2, & f_{14} &=& 9, & f_{24} &=& 73,\\
f_5 &=& 2, & f_{15} &=& 10, & f_{25} &=& 89,\\
f_6 &=& 3, & f_{16} &=& 14, & f_{26} &=& 110,\\
f_7 &=& 3, & f_{17} &=& 16, & f_{27} &=& 140,\\
f_8 &=& 3, & f_{18} &=& 19, & f_{28} &=& 178,\\
f_9 &=& 4, & f_{19} &=& 24, & f_{29} &=& 220,\\
f_{10} &=& 4, & f_{20} &=& 32, & f_{30} &=& 282;
\end{array}$$

suggesting that f_d may also grow like c^d with c around 1.2 or 1.3. Attaining f_d seems to be a rare and unpredictable distinction: it is held

in G_1, only by edges leading into 1;

in G_2, only by edges leading into 12;

in G_3, only by edges leading into 121;

in G_4, only by edges leading into 1211;

in G_5, only by edges leading into 11212,
12111,
12112;

in G_6, only by edges leading into 112122,
121222;

in G_7, only by edges leading into 1121221,
1211112,
1212212,
1212221;

in G_8, only by edges leading into	11111222, 11211112, 11212211, 11212212, 12111121, 12122112, 12122121, 12122122, 12122211, 12221212;
in G_9, only by edges leading into	112211122, 121222222;
in G_{10}, only by edges leading into	1111221112, 1121112122, 1121211122, 1122111221, 1211121222, 1212222212, 1212222221, 1212222222, 2112222222, 2221211122;
in G_{11}, only by edges leading into	12111211122, 12122222222;
in G_{12}, only by edges leading into	112111122222, 121222111122;
in G_{13}, only by edges leading into	1121111122122, 1121111222221, 1212212111222, 1212221111221;
in G_{14}, only by edges leading into	11111211211222, 11211112222122, 11211211221122, 12122221212222;

in G_{15}, only by edges leading into	111122211111122,
	111122212122122,
	112111111111122,
	112111122221212,
	112112211111222,
	112112212212222,
	112122121111222,
	121212222122222,
	121221122122112,
	121221122122122,
	121221122122222,
	122212222221122;
in G_{16}, only by edges leading into	1111112212211122,
	1212211221222222;
in G_{17}, only by edges leading into	11211111111122122,
	12221222212111222;
in G_{18}, only by edges leading into	111212212222211222,
	112111122221222122,
	112112111121222122,
	112112211122211222,
	121221211121211122,
	121221222112222222;
in G_{19}, only by edges leading into	1121121122212212222,
	1122211222222221122;
in G_{20}, only by edges leading into	11211111111122122122,
	11211112222111111122,
	11222211221222122222,
	12122222212121111222;
in G_{21}, only by edges leading into	111112212122221221222,
	111121222211212211122,
	112222112212221222222,
	121222121112221212122;
in G_{22}, only by edges leading into	1111222111111211122222,
	1222111221222122111122;

in G_{23}, only by edges leading into 11222221121212222211122,
12111112121112211222222,
21111112121112211222222,
22222221121212222211122;

in G_{24}, only by edges leading into 112111222112212112212222,
112121122222222221221222,
112211112111212221212122,
122221122212222122221122;

in G_{25}, only by edges leading into 1122111212212111122211122,
1221222112222221111222222;

in G_{26}, only by edges leading into 11212221111221112222211122,
12111221111111212211222222,
21111221111111212211222222,
22212221111221112222211122;

in G_{27}, only by edges leading into 112111112212211221111222222,
112211122112221111122211122;

in G_{28}, only by edges leading into 1111111221212111222212212222,
1121212221222211122222221122;

in G_{29}, only by edges leading into 11212121211122121222212212222,
12112111211222221122222221122,
21112111211222221122222221122,
22212121211122121222212212222;

in G_{30}, only by edges leading into 112212122121111111212112212122,
121222221212121111212212221222,
211222221212121111212212221222,
222212122121111111212112212122.

ACKNOWLEDGMENTS

I learned about Keane's question from Eric Milner at a workshop organized by Geňa Hahn. Adrian Bondy thought about the question with me during the workshop and encouraged me in my computer programming afterwards. When I was still in the dark as to the origins of the problem, Leo Khachiyan gave me a preprint of [6]; then I was led step by step to Jeff Lagarias, Mike Keane, and Michel Dekking, who provided me with additional papers on the subject. Dominique Sotteau pointed out rough spots in a draft of this note. I want to thank all of them.

References

[1] F. M. Dekking, Regularity and irregularity of sequences generated by automata, *Séminaire de Théorie des Nombres de Bordeaux*, 1979–1980, exposé n° 9.

[2] F. M. Dekking, On the structure of selfgenerating sequences, *Séminaire de Théorie des Nombres de Bordeaux*, 1980–1981, exposé n° 31.

[3] M. S. Keane, Ergodic theory and subshifts of finite type, in: *Ergodic Theory, Symbolic Dynamics and Hyperbolic Spaces*, T. Bedford, M. Keane, C. Series (Eds.), Oxford University Press, Oxford, 1991.

[4] C. Kimberling, Advanced Problem 6281*, *American Math. Monthly* **86** (1979), 793.

[5] W. Kolakoski, Self Generating Runs, Problem 5304, *American Math. Monthly* **72** (1965), 674. Solution: *American Math. Monthly* **73** (1966), 681–682.

[6] J. C. Lagarias, Number theory and dynamical systems, in: *The Unreasonable Effectiveness of Number Theory*, S. A. Burr (Ed.), American Math. Society, Providence, R.I., 1992.

[7] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[8] N. J. A. Sloane, *A handbook of integer sequences*, Academic Press, New York, 1973. See sequence 70.

[9] W. D. Weakley, On the number of C^∞-words of each length, *J. Combin. Theory*, Ser. A, **51** (1989), 55–62.

APPENDIX: COMPUTER PROGRAMS

Here we describe the computer program used to find the values of u_d with $d \leq 22$; the programs used to find the values of w_d and e_d are similar and the program used to find the values of f_d is simpler.

The program is written in ANSI C and was run on a Sun SPARC station 10/41 with 64Mb memory. Vertices of G_d are encoded as the integers $0, 1, \ldots, 2^d - 1$: each type $t_1 t_2 \ldots t_d$ is encoded as the integer $\sum (t_i - 1) 2^{d-i}$. In this representation, for each vertex i of G_d there are integers p and c such that $2p \xrightarrow{s} i$, $2p + 1 \xrightarrow{s} i$, and the content of (that is, the number of 1's in) s is c; writing $p(i, d) = p$ and $c(i, d) = c$, we have

$$p(0,1) = p(1,1) = 0, \quad c(0,1) = 1, \quad c(1,1) = 0$$

and

$$\begin{aligned}
p(4i+2, j) &= 2p(2i+1, j-1), \\
p(4i \phantom{{}+2}, j) &= 2p(2i, j-1) + 1, \\
p(4i+3, j) &= 2p(2p(2i+1, j-1) + 1, j-1), \\
p(4i+1, j) &= 2p(2p(2i, j-1), j-1) + 1, \\
c(2i \phantom{{}+2}, j) &= c(i, j-1), \\
c(4i+3, j) &= c(2p(2i+1, j-1) + 1, j-1) + c(2i+1, j-1), \\
c(4i+1, j) &= c(2p(2i, j-1), j-1) + c(2i, j-1)
\end{aligned}$$

whenever $j \geq 2$. The following function uses these recursive formulas to return $p(i, j)$ and $c(i, j)$ as members `p` and `c` of a structure `pair`. (The value of `INT_MAX` is $2^{31} - 1$; function `strange` prints an appropriate message and calls `exit`.)

```
pair getpair(int i, int j)
{
  pair r,s;

  if(j==breakpoint) return table[i];

  if((i&1)==0)
    {
      if((i&2)==0)
        {
          /* i encodes ...11 */
          r=getpair(i/2,j-1);
          r.p=2*(r.p)+1;
          return r;
        }
      /* i encodes ...21 */
      r=getpair(i/2,j-1);
      r.p=2*(r.p);
      return r;
    }
  if((i&2)==0)
    {
      /* i encodes ...12 */
      r=getpair(i/2,j-1);
      s=getpair(2*(r.p),j-1);
      r.p=2*(s.p)+1;
      r.c+=s.c;
      if(abs(r.c)>INT_MAX/2) strange(1);
      return r;
    }
  /* i encodes ...22 */
  r=getpair(i/2,j-1);
  s=getpair((2*(r.p)+1),j-1);
  r.p=2*(s.p);
  r.c+=s.c;
  if(abs(r.c)>INT_MAX/2) strange(1);
  return r;
}
```

Here, we could set simply `breakpoint=1` and initialize the external array `table` by

```
table[0].p=0, table[0].c=1, table[1].p=0, table[1].c=0;
```

to make `getpair` run faster, we set `breakpoint` at $\min(d, 20)$ and initialize `table` of size $2^{\texttt{breakpoint}}$ by the following function with `c0=1, c1=0`.

```
void maketable(int c0, int c1)
{
  int i,step,halfstep;
  char parity;
  pair temp;

  table[0].p=table[tablesize/2].p=0;
  table[0].c=c0, table[tablesize/2].c=c1;

  for(step=tablesize/2; step>1; step/=2)
    {
      /* at the beginning of the j-th iteration,
         table[i*step].p=p(i,j) and table[i*step].c=c(i,j) */
      halfstep=step/2;

      parity=0;
      for(i=halfstep; i<tablesize; i+=step)
        {
          table[i]=table[i-halfstep];
          temp=table[(2*table[i].p+parity)*step];
          table[i].p=2*temp.p+1-parity;
          table[i].c+=temp.c;
          if(abs(table[i].c)>INT_MAX/2) strange(2);
          parity=1-parity;
        }

      parity=0;
      for(i=0; i<tablesize; i+=step)
        {
          table[i].p=2*table[i].p+1-parity;
          parity=1-parity;
        }
    }
}
```

More generally, for an arbitrary choice of integers a and b, define the *cost* of an edge as a times the content of its label plus b times the length of its label; as soon as `maketable(a+b,b)` is called, each subsequent call of `getpair(i,d)` makes member `.c` of its return value equal to the common cost of the two edges entering i in G_d.

Function main, called with a value of d in the command-line argument, sets the values of external variables nodes, breakpoint, and tablesize; then it keeps calling function improve that, given any lower bound on u_k (represented as a ratio try.num/try.den of two integers), either returns a greater lower bound or sets an external variable done to YES, indicating that the current lower bound is sharp.

```
void main(int argc, char *argv[])
{
  int i;
  ratio try;

  if (argc != 2)
    {
      printf("SPECIFY THE VALUE OF d!\n");
      exit(1);
    }
  sscanf(argv[1],"%d",&d);
  printf("COMPUTING u_%d:\n", d);

  nodes=1;
  for(i=0; i<d; i++) nodes*=2;

  breakpoint=(d < MAXBREAKPOINT ? d : MAXBREAKPOINT);
  tablesize=1;
  for(i=0; i<breakpoint; i++) tablesize*=2;

  done=NO;
  for(try.num=1, try.den=2; done==NO; try=improve(try))
      printf("trying num=%d and den=%d ", try.num, try.den);
  exit(0);
}
```

The job of improve(try) amounts to a search for a negative cycle in G_d with edge-costs defined as try.num times length minus try.den times content. If a directed cycle with a negative total cost is found then improve(try) returns the content and the weight of this cycle; if no such cycle exists then $u_k \leq$ try.num/try.den (with the sign of equality since try.num and try.den are the content and the weight of some directed cycle in G_d).

The search for a negative cycle is a variation on the theme presented in Chapter 3 of [7]. Each vertex is assigned a *potential* and a *predecessor*, which are stored in external arrays pot and pred. The value of each pot[i] is an integer; the value of each pred[i] is either a vertex of G_d or the symbolic constant NONE (defined as -1). The actual work is done by three functions:

- if `changepot()` returns the symbolic constant `NO` then it has set the values of `pot` in such a way that `pot[i]`+cost(`i`$\to$`j`)$\geq$ `pot[j]` for all edges `i`$\to$`j` of G_d (which certifies that every directed cycle in G_d is nonnegative),
- if `cycle()` returns a vertex `i` (rather than the symbolic constant `NONE`) then this vertex lies on some negative cycle C (all of whose edges have the form `pred[j]`$\to$`j`), in which case
- `newratio(i)` delivers the content and the length of C as the `.num` and the `.den` members of its return value.

```
ratio improve(ratio try)
{
  int i,counter;

  maketable(try.num-try.den, try.num);
  for(i=0; i<nodes; i++) pot[i]=INT_MAX, pred[i]=NONE;
  pot[0]=0;

  for(counter=1;;counter++)
    {
      if(changepot()==NO)
        {
          done=YES;
          printf("took %d iterations to success.\n", counter);
          return;
        }

      i=cycle();
      if(i!=NONE)
        {
          printf("took %d iterations,\n", counter);
          return newratio(i);
        }
    }
}
```

```
char changepot(void)
{
  char change=NO;
  int j;
  pair t;
  int pred1,pred2,newpot;

  for(j=0; j<nodes; j++)
    {
      t=getpair(j,d);
      pred1=2*(t.p), pred2=pred1+1;

      if(pot[pred1]<pot[pred2])
        {
          newpot=pot[pred1]+t.c;
          if(newpot<pot[j])
            {
              if(abs(newpot)>INT_MAX/2) strange(3);
              pot[j]=newpot, pred[j]=pred1, change=YES;
            }
        }
      else
        {
          if(pot[pred2]==INT_MAX) continue;
          newpot=pot[pred2]+t.c;
          if(newpot<pot[j])
            {
              if(abs(newpot)>INT_MAX/2) strange(3);
              pot[j]=newpot, pred[j]=pred2, change=YES;
            }
        }
    }

  if(change==YES) return YES;
  for(j=0; j<nodes; j++) if(pot[j]==INT_MAX) strange(4);
  return NO;
}
```

Here, the statement

```
for(j=0; j<nodes; j++) if(pot[j]==INT_MAX) strange(4);
```

could be omitted if it were known in advance that G_d is strongly connected, or at least that every vertex of G_d can be reached from vertex $11\ldots1$ (encoded as 0) by a directed path.

Note that `changepot` maintains the following invariant:

$$\texttt{pot[pred[j]]}+\mathrm{cost}(\texttt{pred[j]}\rightarrow\texttt{j}) \leq \texttt{pot[j]} \text{ whenever } \texttt{pred[j]}\neq\texttt{NONE}.$$

If the subgraph of G_d consisting of all the edges `pred[j]`$\rightarrow$`j` contains a directed cycle then this cycle is negative; to see this, consider the last moment at which some `pred[j]` with `j` on this cycle has been reset.

```
int cycle(void)
{
  int i,j;
  char mark[MAXNODES];
  for(j=0; j<nodes; j++) mark[j]=NEW;

  for(j=0; j<nodes; j++)
    {
      for(i=j; i!=NONE && mark[i]==NEW; i=pred[i]) mark[i]=NOW;
      if(i!=NONE && mark[i]==NOW) return i;
      for(i=j; i!=NONE && mark[i]==NOW; i=pred[i]) mark[i]=OLD;
    }
  return NONE;
}
```

We have already observed that switching the first character in every type yields an automorphism of G_d that switches all characters in all edge-labels. Hence for every 1–2 sequence X of length $d-1$, the two sequences $s(1X)$ and $s(2X)$ are mirror images of each other: they have the same length and differ in every term. In particular, their common length is the content of $s(1X)$ plus the content of $s(2X)$; function `newratio` relies on this observation.

```
ratio newratio(int sentinel)
{
  int i,j;
  int half=nodes/2;
  int s,t;
  ratio new;

  maketable(1,0);

  new.num=new.den=0;
  i=sentinel;
  do
    {
      j=(i<half ? i+half : i-half);
      s=getpair(i,d).c, t=getpair(j,d).c;
      new.num+=s, new.den+=s+t;
      if(abs(new.den)>INT_MAX/2) strange(5);
      i=pred[i];
    }
  while(i!=sentinel);

  return new;
}
```

When called with $d = 22$, the program produces the output

```
COMPUTING u_22:
trying num=1 and den=2 took 8 iterations,
trying num=41496 and den=82986 took 9 iterations,
trying num=39498 and den=78949 took 9 iterations,
trying num=22044 and den=44032 took 13 iterations,
trying num=52961 and den=105782 took 17 iterations,
trying num=369512 and den=737965 took 21 iterations,
trying num=130927 and den=261453 took 24 iterations,
trying num=123007 and den=245623 took 25 iterations,
trying num=71384 and den=142533 took 58 iterations,
trying num=511621 and den=1021542 took 108 iterations,
trying num=445945 and den=890406 took 128 iterations,
trying num=315509 and den=629965 took 184 iterations,
trying num=616904 and den=1231743 took 316 iterations to success.
```

in about seven and half hours of user time.

It may be interesting to note what happens if `cycle` is called only after every m iterations of `changepot` for some fixed m greater than one: as m increases, one might expect the total running time to first decrease, until it reaches its minimum at some optimal value of m, and then to increase. Although the running time does follow this trend to some extent, it does so quite erraticaly: changes in the value of m lead to changes in the return value of `improve`, which in turn affect the total number of iterations of `changepot` in unpredictable ways. If $m = 9$ then then the program produces the output

```
COMPUTING u_22:
trying num=1 and den=2 took 9 iterations,
trying num=39498 and den=78949 took 9 iterations,
trying num=22044 and den=44032 took 18 iterations,
trying num=254191 and den=507584 took 27 iterations,
trying num=114208 and den=228048 took 27 iterations,
trying num=71384 and den=142533 took 81 iterations,
trying num=445945 and den=890406 took 135 iterations,
trying num=315509 and den=629965 took 189 iterations,
trying num=616904 and den=1231743 took 316 iterations to success.
```

in about four and half hours of user time.