# Balanced Runlength Limited Codes Using Knuth's Algorithm

Kees A. Schouhamer Immink
Turing Machines Inc.
Willemskade 15b-d, 3016 DK Rotterdam
The Netherlands
Nanyang Technological University, Singapore
Email: immink@turing-machines.com

Jos H. Weber
TU Delft, IRCTR/CWPC,
Mekelweg 4, 2628 CD Delft
The Netherlands
Email: J.H.Weber@ewi.tudelft.nl

Hendrik C. Ferreira
University of Johannesburg,
Dept. E&E Eng. Science
South Africa
Email: hcferreira@uj.ac.za

***Abstract*—Knuth published a very simple algorithm for constructing bipolar codewords with equal numbers of +1's and -1's, called balanced codes. In our paper we will present new code constructions that generate balanced runlength limited sequences using a modification of Knuth's algorithm.**

## I. Introduction

A bipolar $dk$ runlength-limited (RLL) sequence is a sequence whose runlengths lie between $d+1$ and $k+1$, where the runlength is known as the number of consecutive identical symbols occurring in a sequence. Such sequences have been mainly used in conventional magnetic recording systems [1]. Dc-free bipolar RLL sequences have found widespread application in popular optical recording devices such as Compact Disc, DVD and, Blu-Ray [1]. Dc-free sequences have the properties that their power spectral density (PSD) vanishes at the zero frequency and that there is a region of frequencies close to the zero frequency in which the PSD is low. A sequence is 'dc-free' if the running unbalance, that is, the difference between the numbers of transmitted +1's and -1's is at any moment limited [1]. Dc-free runlength-limited codes used in optical storage devices are relatively short, and the encoder may choose between codewords with opposing unbalance aiming to minimize the absolute value of the running unbalance.

A codeword of bipolar symbols is said to be balanced when the numbers of +1's and -1's in that word are equal, that is, the sum of all symbols in the codeword equals zero. Note that the concatenation of balanced codewords is 'dc-free' as the unbalance of the concatenated sequence is limited. The efficient translation of arbitrary data into balanced $dk$ RLL codewords using enumerative coding has been demonstrated by Braun & Immink [2]. Enumerative coding techniques make it possible to very efficiently translate user words into codewords and vice versa by invoking an algorithmic procedure rather than performing the translation with look-up tables. A first drawback of enumerative coding is that it requires look-up tables for coefficient storage that become prohibitively large with mounting codeword size. A second drawback is massive error propagation, where a single transmission error may result in large bursts of decoding errors. It is a desideratum to construct balanced $dk$ runlength-limited codes with a simple construction where mounting size of hardware and massive error propagation are avoided.

To that end, we will study the generation of balanced RLL sequences using a modification of Knuth's algorithm. Knuth's algorithm for constructing balanced codes [3] is very well suited for use with long codewords, since look-up tables are completely absent. Modifications and improvements of the generic Knuth scheme are discussed by Alon *et al.* [4], Al-Bassam & Bose [5], Tallini, Capocelli & Bose [6], Weber & Immink [7], and Immink & Weber [8].

The basic idea of our new constructions is very simple. It is assumed that the RLL sequence is generated by a suitable prior art method. A judiciously chosen $q$-bit buffering runlength-limited sequence, called *interfix*, is inserted at a judiciously chosen position within the given runlength limited word. The index of the position is conveyed by a $p$-bit runlength limited sequence, called *prefix*, which is sent to the receiver. Given the received prefix, the receiver can decode the index, remove the interfix at the index found, and recover the original RLL input word, which, in turn, can be decoded under the rules of the prior art code. The redundancy of the construction is simply $p+q$.

In Sections II and III, we start with some basic notational conventions, and a description of Knuth's algorithm. In Section IV, we describe four code constructions, and compare their redundancy. Finally, the results of this paper are discussed in Section V.

## II. Basics

Let $\boldsymbol{x} = (x_1, \ldots, x_m)$ be a word of $m$ binary symbols, $x_i \in \{0,1\}$. The word $\boldsymbol{x}$ is translated into $\boldsymbol{w} = (w_1, \ldots, w_m)$ of bipolar symbols $w_i \in \{-1,1\}$ by modulo 2 integration plus a conversion step to make $w_i \in \{-1,1\}$,

This project was supported by grant *Theory and Practice of Coding and Cryptography*, Award Number: NRF-CRP2-2007-03

that is,

$$\begin{aligned} w'_i &= w'_{i-1} \oplus x_i, \\ w_i &= 2w'_i - 1,\ 1 \le i \le m, \end{aligned} \tag{1}$$

where $w'_0 = 1$ and the $\oplus$ sign indicates modulo two summation. The above operation will be denoted by $\boldsymbol{w} = I(\boldsymbol{x})$. The logical '1's in the sequence $\boldsymbol{x}$ indicate the positions of a transition 1 $\rightarrow$ -1 or -1 $\rightarrow$ 1 of the corresponding sequence $\boldsymbol{w} = I(\boldsymbol{x})$. The original word $\boldsymbol{x}$ can be uniquely restored by a 'differentiation' operation defined by $x_i = (-w_{i-1}w_i+1)/2$, $1 \le i \le m$. The differentiation operation will be denoted by $\boldsymbol{x} = I^{-1}(\boldsymbol{w})$.

The *unbalance* $S(\boldsymbol{x})$ of $\boldsymbol{x}$ is defined as the unbalance of its integrated version $\boldsymbol{w} = I(\boldsymbol{x})$, that is, $S(\boldsymbol{x}) = \sum w_i$.

A word $\boldsymbol{x}$ is said to be a $dk$-limited sequence if the number of 0's between consecutive 1's lies between $d \ge 0$ and $k > d$. Note that the 'integrated' version $\boldsymbol{w} = I(\boldsymbol{x})$ is a $dk$ runlength limited sequence whose runlengths lie between $d+1$ and $k+1$. In this paper we will study properties and constructions for generating balanced runlength limited sequences using Knuth's algorithm as a basic vehicle. In the next section, we will briefly describe Knuth's algorithm for balancing unconstrained bipolar words.

## III. Knuth's code construction

The conventional Knuth algorithm runs as follows. The user data is arranged as a bipolar $m$-tuple $\boldsymbol{w} = (w_1, \ldots, w_m)$, $w_i \in \{-1, 1\}$, $m$ even. (Knuth also presented code constructions for odd $m$, but they will not be discussed here.) We define for $1 \le j \le m$, the bipolar $m$-tuple $\boldsymbol{w}^{[j]} = (w_1, w_2, \ldots, w_j, -w_{j+1}, -w_{j+2}, \ldots, -w_m)$. Knuth showed that for any user data $\boldsymbol{w}$ an index $v$ can be found such that the codeword $\boldsymbol{w}^{[v]}$ is balanced, that is

$$\sum_{i=1}^{v} w_i - \sum_{i=v+1}^{m} w_i = 0.$$

The balanced codeword $\boldsymbol{w}^{[v]}$ plus a balanced $p$-bit prefix, which suitably represents the index, is transmitted. Note that in case $\boldsymbol{w} = I(\boldsymbol{x})$, it suffices to invert a single bit, $x_{v+1}$, to get a balanced sequence $I(\boldsymbol{x}')$, that is, $\boldsymbol{x}' = \boldsymbol{x}$ except $x'_{v+1} = x_{v+1} \oplus 1$. For an efficient code, the redundant prefix should be as small as possible. Knuth showed that in his best construction the redundancy $p$ is roughly equal to [3]

$$\log_2 m, \quad m >> 1. \tag{2}$$

In the next section, we will show how we can use Knuth's algorithm for generating balanced RLL sequences.

## IV. Balanced RLL Codes

We assume that a user word is encoded into an $m$-bit $dk$-constrained word, $\boldsymbol{x}$, by a suitable prior art coding method. Using Knuth's algorithm we find the index $v$ such that $I(\boldsymbol{x})^{[v]}$ is balanced. In case $\boldsymbol{x}$ is a $dk$-limited word, the inversion of bit $x_{v+1}$, as described in the previous section, may lead to a violation of the $dk$ constraints. In order to solve this problem, we assume in the constructions described below, that a properly chosen $q$-bit *interfix* (buffering) word, $q \ge d+1$, is inserted at the balancing position $v$ so that balancing can be obtained without violating the $dk$ constraint. We further assume that the $p$-bit prefix that carries the (balancing) index $v$, is a $dk$ constrained word that can be cascaded with the data word. The redundancy of the code constructions described in the next section is thus $p+q$.

In order to satisfy the $dk$ constraints, we insert a $q$-bit interfix (buffering) word $\boldsymbol{i} = (i_1, \ldots, i_q)$ between the leading $(x_1, \ldots, x_v)$ and trailing segment $(x_{v+1}, \ldots, x_m)$ of $\boldsymbol{x}$. The $(m+q)$-bit word $\boldsymbol{u}$ is defined by $\boldsymbol{u} = (x_1, \ldots, x_v, i_1, \ldots, i_q, x_{v+1}, \ldots, x_m)$. If the interfix $\boldsymbol{i}$ has an odd number of 1's, it is immediate from the definition of $v$ that

$$|S(\boldsymbol{u})| = |S(\boldsymbol{i})|.$$

We will present various worked code constructions. The basic encoding procedure of the new constructions consists of six steps which are described below. The input to the encoder is a string of binary data whose length is irrelevant here.

1) Encode the string of input data into an $m$-bit $dk$-constrained sequence, $\boldsymbol{x}$, using a prior art encoder.
2) Integrate $\boldsymbol{x}$ and obtain the $dk$-limited RLL codeword $I(\boldsymbol{x})$.
3) Find a balancing index $v$ for the $dk$-limited RLL codeword $I(\boldsymbol{x})$, where $1 \le v \le m$.
4) Insert a $q$-bit $dk$-limited interfix $\boldsymbol{i}$ at position $v$ of the sequence $\boldsymbol{x}$. A proper interfix must satisfy various conditions that will be discussed later.
5) Encode the number $v$ into a unique $dk$-limited $p$-bit prefix, and append the $p$-bit prefix at the end of the $(q+m)$-bit word. Note that since the 'prefix' is appended at the rear it is actually a suffix.
6) Integrate and transmit the $(m+p+q)$-bit balanced RLL sequence including prefix and interfix to the receiver.

Figure 1 shows a diagram of the various steps needed in the constructions.

We will start our exposition with Constructions 1 and 2 which are intended for $d$-limited sequences, while Construction 3 deals with the $dk$-limited case, $k \ge 2d$, and Construction 4 with the $k$-limited case ($d = 0$). Construction 1 is characterized by the insertion of a balanced interfix of length $2(d+1)$, while Construction 2 has an interfix of minimum length $d+1$, where a possible unbalance of the interfix will be compensated by the complementary unbalance of the prefix. For Constructions 1 and 2, we assume that the $p$-bit prefix starts with al least $d$ 0's so that cascading is simple.

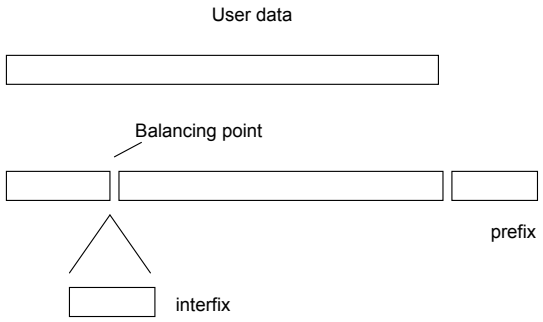


Fig. 1. Diagram of the various steps needed in the constructions.

*Construction 1:* Let $q = 2(d+1)$, and set the $2(d+1)$-bit interfix $\boldsymbol{i}$ = $(0^{d+1}10^d)$, where $0^s$ denotes a string of $s$ 0's. The $2(d+1)$-bit interfix is inserted at the index $v$ found by Knuth's balancing algorithm (see above). The interfix is balanced, i.e. $S(\boldsymbol{i}) = 0$, and it has an odd number of 1's, thus we simply conclude that the prescribed $d$ constraint can be preserved and that the word $I(\boldsymbol{u})$ is balanced.

Let $N_d(n, a)$ denote the number of $d$-limited words of length $n$ that start with at least $d$ 0's, whose unbalance equals $a$. The prefix must uniquely assign the balancing index within the $m$-bit codeword $\boldsymbol{x}$, so that we must choose the prefix length $p$, $p$ even, such that the number of available prefixes is at least $m$, or, in other words, $N_1(p) = N_d(p, 0) \geq m$, where $N_1(p)$ denotes the number of unique prefixes of length $p$ for Construction 1.

*Construction 2:* Let $q = d+1$. Select the $(d+1)$-bit interfix $\boldsymbol{i}$ = $(0^c10^{d-c})$, where the integer $c$ is chosen such that the $d$ constraint in $\boldsymbol{u}$ is preserved. Let $s$ denote the number of trailing 0's of the first segment of $\boldsymbol{x}$, $(x_1, \ldots, x_v)$, then set $c = \max(0, d-s)$. The unbalance of the interfix (and since the number of 1's in $\boldsymbol{i}$ is odd, it also holds for the entire word $I(\boldsymbol{u})$), equals $-q$, $-q+2, \ldots$, $q$. The encoder selects a suitable prefix from a list of $(q+1)$ available prefixes of unbalance $-q$, $-q+2, \ldots$, $q$ in such a way that the prefix unbalance will compensate the unbalance of $I(\boldsymbol{u})$. The size of the list of available prefixes, denoted by $N_2(p)$, of length $p$ is $N_2(p) = \min(N_d(p, -q), N_d(p, -q+2), \ldots, N_d(p, q))$. We choose the prefix length $p$, $p+q$ even, such that $N_2(p) \geq m$. As a result, we can always find a prefix that 1) can uniquely assign the index within a codeword and 2) can balance the interfix and thus the output word.

Tables I and II show outcomes of computations. Table I shows $N_1(r-2(d+1))$ and $N_2(r-d-1)$ versus redundancy $r = p+q$ with $d = 1$, while Table II shows the same quantities for $d = 2$. Tables I and II reveal that the relative redundancy of a code constructed by Construction 2 is a factor of 2-3 lower than that of Construction 1.

TABLE I
NUMBER OF PREFIXES $N_1(r-2(d+1))$ AND $N_2(r-d-1)$ VERSUS REDUNDANCY $r$ WITH $d = 1$ USING CONSTRUCTIONS 1 AND 2.

| $r$ | $N_1$ | $N_2$ |
|---|---|---|
| 12 | 9 | 15 |
| 16 | 50 | 96 |
| 20 | 296 | 611 |
| 24 | 1812 | 3913 |
| 28 | 11328 | 25224 |

TABLE II
NUMBER OF PREFIXES $N_1(r-2(d+1))$ AND $N_2(r-d-1)$ VERSUS REDUNDANCY $r$ WITH $d = 2$ USING CONSTRUCTIONS 1 AND 2.

| $r$ | $N_1$ | $N_2$ |
|---|---|---|
| 16 | 6 | 9 |
| 20 | 24 | 42 |
| 24 | 97 | 196 |
| 28 | 406 | 881 |
| 32 | 1723 | 3955 |

*Example:* We have worked out a simple example for illustrating Construction 2, where $d = 1$ and $m = 10$. We find $q = d+1 = 2$ and the smallest prefix that can accommodate a length $m = 10$ equals $p = 10$, see Table I. The total redundancy is $p+q = 12$. Note that for reasons of exposition we have opted for a relatively small $m$ and as a result the overhead in this example is very high. Let the input word be $\boldsymbol{x}$ = '0000000101'. We find $I(\boldsymbol{x}) = {}'++++++++--+'$. The balancing position is $v = 3$. We stuff the 2-bit interfix at position $v = 3$ and obtain '000 10 0000101' (the spaces are for clarity). The unbalance of this 12-bit word equals -2. The prefix can be taken from Table III from entry $v = 3$ and unbalance=-2, and we find '0010000010'. Note that we choose a prefix with unbalance -2 (and not +2) since $\boldsymbol{u}$ has an odd number of 1's, that is $I(\boldsymbol{u})$ ends with a -1. The total 22-bit word is '000 10 0000101 0010000010'. After integration we obtain

$$'++-------++---++++++--',$$

TABLE III
TABLE OF PREFIXES, $d = 1$ AND $p = 10$.

| $v$ | unbalance=-2 | unbalance=0 | unbalance=+2 |
|---|---|---|---|
| 1 | 0000100000 | 0000010000 | 0000001000 |
| 2 | 0001000001 | 0000100001 | 0000010001 |
| 3 | 0010000010 | 0001000010 | 0000100010 |
| 4 | 0010000101 | 0001000101 | 0000100101 |
| 5 | 0010001010 | 0001001010 | 0000101010 |
| 6 | 0010010100 | 0001010100 | 0001000100 |
| 7 | 0010101000 | 0010000100 | 0001001001 |
| 8 | 0100000100 | 0010001001 | 0001010010 |
| 9 | 0100001001 | 0010010010 | 0001010101 |
| 10 | 0100010010 | 0010010101 | 0010001000 |

TABLE IV
NUMBER OF PREFIXES, $N_3(r-d-1)$, VERSUS REDUNDANCY $r$ WITH $d=1$ AND $k=3$.

| $r$ | $N_3$ |
|---|---|
| 20 | 71 |
| 24 | 341 |
| 28 | 1575 |
| 32 | 7361 |

TABLE V
NUMBER OF PREFIXES, $N_4(r-1)$, VERSUS REDUNDANCY $r$ FOR $k=3$.

| $r$ | $N_4$ |
|---|---|
| 12 | 185 |
| 14 | 662 |
| 16 | 2372 |
| 18 | 8510 |

which, as we can verify, is balanced and satisfies the $dk$ constraints. ■

Construction 3, described below, deals with the general $dk$-limited case, $k \geq 2d$.

*Construction 3:* Let $q = d+1$. The interfix is selected in the same way as in Construction 2. In order to be able to cascade the prefix with $\boldsymbol{u}$, we select a prefix such that it starts with the string $10^{c_1}1$, where $d \leq c_1 \leq k-d$. Let $N^0_{dk}(n,a)$ denote the number of $dk$-limited sequences of length $n$ that start with the string $10^{c_1}1$, $d \leq c_1 \leq k-d$, and whose unbalance equals $a$. In case $\boldsymbol{u}$ ends with a string of $s < d$ 0's we invert the first bit of the prefix from '1' into '0' so that the $d$ and $k$ constraints are preserved. Note that the inversion of the first bit of the prefix changes the polarity of prefix unbalance. The unbalance of the interfix (and thus the entire word) equals, as we can easily verify, $-q$, $-q+2$, $\ldots, q$. We choose the prefix length $p$, $p+q$ even, such that $N_3(p) = \min(N^0_{dk}(p,-q), N^0_{dk}(p,-q+2), \ldots, N^0_{dk}(p,q)) \geq m$, where the quantity $N_3(p)$ denotes the number of available prefixes of length $p$ for Construction 3. Table IV shows $N_3(r-q)$ versus redundancy $r$ for $d=1$ and $k=3$. Construction 4, for the $k$-limited case ($d=0$), is a special case of Construction 3.

*Construction 4:* Let $d=0$ and let $\boldsymbol{x}$ be a $k$-limited word of length $m$, then the 1-bit interfix is set to '1'. A prefix starts with a '1'. Let $N^1_{dk}(n,a)$ denote the number of $k$-limited sequences of length $n$ that start with a '1', and whose unbalance equals $a$. The unbalance of the interfix equals $\mp 1$. We choose the prefix length $p$, $p+q$ even, such that $N_4(p) = \min(N^1_{dk}(p,-1),\ N^1_{dk}(p,+1)) \geq m$, where the quantity $N_4(p)$ denotes the number of available prefixes of length $p$ for Construction 4. Table V shows $N_4(r-1)$ versus redundancy $r$ for $k=3$.

## V. CONCLUSIONS

We have studied the construction of balanced runlength limited codes that are based on a modification of Knuth's algorithm. We have presented various code constructions, where a judiciously chosen $q$-bit buffering runlength-limited sequence, called interfix, is inserted at a judiciously chosen position, $v$, within the input runlength-limited word. The position $v$ is conveyed by a $p$-bit prefix that satisfies the given runlength constraints. Given the prefix, the receiver can locate and remove the interfix, and recover the input word. The redundancy of the new construction equals $p+q$.

## REFERENCES


[1] K.A.S. Immink, *Codes for Mass Data Storage Systems,* Second Edition, ISBN 90-74249-27-2, Shannon Foundation Publishers, Eindhoven, Netherlands, 2004.

[2] V. Braun and K.A.S. Immink, 'An Enumerative Coding Technique for DC-free Runlength-Limited Sequences', *IEEE Trans on Communications,*vol. 48, no. 12, pp. 2024-2031, Dec. 2000.

[3] D.E. Knuth, 'Efficient Balanced Codes', *IEEE Trans. Inform. Theory,* vol. IT-32, no. 1, pp. 51-53, Jan. 1986.

[4] N. Alon, E.E. Bergmann, D. Coppersmith, and A.M. Odlyzko, 'Balancing Sets of Vectors', *IEEE Trans. Inform. Theory,* vol. IT-34, no. 1, pp. 128-130, Jan. 1988.

[5] S. Al-Bassam and B. Bose, 'On Balanced Codes', *IEEE Trans. Inform. Theory,* vol. IT-36, no. 2, pp. 406-408, March 1990.

[6] L.G. Tallini, R.M. Capocelli, and B. Bose, 'Design of some New Balanced Codes', *IEEE Trans. Inform. Theory,* vol. IT-42, pp. 790-802, May 1996.

[7] J.H. Weber and K.A.S. Immink, 'Knuth's Balancing of Codewords Revisited', *IEEE Trans Inform Theory,* vol. 56, no. 4, pp.1673-1679, 2010.

[8] K.A.S. Immink and J. Weber, 'Very Efficient Balancing of Codewords', *IEEE Journal on Selected Areas of Communications,* vol. 28, pp. 188-192, 2010.