Polynacci Rotary Encoding

Tom Gibara

October 2009

Abstract

This document presents an overview of a circular binary encoding developed as part of the Moseycode[1] system. It allows for the efficiently encoding and decoding of integer values while ensuring that the orientation of the encoding remains unambiguous. The encoding does not provide error detection or correction.

1 Context

Suppose that we want to encode an integer in a circular pattern of dots. The positions at which dots may be placed are evenly spaced around the circle and their number is fixed. Suppose further that we also want to know, based only on the pattern of dots, the orientation of the encoding. That is to know unambiguously the dot at which the encoding starts. It is assumed that the dots are sufficiently small that they do not overlap.

Figure 1 shows a set of circular dot patterns over five positions. Each pattern in this set has a uniquely distinguishable orientation and is distinct from the other patterns up to rotation. This set could form the basis of an encoding for the first five natural numbers.

Figure 1: A set of circular dot patterns in which each pattern is distinct from the others and has an identifiable orientation.

[1] http://www.tomgibara.com/android/moseycode

2 Problem

Can we find a practical and efficient algorithm that can encode (resp. decode) any sufficiently small positive integer to (resp. from) a circular arrangement of black dots (under the conditions described above) such that the orientation of the encoding is unambiguous?

In this context, efficient means that the number of dot positions d, required to encode any integer in the interval $[0, n-1]$, should be close to the necessary minimum; this may be argued (informally) to be $log_2(n) - log_2(log_2(n))$. Equivalently, if $N(d)$ is the number of uniquely representable integers on d dots, then $N(d)$ should be close to $2^{d-log2(d)}$.

For the algorithm to be practical, it must be compact, require few (if any) data tables and operate within a low polynomial time.

3 Solution

Below we derive an algorithm based on a generalization of the Fibonacci numbers which is $O(d)$.

3.1 Marking the Orientation

Arguably the simplest way of unambiguously exhibiting the orientation of a circular dot pattern is to include a maximal sequence: a sequence of adjacent dots that is greater in length than any other such sequence in the pattern. Let p be the number of dots in a maximal sequence; p cannot be zero since that would imply there were no dots leaving no feature with which to identify the orientation. By definition, we require a gap at both ends of this sequence of p dots so, assuming that $p < d-2$, there are $d-p-2$ positions remaining on the circle. The pattern of dots in these positions is free except for the constraint that there is no sequence of adjacent dots that equals or exceeds p in length. In the case where $p = d-2$ or $p = d-1$ no positions remain to form any pattern. The case $p = d$ is clearly not permissable since, with all positions occupied by dots, there is no possible way to identify the orientation.

3.2 Polynacci Codes

For $p < d-2$, let the number of positions available in the pattern be $r = d-p-2$. We are interested in dot-patterns of length r in which no sequence of dots has length p or greater. Let N_{pr} be the number of such patterns. Can we compute N_{pr} and is there a practical way that the patterns can be used to encode $[0, \ldots, N_{pr}-1]$?

In the case where $p = 1$, there is only one possible pattern, the 'empty' pattern that contains no dots, thus $N_{1r} = 1$ for all r.

It is instructive to examine the case where $p = 2$. There is an established coding, called the Fibonacci coding, based on the Fibonacci sequence which

generates binary integer encodings that have no adjacent 1s.[2] This is identical to our requirement that there are no two successive dots in the case where $p = 2$. Based on this encoding it is easy to see that $N_{2r} = F(r+1)$, the smallest number not representable in r digits of a Fibonacci coding. In what follows, it is assumed that the reader is familiar with the Fibonacci coding.

To address the case where $p > 2$, we generalize F. Let F_n be a generalized Fibonacci ("polynacci") sequence with:

$$F_n(i) = \begin{cases} 2^i & i < n \\ \sum_{k=1}^{n} F_n(i-k) & otherwise \end{cases}$$

Though it differs in indexing, F_2 clearly defines the familiar Fibonacci sequence. With this new definition, $N_{2r} = F_2(r)$.

The Fibonacci coding uses a positional number system where the value of the nth position is $F(n)$. By replacing this place-value system with the values obtained from F_p we create a family of encodings indexed by p. Each encoding having the desired property that no successive p digits are 1's.

This generalized Fibonacci encoding provides both the algorithm (which is identical to that in Fibonacci coding) and the equation for the number of representable integers, $N_{pr} = F_p(r)$.

3.3 Combined Encoding

For a fixed d, circular patterns of dots that are generated with a unique maximal sequence of adjacent dots, are clearly partitioned by the length of their maximal sequence. This means that each set of circular patterns with a given maximal sequence length can unambiguously encode a different integer range. The total number of distinct encodable values is thus

$$\begin{aligned} N_d &= 2 + \sum_{p=1}^{d-3} N_{pr} \\ &= 2 + \sum_{p=1}^{d-3} F_p(d-p-2) \\ &= 2 + \sum_{i=3}^{d-1} F_i(i-2) \end{aligned}$$

The additional 2 encodings arise from the two single patterns with $p = d-1$ and $p = d-2$.

[2] Wikipedia provides some information on Fibonacci coding, see http://en.wikipedia.org/wiki/Fibonacci_coding

Algorithm 1 Encoding

Input: An integer value v.
Output: A sequence of d dots for display in a circle.

1. If v is out of range ($v < 0$ or $v > S_{d-2}$) raise an error
2. Choose the greatest i such that $S_i \leq v$
3. If $i = 0$, output $d - 1$ successive dots
4. If $i = 1$, output $d - 2$ successive dots
5. If $i > 1$ then
   let $p = d - (i + 1)$,
   output p successive dots,
   skip one position,
   compute the pth Fibonacci encoding of $v - S_i$ and
   for each 1 in the encoding, output a dot and for each 0 skip a position

Note: Output is assumed to proceed in a fixed direction around the circle, from a fixed point on the circle.

3.4 Algorithms

Given the observations that have already been made, it is straightforward to compose an algorithm that first identifies the necessary polynacci coding based on the size of the value to be encoded before outputing the maximal dot sequence followed by any dots generated from the polynacci encoding.

In the encoding and decoding algorithms, we assume the availability of a sequence $(S_0, S_1, \ldots, S_{d-2})$ that identifies the lower bound of values encoded by a given polynacci code. This can be computed with:

$$S_i = \begin{cases} i & i < 3 \\ S_{i-1} + F_{d-(i+1)}(i-1) & \textit{otherwise} \end{cases}$$

Note: Since d is fixed, as per the context of the problem, this sequence can be pre-computed.

Algorithm 2 Decoding

Input: A circular pattern of d dots.
Output: The encoded value and its orientation.

1. Given a circular dot pattern identify the start of the maximal sequence.
2. If this start position is not unique (due to multiple maximal sequences or no empty positions) raise an error.
3. The orientation is given by the position of the start of the maximal sequence.
4. Let $i = d - p - 1$ where p is the number of dots in the maximal sequence.
5. Read from each of the dot positions that are not included in, or adjacent to, the maximal sequence: a 1 for each dot and a 0 for each blank position.
6. The decoded value is the sum of S_i and the pth Fibonacci decoding of this binary sequence.

Note: the direction in which dots are input is dependent on the direction chosen for output.

Figure 2: Sample output from the encoding algorithm. This is the result of encoding all possible values on 7 dot positions. In this rendering, adjacent dots have been consolidated.

4 Analysis

This section briefly presents some figures and diagrams that demonstrate the efficiency of the encoding.

d	$N(d)$	*bits*	*optimum*	*loss*	*ratio*
3	2	1.00	1.42	0.415	0.707
4	3	1.58	2.00	0.415	0.792
5	5	2.32	2.68	0.356	0.867
6	8	3.00	3.42	0.415	0.878
7	14	3.81	4.19	0.385	0.908
8	24	4.58	5.00	0.415	0.917
9	43	5.43	5.83	0.404	0.931
10	77	6.27	6.68	0.411	0.938
11	140	7.13	7.54	0.411	0.945
12	256	8.00	8.42	0.415	0.951
13	472	8.88	9.30	0.417	0.955
14	874	9.77	10.19	0.421	0.959
15	1628	10.67	11.09	0.424	0.962
16	3045	11.57	12.00	0.428	0.964
17	5719	12.48	12.91	0.431	0.967
18	10780	13.40	13.83	0.434	0.969
19	20388	14.32	14.75	0.437	0.970
20	38674	15.24	15.68	0.439	0.972
21	73562	16.17	16.61	0.441	0.973
22	140268	17.10	17.54	0.443	0.975
23	268066	18.03	18.48	0.444	0.976
24	513350	18.97	19.42	0.445	0.977
25	984911	19.91	20.36	0.447	0.978
26	1892875	20.85	21.30	0.447	0.979
27	3643570	21.80	22.25	0.448	0.980
28	7023562	22.74	23.19	0.449	0.981
29	13557020	23.69	24.14	0.449	0.981
30	26200182	24.64	25.09	0.450	0.982
31	50691978	25.60	26.05	0.451	0.983

Table 1: *bits* is the number of binary bits of data that can be encoded in one circular pattern ($log_2(N(d))$), *optimum* is the theoretical maximum number of bits which could be stored ($log_2(n) - log_2(log_2(n))$), *loss* is the number of bits lost in the encoding ($optimum - bits$) and *ratio* is the number of lost bits as a proportion of the optimum ($ratio/optimum$).

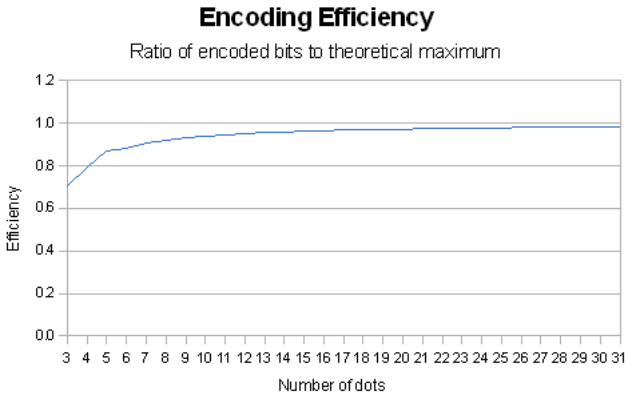

Figure 3: The efficiency of the encoding (as defined in Table 1) plotted against the number of dot positions.

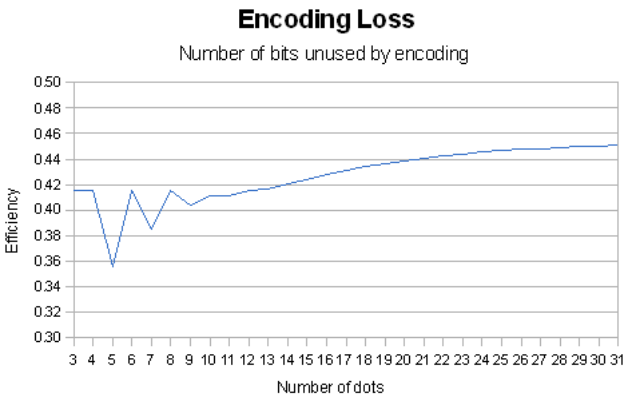

Figure 4: The loss of the encoding (as defined in Table 1) plotted against the number of dot positions.