Constant Weight Codes: An Approach Based on Knuth's Balancing Method

Vitaly Skachek and Kees A. Schouhamer Immink

***Abstract*—In this article, we study properties and algorithms for constructing sets of constant weight codewords with bipolar symbols, where the sum of the symbols is a constant q, $q \neq 0$. We show various code constructions that extend Knuth's balancing vector scheme, $q = 0$, to the case where $q > 0$. We compute the redundancy of the new coding methods. Finally, we generalize the proposed methods to encoding of imbalanced arrays in two or more dimensions.**

***Index Terms*—Balanced code, channel capacity, constant-weight code, constrained code, magnetic recording, optical recording.**

I. Introduction

Let q be an integer. A set $\mathcal{C}$, which is a subset of

$$\left\{ \boldsymbol{w} = (w_1, w_2, \ldots, w_n) \in \{-1, +1\}^n \ : \ \sum_{i=1}^{n} w_i = q \right\} ,$$

is called a *constant weight code* of length n. If $q = 0$, the code $\mathcal{C}$ is called a *balanced code*. A vector $\boldsymbol{w} \in \mathcal{C}$ is called a *code word*. The *weight* of a word equals the number of '$+1$'s in it. An *encoder* for the code $\mathcal{C}$ is an invertible mapping from the set of all vectors $\boldsymbol{w} \in \{-1, +1\}^m$ onto the vectors in $\mathcal{C} \subseteq \{-1, +1\}^n$, for some integer $m \geq 1$. The *redundancy* of the encoder is $n-m$. For practical purposes, we are interested in encoders which are simple and computationally efficient, and whose inverse, *decoder*, is also computationally efficient.

Balanced codes have found application in cable transmission, optical and magnetic recording. A survey of properties and methods for constructing balanced codes can be found in [9]. A simple encoding technique for generating balanced codewords, which is capable of handling (very) large blocks was described by Knuth [11] in 1986.

For a general value of q, constant weight codes are useful in a variety of applications such as data storage, fault-tolerant circuit design and computing, pattern generation for circuit testing, identification coding, and optical overlay networks. For thoroughful discussion about various applications of constant weight codes, see [17] and the references therein.

This work was supported by the National Research Foundation of Singapore under the grant *Theory and Practice of Coding and Cryptography*, award number: NRF-CRP2-2007-03. It was presented in part at the *IEEE International Symposium on Information Theory 2011*, St. Petersburg, Russia.

Vitaly Skachek is with the Institute of Computer Science, University of Tartu, J. Liivi 2-216, Tartu 50409, Estonia. E-mail: vitaly.skachek@gmail.com.

Kees A. Schouhamer Immink is with Turing Machines Inc, Willemskade 15b-d, 3016 DK Rotterdam, The Netherlands. E-mail: immink@turing-machines.com.

This work was done in part while the authors were with the School of Physical and Mathematical Sciences, Nanyang Technological University, 21 Nanyang Link, Singapore 637371.

For example, the use of constant weight codes in off-chip signaling allows for minimizing the power-supply current changes. These current fluctuations cause a voltage drop between the chip and circuit board, and lead to significant system noise [16], [17]. Constant weight codes are useful in detection of unidirectional errors in memories and systems-on-chip [3], [23, pp. 159–160]. They can be employed in delay-insensitive communication in asynchronous systems [4], [22].

Constant weight codes were recently proposed for use with rank modulation in NAND flash memory devices [7]. They were also shown to be efficient in coping with electrical charge leakage in the cells, when used with dynamic reading thresholds [24], [25].

Efficient encoding methods for constant weight codes, suitable for use in VLSI circuits, were studied in [17]. That paper focuses primarily on hardware implementation of such encoders. In particular, the authors propose a parallel implementation, which can be efficiently implemented on a single chip. In the implementation proposed in [17], by building on the idea proposed in [2], a set of *balancing functions* is applied in parallel to the input data. If a sufficiently large set of balancing functions is used, then at least one function returns the desired imbalanced output.

In contrast with [17], we consider sequential encoding algorithms. It should be noticed that some of the methods proposed in our work can be efficiently implemented in a parallel manner, while the others can not. Generally, we leave the question of parallelization of the discussed algorithms outside of the scope of this paper.

As of today, a simple sequential encoding algorithm for producing a word with a prescribed *imbalance* $q \neq 0$ is not known. The publications dealing with constructions of unbalanced codes that the authors are aware of are the geometric approach by Tian *et al.* [20] and the enumerative method by Schalkwijk [14] (see also [5]). Below, we briefly discuss these two methods.

An interesting geometric encoding method proposed in [20] uses a w-dimensional Euclidian space representation of the codewords, where $\mathsf{w} = (n+q)/2$ is the weight of the encoded word. The encoding is done by dissectioning a polytope in that space. The time complexity of the proposed method is $\Theta(\mathsf{w}^2)$, and if $\mathsf{w} = \Theta(n)$, then the complexity of this method behaves as $\Theta(n^2)$. Moreover, the rate of the produced code is not optimal for the targeted small values of the weight w. As the authors of [20] mention, for the targeted regime, the produced redundancy is proportional to n.

By contrast, the redundancy produced by Schalkwijk's enumeration method [14] is essentially optimal. However, the

enumeration method has some drawbacks. First, it requires large registers. Second, it suffers from error propagation, namely even a small number of corrupted bits leads to a massive destruction of data when the decoding algorithm is applied. Third, this method has higher complexity. For example, if the enumerative algorithm is implemented in a rather straightforward manner, the resulting complexity scales at least $\Omega(\mathsf{w}^2)$ operations over integers. If q is small, the complexity of this method is too high. It is worth mentioning that Ryabko [13] has reduced the complexity of enumerative algorithms to $O(n(\log n)^c)$ bit operations, where $c \geq 2$ is some fixed constant.

In this article we study various simple and efficient methods of generating bipolar codewords of imbalance q, $q \neq 0$. These methods can be readily implemented in hardware, where massive error propagation can be avoided. Without loss of generality we assume $q > 0$, since, for $q < 0$, codewords can simply be obtained by inverting all m symbols of a codeword with $q > 0$. We mainly target a regime where q is small compared to n. In that range of q, we aim at developing encoding algorithms, which produce rather small redundancy. Observe however, that for the case $q = 0$, the original Knuth's algorithm produces redundancy which is suboptimal. Therefore, in our case for $q > 0$, we aim at developing methods which produce small additional redundancy relative to what Knuth's algorithm produces.

The paper is structured as follows. In Section II we revisit Knuth's algorithm. In Section III we present several encoding schemes for codes with imbalance $q \neq 0$. In Section IV, we present the encoding scheme for two-dimensional imbalanced arrays. In Section V, we summarize the results in the paper.

II. Background

Let the word be $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)$, $w_i \in \{-1, +1\}$, and let q be the imbalance of the user symbols defined by

$$q = \sum_{i=1}^{n} w_i. \tag{1}$$

The cardinality of the set of all q-balanced words of length n, n and q even, can be approximated by [15]

$$\binom{n}{\frac{n+q}{2}} \approx \frac{2^n}{\sqrt{\frac{n\pi}{2}}} e^{-\frac{q^2}{2n}}, \quad n >> 1.$$

Then the redundancy of a full set of q-balanced codewords is

$$\frac{1}{2}\log_2 n + \frac{q^2}{2n}\log_2 e + 0.326, \quad n >> 1. \tag{2}$$

We notice that the redundancy consists of two terms, namely the term $\frac{1}{2}\log_2 n + 0.326$, the redundancy of the set of balanced words, and the q-dependent term $\frac{q^2}{2n}\log_2 e$. Observe that the q-dependent term decreases with increasing n.

Knuth's encoding scheme

Knuth published an algorithm for generating sets of bipolar codewords with equal numbers of '+1's and '−1's [11]. In the simplest embodiment of Knuth's method, an m-bit user word, m even, which consists of bipolar symbols valued '± 1', is forwarded to the encoder. The encoder inverts the first k bits of the user word, where k is chosen in such a way that the modified word has equal numbers of '+1's and '−1's. Such an index k can always be found. The integer k is represented by a balanced (prefix) word $\boldsymbol{u}$ of length p. The p-bit prefix word followed by the modified m-bit user word are both transmitted, so that the rate of the code is $m/(m+p)$. The receiver can easily undo the inversion of the first k bits received by decoding the prefix. Both encoder and decoder do not require large look-up tables, and Knuth's algorithm is therefore very attractive for constructing long balanced codewords. Knuth showed that the redundancy p is roughly equal to $\log_2 m$, for $m >> 1$ [11], [10].

Modifications of the generic scheme are discussed in Knuth [11], Alon *et al.* [2], Al-Bassam & Bose [1], Tallini, Capocelli & Bose [18], and Weber & Immink [10].

We will now try to answer the question of how to construct a constant weight code with codewords with unbalance $q \neq 0$.

III. Methods for constructing constant weight codes

A. General setup

Let m be an even number. In the sequel, for any $\boldsymbol{w} = (w_1, w_2, \ldots, w_m)$, $w_i \in \{-1, +1\}$, we use the notation

$$d(\boldsymbol{w}) \triangleq \sum_{i=1}^{m} w_i \tag{3}$$

to denote the imbalance of $\boldsymbol{w}$. We also denote by $\boldsymbol{w}^{(k)}$ the word $\boldsymbol{w}$ with its first k bits inverted, and by $\sigma_k(\boldsymbol{w})$ we denote $d(\boldsymbol{w}^{(k)})$.

Now, assume that $\boldsymbol{w} = (w_1, w_2, \ldots, w_m)$ is the bipolar input word. For convenience, we denote $q' \triangleq d(\boldsymbol{w})$. Then, we have

$$\sigma_k(\boldsymbol{w}) = -2\sum_{i=1}^{k} w_i + q' \ . \tag{4}$$

By far the simplest method for generating q-balanced codewords first generates a balanced codeword by using Knuth's method, and appends q '+1' symbols to the balanced codeword. The method is attractive as it is simple to implement, and as it avoids mass error propagation. The redundancy of this scheme is $p + q$, where $p \approx \log_2 m$ is the redundancy of Knuth's base scheme. From (2) it follows that this simple method is far from optimal since the additional redundancy (relative to the redundancy produced by Knuth's algorithm) does not decrease with increasing m.

Alternative methods, to be discussed below, first use a slightly modified version of Knuth's algorithm for generating q-balanced words. These methods may fail to generate such words and, in a second step, the 'failed' words are modified. Information regarding this modification made is, as in Knuth's scheme, carried by the prefix.

Assume that we apply Knuth's algorithm to $\boldsymbol{w}$ to generate a q-balanced codeword, $0 \leq q \leq m$. That is, we scan $\boldsymbol{w}$ and seek an index k, $0 \leq k \leq m$, such that $\sigma_k(\boldsymbol{w}) = q$. As shown by Knuth, such an index k can always be found for

$q = 0$. For $q \neq 0$, however, the encoder may fail to find such an index k. Since, from the above, $\sigma_0(\boldsymbol{w}) = q'$ (no symbols inverted) and $\sigma_m(\boldsymbol{w}) = -q'$ (all m symbols inverted), we conclude that words $\boldsymbol{w}$ with

$$\sigma_k(\boldsymbol{w}) < q \ , \quad \text{for all } 0 \leq k \leq m \ , \tag{5}$$

cannot be translated into a q-balanced codeword (in particular, for such words we have $|q'| < q$). User words satisfying (5) are called *delinquent* words. Evidently, delinquent words cannot be encoded by Knuth's algorithm, and thus some alternative method has to be found. In the sequel, we present various code constructions that encode delinquent words.

B. Unbalanced prefix

It is assumed that the index k in Knuth's algorithm is represented by a p-bit balanced prefix $\boldsymbol{u}$, for p even. Delinquent words can be balanced by the introduction of prefixes of imbalance q_p, $2 \leq q_p \leq p$. If the imbalance of a delinquent word is negative, i.e., $q' < 0$, then we invert all symbols of that user word. A delinquent word or its inverted version may be balanced by choosing a prefix with imbalance $q_p = q - |q'|$.

An inversion made can be signalled by a unique prefix, thus requiring two distinct prefixes of total imbalance q_p, $2 \leq q_p < q$. For $q_p = q$, since $q' = 0$, we only need one prefix, the all-ones prefix. The unbalanced prefix is an indication to the decoding side that the user word $\boldsymbol{w}$ was not modified. Thus, for the case $q \leq p$, it is quite straightforward to generate a codeword having imbalance q by invoking a simple modification of Knuth's method. The redundancy of this method is $\log_2 m + o(\log m)$ for $q \leq p$, the same as that of Knuth's method for $q = 0$.

C. Flipping tail patterns

Method description

In an alternative way of encoding, the encoder locates $(q - q')/2$ symbols valued '-1' in the delinquent $\boldsymbol{w}$, and these '-1' symbols are inverted into '$+1$' symbols. Note that, since $|q'| \leq q - 2$, at most $(q - 1)$ '-1' symbols must be inverted. The positional information of the $(q-q')/2$ inverted symbols will be conveyed to the receiver by the prefix. The quantity N_p will denote the number of distinct position combinations of inverted symbols needed to unbalance all possible delinquent words. The prefix conveys either information on the index k (in case Knuth's algorithm does not fail) or on the positional information of the inverted symbols (in case Knuth's algorithm fails). By decoding the prefix, the decoder can undo the modifications made.

The index $k \in \{0, \ldots, m\}$ requires $m + 1$ combinations of the prefix, while the positional information of the inverted symbols requires N_p combinations, thus totaling $m + 1 + N_p$ prefix combinations. We will now identify a segment of a delinquent word $\boldsymbol{w}$, where the $(q - q')/2$ symbols '-1' can be found, so that N_p and the redundancy of the new method can be calculated. We will first exemplify the above with a simple example, where $q = 2$.

Example 3.1: For $q = 2$, we find that all delinquent words $\boldsymbol{w}$ are characterized by the following properties

(a) $q' = 0$;
(b) $w_1 = +1$;
(c) $w_m = -1$.

(Please note, however, that there are many words $\boldsymbol{w}$ satisfying (a)-(c), which are not delinquent.)

From $q' = 0$, we conclude that only one '-1' has to be located and inverted in the delinquent word. As the tail symbol of all delinquent words is $w_m = -1$, we conclude that by inverting the tail symbol, we can translate any user word $\boldsymbol{w}$ into a codeword with imbalance $q = 2$. Clearly, we require $m + 2$ different prefixes to uniquely identify the modification made at the transmitter's site.

Lemma 3.1: Let $\boldsymbol{w}$ be a delinquent word of even length m, $q \geq 2$ and q be even, and let ℓ be an integer, $1 \leq \ell \leq m$. Then there are at least

$$\frac{\ell + 1}{2} - \frac{q + q'}{4}$$

symbols '-1' in the ℓ tail bits $(w_{m-\ell+1}, w_{m-\ell+2}, \cdots, w_m)$.

Proof: In case Knuth's encoder fails, we have for any $0 \leq k \leq m$ that $\sigma_k(\boldsymbol{w}) < q$ and $\sigma_k(\boldsymbol{w})$ is even. Thus from (4) we obtain

$$\sigma_k(\boldsymbol{w}) = -2\sum_{i=1}^{k} w_i + q' \leq q - 2, \quad 0 \leq k \leq m \ , \tag{6}$$

or

$$2\sum_{i=k+1}^{m} w_i \leq q + q' - 2, \quad 0 \leq k \leq m \ . \tag{7}$$

Let ℓ^+ be the number of '$+1$'s in the last ℓ positions of $\boldsymbol{w}$ and ℓ^- be the number of '-1's in these ℓ positions. We have that

$$\ell^+ + \ell^- = \ell \ . \tag{8}$$

From (7) (with $k = m - \ell$), we also have

$$2\ell^+ - 2\ell^- \leq q + q' - 2 \ . \tag{9}$$

We conclude from (8) and (9) that

$$\ell^- \geq \ell/2 - (q + q')/4 + 1/2 \ ,$$

as required. ■

The following Theorem will be used in the encoding method, which will be presented in the sequel.

Theorem 3.2: Let $\boldsymbol{w}$ be a delinquent word of even length, $q \geq 2$ and q be even. Then there are at least $(q - q')/2$ symbols valued '-1' in the $\ell = (3q - q')/2 - 2$ tail bits $(w_{m-\ell+1}, w_{m-\ell+2}, \cdots, w_m)$.

Proof: By setting $\ell = (3q - q')/2 - 2$ in Lemma 3.1, we obtain that the number of symbols valued '-1' is at least

$$(q - q')/2 - 1/2 \ .$$

Since q and q' are both even, and the number of symbols is integer, we obtain the desired result. ■

The following corollary also follows from Lemma 3.1.

Corollary 1: In the $q-2+\tau$ tail bits of a delinquent word of even length there are at least $\tau/2$ symbols valued '-1', for $1 \le \tau \le q-1$.

Proof: From Lemma 3.1, and by taking $\ell = q-2+\tau$, we obtain that the number of symbols valued '-1' is at least

$$\frac{\ell+1}{2} - \frac{q+q'}{4} = \frac{q-1+\tau}{2} - \frac{q+q'}{4} = \frac{\tau}{2} + \frac{q-2-q'}{4} \ge \frac{\tau}{2} ,$$

where the last transition is due to $|q'| \le q-2$. ∎

Example 3.2: Consider the input word $\boldsymbol{w}$

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

where we use '$+$' for '$+1$' and '$-$' for '-1'. For this $\boldsymbol{w}$ we have $q' = 2$. Assume that the required imbalance $q = 4$. Observe that if we apply Knuth's algorithm by flipping bits sequentially, the resulting word will never have imbalance 4. Then, according to Theorem 3.2, there is at least one symbol '-1' in the last three bits in the word $\boldsymbol{w}$. Indeed, we see that there is a '-1' symbol in the third to last position. After flipping this bit, the total imbalance equals $q = 4$, as required.

Example 3.3: Consider the input word $\boldsymbol{w}$

$$-+-+--+-+-+-+- .$$

The unbalance of $\boldsymbol{w}$ equals $q' = -2$. Assume that the required imbalance $q = 4$, then observe that this cannot be achieved by flipping the bits sequentially, as in Knuth's algorithm. According to Theorem 3.2, there are at least three symbols '-1' in the last five positions in the word $\boldsymbol{w}$. By flipping these bits, the total imbalance becomes $q = 4$, as required.

We therefore conclude that only a limited number of inversion combinations need to be taken into account. Then, in case Knuth's algorithm fails to produce an unbalanced codeword, we can take care of that 'failed' word $\boldsymbol{w}$ by inverting $(q-q')/2$ symbols '-1' in the $(3q-q')/2-2$ tail bits of $\boldsymbol{w}$. In the worst case, $q-1$ symbols '-1' have to be inverted into a '$+1$' in $2q-3$ tail symbols.

In the next section, we compute the number of modifications, N_p, that have to be made to delinquent words in order to generate a q-balanced codeword.

Redundancy computation

The computation of N_p is directly related to the computation of the redundancy of the new method. Note that N_p is independent of the word length m. The number, N_p, of prefixes required to identify the $(q-q')/2$ symbol inversions made in the $\ell = (3q-q')/2-2$ tail bits is upper bounded by

$$N_p < \sum_{\substack{q'=-q+2 \\ q' \ even}}^{q-2} \binom{\frac{3q-q'}{2}-2}{\frac{q-q'}{2}},$$

TABLE I
SET OF SHORTEST TAIL STRINGS FOR $q = 4$.

$q'=2$	$q'=0$	$q'=-2$
-1	$-1-1$	$-1-1-1$
$-1+1$	$-1+1-1$	$-1+1-1-1$
$-1+1+1$	$-1-1+1$	$-1-1+1-1$
	$-1+1+1-1$	$-1+1+1-1-1$
	$-1+1-1+1$	$-1+1-1+1-1$

or after replacing $(q'+q)/2$ by i, where $i = 1, 2, \cdots, q-1$, we obtain

$$N_p < \sum_{i=1}^{q-1} \binom{2q-2-i}{q-i}.$$

The exact computation of N_p can be accomplished by setting up a forest of $q-1$ binary trees for all possible values of

$$q' \in \{-q+2, -q+4, \ldots, q-2\} .$$

Starting from the root with w_m, we generate all possible valid tail strings $(w_{m-k}, \ldots, w_m)$ of length $k+1$, where a string is valid if $2\sum_{i=m-k}^{m} w_i \le q+q'-2$. We terminate a string $(\ldots, w_{m-2}, w_{m-1}, w_m)$ when the string contains $(q-q')/2$ symbols '-1'. Theorem 3.2 guarantees that the length of a string is at most $2q-3$. The value of N_p is computed by summing all strings of all $q-1$ trees.

For $q = 4$, the 13 shortest tail strings are given in Table I, where the right-most symbol of the strings is associated with the tail symbol w_m of a codeword and the tree root. Appending the above ℓ-bit tail strings to $m-\ell$ leading '$+1$' symbols yields the set of m-bit "unbalancing" vectors $\{\boldsymbol{b}_j\}$, whose entries are from the set $\{-1, +1\}$. The delinquent input word $\boldsymbol{w}$ is modified using its component-wise product $\boldsymbol{w} \cdot \boldsymbol{b}_j$ with a judiciously chosen "unbalancing" vector $\boldsymbol{b}_j$. Thus, from the above 13 m-bit "unbalancing" vectors plus the set of $m+1$ Knuth's balancing vectors of the form

$$(\underbrace{-1\ -1\ \cdots\ -1}_{k}\ \underbrace{+1\ +1\ \cdots\ +1}_{m-k}) ,$$

$0 \le k \le m$, we can select at least one vector such that the inner product of the selected vector and the input word $\boldsymbol{w}$ equals $q = 4$ for any even value of $m > 4$. This component-wise product $\boldsymbol{w} \cdot \boldsymbol{b}_j$ as above is appended to the encoded index of the chosen vector $\{1, 2, \cdots, m+1+N_p\}$, thus resulting in the word of total imbalance q, as required.

The unbalancing vector $\boldsymbol{b}_j$ is uniquely determined by the positions of the symbols '-1' in the tail of length at most $2q-3$ in the encoded word. If we assume that q is fixed, then the encoding can be done by using a look-up table, or by using an enumerative coding on the tail of length up to $2q-3$. Since q is constant, such encoding takes constant time.

We have computed N_p for various values of q. The second column in Table II shows the results of our computations.

We can improve the redundancy by a small modification of the tree search: in case $q' < 0$, we invert many symbols in a large tail. This costs a lot of bits of redundancy. Thus, in the worst case, $q' = -q+2$, we invert $q-1$ symbols in $2q-3$ tail symbols. If we assume that we invert all bits of that user

TABLE II
NUMBER OF PREFIXES, N_p, VERSUS q.

q	N_p
2	1
4	13
6	131
8	1429
10	16795

word, then the imbalance q' becomes $-q'$. The worst case $q' = -q+2$ becomes $q' = q-2$, where only one symbol has to be inverted in $q-1$ tail bits. For this inverted word, we search for the smallest tail string that contains $(q+q')/2$ symbols valued '-1'.

In a systematic way, we set up a forest of $q-1$ binary trees as described above, where we follow a valid path until either we tallied $(q-q')/2$ symbols valued '-1' (as described above in the first method) or we tallied $(q+q')/2$ symbols '$+1$'. The m-bit unbalancing vectors are obtained as follows. The ℓ-bit tail strings with $(q-q')/2$ symbols '-1' are appended to $m-\ell$ symbols '$+1$', while the ℓ-bit tail strings with $(q+q')/2$ symbols '$+1$' are appended to $m-\ell$ symbols '-1'. We tally the leaves of all trees, which yields N'_p, the number of tail strings found.

The value of N_p is rapidly mounting with increasing imbalance q. Since N_p+1 can be bounded from above by 2^{2q-2}, the redundancy produced by this method is $\log_2(m+1+N_p) \leq \max\{\log_2 m + 1, 2q-1\}$. We conclude that the proposed method is as optimal as Knuth's method for $2q << \log_2 m$.

D. An alternative approach with several operational modes

In order to encode an arbitrary block of '± 1's of length $m \geq 16$ into a block of imbalance q, the methods discussed in Sections III-B and III-C produce at least $\approx q$ redundant bits, in addition to redundancy produced by Knuth's algorithm. Below, we introduce an alternative encoding method which encodes an arbitrary block of data of length m into a block with imbalance $q = \frac{3}{2}\log_2 m$ with only $\log_2 m + o(\log m)$ redundant bits. Hereafter we assume that $\log_2 m$ is even integer.

Recall that if $q' < 0$, we first invert the word. Then, there are five different modes of operation: modes 1, 2, 3, 4 and 5. Mode 5 contains four different submodes, denoted by 5-1 – 5-4. The choice of mode depends on the value of q' and the structure of the input word $\boldsymbol{w}$. The selected mode and a possible inversion (or not) are denoted by a special prefix. There are eight different modes, for each mode there are also two possibilities for inversion. These 16 possibilities are represented by balanced prefixes with six bits.

We also make use of a balanced suffix of length $\log_2 m + o(\log m)$. In Modes 1 and 5-1 this suffix denotes the number of bit flips. In Modes 2 and 5-2 the suffix carries additional weight, thus making the total imbalance of the word equal q. In Modes 3 and 5-3, the suffix serves as an index of an entry in the input word. Modes 4 and 5-4 do not use suffix. Instead, in these modes, the values of the last $\log_2 m - 1$ bits in the tail of the word are encoded by a *location* of a special string of length $2\log_2 m + 1$, which is inserted into the input word. The tail is then discarded. Therefore, in these two modes, the resulting length of the word increases by $\log_2 m + o(\log m)$ bits as well.

Recall that $\boldsymbol{w} = (w_1, w_2, \cdots, w_m)$ is an arbitrary input word, m is even, and $q' = \sum_{i=1}^{m} w_i$. Let $q = \frac{3}{2}\log_2 m$ be an even integer number. Assume without loss of generality, $q' \geq 0$, otherwise flip all the bits, and mark this in the prefix of the encoded word. Denote by

$$\mu \triangleq \sum_{i=m-\log_2 m+2}^{m} w_i$$

a sum of the last $\log_2 m - 1$ symbols in $\boldsymbol{w}$.

Mode 1. $q' > \frac{3}{2}\log_2 m$*:* In this case, we simply apply Knuth's algorithm. We append a balanced suffix of $\log_2 m + o(\log m)$ bits at the tail of the block, to denote the actual number of bit flips.

Mode 2. $\frac{1}{2}\log_2 m - 1 \leq q' \leq \frac{3}{2}\log_2 m$*:* In this case, we simply append the suffix of the total imbalance $q_p = q - q' \leq \log_2 m + 1$.

Mode 3. $0 \leq q' < \frac{1}{2}\log_2 m - 1$ *and the word* $\boldsymbol{w}$ *contains a subsequence* "$+1+1+1 \ldots +1$" *of length* σ*, where* $\sigma \geq \log_2 m$*:* Then, the sum of all bits in $\boldsymbol{w}$, except for this subsequence, is in the interval $[-\sigma, \frac{1}{2}\log_2 m - 2 - \sigma]$.

If the subsequence of symbols '$+1$' does not appear at the end of the word $\boldsymbol{w}$, then by flipping all the bits in $\boldsymbol{w}$, except for the above subsequence and one '-1' immediately after it, we make the total imbalance to lie in the interval $[2\sigma - \frac{1}{2}\log_2 m, 2\sigma - 2]$. If, however, the subsequence of symbols '$+1$' appears at the end of $\boldsymbol{w}$, then we flip all the bits in $\boldsymbol{w}$, except for the above subsequence. The resulting total imbalance then lies in the interval $[2\sigma - \frac{1}{2}\log_2 m + 2, 2\sigma]$.

Then, we sequentially flip the bits in the subsequence "$+1+1+1 \ldots +1$", from the first to the penultimate, to make the total imbalance equal q. It is straight-forward to see that this imbalance can always be achieved. We need $\log_2 m + o(\log m)$ bits to encode the index of the first flipped bit in this subsequence by the balanced suffix. Observe, that from knowing that index, the flipped region can be uniquely determined, and so the whole operation is invertible.

Mode 4. $0 \leq q' < \frac{1}{2}\log_2 m - 1$*, the word* $\boldsymbol{w}$ *contains no subsequence* "$+1+1+1 \ldots +1$" *of length* $\geq \log_2 m$ *and* $q' \geq \mu$*:* Denote $\hat{\boldsymbol{w}} = (w_1, w_2, \ldots, w_{m-\log_2 m+1})$. We insert the following string between two consecutive symbols in $\hat{\boldsymbol{w}}$:

$$-1\ \underbrace{+1+1+1 \ldots +1}_{s_1}\ \underbrace{-1-1-1 \ldots -1}_{s_2}\ +1\ , \qquad (10)$$

$s_1 + s_2 = 2\log_2 m - 1$, $s_1 \geq \log_2 m$ and $s_2 \geq 1$. Denote the sum of the elements in this string by η. Under the given constraints, η can take any odd value in $[1, 2\log_2 m - 3]$.

We select the values of s_1 and s_2 such that after the insertion the total imbalance of the resulting word (i.e., $\hat{\boldsymbol{w}}$ with the inserted sequence) becomes $\frac{3}{2}\log_2 m$. Note, that after the insertion of the string, the imbalance becomes

$q' - \mu + \eta$. Since $q' \geq \mu$, we obtain that

$$0 \leq q' - \mu < \frac{3}{2}\log_2 m - 2 \,,$$

and is odd, and so there exists a string as above such that $q' - \mu + \eta = \frac{3}{2}\log_2 m$.

Next, we decide on the location of this insertion. The location of the insertion $i \in \{0, 1, 2, \ldots, m - \log_2 m + 1\}$, represents the encoding of the last $\log_2 m - 1$ bits that were removed from $\boldsymbol{w}$. These bits can take

$$2^{\log_2 m - 1} = \frac{m}{2}$$

different values. Since $\frac{m}{2} < m - \log_2 m + 1$ (for all $m \geq 4$), we have enough different locations to encode the values of these bits.

There are two possible cases:

1) There is no other substring "$+1\,+1\,+1\,\ldots\,+1$" of length $\geq \log_2 m$ in $\hat{\boldsymbol{w}}$, and so the inserted string can be uniquely identified.
2) There is another substring "$+1\,+1\,+1\,\ldots\,+1$" of length $\log_2 m$ in $\hat{\boldsymbol{w}}$. This can happen only if the inserted string appears immediately before the sequence of $\log_2 m - 1$ symbols '$+1$' in $\hat{\boldsymbol{w}}$. In that case the inserted string can also be uniquely identified by using the leftmost substring of at least $\log_2 m$ symbols '$+1$'.

Therefore, we see that all steps are invertible.

Mode 5. $0 \leq q' < \frac{1}{2}\log_2 m - 1$*, the word* $\boldsymbol{w}$ *contains no subsequence* "$+1\,+1\,+1\,\ldots\,+1$" *of length* $\geq \log_2 m$ *and* $q' < \mu$*:* Consider the word $\tilde{\boldsymbol{w}}$ obtained by flipping the first $m - \log_2 m + 1$ bits in $\boldsymbol{w}$, in other words $\tilde{\boldsymbol{w}} = \boldsymbol{w}^{(m - \log_2 m + 1)}$. Since

$$\sum_{i=1}^{m - \log_2 m + 1} w_i = q' - \mu < 0 \,,$$

we obtain that

$$d(\tilde{\boldsymbol{w}}) = (\mu - q') + \mu > \mu \geq 0 \,.$$

Thus, the resulting $\tilde{\boldsymbol{w}}$ falls under one of the cases considered in Modes 1–4. We apply the corresponding encoding method in Modes 1–4 to $\tilde{\boldsymbol{w}}$ (we refer to these cases as Modes 5-1 – 5-4, respectively).

As we can see, all encoding modes are invertible. Therefore, the whole encoding algorithm is invertible.

Time complexity: It can be verified that each mode requires up to $O(m)$ bit operations and $O(m)$ increments/decrements of counters of size $O(\log m)$ bits.

Example 3.4: In the following toy example assume that we want to encode an input sequence $\boldsymbol{w}$ of length $m = 16$ into an imbalanced codeword with $q = \frac{3}{2}\log_2 m = 6$. Below, we use '$+$' for '$+1$' and '$-$' for '$-1$'. Let $\boldsymbol{w}$ be

$$- + - + - - + + + + - + - - - + \,.$$

For this input word, we have $d(\boldsymbol{w}) = 0$. Moreover, $\boldsymbol{w}$ contains a sequence of four consecutive '$+1$'s, and thus we conclude that Mode 3 of the encoding algorithm should be applied. We flip all bits in $\boldsymbol{w}$, except for the sequence of four consecutive '$+1$'s and one '-1' immediately after it. We obtain:

$$+ - + - + + \quad + + + + - \quad - + + + + - \,.$$

The total imbalance is now 6. There is no need to apply more bit flips. Yet we have to store a pointer to the 7-th symbol (the first '$+1$' in the sequence of four '$+1$'s) in order to make the encoding invertible.

To encoded word, we add a prefix and a suffix. The prefix denotes that the overall word was not inverted, and that Mode 3 was used. The suffix stores balanced encoding of index 7.

To invert the encoded word, the decoder recovers the mode and the value of the suffix, and inverts all '-1' from position 7 till the first '$+1$' (in this case there will be zero such inversions). Then, the decoder inverts all the bits except the long sequence of '$+1$'s (that starts at position 7) and one consecutive '-1'.

Example 3.5: As in the previous example, assume that we want to encode an input sequence $\boldsymbol{w}$ of length $m = 16$ into an imbalanced codeword with $q = 6$. Let $\boldsymbol{w}$ be

$$- + - - + + - + + - + - + - - + \,.$$

In this case $d(\boldsymbol{w}) = 0$ and $\mu = -1$, and so Mode 4 is applicable.

Then, we insert the string

$$- + + + + + + + - +$$

of length 9 and imbalance 5, in location which is encoded by the last 3 bits. Thus $- - +$ represents the value of 1 (if '-1' is regarded as binary 0, and '$+1$' is regarded as binary 1). The resulting encoded word becomes (after the last three bits are discarded):

$$- \quad - + + + + + + + - + \quad + - - + + - + + - + - + \,.$$

A prefix needs to be added to this word. This prefix denotes that the overall word was not inverted, and that Mode 4 was used.

To invert the encoded word, the decoder recovers the mode. Then, the decoder identifies the substring of the form (10) at location 1. It removes this substring, and appends the suffix that represents encoding of its location, namely "$- - +$".

Example 3.6: As in two previous examples, assume that we want to encode an input sequence $\boldsymbol{w}$ of length $m = 16$ into an imbalanced codeword with $q = 6$, and $\boldsymbol{w}$ is

$$- + - - + + - + + - + - - + - + \,.$$

In this case $d(\boldsymbol{w}) = 0$ and $\mu = +1$, and so Mode 5 is applicable. Thus, the encoder inverts the first $m - \log_2 m + 1 = 13$ bits to obtain the word $\tilde{\boldsymbol{w}}$ equal

$$+ - + + - - + - - + - + + + - + \,.$$

We have that $d(\tilde{\boldsymbol{w}}) = 2$, and so the encoder proceeds as in Mode 2.

Recall that the prefix needs to be added, in order to encode the mode number, which is Mode 5-2.

Other parameter settings

The method described in this section can also be adopted for some other parameter combinations. For example, for the input word $\boldsymbol{w}$ of length m and desired imbalance q, such that $m < 2^{2q/3}$, we can combine the encoding method in this section with the method outlined in Section III-B. Alternatively, we may append the "missing" number of '+1's to make the total imbalance equal to q, as suggested in Section III-A.

If the length m and desired imbalance q are such that $m > 2^{2q/3}$, we can use a combined decoder that applies Knuth's algorithm to the first $m - 2^{2q/3}$ input bits, and the encoding algorithm in this section to the remaining $2^{2q/3}$ bits. The resulting encoder outputs words with imbalance q and redundancy

$$\log_2(m - 2^{2q/3}) + \frac{2}{3}q + o(\log m) \ .$$

Alternatively, for the case $m > 2^{2q/3}$, we can use a combined decoder when we first apply the encoding algorithm in this section (resulting in the redundancy larger than the required q), and then apply Knuth-like algorithm to reduce the total imbalance down to q. The resulting encoder similarly outputs words with imbalance q and redundancy $2\log_2 m + o(\log m)$. However, the produced redundancy in this case is slightly larger than in the approach outlined in the previous paragraph.

IV. Codes in Two Dimensions

In this section, we generalize the results in Section III towards two-dimensional imbalanced arrays. More specifically, we consider a problem of encoding an arbitrary sequence of information bits into a two-dimensional $n_1 \times n_2$ imbalanced array. Here, every row in the encoded array has imbalance q_1 and every column has imbalance q_2. It must hold that $n_1 \cdot q_1 = n_2 \cdot q_2$. We propose an efficient algorithm to solve this task, which can use one of the encoding algorithms for imbalanced (one-dimensional) words in Sections III as a subroutine.

Two-dimensional constrained codes were considered for use in a variety of applications, such as magnetic and optical disks [8], multi-track storage devices [12] and holographic storage [21]. In particular, two-dimensional balanced codes with corresponding encoding and decoding algorithms were proposed in [19]. For these codes, defined over the binary alphabet, the number of zeros and ones in every row is equal, and the numbers of zeros and ones in every column are equal. Therefore, those codes naturally generalize balanced codes towards two-dimensional settings.

The algorithm, that we propose in this section, closely follows the structure of the two-dimensional balancing algorithm by Talyansky *et al.* [19]. In particular, when $q_1 = q_2 = 0$, this algorithm is essentially equivalent to its counterpart [19]. However, as we will show, by implementing a small modification, the algorithm can handle the case when $q_1 \neq 0$ (and so $q_2 \neq 0$).

The proposed algorithm, named Algorithm ENCODE, is presented in Figure 1 (although it is quite similar to its counterpart in [19], we show it for the sake of completeness of the presentation.) For simplicity, we assume here that both n_1 and n_2 are powers of two. Later, we will discuss the modifications to the algorithm needed to take care of the cases where n_1 and n_2 are not powers of two.

The algorithm first applies a one-dimensional unbalancing encoding step to the input string, on a row-by-row basis (Step 2). Then, in Step 3, Procedure SWAP, which swaps bits between two two-dimensional subarrays, is recursively used to make the number of ones in every column of the array equal. In Step 4, the redundant bits, obtained as a by-product of the execution of Procedure SWAP, are encoded using the recursive call to Algorithm ENCODE. In Step 5, the result of the encoding in Step 4 is appended to the array produced in Step 2.

The parameter η in the algorithm corresponds to the smallest size of information sequence, for which the look-up table is not implemented.

Algorithm ENCODE (**Inputs:** integer n_2, word $\boldsymbol{w}$).

Step 1: If length of $\boldsymbol{w}$ is smaller than η, encode it into Γ using look-up table and return.

Step 2: Encode $\boldsymbol{w}$ into an $s \times n_2$ array Γ_0, in which every row has imbalance q_1.

Step 3: $(\Gamma_0, \boldsymbol{w}') = \text{SWAP}\ (s, n_2, \Gamma_0)$.

Step 4: Let $(n_1', \Gamma_1) = \text{ENCODE}(n_2, \boldsymbol{w}')$.

Step 5: Let ${\Gamma}^T = ({\Gamma_0}^T \mid {\Gamma_1}^T)$, and $n_1 = s + n_1'$.

Output: integer n_1 and $n_1 \times n_2$ array Γ.

Fig. 1. Encoding algorithm for two-dimensional imbalanced arrays.

Procedure SWAP is presented in Figure 2. This procedure is used to make the number of ones in every column of the input array equal. It divides the input array into two parts, each part consists of half the number of the columns in the input. The bits are sequentially exchanged between the two parts, until the number of ones in each part becomes equal. The number of exchanged bits is recorded, and Procedure SWAP is recursively applied to each of the produced parts.

Generally, the structure of the algorithm is similar to that of its counterpart in [19], yet there are some differences between the two algorithms. Thus, the algorithm ENCODE in Figure 1, uses a slightly modified version of Procedure SWAP. The new version of the procedure swaps the bits between two arrays Δ_1 and Δ_2 until the number of '+1's in both arrays becomes equal, and so the total imbalance is equal. In Step 2, Algorithm ENCODE starts with the arrays, where every row has imbalance q_1. In Step 4, Algorithm ENCODE calls itself recursively to encode the indices used in swapping the data.

We claim that the proposed algorithm will always terminate with the imbalanced $n_1 \times n_2$ array as required. To see this, observe that the imbalance of any row is q_1 due to the correctness of the one-dimensional encoder used, and due to the fact that during the run of the algorithm, the symbols

Procedure SWAP (**Inputs:** integers p_1, p_2, and $p_1 \times p_2$ array Δ).

Step 1: Let $\boldsymbol{w}'$ be an empty string.

Step 2: Let Δ_1 and Δ_2 be arrays that consist of the first and the last $p_2/2$ columns of Δ, respectively.

Step 3: For $j = 1, 2, \ldots, p_2/2$ and for $i = 1, 2, \ldots, p_1$:

swap the bits $\Delta_1(i, j)$ and $\Delta_2(i, j)$,

until Δ_1 and Δ_2 contain equal number of '+1's.

Step 4: Append to $\boldsymbol{w}'$ the encoding of the indices (i, j) computed at Step 3.

Step 5: If $p_2 > 2$ then $(\Delta_1, \boldsymbol{w}'') = \text{SWAP}(p_1, p_2/2, \Delta_1)$ and $(\Delta_2, \boldsymbol{w}''') = \text{SWAP}(p_1, p_2/2, \Delta_2)$.

Output: $p_1 \times p_2$ array Δ, and the string $(\boldsymbol{w}', \boldsymbol{w}'', \boldsymbol{w}''')$.

Fig. 2. Procedure SWAP.

are not moved between different rows. Moreover, due to the selection of the parameters, the total imbalance at the end of the algorithm run is $n_1 \cdot q_1 = n_2 \cdot q_2$. Procedure SWAP ensures that the imbalances in all columns of the produced array are equal. Therefore, the imbalance of each column is q_2, as required.

Example 4.1: Assume that we want to encode a string $\boldsymbol{w}$ given by

$$-- \;\; -+ \;\; -- \;\; +-$$

of length $m = 8$ into 8×4 array, with a row imbalance $q_1 = 2$ and a column imbalance $q_2 = 4$. Assume that we can use a look-up table that maps an arbitrary input word onto a two-dimensional imbalanced array with row imbalance $q_1 = 2$ and number of columns $n_2 = 4$, for all input strings of length up to seven.

We will use an one-dimensional unbalancing encoder that maps an arbitrary vector of length 2 onto a vector of length $n = 4$ with imbalance $q = 2$, according to the following encoding table:

Information word	Codeword with $q = 2$
$--$	$++-$
$-+$	$-++$
$+-$	$+-++$
$++$	$++-+$

Then, the input and its row-by-row encoding can be written as the following two-dimensional arrays, respectively:

$$\boldsymbol{w} = \begin{bmatrix} -- \\ -+ \\ -- \\ +- \end{bmatrix} \quad \text{and} \quad \Gamma_0 = \begin{bmatrix} ++- \\ -++ \\ +++- \\ +-++ \end{bmatrix}.$$

Next, Procedure SWAP is applied to Γ_0 in order to obtain an equal number of ones in Δ_1 and Δ_2, where

$$\Delta_1 = \begin{bmatrix} ++ \\ -+ \\ ++ \\ +- \end{bmatrix} \quad \text{and} \quad \Delta_2 = \begin{bmatrix} +- \\ ++ \\ +- \\ ++ \end{bmatrix}.$$

In our case, no swaps of bits are needed, and $\boldsymbol{w}'$ is set to "$---$", the binary encoding of an integer 0. Then, Procedure SWAP is recursively applied to each of Δ_1 and Δ_2. Δ_1 has already equal number of ones in its first and second columns, and so it is not changed, and $\boldsymbol{w}''$ is set to "$--$" to represent 0 bit swaps. By contrast, Δ_2 becomes

$$\begin{bmatrix} -+ \\ ++ \\ +- \\ ++ \end{bmatrix},$$

and $\boldsymbol{w}'''$ is set to "$-+$" to represent 1 bit swap. The resulting string $(\boldsymbol{w}', \boldsymbol{w}'', \boldsymbol{w}''')$ is

$$--- \;\; -- \;\; -+,$$

which is encoded recursively into Γ_1 using a look-up table. The resulting output of the algorithm is $({\Gamma_0}^T \mid {\Gamma_1}^T)^T$, where

$$\Gamma_0 = \begin{bmatrix} ++-+ \\ -++ \\ +++- \\ +-++ \end{bmatrix}$$

and Γ_1 is obtained from the look-up table.

Next, we turn to estimate the redundancy of the proposed algorithm. Since the redundancy produced by the one-dimensional unbalancing algorithm used within the procedure ENCODE can vary, the redundancy analysis herein is somewhat different from its counterpart in [19]. Hereafter, we denote by $\mu(n_2)$ a redundancy produced by the unbalancing algorithm in Step 2 (where n_2 is the length of the resulting imbalanced vector).

Assume that the algorithm outputs array of size $n_1 \times n_2$, where $n_1 = s + n_1'$. Let $\rho(n_1 \times n_2)$ be the redundancy corresponding to encoding information into $n_1 \times n_2$ arrays. The redundancy produced in all levels of recursion in Step 2 is $n_1 \cdot \mu(n_2)$. Analogous to [19], the redundancy, produced in (the first level of recursion in) Step 3 is less than $n_2(1 + \lceil \log_2 s \rceil)$. In the second level of recursive call to procedure ENCODE, the value of s becomes at most

$$\left\lceil \frac{n_2(1 + \lceil \log_2 s \rceil)}{n_2 - \mu(n_2)} \right\rceil. \tag{11}$$

If we assume that $\mu(n_2) \leq \eta n_2$ for some constant $\eta < 1$ for all n_2 large enough[1], then the expression in (11) can be written as $O(\log s)$. We conclude that the value of s decreases as a log-function with the number of levels of recursion.

[1]The assumption that $\mu(n_2) \leq \eta n_2$ holds for all encoding methods presented in Section III and in [5], [14] and [20].

To summarize, we have

$$\begin{aligned}\rho(n_1 \times n_2) &\leq n_1 \cdot \mu(n_2) + n_2 \cdot \lceil \log_2 s + 1 \rceil + O(n_2 \log s) \\ &\leq n_1 \cdot \mu(n_2) + n_2 \cdot \log_2 n_1 + O(n_2 \log n_1) \ .\end{aligned}$$

It can be seen that the optimality of the produced array strongly depends on the optimality of $\mu(\cdot)$, the redundancy produced by the used encoding method for one-dimensional vectors.

The analysis of the encoding time complexity follows the lines of [19]. Step 1 (at all levels of recursion) requires applying a one-dimensional imbalancing algorithm to $O(n_1)$ rows. If the algorithm with time complexity $O(n_2)$ is used, then Step 1 costs $O(n_1 n_2)$ operations. Procedure SWAP works analogously to its counterpart in [19], and so Step 3 in the algorithm (at all levels of recursion) takes $O(n_1 n_2 \cdot \log n_2)$ operations overall. Thus, the total time complexity is bounded by $O(n_1 n_2 \cdot \log n_2)$.

The Procedure SWAP can be modified to handle arrays of sizes which are not powers of two. The required modification is slightly different from that done in [19]. Assume that Δ is odd. Suppose also that Δ_1 consists of the first $(p_2 - 1)/2$ columns of Δ, and Δ_2 consists of the remaining $(p_2 + 1)/2$ columns. If the total imbalance of Δ_1 is $q_2 \cdot (p_2 - 1)/2$, then no further processing is needed. Otherwise, w.l.o.g., assume that Δ_1 has imbalance larger than $q_2 \cdot (p_2 - 1)/2$. Select Δ_2' to be an $p_1 \times (p_2 - 1)/2$ subarray of Δ_2 that does not contain the column with the maximal imbalance (among all the columns in Δ_2). Then, the imbalance of Δ_2' is smaller than $q_2 \cdot (p_2 - 1)/2$. Therefore, by applying SWAP on the arrays Δ_1 and Δ_2', the imbalance of Δ_1 can be made exactly $q_2 \cdot (p_2 - 1)/2$. Then, the Procedure SWAP should be recursively applied to each of Δ_1 and Δ_2. Additionally, the index of the column excluded from Δ_2' should be appended to $\boldsymbol{w}'$.

Higher Dimensional Codes

Using similar ideas, the proposed encoding and decoding method can be further generalized to arrays of dimension higher than two. We leave that discussion outside the scope of this paper.

V. CONCLUSIONS

We have studied various constructions of simple codes that translate arbitrary user words into codewords with a prescribed imbalance $q > 0$.

We have shown that we can extend Knuth's scheme that produces balanced codewords, $q = 0$, to the "unbalanced" case $q > 0$. Part of the input words can be converted into a codeword of imbalance q by Knuth's algorithm. The other 'delinquent' input words require a second encoding step. We have presented a construction where delinquent words are balanced by an imbalance of the prefix. In a second construction, at most $(q - q')/2$ symbols '-1' in the delinquent word are inverted, where q' denotes the imbalance of the user word. The information regarding the changes made is carried by the prefix word. For $q = 2$, it suffices to invert the tail bit of the delinquent word. For larger values of q, more combinations of symbol inversions have to be defined which have to be carried by the prefix costing more redundancy. We have computed the redundancy of the new method for various values of m and q. We conclude that the method is either efficient for small q or very large m.

We also presented a new method which allows us to encode an arbitrary block of data of length m bits into a block of imbalance $q = \frac{3}{2} \log_2 m$ with only $\log_2 m + o(\log m)$ redundant bits. The time complexity of this method is $O(m)$ bit operations and $O(m)$ increments/decrements of counters of size $O(\log_2 m)$ bits. This method can be used in the algorithm that encodes arbitrary binary input words of length $m > 2^{2q/3}$ into words with imbalance q and redundancy $\log_2(m - 2^{2q/3}) + \frac{2}{3}q + o(\log m)$. Finally, we presented a modification of the encoder proposed by Talyansky *et al.* [19] that encodes an arbitrary binary data into a two-dimensional $n_1 \times n_2$ array, such that all its rows have imbalance q_1 and all its columns have imbalance q_2, where $n_1 q_1 = n_2 q_2$. The redundancy produced by this method is at most

$$\leq n_1 \cdot \mu(n_2) + n_2 \cdot \log_2 n_1 + O(n_2 \log n_1) \ ,$$

where $\mu(n_2)$ is the corresponding redundancy of any method for encoding one-dimensional arrays of length n_2 and imbalance q_1. The total time complexity of this method is $O(n_1 n_2 \cdot \log n_2)$. The method can be further extended to arrays in any dimension higher than two.

REFERENCES

[1] S. Al-Bassam and B. Bose, "On balanced codes," *IEEE Trans. Inform. Theory,* vol. IT-36, no. 2, pp. 406–408, March 1990.

[2] 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.

[3] M. Blaum, "Codes for detecting and correcting unidirectional errors," Los Alamitos, CA.: IEEE CS Press 1993.

[4] M. Blaum and J. Bruck "Delay-intensitive pipelined communication on parallel buses," *IEEE Trans. Computers,* vol. 44, no. 5, pp. 660–668, May 1995.

[5] Y.M. Chee, C.J. Colbourn, and A.C.H. Ling, "Optimal memoryless encoding for low power off-chip data buses," Proceedings of the 2006 IEEE/ACM international Conference on Computer-Aided Design, San Jose, California, November 05 - 09, 2006.

[6] T.M. Chien, "Upper bound on the efficiency of dc-constrained codes," *Bell Syst. Tech. J.,* vol. 49, pp. 2267–2287, Nov. 1970.

[7] E. En Gad, M. Langberg, M. Schwartz and J. Bruck, "Constant-weight Gray codes for local rank modulation," *IEEE Transactions on Information Theory,* vol. 57, pp. 7431–7442, Nov 2011

[8] T. Etzion, "Cascading methods for runlength-limited arrays," *IEEE Trans. Inform. Theory*, vol. 43, pp. 319–324, 1997.

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

[10] 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.

[11] D.E. Knuth, "Efficient balanced codes," *IEEE Trans. Inform. Theory,* vol. IT-32, no. 1, pp. 51–53, Jan. 1986.

[12] M. Marcellin and H. Weber, "Two-dimensional modulation codes," *IEEE J. Selected Areas Commun.,* vol. 10, pp. 254–266, 1992.

[13] B. Ryabko, "Fast enumeration of combinatorial objects," *Discrete Math. and Applications,* vol. 8, no. 2, pp. 163–182, 1998.

[14] J.P.M. Schalkwijk, "An algorithm for source coding," *IEEE Trans. Inform. Theory,* IT-18, pp. 395–399, 1972.

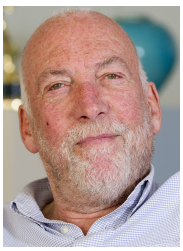
[15] P. Stanica, "Good lower and upper bounds on binomial coefficients," *Journal of Inequalities in Pure and Applied Mathematics,* vol. 2, Art. 30, 2001.

[16] J. Tabor, "Noise reduction using low weight and constant weight coding techniques," Technical Report AI-TR 1232, MIT, Artificial Intelligence Laboratory, June 1990.
[17] L.G. Tallini and B. Bose, "Design of balanced and constant weight codes for VLSI systems," *IEEE Trans. on Computers,* vol. 47, pp. 556–572, May 1998.
[18] 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.
[19] R. Talyansky, T. Etzion, and R.M. Roth, "Efficient code constructions for certain two-dimensional constraints," *IEEE Trans. Inform. Theory,* vol. IT-45, pp. 794–799, Mar. 1999.
[20] C. Tian, V.A. Vaishampayan, and N. Sloane, "A coding algorithm for constant weight vectors: a geometric approach based on dissections," *IEEE Trans. Inform. Theory,* vol. IT-55, no. 3, pp. 1051–1060, March 2009.
[21] A. Vardy, M. Blaum, P. H. Siegel, and G. T. Sincerbox, "Conservative arrays: multidimensional modulation codes for holographic recording," *IEEE Trans. Inform. Theory,* vol. 42, pp. 227–230, 1996.
[22] T. Verhoeff, "Delay-insensitive codes – an overview," *Distributed computing,* vol. 3, pp. 1–8, May 1988.
[23] L.-T. Wang, C.E. Stroud, and N.A. Touba, "System-on-chip test architectures: nanometer design for testability," Elsevier Science Limited, 2008.
[24] H. Zhou, A. Jiang, and J. Bruck, "Error-correcting schemes with dynamic thresholds in nonvolatile memories," *Proc. IEEE Intern. Symp. on Inform. Theory (ISIT),* pp. 2109–2113, St. Petersburg, Russia, 2011.
[25] H. Zhou, A. Jiang, and J. Bruck, "Balanced modulation for nonvolatile memories," ArXiv report *http://arxiv.org/abs/1209.0744*.

Vitaly Skachek received the B.A. (Cum Laude), M.Sc. and Ph.D. degrees in computer science from the Technion—Israel Institute of Technology, in 1994, 1998 and 2007, respectively.

In the summer of 2004, he visited the Mathematics of Communications Department at Bell Laboratories under the DIMACS Special Focus Program in Computational Information Theory and Coding. During 2007-2012, he held research positions with the Claude Shannon Institute, University College Dublin, with the School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, with the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, and with the Department of Electrical and Computer Engineering, McGill University, Montréal. He is now a Senior Lecturer with the Institute of Computer Science, University of Tartu.

Dr. Skachek is a recipient of the Permanent Excellent Faculty Instructor award, given by Technion.



Kees Schouhamer Immink received his PhD degree from the Eindhoven University of Technology in 1985. He is, since 1994, an adjunct professor at the Institute for Experimental Mathematics, Essen-Duisberg University, Germany. He founded and became president of Turing Machines Inc. in 1998. He contributed to consumer-type digital audio and video recording products, such as Compact Disc, CD-ROM, CD-Video, Digital Audio Tape recorder, Digital Compact Cassette system, DCC, Digital Versatile Disc, DVD, and Blu-ray Disc.

He received widespread recognition for his many contributions to the technologies of video, audio, and data recording. He received a Knighthood in 2000, an Emmy award in 2004, the 1996 IEEE Masaru Ibuka Consumer Electronics Award, the 1998 IEEE Edison Medal, the 1999 AES Gold Medal, and the 2004 SMPTE Progress Medal. He was named a fellow of the IEEE, AES, and SMPTE, and was inducted into the Consumer Electronics Hall of Fame, elected into the Royal Netherlands Academy of Arts and Sciences, and the US National Academy of Engineering. He served the profession as President of the Audio Engineering Society inc., New York, in 2003.