Rapid Encoding of Run Lengths and Pyramid Cubic Lattices

Rafail E. Krichevskiy, Member, IEEE, and Vladimir N. Potapov

Abstract—We can encode rare events with an overhead of about 1.56 bits/event. The contribution of the overhead to the total length of the code is negligible. The encoding and decoding time counted in operations over bits per bit of the code does not depend on the number of appearances of the events. We present also algorithms of the same speed which enumerate the pyramid cubic lattices with an overhead of about 1.56 bits/dimension.

The overhead is the price for reaching the ultimate (to within a constant factor) encoding and decoding speed.

Index Terms—Coding overhead, coding speed, enumeration, rare events, redundancy-speed tradeoff.

I. INTRODUCTION

An injective mapping of a set of words $S$ into a set of binary words of a length $L$ is called an enumeration of $S$; the number

$$ R = L - \log |S| $$

is called the redundancy or the overhead of the enumeration, $|S|$ being the cardinality of $S$, $\log$ being binary.

The encoding and decoding are performed by a computer. We count its running time in operations over bits, which we call bit ops. We want to reach a very high coding and decoding speed keeping the redundancy below some level. We say that an algorithm is per bit linear if its running time equals the length of the input to within a constant factor.

We are concerned with two sets: the set of rare events and the set of pyramid cubic lattices.

First, the set of rare events $S(n, p)$. It is defined as the set of binary words of length $n$ in which the one (rare event) appears $p$ times, $n \to \infty$, $p \to \infty$, $p/n \to 0$.

Second, pyramid cubic lattices. We denote by $||x||$ the $l_1$-norm of a vector $x = (x_1, \ldots, x_p)$. A $p$-dimensional pyramid with side $n$ is the set $\{ x : ||x|| \leq n \}$, $n > 0$; its surface is the set $\{ x : ||x|| = n \}$. A (surface) pyramid cubic lattice is the set of vectors with nonnegative integer coordinates which belong to a (surface) pyramid. An enumeration of pyramid surface lattices gives an enumeration of pyramid lattices.

The problems of encoding the mentioned sets are equivalent. Any binary word can be represented as a sequence of the lengths of runs of zeros separated by ones.

For simplicity, we suppose in the sequel $p = 2^t$, $t = 0, 1, \ldots$. We present per bit linear enumerations of the above sets. Their redundancy divided either by the number of events or by the dimension is asymptotically bounded. We estimate the bound for rare events and pyramid cubic lattices as 1.56.

Manuscript received November 7, 1996; revised April 25, 1997. This work was supported by the Russian Foundation of Basic Research under Grant N-99-01-99531, and by NSF under Grant 9703844.

R. E. Krichevskiy is with the College of Engineering, University of California Riverside, Riverside, CA 92507 USA (e-mail: raafai@ee.ucr.edu) and the Sobolev Mathematical Institute, Novosibirsk 630090 Russia (e-mail: rekri@math.nsc.ru).

V. N. Potapov is with the Sobolev Mathematical Institute, Novosibirsk 630090 Russia.

Communicated by N. Merhav, Associate Editor for Source Coding.

II. HISTORY

Now we consider briefly well-known algorithms of enumerations of pyramid cubic lattices and runs of rare events. Babkin [7] proposed the following method. Let $h(x_1, x_2, \ldots, x_{n-1}, 0)$ be the binary digit be the number of elements of $S(n, p)$ having sequence $x_1, x_2, \ldots, x_{n-1}, 0$ as their prefix. Then the numbers of $(x_1, \ldots, x_n) \in S(n, p)$ in lexicographic order equal

$$ \sum_{i=1}^{n} x_i h(x_1, \ldots, x_{i-1}, 0). $$

Babkin’s code of $(x_1, \ldots, x_n)$ is the binary notation of this sum. It is clear that Babkin’s algorithm enumerates $S(n, p)$ without redundancy. The length of Babkin’s code of $x \in S(n, p)$ is asymptotically equal to

$$ p \log (n/p) + (\log e + o(1))p $$

where $n \to \infty$, $p \to \infty$, $p/n \to 0$.

Let us estimate coding and decoding speed of this algorithm. If the sum $\sum_{i=1}^{n} x_i = w$ then

$$ h(x_1, x_2, \ldots, x_{n-1}, 0) = \left( \frac{n-k}{p-w} \right). $$

To find the codeword it is necessary to add $p$ numbers $(\binom{n-k}{p-w})$ at least. It requires about $p^2 \log n$ bit ops or about $p$ times greater than the length of input, i.e., Babkin’s algorithm is not per bit linear. In addition it is necessary to calculate or to store binomial coefficients. Cover’s [8] algorithm is analogous. Pan and Fischer’s [5] extend this algorithm for enumeration of the cubic lattices of weighted pyramids.

For example we consider Babkin’s code of $(127, 7)$. The cardinality of $(127, 7)$ is $8936451775$. The length of the code is equal to $\log(8936451775) + 1 = 37$. Let $x \in S(127, 7)$ be the sequence containing ones at $5, 31, 40, 58, 62, 77, 107$ places. Babkin’s code of $x$ is the binary notation of the number

$$ \left( \binom{127}{7} + \binom{69}{5} + \binom{69}{4} + \binom{69}{3} + \binom{69}{2} + \binom{20}{1} \right) = 67948545451 $$

eq, it equals 0111111010010000011001000101101011.
Shannon’s algorithm [1] is speedier, but it is redundant. The runs of zeros are enumerated by binary words with no \( m \) ones in succession, \( m > 1 \). Namely, we rewrite the lengths of runs of zeros in \( \left( 2^m - 1 \right) \)-cimal system. Every digit of \( 2^m - 1 \) we denote via an \( m \) digit binary number. We use the remaining \( m \) digit binary number as a comma. Let \( m = 3 \) and \( n \) be the comma. Then Shannon’s code of 4 is 100 and Shannon’s code of 29 is 100001. Elias showed [2] that \( L / \Pi \to 1 \) as \( p / n \to 0 \) and \( m \to \infty \). Precisely, the length of Shannon’s code of \( x \in S(n, p) \) is equal to

\[
(1 + \frac{1}{m} 2^m + o(1)) p \log(n/p)
\]

i.e., for any fixed \( m \) the per event redundancy \( R/p \to \infty \) as \( p/n \to 0 \).

Consider the previous example: \( x \in S(127, 7) \) is the sequence having ones at 5, 31, 40, 58, 62, 77, 107 places. Shannon’s code of \( x \) is the sequence of 7-cimal notation of numbers 4, 25, 8, 17, 13, 14, 29 separated by 111. It is equal to

\[
10011001011010101000100101000001110000111010
\]

The length of Shannon’s code of \( x \) is 54.

To enumerate the lengths of runs of zeros we can use any prefix code of integers. Elias [3] developed some codes of the kind. The most simple of them is \( \gamma' \). The code \( \gamma'(y) \) is the binary notation of \( y \) having \( \lfloor \log y \rfloor \) zeros as a prefix. For example, the code of 5 is 00101. \( \gamma'(0) \) is not defined. Then \( \gamma' \) code of the same \( x \in S(127, 7) \) is the concatenation of \( \gamma'(5), \gamma'(26), \gamma'(9), \ldots \). It is equal to

\[
0011001011010101000100101000001110000111010
\]

The length of \( \gamma' \) code of \( x \) is 51.

The most condensed (for large integer) Elias’s code \( \omega \) [3] is analogous to Levenshtein’s code [4]. Let \( Bin'y \) be the binary notation of an integer \( y \) without the first one. The series of the functions \( \log^{(n)} y \) is defined by induction: \( \log^{(1)} y \) \( \equiv \lfloor \log y \rfloor \) and \( \log^{(n)} y \) \( \equiv [\log^{(n-1)} y] \). The function \( \log^* \) \( y \) is defined by the equation

\[
\log^* y \equiv 0.
\]

For example, \( \log^* 3 \equiv 2 \). Levenshtein’s code of \( y \) is defined as

\[
1^{\log^* y} 0\text{Bin}'(\log \log^* y \cdot 2^{(n-2)} \cdot \ldots \cdot \text{Bin}' y).
\]

Levenshtein’s code of 4 is 1110000 and Levenshtein’s code of 29 is 1111000011010. It is obvious that the length of Levenshtein’s code of \( x \in S(n, p) \) is asymptotically equal to

\[
p \log(n/p) + (1 + o(1)) p \log(n/p)
\]

where \( n \to \infty \), \( p \to \infty \), \( p/n \to 0 \), i.e., \( R/p \to \infty \).

Levenshtein’s code of \( x \) is the concatenation of codes of integers 4, 25, 8, 17, 13, 14, 29. It is equal to

\[
11100001110000110011000011100001111010
\]

The length of Levenshtein’s code of \( x \) is 63.

Some original prefix codes of integers are constructed in [10]–[12]. Ahlsweide et al. [9] gave detailed review of all known algorithms of prefix coding integers. In addition, they proved asymptotic (where \( y \to \infty \)) inequality for any prefix code \( f \) based on Kraft inequality

\[
|f(y)| > \log y + \log^{(2)} y + \ldots + \log^{(r)} y - r \log \log e
\]

where \( r = \log^* y \), and \( e \) is the base of natural logarithm. Thus the lengths of prefix codes of \( p \) integers coded separately are asymptotically greater then

\[
p \log n + p \log^{(2)} n.
\]

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Redundancy per rare event or dimension</th>
<th>Running time per bit input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Babkin’s</td>
<td>( 1/p )</td>
<td>( O(p)^* )</td>
</tr>
<tr>
<td>Shannon’s</td>
<td>( \frac{1}{2} \log \log(n/p) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>prefix code of integers</td>
<td>( \geq \log \log(n/p) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>our code</td>
<td>( \leq 1, 56 )</td>
<td>( O(1) )</td>
</tr>
</tbody>
</table>

where \( n \) is the sum of integers. All mentioned prefix codes have very high coding and decoding speed but their redundancy is greater then \( \log^{(2)} (n/p) \) per bit.

The Fast Golomb’s code [13] is efficient as \( p/n \to \varepsilon \), where \( 0 < \varepsilon \ll 1 \). The redundancy of this code was estimated in [14].

Table I displays the results discussed.

III. ENUMERATION OF PYRAMID CUBIC LATTICES

We introduce first some definitions and relations. Let \( Bin'x \) be the binary notation of an integer \( x: Bin0 = 0, Bin1 = 1 \), it \( Bin2 = 10 \), etc. We denote by \([w] \) the length of a binary word \( w \). Then \([Bin0] = [Bin1] = 1, [Bin2] = 2 \). Obviously,

\[
[Bin x] \leq \log(x + 1) + 1, \quad x \geq 0.
\]

Suppose we have nonnegative numbers \( y_1, \ldots, y_k, k \geq 1 \). By (2) we obtain for the total length of their binary notations

\[
\sum_{i=1}^{k} [Bin y_i] \leq k + \sum_{i=1}^{k} \log(y_i + 1).
\]

The arithmetic mean–geometric mean inequality yields

\[
\sum_{i=1}^{k} \log(y_i + 1) \leq k \log \left( 1 + \frac{1}{k} \sum_{i=1}^{k} y_i \right).
\]

From (3) and (4)

\[
\sum_{i=1}^{k} [Bin y_i] \leq k + k \log \left( 1 + \frac{1}{k} \sum_{i=1}^{k} y_i \right).
\]

We need binary trees. Each node of a tree is identified with a binary word \( \sigma \). The root is identified with \( \emptyset \); the left (right) son of \( \sigma \) is \( 0(1) \).

A node \( \sigma \) belongs to the \( |\sigma| \)th level of the tree.

Every node is put into correspondence with a subvector of \( x = (x_1, x_2, \ldots, x_p) \). The root corresponds to \( x \), leaves correspond to coordinates of \( x \). If \( \sigma \) corresponds to a subvector \( x_\tau \), then \( \sigma 0 (\sigma 1) \) corresponds to the left (right) half of \( x \). For example, \( 0 \to (x_1, \ldots, x_{p/2}), 1 \to (x_{p+2}, \ldots, x_p) \).

By definition

\[
||x_\tau|| = ||x_0|| + ||x_1||
\]

\[
\sum_{|\tau| = i} ||x_\tau|| = ||x||, \quad i = 0, 1, \ldots, t.
\]

**Theorem I:** There is a per bit linear enumeration of a \( p \)-dimensional pyramid cubic lattice with side \( n \) by \( L \)-length binary words, \( L \leq p \log(n/p) + 3p - \log n + (p^2/n) \log e \). The denumeration is per bit linear as well.

The input of the enumeration algorithm is a sequence of binary words \( x_1, x_2, \ldots, x_p \). The words are separated by commas. The output is an \( L \)-length binary word. The running time is \( O(L) \) bit ops. The denumeration produces those words from their \( L \)-length code.
Proof:

1) Enumeration: Take a vector \( x = (x_1, \cdots, x_p) \), \( \|x\| = n \).

We suppose \( p = 2^t, t > 0 \), for simplicity sake. We begin the enumeration by building a \( t \)-level binary tree whose nodes correspond to subvectors of \( x \). Next we label its nodes.

Let \( \sigma \) be a node, \( x_\sigma \) be the corresponding subvector. We label \( \sigma \) by a binary word \( \text{label}(\sigma) \), where

\[
\text{label}(\sigma) = 0^{[n_0 \|x_\sigma\|]} 1^{[n_\sigma \|x_\sigma\|]} \text{Bin}^{\|x_{\sigma 0}\|}.
\]

Here \( 0^k \) means \( k \) consequent zeros. The label of \( \sigma \) is the number \( \|x_\sigma\| \) in binary supplemented by senior zeros up to the length \( \|\text{Bin}^{\|x_\sigma\|}\|_2 \).

Leaves are not labeled. By the definition

\[
\text{label}(\sigma) = \text{Bin}^{\|x_\sigma\|}.
\]

For the total length \( L \) of all labels we have

\[
L = \sum_\sigma |\text{label}(\sigma)|.
\]

We make summation in (10) first over levels, then over nodes of a level

\[
L = \sum_{i=1}^{t-1} \sum_{\sigma, |x_\sigma| = i} |\text{label}(\sigma)|.
\]

By virtue of (9), we rewrite (11) as

\[
L = \sum_{i=1}^{t-1} \sum_{\sigma, |x_\sigma| = i} \|\text{Bin}^{\|x_\sigma\|}\|_2.
\]

We apply inequality (5) to the inner sum in (12), taking into account (7). It yields

\[
L \leq \sum_{i=1}^{t-1} 2^i + 2^i \log \left( 1 + \frac{\|x\|}{2^i} \right).
\]

We need the formula for the sum of a geometric progression

\[
\sum_{i=1}^{t-1} 2^i = 2^t - 1
\]

and its derivative

\[
\sum_{i=1}^{t-1} i 2^i = (t - 2) 2^t + 2.
\]

These formulas and (13) give an upper bound

\[
L \leq p \left( \log \left( 1 + \frac{n}{p} \right) \right) + 3p - \log n, \quad n = \|x\|, \quad p = 2^t.
\]

An inequality \( \ln(1 + x) \leq x \) allows us to rewrite (14) as

\[
L \leq p \log \frac{n}{p} + 3p - \log n + \log e \frac{p^2}{n}.
\]

We build our tree inductively from the \((t-1)\)th level. According to (8), the labels of the \((t-1)\)-th level are \( 0^{[n_0 \|x\|]} 1^{[n_\sigma \|x\|]} \text{Bin}^{\|x_{\sigma 0}\|} \) etc. The computation of labels takes

\[
O\left( \sum_{\sigma, |x_\sigma| = t-1} \|\text{Bin}^{\|x_\sigma\|}\|_2 \right) \text{ bit ops.}
\]

Change the vector \( x = (x_1, \cdots, x_p) \) for the vector \( (x_1 + x_2, x_3 + x_4, \cdots, x_{2^{t-1}} + x_{2^t}) \). Apply to it the same operation. It gives \( t - 2 \)th level of the tree, etc. The total number of the bit ops is estimated by (12) which exceeds \( L \) by a constant factor.

The code of the vector \( x \) is the concatenation of all labels of the tree: the root, then the labels of the first level from left to right, then the labels of the second level in the same order, etc. The length of the code and the time of encoding are as required.

2) Decimation: A vector \( x \) can be reconstructed from its code by a per bit linear algorithm. We separate the coordinates of \( x \) from the code of \( x \) one by one. The number \( \|x\| = n \) is known.

The decoding is done starting from the root. By (9), the length of the label of the root is \( \|\text{label}(0)\| = \|\text{Bin}^{\|x\|}\| = \|\text{Bin}^{n}\| \). Take the initial \( \|\text{Bin}^{n}\| \) letters of the code. Those letters make the word \( \text{label}(0) = x_1 + \cdots + x_{n/2} = \|x_\sigma\| \). By (6), we find

\[
\|x_\sigma\| = x_{1+n/2} + \cdots + x_{n} = n - \text{label}(0).
\]

The computation takes \( O\left( \|\text{Bin}^{\|x\|}\| \right) \) bit ops. Go to the first level. Take the next \( \|\text{Bin}^{\|x\|}\| \) bits of the code. They make \( \text{label}(0) \). The next \( \|\text{Bin}^{\|x\|}\| \) bits give \( \text{label}(1) \). The first level is done. Proceeding this way we find from the code the initial tree and all coordinates of \( x \). The total number of bit ops is

\[
O\left( \sum_{\sigma, |x_\sigma| = i} \|\text{Bin}^{\|x_\sigma\|}\|_2 \right) = O(L).
\]

Consider an example. Let \( x = (4, 25, 8, 17, 3, 14, 29, 20) \). The corresponding tree with eight leaves is shown in Fig. 1. The leaves correspond to coordinates of \( x \). By (8), the label of the root is

\[
0^{[n_0 \|x\|]} 1^{[n_\sigma \|x\|]} \text{Bin}^{\|x_{\sigma 0}\|}.
\]

The label of the left son of the root is

\[
0^{[n_0 \|x\|]} 1^{[n_\sigma \|x\|]} \text{Bin}^{\|x_{\sigma 0}\|}.
\]

the label of the right son is

\[
0^{[n_0 \|x\|]} 1^{[n_\sigma \|x\|]} \text{Bin}^{\|x_{\sigma 0}\|}
\]

and so on.

We have

\[
\|\text{Bin}^{\|x\|}\| = \text{Bin}(4 + 25 + 8 + 17 + 3 + 14 + 29 + 20 = 120) = 11110000
\]

\[
\|\text{Bin}^{\|x\|}\| = \text{Bin}(4 + 25 + 8 + 17 = 54) = 110110
\]

\[
\|\text{Bin}^{\|x\|}\| = \text{Bin}(3 + 14 + 29 + 20 = 66) = 1000010;
\]

\[
\text{label}(0) = 0^3 \text{Bin}^{\|x\|}, \quad \text{label}(1) = 0^3 \text{Bin}^{\|x\|}
\]

\[
\text{label}(0) = 0^3 \text{Bin}^{\|x\|}, \quad \text{label}(1) = 0^3 \text{Bin}^{\|x\|}
\]

\[
\text{label}(0) = 0^3 \text{Bin}^{\|x\|}, \quad \text{label}(1) = 0^3 \text{Bin}^{\|x\|}
\]

\[
\text{label}(0) = 0^3 \text{Bin}^{\|x\|}, \quad \text{label}(1) = 0^3 \text{Bin}^{\|x\|}
\]

and so on. The labeled tree is shown on Fig. 2. The code of \( x \) is

\[
01101101110100100010010000000110111101.
\]

The length of the code of \( x \) is 41.
To decode we take $|\text{Bin}| = 120$ letters of the code. We get 0110110. It is $||x_0||; |\text{Bin}| ||x_0|| = 11110000 - 0110110 = 0000010; |\text{Bin}| ||x_0|| = 7$. Take the next $|\text{Bin}| ||x_0|| = 6$ letter, which is 011101. Hence, $|x_2||; |\text{Bin}| ||x_2|| = 29$ and $|\text{Bin}| ||x_2|| = 5; \|x_0|| = \|x_0|| - \|x_0|| = 54 - 29 = 25$ and $|\text{Bin}| ||x_0|| = 5$. Take the next $|\text{Bin}| ||x_2|| = 7$ letters, which are 0010001. Hence, $|x_2||; |\text{Bin}| ||x_2|| = 10001 = 17$ and $|\text{Bin}| ||x_2|| = 5$. Finally, $|x_1|| = \|x_1|| - \|x_0|| = 66 - 17 = 49$ and $|\text{Bin}| ||x_1|| = 6$. Taking the next 5, 5, and 6 letters of code we obtain 00100 = 8, 00110 = 3, and 011101 = 29. Subtracting 29 - 4, 25 - 8, 17 - 3, 49 - 29 we get initial sequence: 4, 25, 8, 17, 3, 14, 29, 20.

**Corollary 1:** The per dimensional redundancy of the above encoding of a surface pyramid cubic lattice asymptotically does not exceed $3 - \log_2 e \approx 1.56$ bits, as $p/n \to 0$, i.e.,

$$\limsup_{p/n \to 0} R/p \leq 1.56.$$  \hspace{1cm} (16)

**Proof:** The cardinality $|\tilde{S}(n, p)|$ of a surface pyramid cubic lattice is

$$|\tilde{S}(n, p)| = \binom{n + p - 1}{p - 1}.$$  \hspace{1cm} (17)

If $p/n \to 0$, $n \to \infty$, $p \to \infty$, then, by the Stirling formula we have from (16)

$$\log |\tilde{S}(n, p)| \approx p \log \frac{n + p}{p} + p \log e + o(p).$$  \hspace{1cm} (18)

Now Corollary 1 follows from (17) and the theorem.

**Corollary 2:** There is a per bit linear encoding of rare events whose per event redundancy asymptotically does not exceed $3 - \log_2 e \approx 1.56$ bits as $p/n \to 0$, i.e.,

$$\limsup_{p/n \to 0} R/p \leq 1.56.$$  \hspace{1cm} (19)

**Proof:** Take a word $x \in S(n, p)$. It consists of runs of zeros separated by ones. Let the lengths of those runs be $x_1, \ldots, x_{p+1}$.

$$x_i \geq 0$$

$$x_1 + \cdots + x_{p+1} = n - p.$$  \hspace{1cm} (20)

We conclude from (18) that the vector $(x_1, \ldots, x_{p+1})$ belongs to a $p + 1$-dimensional pyramid cubic lattice with the side $n - p$. Since $p/n \to 0$, the log-cardinality of this lattice is given by (17). Now Corollary 2 follows from (17) and the theorem.

**References**


