Prefixless q-ary Balanced Codes with Fast Syndrome-based Error Correction

Theo G. Swart, *Senior Member, IEEE*, Jos H. Weber, *Senior Member, IEEE*, and Kees A. Schouhamer Immink, *Fellow, IEEE*

***Abstract*—We investigate a Knuth-like scheme for balancing q-ary codewords, which has the virtue that look-up tables for coding and decoding the prefix are avoided by using precoding and error correction techniques. We show how the scheme can be extended to allow for error correction of single channel errors using a fast decoding algorithm that depends on syndromes only, making it considerably faster compared to the prior art exhaustive decoding strategy. A comparison between the new and prior art schemes, both in terms of redundancy and error performance, completes the study.**

***Index Terms*—Balanced code, constrained code, error correction, Knuth code, running digital sum.**

I. Introduction

BALANCED, sometimes called dc-free, q-ary sequences have found widespread application in popular optical recording devices such as CD, DVD, and Blu-Ray [1], cable communication [2], and recently in non-volatile (flash) memories [3]. A sequence of symbols is said to be balanced if the sum of the symbols equals the prescribed balancing value. The study of simple and efficient methods for translating arbitrary source sequences into balanced q-ary sequences has been an active field of research. If the sequences are not too long, look-up translation tables can be used. For handling extremely long binary ($q = 2$) blocks, where look-up tables are impractically large, Knuth [4] has devised two simple algorithms for generating binary balanced codewords, namely a *parallel* algorithm and a *serial* algorithm.

In the parallel algorithm, the encoder splits the user word into two segments: the first consisting of the first v bits of the user word, and the second consisting of the remaining $m - v$ bits. The encoder inverts the first segment by adding (modulo 2) a '1' to the v symbols in the first segment. The index v is chosen in such a way that the modified word is balanced. Knuth showed that such an index v can always be found. In the simplest embodiment of Knuth's algorithm, the index v is represented by a p-bit balanced word, called a *prefix*. The p-bit prefix is appended to the m-bit modified user word, and the sequence of $p+m$ bits is transmitted. The rate of the code is $m/(m+p)$. The receiver, after observing the prefix, decodes the index v, and subsequently it undoes the modifications made to the user word. Note that both the encoder and decoder do not require large, m-bit wide, look-up tables, making Knuth's algorithm very attractive for balancing long user words. The serial algorithm adds a p-bit prefix (not necessarily balanced) that describes the weight of the original m-bit user word. In this case, the encoder splits the sequence, consisting of the prefix and user word together, into two segments and finds an index v that balances the overall sequence. The receiver undoes the modification by inverting the bits until the original weight, captured by the prefix, is attained. Modifications and embellishments of Knuth's binary schemes have been presented by Al-Bassam and Bose [5], Tallini *et al.* [6], and Weber and Immink [7].

Binary balancing schemes that enable correction of errors have been presented by van Tilborg and Blaum [8], Al-Bassam and Bose [9] and Weber *et al*. [10], among others. In [8], the idea is to consider short balanced sequences as symbols of a non-binary alphabet and to construct error-correcting codes over that alphabet. In [9], balanced codes that correct a single error are constructed. These codes can be extended using concatenation techniques to correct up to four errors. In [10], a combination of conventional error correction techniques and Knuth's balancing method is used.

Methods for balancing q-ary, $q > 2$, codewords can be found, for example, in Capocelli *et al*. [11], Tallini and Vacaro [12], Al-Bassam [13], Swart and Weber [14], and Pelusi *et al*. [15]. Balancing is achieved in [14], as in Knuth's parallel scheme, by splitting the user word into a first and second segment of v and $m - v$ symbols, respectively. The encoder adds (modulo q) an integer $s + 1$ to the symbols in the first segment, and an integer s to the symbols in the second segment. A further improvement of [14] was presented by Pelusi *et al*. [15]. More details regarding Swart and Weber's q-ary scheme are provided in the next section.

Other work closely related to q-ary balancing, but with different alphabets or constraints, include balancing codes over the q-th roots of unity [16], [17] and balancing codes that are invariant under symbol permutation [18]. For the former, the non-binary, complex alphabet is chosen as the q-th roots of unity, e.g. when $q = 4$, the alphabet is $\{+1, +j, -1, -j\}$, and the complex sum of the symbols in a codeword must be zero. For the latter, each alphabet symbol occurs as many times as any other symbol in the codeword, and can thus be seen as a

This paper was presented in part at the IEEE Information Theory Workshop, Seville, Spain, September 2013.

T. G. Swart is with the Department of Electrical and Electronic Engineering Science, University of Johannesburg, Auckland Park, 2006, South Africa. (e-mail: tgswart@uj.ac.za). J. H. Weber is with the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2628 Delft, The Netherlands and a visiting professor with the Department of Electrical and Electronic Engineering Science, University of Johannesburg, Auckland Park, 2006, South Africa (e-mail: j.h.weber@tudelft.nl). K. A. Schouhamer Immink is with Turing Machines Inc, 3016 DK Rotterdam, The Netherlands (e-mail: immink@turing-machines.com).

This work is based on research supported in part by the National Research Foundation of South Africa (UID 77596).

special case of q-ary balancing.

In most Knuth-like balancing schemes, both binary and non-binary, the encoder appends a prefix, which is required by the decoder for restoring the original user word. However, for some of these schemes look-up tables are required for encoding and decoding the prefix, which is undesirable for certain high-speed applications. Schemes like [12] require either a very small look-up table or the check word (i.e. prefix) can be obtained by direct computation of the user word's weight. In this paper, we present a simple prefixless scheme, which extends the work by Swart and Immink [19], see also Section III. As in [19], we add error correction capabilities to the balancing scheme, which can correct single channel errors, and in this paper we show that fast decoding can be done based on syndromes only.

In Sections II and III, we present relevant results from the literature. We also detail Swart and Immink's [19] method for constructing prefixless q-ary balanced codes in Section III. In Section IV, we show how error correction capabilities can be efficiently added to the balancing act. In Section V, we investigate the redundancy, complexity, and performance of the new scheme. In Section VI we discuss and highlight certain aspects of the work along with directions for future research, and finally in Section VII we present our conclusions.

II. Balancing of q-ary Sequences

The following definitions will be used in this paper. Let $\boldsymbol{x} = (x_1, \ldots, x_m)$ be a sequence of m symbols taken from the q-ary alphabet $\mathcal{Q} = \{0, 1, \ldots, q-1\}$, with q and m positive integers and $q \geq 2$. The *weight* of $\boldsymbol{x}$, denoted by weight($\boldsymbol{x}$), is defined as the real sum of the m q-ary symbols, i.e.

$$\text{weight}(\boldsymbol{x}) = \sum_{i=1}^{m} x_i.$$

We further define the *balancing value* by

$$\Omega_{q,m} = \frac{m(q-1)}{2}, \tag{1}$$

where q and m are chosen such that $\Omega_{q,m}$ is an integer. A codeword $\boldsymbol{x}$ of length m is said to be balanced if

$$\text{weight}(\boldsymbol{x}) = \Omega_{q,m}. \tag{2}$$

Alternatively, an alphabet with polar symbols can be considered, where

$$\mathcal{Q}_{\text{odd}} = \{-\tfrac{q-1}{2}, \ldots, -2, -1, 0, +1, +2, \ldots, +\tfrac{q-1}{2}\},$$

if q is odd, and

$$\mathcal{Q}_{\text{even}} = \{-(q-1), \ldots, -3, -1, +1, +3, \ldots, +(q-1)\},$$

if q is even. In this case, balancing is achieved when the symbol sum equals zero, i.e. when weight($\boldsymbol{x}$) $= 0$. The conversion between the two representations is straightforward, thus for clerical convenience we only use the former representation.

To keep this paper as self-contained as possible, we summarize the most important results from [14] without proofs.

A q-ary sequence can be balanced by adding (modulo q) an appropriate q-ary *balancing sequence*, as defined in the following.

Definition 1: A q-ary balancing sequence of length m is denoted by $\boldsymbol{b}_{s,v} = (b_1, \ldots, b_m)$, $s \in \mathcal{Q}$, $v \in \{1, \ldots, m\}$, with

$$b_i = \begin{cases} s+1 \pmod q, & i \leq v, \\ s, & \text{otherwise.} \end{cases}$$

The balancing sequence can be seen as two sequences, consisting of a q-ary sequence (the all-s sequence) and a "binary" sequence (indicating the position). Then,

$$\boldsymbol{b}_{s,v} = (s, s, s, \ldots, s) \oplus_q (\overbrace{1, \ldots, 1}^{v}, 0, \ldots, 0),$$

where $\oplus_q$ represents modulo q summation.

Example 1: Consider a 4-ary sequence $\boldsymbol{x} = (0, 2, 3, 3, 3, 1, 3, 2)$, of weight 17. By adding the sequence $(2, 2, 2, 2, 2, 2, 2, 1)$, we can obtain a balanced sequence $(2, 0, 1, 1, 1, 3, 1, 3)$ with the weight equal to $\Omega_{4,8} = 12$. The balancing sequence is not unique, and in this case three more balancing sequences can be found, namely $(3, 3, 3, 2, 2, 2, 2, 2)$, $(3, 3, 3, 3, 3, 3, 3, 2)$ and $(0, 0, 0, 3, 3, 3, 3, 3)$.

To see how the balancing sequences affect the weight, let $\boldsymbol{b}_z$ denote the z-th balancing sequence, with

$$z = sm + v, \quad 1 \leq z \leq qm,$$

and let $\omega(z) = \text{weight}(\boldsymbol{x} \oplus_q \boldsymbol{b}_z)$. Note there are qm possible balancing sequences.

For the binary case, we know from [4] that the minimum and maximum values of $\omega(z)$ will always be such that $\min\{\omega(z)\} \leq \Omega_{2,m} \leq \max\{\omega(z)\}$. The increase and decrease in $\omega(z)$ plotted against z will always be one, and therefore it must pass through $\Omega_{2,m}$ at some stage. A progression of this nature, that consists of a succession of random steps, is called a *random walk*, which is the basis of Knuth's proof. For q-ary balancing in [12], Tallini and Vaccaro construct single or double maps in such a way that random walks are also achieved, with $\omega(z)$ changing by -1, 0 or $+1$ for each step, thereby ensuring that it will pass through $\Omega_{q,m}$. The approach from [14] also results in random walks, but the value of $\omega(z)$ does not change by -1, 0 or $+1$, as described in the following lemma.

Lemma 1: When adding $\boldsymbol{b}_z$ to $\boldsymbol{x}$, $\omega(z)$ forms a random walk with increases of 1 and decreases of $q-1$.

An exact minimum and maximum value for $\omega(z)$ is hard to find since it is sequence dependent, but it can be shown that a bound exists on these values for all sequences.

Lemma 2: The $\omega(z)$-random walk has $\min\{\omega(z)\} \leq \Omega_{q,m} \leq \max\{\omega(z)\}$.

Using Lemmas 1 and 2 we can state the main result from [14]:

Theorem 1: There is at least one pair of integers, s and v, $s \in \mathcal{Q}$, $v \in \{1, \ldots, m\}$, such that $\boldsymbol{x} \oplus_q \boldsymbol{b}_{s,v}$ is balanced.

This result is used in the next section to construct the prefixless balanced codes from [19].

III. Prefixless Balanced Codes

As before, let $\boldsymbol{x} = (x_1, \ldots, x_m)$ be a q-ary word of m symbols, $x_i \in \mathcal{Q}$. The word $\boldsymbol{d} = (d_1, \ldots, d_m)$ is obtained by

modulo q integration[1] of $\boldsymbol{x}$

$$d_i = d_{i+1} \oplus_q x_i, \quad 1 \le i \le m,$$

where $d_{m+1} = 0$. The above integration operation will be denoted by $\boldsymbol{d} = I(\boldsymbol{x})$. Note that the original word $\boldsymbol{x}$ can be uniquely restored by modulo q *differentiation*:

$$x_i = d_i \ominus_q d_{i+1}, \quad 1 \le i \le m, \tag{3}$$

where $\ominus_q$ indicates modulo q subtraction. The above differentiation operation will be denoted by $\boldsymbol{x} = I^{-1}(\boldsymbol{d})$. Clearly, $I^{-1}(I(\boldsymbol{x})) = \boldsymbol{x}$.

Define the binary m-bit word $\boldsymbol{u}_v = (\overbrace{0\ldots0}^{v-1}1\overbrace{0\ldots0}^{m-v})$. We are now in a position to formulate Theorem 2.

Theorem 2: There is at least one pair of integers, s and v, $s \in \mathcal{Q}$, $v \in \{1, \ldots, m\}$, such that $\boldsymbol{w} = I(\boldsymbol{x} \oplus_q \boldsymbol{u}_v \oplus_q s\boldsymbol{u}_m)$ is balanced, that is weight$(\boldsymbol{w}) = \Omega_{q,m}$.

Since $\boldsymbol{u}_v$ and $s\boldsymbol{u}_m$ under modulo q integration are equivalent to $\boldsymbol{b}_{s,v}$, according to Theorem 1 balancing of $\boldsymbol{x}$ will always be possible. Note that in the binary case, $q = 2$, the search simplifies to finding the balancing index v only, since such an index will always be found for $s = 0$. The following example illustrates the method.

Example 2: Let $q = 5$ and $m = 7$, and let the input be $\boldsymbol{x} = (3, 2, 0, 1, 1, 4, 0)$. After a search we find that the choice of $s = 3$ and $v = 3$ balances the integrated sequence $I(\boldsymbol{x}) = (1, 3, 1, 1, 0, 4, 0)$. Adding $\boldsymbol{u}_3 \oplus_5 3\boldsymbol{u}_7 = (0, 0, 1, 0, 0, 0, 3)$ to the input yields $(3, 2, 1, 1, 1, 4, 3)$, and after integration we obtain $\boldsymbol{w} = I((3, 2, 1, 1, 1, 4, 3)) = (0, 2, 0, 4, 3, 2, 3)$. Note that $\boldsymbol{w}$ is balanced since the sum of its components equals $\Omega_{5,7} = 14$.

It is worth noting that determining s and v can be simplified by first finding an s such that weight$(\boldsymbol{x} \oplus_q \boldsymbol{b}_{s,0}) \le \Omega_{q,m}$ and weight$(\boldsymbol{x} \oplus_q \boldsymbol{b}_{s+1,0}) \ge \Omega_{q,m}$ and then finding the v that balances the sequence. We will elaborate on the complexity for this in Section V-B.

From the receiver's point of view, $\boldsymbol{u}_v$ introduced an error of magnitude one in $\boldsymbol{x}$ in an unknown position. In the rest of the paper we will refer to this as the *magnitude-one error*. The next encoding and decoding algorithms exploit Theorem 2 and we will show that in conjunction with error correcting techniques to correct the magnitude-one error, it will be possible to efficiently balance q-ary words, and circumvent the encoding and decoding of the prefix in the prior art constructions.

A. Encoding

We will make use of a q-ary $(m-1, k)$ linear block code, denoted as $\mathcal{C}$, of dimension k and length $m-1$ to encode the user word, $\boldsymbol{a} = (a_1, \ldots, a_k)$, of length k. Let $r' = m-1-k$ be the redundancy of the block code, and define the $r' \times (m-1)$ matrix $\boldsymbol{H}_{q,r'}$ whose i-th column $\boldsymbol{h}_i$ is the q-ary representation of the integer i, $1 \le i \le m-1$, $m \le q^{r'}$. For example, for $q = 3$, $r' = 2$, and $m = 9$ we obtain

$$\boldsymbol{H}_{3,2} = \begin{bmatrix} 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \end{bmatrix}. \tag{4}$$

We call $\boldsymbol{H}_{q,r'}$ a *check matrix*, for which we have an easy syndrome decoding available similar to that of binary Hamming codes [20]. The code $\mathcal{C}$ is a single, magnitude-one error correction code, or a single error detection code, as described in [21], [22], [23]. The maximum row length of the check matrix $\boldsymbol{H}_{q,r'}$ is $q^{r'} - 1$, $r' > 1$. The encoding function is denoted by $\boldsymbol{x} = \phi_q(\boldsymbol{a})$, and is defined in such a way that $\boldsymbol{x}$ satisfies $\boldsymbol{H}_{q,r'}\boldsymbol{x}^T = \boldsymbol{0}^T$.

The encoding procedure consists of the following three steps:

Step 1: The k-symbol user word, $\boldsymbol{a}$, is encoded into the codeword $\boldsymbol{x} = (x_1, \ldots, x_{m-1})$ using the q-ary $(m-1, k)$ linear block code, i.e. $\boldsymbol{x} = \phi_q(\boldsymbol{a})$.

Step 2: The m-symbol word $\boldsymbol{x}'$ is obtained by appending a redundant '0' to $\boldsymbol{x}$, that is, $\boldsymbol{x}' = (x_1, \ldots, x_{m-1}, 0)$.

Step 3: Find a pair of integers, $s \in \mathcal{Q}$ and $v \in \{1, \ldots, m\}$, such that $\boldsymbol{w} = I(\boldsymbol{x}' \oplus_q \boldsymbol{u}_v \oplus_q s\boldsymbol{u}_m)$, with weight$(\boldsymbol{w}) = \Omega_{q,m}$. (According to Theorem 1, such a pair of integers s and v can always be found.)

Example 3: Let $q = 5$ and $k = 2$, and let the user word be $\boldsymbol{a} = (3, 2)$. Using the shortened linear code with generator matrix

$$\boldsymbol{G}_{5,2} = \begin{bmatrix} 1 & 0 & 1 & 1 & 3 & 2 \\ 0 & 1 & 1 & 4 & 1 & 4 \end{bmatrix},$$

and check matrix

$$\boldsymbol{H}_{5,2} = \begin{bmatrix} 1 & 2 & 3 & 4 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix},$$

the user word is encoded as $\boldsymbol{x} = (3, 2, 0, 1, 1, 4)$ and after appending a redundant '0' we have $\boldsymbol{x}' = (3, 2, 0, 1, 1, 4, 0)$. This is the same sequence used in Example 2 and will thus be balanced as $\boldsymbol{w} = (0, 2, 0, 4, 3, 2, 3)$.

B. Decoding

At the receiver side, the m-symbol word $\boldsymbol{y}'$ is retrieved from the received $\boldsymbol{w}$ by modulo q differentiation, i.e.

$$\boldsymbol{y}' = I^{-1}(\boldsymbol{w}) = \boldsymbol{x}' \oplus_q \boldsymbol{u}_v \oplus_q s\boldsymbol{u}_m.$$

We drop the last symbol, 's' (or '$s+1$' if $v = m$), of $\boldsymbol{y}'$ and thereby obtain $\boldsymbol{y}$ of length $m-1$. Then either the words $\boldsymbol{y}$ and $\boldsymbol{x}$ differ only at an unknown index position v if $v \ne m$, because of the magnitude-one "error" introduced during encoding, or $\boldsymbol{y} = \boldsymbol{x}$ if $v = m$.

As $\boldsymbol{x}$ satisfies $\boldsymbol{H}_{q,r'}\boldsymbol{x}^T = \boldsymbol{0}^T$, we have

$$\boldsymbol{H}_{q,r'}\boldsymbol{y}^T = \boldsymbol{H}_{q,r'}(\boldsymbol{x} \oplus_q \boldsymbol{u}_v)^T = \begin{cases} \boldsymbol{h}_v, & \text{if } 1 \le v \le m-1, \\ \boldsymbol{0}^T, & \text{if } v = m, \end{cases}$$

where $\boldsymbol{h}_i$ is the i-th column of $\boldsymbol{H}_{q,r'}$. Thus, we can uniquely retrieve the index v, and restore the original word by subtracting '1' from y_v, i.e. $\boldsymbol{x} = \boldsymbol{y} \ominus_q \boldsymbol{u}_v$.

By removing the redundant symbols, we obtain the original user word $\boldsymbol{a}$.

[1]Note that for convenience we perform the integration from right to left. Similar results will be obtained if it was performed from left to right, i.e. $d_{i+1} = d_i \oplus_q x_i$ with $d_0 = 0$.

Example 4: Using the balanced codeword obtained in Example 3 as our received word $\boldsymbol{w} = (0,2,0,4,3,2,3)$, we apply modulo q differentiation to obtain $\boldsymbol{y}' = I^{-1}(0,2,0,4,3,2,3) = (3,2,1,1,1,4,3)$. We then drop the last symbol to get $\boldsymbol{y} = (3,2,1,1,1,4)$. By multiplying $\boldsymbol{y}$ with the shortened check matrix

$$\begin{bmatrix}1&2&3&4&0&1\\0&0&0&0&1&1\end{bmatrix} \times \begin{bmatrix}3&2&1&1&1&4\end{bmatrix}^T = \begin{bmatrix}3\\0\end{bmatrix},$$

the third column is identified representing $v = 3$, thus $\boldsymbol{x} = (3,2,0,1,1,4)$ and the original information is retrieved as $\boldsymbol{a} = (3,2)$.

IV. Adding Error Correction

So far the error correction techniques employed were used to identify the index v that was used during the encoding process. We will now extend the error correction code that was used in such a way that we will be able to correct single channel errors.

From (3), any single channel error in, say w_j, will be transformed into a double adjacent error, in x_{j-1} and x_j. This, together with the single "error" we introduced during balancing, means that we must be able to correct three errors. However, this would come at a price of much more redundancy. We can avoid this by extending our code used in the previous section and by introducing interleaving.

A. Encoding

We start with a code similar to that used in Section III-A, but extend it by adding a check symbol that checks all the previous symbols and denote this extended code by $\mathcal{C}^*$. If q is odd[2] we then choose an odd value for m as well and set $n = \frac{m-1}{2}$. The code $\mathcal{C}^*$ then forms an (n,k) linear block code, with redundancy of $r^* = n-k$ and check matrix $\boldsymbol{H}^*_{q,r^*}$. In general, the check matrix will be of size $r^* \times n$, with the i-th column $\boldsymbol{h}^*_i$ being the q-ary representation of the integer $(q^{r^*-1}+i)$, $1 \le i \le n$. As an example, the check matrix in (4) becomes

$$\boldsymbol{H}^*_{3,3} = \begin{bmatrix}1&2&0&1&2&0&1&2\\0&0&1&1&1&2&2&2\\1&1&1&1&1&1&1&1\end{bmatrix}.$$

We now formally define the syndrome, $\boldsymbol{s} = (s_1, s_2, \ldots, s_{r^*})^T$, as it is generally used for Hamming codes, by

$$\boldsymbol{s} = \boldsymbol{H}^*_{q,r^*}\hat{\boldsymbol{w}}^T,$$

where $\hat{\boldsymbol{w}}$ is the (possibly corrupted) received codeword. The syndrome is then equal to the summation (modulo q) of those columns, multiplied by the error magnitudes, where the errors occurred. For clerical convenience in the rest of the paper, we neglect indicating that modular arithmetic is used when calculating the syndromes.

Note that this code is a special case of the class of t symmetric error correction and all unidirectional error detection (tEC-AUED) codes with $t = 1$. The reader can refer to [21], [22], [23], [24] and references therein for more details. We will further elaborate on the use of these codes in Section VI. To make this paper self-contained, we include the following lemma.

Lemma 3: The code $\mathcal{C}^*$ with check matrix $\boldsymbol{H}^*_{q,r^*}$ can:

(i) detect a single magnitude-one error and a single random error, for any value of q, or
(ii) correct a single magnitude-one error, for any value of q, or
(iii) correct a single random error, for q any integer power of a prime value.

Proof: Let the magnitude-one error be in position i and the random error, with error magnitude e, $e \in \mathcal{Q}$, be in position j. The resulting syndrome is $\boldsymbol{s} = \boldsymbol{h}^*_i + e\boldsymbol{h}^*_j$. We prove each case individually:

(i) $s_{r^*} \neq 0$ for all e except when $e = q-1$ (since $s_{r^*} = 1 \oplus_q (q-1) = 0$). If $e = q-1$, then $\boldsymbol{s} = \boldsymbol{0}^T$ only if $i = j$, which would mean that the magnitude-one error and the random error "cancel" each other out. Therefore, a single magnitude-one error and a single random error can always be detected based on $\boldsymbol{s} \neq \boldsymbol{0}^T$.

(ii) If $e = 0$ (there is no random error), then with $\boldsymbol{s} = \boldsymbol{h}^*_i$ a magnitude-one error can always be corrected.

(iii) If there is no magnitude-one error, then $\boldsymbol{s} = e\boldsymbol{h}^*_j$. Since we can find e from s_{r^*}, we can retrieve j from $\boldsymbol{h}^*_j = e^{-1}\boldsymbol{s}$. As the modular multiplicative inverse, e^{-1}, is needed, this case is limited to q being integer powers of a prime value. Therefore, a single random error can always be corrected. ∎

We have two user words $\boldsymbol{a}$ and $\boldsymbol{a}'$, each of length k, that are encoded into codewords of length n, $\boldsymbol{c} = (c_1, c_2, \ldots, c_n)$ and $\boldsymbol{c}' = (c'_1, c'_2, \ldots, c'_n)$ respectively, using the code $\mathcal{C}^*$. The encoding function for this code is defined as $\boldsymbol{c} = \phi^*_q(\boldsymbol{a})$. Interleave these two codewords to a depth of two, to form

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_{m-1}) = (c_1, c'_1, c_2, c'_2, \ldots, c_n, c'_n).$$

The encoding now follows the same steps as in Section III-A to add a redundant '0', to find the values s and v to balance the sequence and to encode it into $\boldsymbol{w}$. The final encoding step is to append symbols α and β to $\boldsymbol{w}$, where

$$\alpha = w_1 \oplus_q w_3 \oplus_q \cdots \oplus_q w_m \oplus_q \delta_{q,m}, \text{ and}$$
$$\beta = w_2 \oplus_q w_4 \oplus_q \cdots \oplus_q w_{m-1},$$

with $\delta_{q,m} \equiv (q-1) - \Omega_{q,m} \pmod q$.

Lemma 4: The sequence (α, β) is balanced.

Proof: Adding the two check symbols together:

$$\begin{aligned}\alpha + \beta &\equiv w_1 + w_3 + \cdots + w_m + \delta_{q,m} + w_2 \\ &\quad + w_4 + \cdots + w_{m-1} \pmod q \\ &\equiv w_1 + w_2 + w_3 + w_4 + \cdots + w_{m-1} \\ &\quad + w_m + (q-1) - \Omega_{q,m} \pmod q \\ &\equiv \Omega_{q,m} + (q-1) - \Omega_{q,m} \pmod q \\ &\equiv q - 1 \pmod q.\end{aligned}$$

Since $0 \le \alpha, \beta \le q-1$, it must hold that $\alpha + \beta = q-1$, and thus (α, β) is balanced. ∎

[2]If q is even, then instead of using $n = \frac{m-1}{2}$ and $\boldsymbol{x}' = (x'_1, \ldots, x'_{m-1}, 0)$, use $n = \frac{m-2}{2}$ and $\boldsymbol{x}' = (x'_1, \ldots, x'_{m-2}, 0, 0)$ with m even, so that the overall length is even and balancing can be achieved.

In essence, α and β are check symbols over the odd and even symbols respectively, and $\delta_{q,m}$ is added to ensure that α and β together are balanced. The sender then sends the balanced sequence $(w_1, w_2, \ldots, w_m, \alpha, \beta)$ to the receiver. The encoding process is summarized in Fig. 1.

The following example illustrates the encoding process.

Example 5: We consider a $q = 5$, $(4, 2)$ linear block code with a generator matrix

$$\boldsymbol{G}^*_{5,2} = \begin{bmatrix} 1 & 0 & 2 & 2 \\ 0 & 1 & 3 & 1 \end{bmatrix},$$

and check matrix

$$\boldsymbol{H}^*_{5,2} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Let the user words be $\boldsymbol{a} = (4, 0)$ and $\boldsymbol{a}' = (2, 1)$. Using $\boldsymbol{G}^*_{5,2}$, these are encoded as $\boldsymbol{c} = (4, 0, 3, 3)$ and $\boldsymbol{c}' = (2, 1, 2, 0)$. After interleaving and adding the redundant zero, we have $\boldsymbol{x}' = (4, 2, 0, 1, 3, 2, 3, 0, 0)$. According to Theorem 2, we can find $s = 1$ and $v = 4$, then $\boldsymbol{w} = I((4, 2, 0, 1, 3, 2, 3, 0, 0) \oplus_5 (0, 0, 0, 0, 0, 0, 0, 0, 1) \oplus_5 (0, 0, 0, 1, 0, 0, 0, 0, 0)) = I((4, 2, 0, 2, 3, 2, 3, 0, 1)) = (2, 3, 1, 1, 4, 1, 4, 1, 1)$. The check symbols are calculated as $\alpha = 3$ and $\beta = 1$, with $\delta_{5,9} = 1$. The transmitted sequence $(2, 3, 1, 1, 4, 1, 4, 1, 1, 3, 1)$ is balanced with $\Omega_{5,11} = 22$.

B. Decoding

Previously in [19], decoding was done exhaustively by trying to correct the error in every even (or odd) position, until the syndromes were found to be zero. For each attempt at correcting the error, modulo q differentiation, deinterleaving and syndrome calculation had to be performed. However, this can negatively affect the decoding time if the length of the sequence becomes very long. Instead of this exhaustive decoding, we will show that we can decode once and correct the error by making use of the syndromes directly.

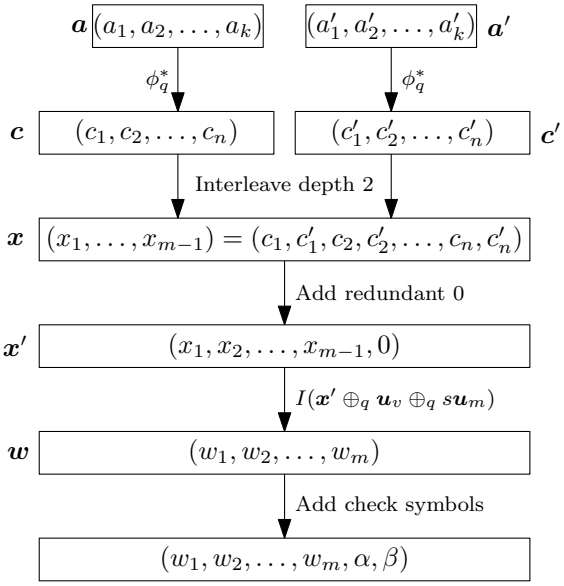

Fig. 1. Summary of encoding algorithm for q odd

Now we will define notations to be used in the following description of the decoding process. Let $\hat{\boldsymbol{w}}$ be the (possibly corrupted) received codeword and $\hat{\alpha}$ and $\hat{\beta}$ the (possibly corrupted) received check symbols. Let $\hat{\boldsymbol{x}}$ be the sequence after applying modulo q differentiation and dropping the last redundant symbol, with $\hat{\boldsymbol{c}}$ and $\hat{\boldsymbol{c}}'$ the codewords recovered after deinterleaving. Let $\boldsymbol{s} = (s_1, s_2, \ldots, s_{r^*})^T$ and $\boldsymbol{s}' = (s'_1, s'_2, \ldots, s'_{r^*})^T$ be the syndromes calculated by multiplying $\hat{\boldsymbol{c}}$ and $\hat{\boldsymbol{c}}'$ with the check matrix $\boldsymbol{H}^*_{q,r^*}$, respectively. Finally, let $\bar{\boldsymbol{c}}$ and $\bar{\boldsymbol{c}}'$ be the codewords after correction is applied, with $\hat{\boldsymbol{a}}$ and $\hat{\boldsymbol{a}}'$ the recovered user words. The decoding process is summarized in Fig. 2, where $(\phi^*_q)^{-1}$ is used to denote the inverse operation of ϕ^*_q. If the error correction was successful, then we will have $\boldsymbol{a} = \hat{\boldsymbol{a}}$ and $\boldsymbol{a}' = \hat{\boldsymbol{a}}'$.

We define the *imbalance* as the difference between the weight of the received sequence and the known weight of the balanced sequence, i.e. $\sum_{i=1}^{m} \hat{w}_i - \Omega_{q,m}$. From this imbalance the error magnitude can be determined, provided a single channel error occurred. By using check symbols α and β we can also determine whether the channel error occurred in an even or odd position.

As was seen in the proof of Lemma 3, the modular multiplicative inverse will be needed during decoding, which means that the algorithm as described here is limited to prime values of alphabet size q, or integer powers of a prime value if it is adapted to work in an extension field.

We assume that the random channel error occurred in position t when considering $\hat{\boldsymbol{w}}$, and that the intentional magnitude-one error occurred in position v when considering $\hat{\boldsymbol{x}}$.

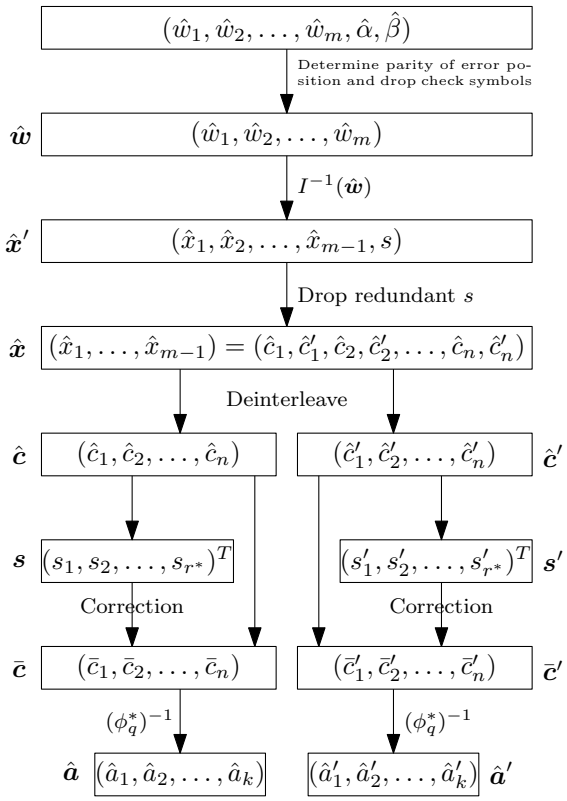

Fig. 2. Summary of decoding algorithm for q odd.

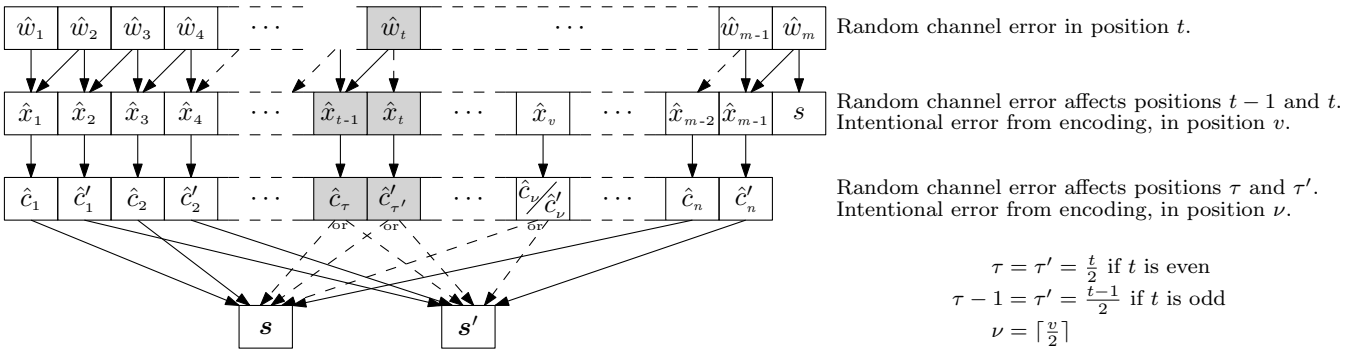

Fig. 3. Effect of errors on symbols from one decoding stage to the next

TABLE I
CLASSIFICATION OF ERROR LOCATIONS FOR THE MAGNITUDE-ONE ERROR AND THE CHANNEL ERROR

Magnitude-one error in position v		Channel error in position t	
State	Error Locations	State	Error Locations
A	v odd, $1 \le v \le m-2$	0	No channel error
B	v even, $2 \le v \le m-1$	1	$t = 1$
C	$v = m$	2	t even, $2 \le t \le m-1$
		3	t odd, $3 \le t \le m-2$
		4	$t = m$

The following observations are crucial to note, using Fig. 3 as a guide:

- a random channel error in $\hat{w}_1$ will only affect $\hat{x}_1$ and thus only $\hat{c}_1$, and similarly an error in $\hat{w}_m$ will only affect $\hat{x}_{m-1}$ and thus only $\hat{c}'_n$,
- a random channel error of magnitude e in $\hat{w}_t$, $2 \le t \le m-1$, will affect $\hat{x}_t$ by e and $\hat{x}_{t-1}$ by $-e$ after modulo q differentiation,
- similarly, a random channel error in $\hat{w}_t$, $2 \le t \le m-1$, will affect $\hat{c}_\tau$ and $\hat{c}'_{\tau'}$ after modulo q differentiation and deinterleaving, where $\tau = \tau' = \frac{t}{2}$ if t is even, or $\tau - 1 = \tau' = \frac{t-1}{2}$ if t is odd, with $1 \le \tau, \tau' \le n$,
- an intentional magnitude-one error in position v in $\hat{\boldsymbol{x}}$, will be in position $\nu = \lceil \frac{v}{2} \rceil$, $1 \le \nu \le n$, in $\hat{\boldsymbol{c}}$ if v is even or $\hat{\boldsymbol{c}}'$ if v is odd, after deinterleaving,
- $\hat{x}_m$ does not affect any syndromes, $\hat{x}_1, \hat{x}_3, \hat{x}_5, \ldots, \hat{x}_{m-2}$ affect $\boldsymbol{s}$, and $\hat{x}_2, \hat{x}_4, \hat{x}_6, \ldots, \hat{x}_{m-1}$ affect $\boldsymbol{s}'$,
- a random channel error of magnitude e in position τ will result in a syndrome $\boldsymbol{s} = e\boldsymbol{h}^*_\tau$, and consequently we have to multiply by e^{-1} before we can determine the τ-th column of $\boldsymbol{H}^*_{q,r^*}$, i.e. $\boldsymbol{h}^*_\tau = e^{-1}\boldsymbol{s}$ with

$$\tau = \sum_{i=1}^{r^*-1} q^{i-1}[e^{-1}s_i \pmod q]. \tag{5}$$

  The same applies to τ' and $\boldsymbol{s}'$.
- a random channel error's effect on $\hat{\boldsymbol{c}}$ and $\hat{\boldsymbol{c}}'$ are determined from s_{r^*} and s'_{r^*} respectively, since all columns of $\boldsymbol{H}^*_{q,r^*}$ have ones in the r^*-th position.

Table I now lists the possible error locations that will be affected by both types of errors and classifies them into states, to be used in the decoding process. By pairing the states together, we obtain all the possible error scenarios as listed in Table II, along with the complete syndromes. The values of s_{r^*} and s'_{r^*} are used to determine the error state. It is now also evident why it is necessary to determine the position parity of the channel error, e.g. for $q = 7$ and $e = 3$ one is unable to distinguish between states B2 and A3 as both are valid with $s_{r^*} = 4$ and $s'_{r^*} = 4$.

TABLE II
SYNDROME VALUES BASED ON THE POSSIBLE ERROR STATES (ALL OPERATIONS PERFORMED MODULO q)

Error State	$\boldsymbol{s}$	$\boldsymbol{s}'$	s_{r^*}	s'_{r^*}	Note
A0	$\boldsymbol{h}^*_\nu$	$\boldsymbol{0}$	1	0	
A1	$\boldsymbol{h}^*_\nu + e\boldsymbol{h}^*_\tau$	$\boldsymbol{0}$	$1+e$	0	$\tau = 1$
A2	$\boldsymbol{h}^*_\nu - e\boldsymbol{h}^*_\tau$	$e\boldsymbol{h}^*_{\tau'}$	$1-e$	e	$\tau = \tau'$
A3	$\boldsymbol{h}^*_\nu + e\boldsymbol{h}^*_\tau$	$-e\boldsymbol{h}^*_{\tau'}$	$1+e$	$-e$	$\tau - 1 = \tau'$
A4	$\boldsymbol{h}^*_\nu$	$-e\boldsymbol{h}^*_{\tau'}$	1	$-e$	$\tau' = n$
B0	$\boldsymbol{0}$	$\boldsymbol{h}^*_\nu$	0	1	
B1	$e\boldsymbol{h}^*_\tau$	$\boldsymbol{h}^*_\nu$	e	1	$\tau = 1$
B2	$-e\boldsymbol{h}^*_\tau$	$\boldsymbol{h}^*_\nu + e\boldsymbol{h}^*_{\tau'}$	$-e$	$1+e$	$\tau = \tau'$
B3	$e\boldsymbol{h}^*_\tau$	$\boldsymbol{h}^*_\nu - e\boldsymbol{h}^*_{\tau'}$	e	$1-e$	$\tau - 1 = \tau'$
B4	$\boldsymbol{0}$	$\boldsymbol{h}^*_\nu - e\boldsymbol{h}^*_{\tau'}$	0	$1-e$	$\tau' = n$
C0	$\boldsymbol{0}$	$\boldsymbol{0}$	0	0	
C1	$e\boldsymbol{h}^*_\tau$	$\boldsymbol{0}$	e	0	$\tau = 1$
C2	$-e\boldsymbol{h}^*_\tau$	$e\boldsymbol{h}^*_{\tau'}$	$-e$	e	$\tau = \tau'$
C3	$e\boldsymbol{h}^*_\tau$	$-e\boldsymbol{h}^*_{\tau'}$	e	$-e$	$\tau - 1 = \tau'$
C4	$\boldsymbol{0}$	$-e\boldsymbol{h}^*_{\tau'}$	0	$-e$	$\tau' = n$

$\nu = \lceil \frac{v}{2} \rceil$, $\tau = \tau' = \frac{t}{2}$ if t is even, $\tau - 1 = \tau' = \frac{t-1}{2}$ if t is odd

The flow diagram, see Fig. 4, and the values of the parameters e, s_{r^*} and s'_{r^*} determine the error state. Once the error state is determined, then $\boldsymbol{s}$ and $\boldsymbol{s}'$ in Table II can be used to determine the error positions using (5). Depending on whether $e = 1$ or $e = q - 1$, certain states may appear equivalent, and during decoding we need to be able to differentiate between these. Here we consider two states to be equivalent if their s_{r^*} and s'_{r^*} values are the same. In that case, we need to determine the error positions to distinguish between the two states.

If only a single error occurred, then decoding will be straightforward as no ambiguity exists. However, since we are working with a constrained code, we can use the constraints to check at different stages in the decoding whether multiple

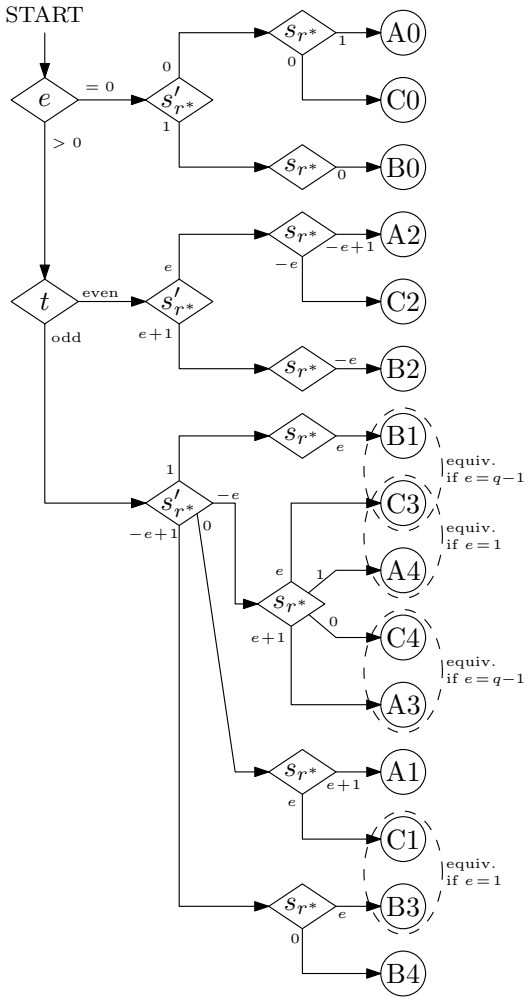

Fig. 4. Flow diagram to determine error states based on syndrome values.

errors possibly occurred and declare a decoding failure if this is detected. Decoding is done according to the following steps.

Step 1: Let Δ be the imbalance of the received sequence, with:

$$\Delta = \sum_{i=1}^{m} \hat{w}_i - \Omega_{q,m}.$$

If $|\Delta| > q-1$, conclude that multiple errors occurred, declare a decoding failure and STOP. Otherwise, proceed to calculate all the necessary values by determining the error magnitude[3] as $e \equiv \Delta \pmod q$ and the check symbols as

$$\gamma = \hat{w}_1 \oplus_q \hat{w}_3 \oplus_q \cdots \oplus_q \hat{w}_m \oplus_q \delta_{q,m} \ominus_q \hat{\alpha},$$
$$\gamma' = \hat{w}_2 \oplus_q \hat{w}_4 \oplus_q \cdots \oplus_q \hat{w}_{m-1} \ominus_q \hat{\beta}.$$

Perform modulo q differentiation with $\hat{\boldsymbol{x}}' = I^{-1}(\hat{\boldsymbol{w}})$, drop the redundant last symbol to obtain $\hat{\boldsymbol{x}}$, deinterleave the codewords to $\hat{\boldsymbol{c}}$ and $\hat{\boldsymbol{c}}'$, and determine $\boldsymbol{s}$ and $\boldsymbol{s}'$.

Step 2: If $\boldsymbol{s} = \boldsymbol{s}' = \boldsymbol{0}^T$ and $\Delta = \gamma = \gamma' = 0$, then proceed to Step 8, otherwise proceed to the next step.

[3]The imbalance, Δ, indicates the error magnitude, but its sign also indicates if we are above or below $\Omega_{q,m}$. This information can be used to check if the error correction was performed successfully. The error magnitude, e, used in the decoding algorithm, however, does not make use of the sign information, as $-e \equiv q - e \pmod q$ in all the calculations.

TABLE III
ERROR STATES AND CORRESPONDING CORRECTIVE ACTION

Error State	Corrective action (all operations done modulo q)
A0	Subtract 1 from $\hat{c}_\nu$
A1	Subtract 1 from $\hat{c}_\nu$, subtract e from $\hat{c}_1$
A2	Subtract 1 from $\hat{c}_\nu$, add e to $\hat{c}_\tau$, subtract e from $\hat{c}'_\tau$
A3	Subtract 1 from $\hat{c}_\nu$, subtract e from $\hat{c}_\tau$, add e to $\hat{c}'_{\tau'}$
A4	Subtract 1 from $\hat{c}_\nu$, add e to $\hat{c}'_n$
B0	Subtract 1 from $\hat{c}'_\nu$
B1	Subtract 1 from $\hat{c}'_\nu$, subtract e from $\hat{c}_1$
B2	Subtract 1 from $\hat{c}'_\nu$, add e to $\hat{c}_\tau$, subtract e from $\hat{c}'_\tau$
B3	Subtract 1 from $\hat{c}'_\nu$, subtract e from $\hat{c}_\tau$, add e to $\hat{c}'_{\tau'}$
B4	Subtract 1 from $\hat{c}'_\nu$, add e to $\hat{c}'_n$
C0	No correction necessary
C1	Subtract e from $\hat{c}_1$
C2	Add e to $\hat{c}_\tau$, subtract e from $\hat{c}'_\tau$
C3	Subtract e from $\hat{c}_\tau$, add e to $\hat{c}'_{\tau'}$
C4	Add e to $\hat{c}'_n$

Step 3: If $\gamma \neq 0$ and $\gamma' = 0$, then t is odd, or if $\gamma = 0$ and $\gamma' \neq 0$, then t is even, and proceed to the next step. Otherwise, if $\Delta \neq 0$ (and since either $\gamma \neq 0$, $\gamma' \neq 0$ or $\gamma = \gamma' = 0$, and the parity of t cannot be determined), then multiple errors occurred. In that case, declare a decoding failure and STOP. Otherwise, proceed.

Step 4: Use e, s_{r^*} and s'_{r^*} together with Table I and Fig. 4 to determine the error state(s). If equivalent error states are obtained, for $e = 1$ (C3 $\equiv$ A4 and C1 $\equiv$ B3) or for $e = q-1$ (C3 $\equiv$ B1 and C4 $\equiv$ A3), then use the next step to determine whether $\tau = 1$ or $\tau' = n$.

Step 5: For the determined error state, use Table II together with the syndromes to solve for τ, τ' and ν, making use of (5). If $\tau \neq \tau'$ for t even, or $\tau+1 \neq \tau'$ for t odd[4], or $0 \not< \tau, \tau' \not< n$, then declare a decoding failure and STOP. If $\hat{w}_{2\tau} - \Delta \notin \mathcal{Q}$ for t even, or if $\hat{w}_{2\tau-1} - \Delta \notin \mathcal{Q}$ for t odd, then declare a decoding failure and STOP. (Note that in this case we are not doing modulo q subtraction, as we are testing whether correcting the original imbalance, Δ, in position t would have resulted in an invalid channel symbol.)

Step 6: Correct the errors by applying the corrective action as described in Table III, using e, τ, τ' and ν.

Step 7: Recalculate the syndromes for the corrected codewords. If $\boldsymbol{s} = \boldsymbol{s}' = \boldsymbol{0}^T$, then proceed to the next step, otherwise declare a decoding failure and STOP.

Step 8: Finish decoding by recovering the user words from the codewords.

Theorem 3: Using the fast syndrome-based algorithm described, a single channel error can be corrected, provided that q is an integer power of a prime number.

Proof: Using γ and γ' we can determine the parity of t. Using the imbalance we can determine e, and from $\boldsymbol{s}$ and $\boldsymbol{s}'$ we can obtain s_{r^*} and s'_{r^*}. All these parameters can then be used to determine the error state from those listed in Table II, as

[4]These conditions are checked in the case where τ and τ' can be determined independently, to test whether the expected result is obtained, e.g. for state C2 or C3 in Table II.

well as to distinguish between equivalent states for instances where $e = 1$ or $e = q - 1$, as described in the decoding algorithm.

By using e, e^{-1}, $\boldsymbol{s}$ and $\boldsymbol{s}'$, and according to Lemma 3, we can solve for ν, τ and τ', since we have two syndromes and two unknowns, recalling that τ and τ' are related. Then correction follows from Table III. ∎

We conclude this section with the following example of decoding.

Example 6: We use the $q = 5$ balanced sequence, $(2, 3, 1, 1, 4, 1, 4, 1, 1, 3, 1)$, obtained in Example 5, and consider three received sequences, where the bold symbol(s) indicate the channel error(s).

- Case 1: The received sequence is $(2, 3, 1, 1, 4, \mathbf{3}, 4, 1, 1, 3, 1)$. We proceed through the decoding steps:
  1) $\Delta = 2$, $e = 2$, $\gamma = 0$ and $\gamma' = 2$. After performing modulo 5 differentiation and dropping the redundant last symbol, we obtain $\hat{\boldsymbol{x}} = (4, 2, 0, 2, 1, 4, 3, 0)$. Deinterleaving produces $\hat{\boldsymbol{c}} = (4, 0, 1, 3)$ and $\hat{\boldsymbol{c}}' = (2, 2, 4, 0)$, and multiplying these with $\boldsymbol{H}^*_{5,2}$ results in the syndromes $\boldsymbol{s} = (4, 3)^T$ and $\boldsymbol{s}' = (3, 3)^T$.
  2) None of the conditions are met, and we proceed to the next step.
  3) According to γ and γ', t is even.
  4) From the syndromes we extract $s_2 = 3$ and $s'_2 = 3$, together with $e = 2$, using Fig. 4 to determine the error state to be B2.
  5) Using Table II, we find $-e\boldsymbol{h}^*_\tau = (4, 3)^T$ and $\boldsymbol{h}^*_\nu + e\boldsymbol{h}^*_{\tau'} = (3, 3)^T$. Keeping in mind that $\tau = \tau'$ and $e^{-1} = 3$, we solve the column positions as $\tau = \tau' = 3$ and $\nu = 2$. Testing $\hat{w}_{2\tau} - \Delta = 3 - 2 = 1 \in \mathcal{Q}$ shows that a valid channel symbol is obtained.
  6) Apply the correction from Table III, then $\bar{\boldsymbol{c}} = (4, 0, 3, 3)$ and $\bar{\boldsymbol{c}}' = (2, 1, 2, 0)$.
  7) Recalculating the syndromes results in $\boldsymbol{s} = \boldsymbol{0}^T$ and $\boldsymbol{s}' = \boldsymbol{0}^T$. Proceed to the next step.
  8) According to $\boldsymbol{G}^*_{5,2}$, the information symbols are in positions 1 and 2. Thus $\hat{\boldsymbol{a}} = (4, 0)$ and $\hat{\boldsymbol{a}}' = (2, 1)$, which agrees with the original $\boldsymbol{a}$ and $\boldsymbol{a}'$ from Example 3.
- Case 2: The received sequence is $(\mathbf{1}, 3, 1, 1, 4, 1, 4, 1, 1, 3, 1)$. We proceed through the decoding steps:
  1) $\Delta = -1$, $e = 4$, $\gamma = -1$ and $\gamma' = 0$. We obtain $\hat{\boldsymbol{x}} = (3, 2, 0, 2, 3, 2, 3, 0)$, deinterleaving produces $\hat{\boldsymbol{c}} = (3, 0, 3, 3)$ and $\hat{\boldsymbol{c}}' = (2, 2, 2, 0)$, and the syndromes $\boldsymbol{s} = (4, 4)^T$ and $\boldsymbol{s}' = (2, 1)^T$ are obtained.
  2) None of the conditions are met, and we proceed to the next step.
  3) t is odd.
  4) $s_2 = 4$ and $s'_2 = 1$, together with $e = 4$. Use Fig. 4 to determine the error state to be B1 or C3.
  5) From Table II, since $\boldsymbol{s} = e\boldsymbol{h}^*_\tau$ for both possible error states, solving τ will enable us to distinguish between the two states. Multiplying $\boldsymbol{s}$ by $e^{-1} = 4$, determines that $\tau = 1$, and thus the error state is B1. With τ solved, it can straightforwardly be found that $\nu = 2$. Testing $\hat{w}_{2\tau-1} - \Delta = 1 - (-1) = 2 \in \mathcal{Q}$ shows that a valid channel symbol is obtained.
  6) Apply the correction from Table III, making $\hat{\boldsymbol{c}} = (4, 0, 3, 3)$ and $\hat{\boldsymbol{c}}' = (2, 1, 2, 0)$.
  7) Recalculating the syndromes gives $\boldsymbol{s} = \boldsymbol{0}^T$ and $\boldsymbol{s}' = \boldsymbol{0}^T$. Proceed to the next step.
  8) $\hat{\boldsymbol{a}} = (4, 0)$ and $\hat{\boldsymbol{a}}' = (2, 1)$, which agrees with the original $\boldsymbol{a}$ and $\boldsymbol{a}'$ from Example 3.
- Case 3: The received sequence is $(2, 3, 1, \mathbf{3}, 4, \mathbf{2}, 4, 1, 1, 3, 1)$. We proceed through the decoding steps:
  1) $\Delta = 3$, $e = 3$, $\gamma = 0$ and $\gamma' = 3$. We obtain $\hat{\boldsymbol{x}} = (4, 2, 3, 4, 2, 3, 3, 0)$, deinterleaving produces $\hat{\boldsymbol{c}} = (4, 3, 2, 3)$ and $\hat{\boldsymbol{c}}' = (2, 4, 3, 0)$, and the syndromes $\boldsymbol{s} = (3, 2)^T$ and $\boldsymbol{s}' = (4, 4)^T$ are obtained.
  2) None of the conditions are met, therefore we proceed to the next step.
  3) t is even.
  4) $s_2 = 2$ and $s'_2 = 4$, together with $e = 3$. Use Fig. 4 to determine the error state to be B2.
  5) Using Table II, we find $-e\boldsymbol{h}^*_\tau = (3, 2)^T$ and $\boldsymbol{h}^*_\nu + e\boldsymbol{h}^*_{\tau'} = (4, 4)^T$. Using $\tau = \tau'$ and $e^{-1} = 2$, we solve the column positions as $\nu = 2$ and $\tau = \tau' = 4$. Testing $\hat{w}_{2\tau} - \Delta = 1 - 3 = -2 \notin \mathcal{Q}$ indicates that an invalid channel symbol has been obtained, therefore we declare a decoding failure and stop.

V. Analysis

A. Redundancy

We first look at the redundancy of the balancing scheme in Section III and compare it with those discussed in Sections I and II. Let r denote the total number of redundant symbols of the balanced code, $r = r' + 1$. Since the maximum length of the check matrix $\boldsymbol{H}_{q,r'}$ equals $q^{r'} - 1$, we conclude that the maximum length, $L_q(r)$, of the user word for $q > 2$ is

$$\begin{aligned} L_q(r) &= (q^{r'} - 1) + 1 - r \\ &= q^{r-1} - r. \end{aligned}$$

For the binary case $q = 2$, since only the index v needs to be encoded and not the integer s, we find

$$L_2(r) \;=\; 2^r - r - 1,$$

which is the same value as presented by Knuth [4] using a construction with a prefix. Note that for $q = 2$ the check matrix $C_{2,r}$ defines a regular (binary) Hamming code with redundancy $r' = r$.

Swart and Weber's construction [14] has a balanced prefix of length r, where each prefix uniquely represents the pair of integers s and v. Let $N_q(r)$ denote the number of distinct q-ary balanced prefixes of length r. For this construction the maximum length of the user word, denoted by $L^{\mathrm{Sw}}_q(r)$, is

$$L^{\mathrm{Sw}}_q(r) = \left\lfloor \frac{N_q(r)}{q} \right\rfloor.$$

Using generating functions, we can straightforwardly compute $N_q(r)$ as the largest coefficient of the expansion of $(1 + x +$

TABLE IV
MAXIMUM USER WORD LENGHTS AS A FUNCTION OF r

q	r	$L_q^{\text{Sw}}(r)$	$L_q^{\text{Cap1}}(r)$	$L_q^{\text{Cap2}}(r)$	$L_q^{\text{Pel1}}(r)$	$L_q(r)$	$L_q^{\text{ECC}}(r)$	R_{ECC}
3	4	6	40	76	9	23	—	—
3	5	17	121	237	25	76	—	—
3	6	47	364	722	70	237	—	—
3	7	131	1093	2179	196	722	—	—
3	8	369	3280	6552	553	2179	—	—
3	9	1046	9841	19673	1569	6552	10	0.526
3	10	2984	29524	59038	4476	19673	9	0.474
3	11	8551	88573	177135	12826	59038	44	0.800
3	12	24596	265720	531428	36894	177135	43	0.782
3	13	70980	797161	1594309	106470	531428	150	0.920
3	14	205409	2391484	4782954	308113	1594309	149	0.914
5	4	17	156	308	21	121	—	—
5	5	76	781	1557	95	620	—	—
5	6	350	3906	7806	437	3119	—	—
5	7	1627	19531	39055	2033	15618	4	0.364
5	8	7633	97656	195304	9541	78117	3	0.273
5	9	36065	488281	976553	45081	390616	42	0.824
5	10	171389	2441406	4882802	214236	1953115	41	0.804
5	11	818299	12207031	24414051	1022873	9765614	240	0.956
5	12	3922235	61035156	122070300	4902793	48828113	239	0.952
5	13	18861819	305175781	610351549	23577274	244140612	1238	0.990
5	14	90961151	1525878906	3051757798	113701438	1220703111	1237	0.989

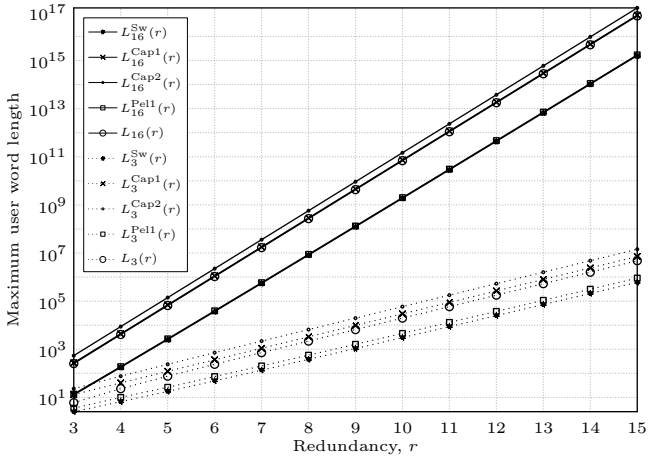

Fig. 5. Maximum user word lengths as a function of r

$x^2+\cdots+x^{q-1})^r$. Then these values correspond to the central binomial coefficients (A001405), central trinomial coefficients (A002426), central quadrinomial coefficients (A005190) and central pentanomial coefficients (A005191) for sequences with $q = 2$, 3, 4 and 5 respectively, where the bracketed numbers indicate the sequences from [25].

Capocelli *et al.* [11] presented two code constructions where the maximum length of the user word for the first construction, $L_q^{\text{Cap1}}(r)$, is

$$L_q^{\text{Cap1}}(r) = \frac{q^r-1}{q-1},$$

and the maximum length of the user word for the second, slightly more complex, construction, $L_q^{\text{Cap2}}(r)$, is

$$L_q^{\text{Cap2}}(r) = 2\frac{q^r-1}{q-1} - r.$$

In Tallini and Vaccaro [12], a generalization of Knuth's complementation method is used to balance sequences that are close to being balanced, while other sequences are compressed with uniquely decodable variable length code and balanced using the saved space. The maximum length of the user word for this construction, $L_q^{\text{Tal}}(r)$, is

$$L_q^{\text{Tal}}(r) = \frac{1}{1-2\alpha}\frac{q^r-1}{q-1} - c_1(q,\alpha)r - c_2(q,\alpha),$$

where c_1 and c_2 are dependent on q and α, with $\alpha \in [0, \frac{1}{2})$. If only the balancing aspect is taken into account and not the compression aspect, then $L_q^{\text{Tal}}(r)$ is equal to $L_q^{\text{Cap2}}(r)$.

Pelusi *et al.* [15] have two constructions with parallel decoding. The first construction has balanced prefixes, similar to [14], and the maximum length of the user word for this construction, $L_q^{\text{Pel1}}(r)$, is

$$L_q^{\text{Pel1}}(r) = \frac{N_q(r) - \{q \bmod 2 + [(q-1)L_q^{\text{Pel1}}(r)] \bmod 2\}}{q-1}.$$

The second construction is a refinement of the first with prefixes that need not be balanced, and has a maximum user word length that is the same as $L_q^{\text{Cap1}}(r)$.

Table IV shows, for $q = 3$ and $q = 5$, the maximum length of the user words as a function of r for the schemes discussed thus far. Fig. 5 graphically shows the maximum length of the user words as a function of r for $q = 3$ and $q = 16$. Table V compares the schemes' redundancies for practical user word lengths. From this we can see that the new scheme has maximum user lengths that are considerably longer than the scheme from [14] and Scheme 1 from [15], comparable to that of Scheme 1 from [11] and Scheme 2 from [15], but roughly half that of Scheme 2 from [11].

This can further be illuminated by looking at the redundancies when the alphabet size becomes large. To proceed, we

TABLE V
REDUNDANCY COMPARISON FOR SPECIFIC INFORMATION LENGTHS

		Redundancy, r					
q	User word length	[14]	Scheme 1 [11]	Scheme 2 [11]	Scheme 1 [15]	Our scheme	Our scheme with ECC
3	64	7	5	4	6	5	12
3	128	7	6	5	7	6	13
3	256	8	6	6	8	7	14
3	512	9	7	6	8	7	16
3	1024	9	7	7	9	8	17
3	2048	10	8	7	10	8	18
3	4096	11	9	8	10	9	19
5	64	5	4	4	5	4	10
5	128	6	4	4	6	5	11
5	256	6	5	4	6	5	12
5	512	7	5	5	7	5	12
5	1024	7	6	5	7	6	13
5	2048	8	6	6	8	6	14
5	4096	8	7	6	8	7	15

make use of Star's approximation [26] for $N_q(r)$, given by

$$N_q(r) = q^r \sqrt{\frac{6}{\pi r(q^2-1)}} \left(1 + \mathcal{O}\left(\frac{1}{r}\right)\right) \approx q^r \sqrt{\frac{6}{\pi r(q^2-1)}},$$

as $r \to \infty$.

Now, letting $q \to \infty$ for the previous redundancies, we find

$$L_q(r) \approx q^{r-1},$$
$$L_q^{\text{Sw}}(r) \approx 1.38 q^{r-2}/\sqrt{r},$$
$$L_q^{\text{Cap1}}(r) = L_q^{\text{Pel2}}(r) \approx q^{r-1},$$
$$L_q^{\text{Cap2}}(r) \approx 2q^{r-1},$$

and

$$L_q^{\text{Pel1}}(r) \approx 1.38 q^{r-2}/\sqrt{r},$$

confirming our earlier observation.

For the redundancy of the balancing scheme with error correction in Section IV, the total redundancy is $r = 2r^* + 3$, taking into account that we use the $\mathcal{C}^*$ code twice with redundancy of r^* for each, and add one redundant symbol for balancing and two more redundant check symbols. Furthermore, the maximum length of the check matrix for one $\mathcal{C}^*$ code is $n = q^{r^*-1} - 1$, since the one row is an all-ones row. The total length of the encoded word is $2(q^{r^*-1} - 1) + 3$. Let $L_q^{\text{ECC}}(r)$ denote the maximum length of the user word, then it can be shown that for q odd

$$\begin{aligned} L_q^{\text{ECC}}(r) &= 2(q^{r^*-1} - 1) + 3 - r \\ &= 2q^{\lfloor \frac{r-3}{2} \rfloor - 1} - r + 3 - 2 \\ &= 2q^{\lfloor \frac{r-5}{2} \rfloor} - r + 1. \end{aligned}$$

For q even, one more redundant symbol is added. The values for $L_q^{\text{ECC}}(r)$ are also shown in Table IV, along with the error correction code rate based on these values, where

$$R_{\text{ECC}} = \frac{L_q^{\text{ECC}}(r)}{L_q^{\text{ECC}}(r) + r}.$$

Again, for the binary case $q = 2$, since only the index v needs to be encoded and not the integer s, we find

$$L_2^{\text{ECC}}(r) = 2^{\lfloor \frac{r-2}{2} \rfloor} - r - 1.$$

B. Complexity

The main contributor to the time and space complexity in our balancing scheme is the ECC component: vector-matrix multiplication affecting the time complexity and the size of the generator/parity check matrices affecting the space complexity. For the complexity analyses we use $r' \approx \log_q k$ which holds as $k \to \infty$, leading to $m \approx k + \log_q k + 1$.

The time complexity for encoding consists of the error correction and balancing aspects. For the error correction, vector-matrix multiplication is needed with the number of operations being $k(m-1) = km - k = k^2 + k\log_q k$, resulting in $\mathcal{O}(k^2)$. Note that we have not considered a parallel vector-matrix multiplication implementation here. Using the method briefly described after Example 2 to find s and v results in complexity $\mathcal{O}((m+q)\log_q m)$. This is obtained by first computing weight$(\boldsymbol{x} \oplus_q \boldsymbol{b}_{0,0})$, requiring $\mathcal{O}(m \log_q m)$ digit operations[5]. Next, the number of appearances of all q-ary symbols in $\boldsymbol{x}$ are computed and stored in a table, also requiring $\mathcal{O}(m \log_q m)$ digit operations. If m_i denotes the number of appearances of symbol i in $\boldsymbol{x}$, $i \in \mathcal{Q}$, then for each $0 \le s \le q-2$, the weight of $\boldsymbol{x} \oplus_q \boldsymbol{b}_{s+1,0}$ can be computed based on the weight of $\boldsymbol{x} \oplus_q \boldsymbol{b}_{s,0}$ using

$$\text{weight}(\boldsymbol{x} \oplus_q \boldsymbol{b}_{s+1,0}) = \text{weight}(\boldsymbol{x} \oplus_q \boldsymbol{b}_{s,0}) + m - qm_{q-1-s},$$

which requires only $\mathcal{O}(\log_q m)$ digit operations. Once an s is found such that weight$(\boldsymbol{x} \oplus_q \boldsymbol{b}_{s,0}) \le \Omega_{q,m} \le$ weight$(\boldsymbol{x} \oplus_q \boldsymbol{b}_{s+1,0})$, a v has to be found that balances the sequence. The final step of modulo q integration has complexity $\mathcal{O}(m)$. Taking all of these into account results in a time complexity for encoding of $\mathcal{O}(k^2 + q)$.

[5]Here we make use of the fact that arithmetic operations with numbers up to $m(q-1)$ requires $\mathcal{O}(m \log_q m)$ q-ary digit operations, assuming that m is larger than q.

TABLE VI
TIME/SPACE COMPLEXITY COMPARISON

	[14]	Scheme 1 and 2 [11]	Scheme 1 [15]	Scheme 2 [15]	Our scheme
Time complexity (encoding)	$\mathcal{O}(k \log_q k)$	$\mathcal{O}(qk \log_q k)$	$\mathcal{O}(qk \log_q k)$	$\mathcal{O}(k\sqrt{\log_q k})$	$\mathcal{O}(k^2 + q)$
Time complexity (decoding)	$\mathcal{O}(1)$	$\mathcal{O}(qk \log_q k)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(k \log_q k)$
Space complexity (encoding)	$\mathcal{O}(qk \log_q k)$	$\mathcal{O}(k + q)$	$\mathcal{O}(qk \log_q k)$	$\mathcal{O}(qk \log_q k)$	$\mathcal{O}(k^2)$
Space complexity (decoding)	$\mathcal{O}(qk \log_q k)$	$\mathcal{O}(k + q)$	$\mathcal{O}(qk \log_q k)$	$\mathcal{O}(qk \log_q k)$	$\mathcal{O}(k \log_q k)$

The time complexity for decoding depends on modulo q differentiation with complexity $\mathcal{O}(m)$ and vector-matrix multiplication to find the syndrome, with the number of operations $r'(m-1) = r'm - r' = k \log_q k + 2 \log_q k$, finally resulting in $\mathcal{O}(k \log_q k)$.

The space complexity for encoding mainly depends on the generator matrix of size $(m-1) \times k$. After substitution for m we have $\mathcal{O}(k^2)$. Similarly, the space complexity for decoding mainly depends on the parity check matrix of size $(m-1) \times r'$, resulting in $\mathcal{O}(k \log_q k)$.

In Table VI we compare the time and space complexity of our balancing scheme, as shown above, with those of previous schemes. In terms of encoding time complexity, the new scheme generally takes longer than the previous ones, except in the special case of large q and short k. Scheme 2 [15] is the most efficient, regardless of q and k. Note that we listed a reduced encoding time complexity for [14], since the same fast encoding method for balancing as described earlier can be used. Our decoding time complexity is a factor of q less than the previous schemes, with time complexity $\mathcal{O}(qk \log_q k)$. However, it cannot compete with the schemes that have parallel decoding with $\mathcal{O}(1)$. Again, it should be pointed out that one could improve the time complexity for our new scheme if a parallel implementation for the vector-matrix multiplication is considered, although this could potentially increase the space complexity.

C. Performance

We look at the performance of four codes:

- $R = 10/19$ code with $q = 3$: an $(8, 5)$ linear block code is used, giving a $(16, 10)$ code after interleaving, and after balancing and adding the extra two check symbols this becomes $(19, 10)$.
- $R = 44/55$ code with $q = 3$: a $(26, 22)$ linear block code is used.
- $R = 4/11$ code with $q = 5$: a $(4, 2)$ linear block code is used. (This is the code used in Example 3.)
- $R = 12/21$ code with $q = 5$: a $(9, 6)$ linear block code is used.[6]

In general, if we start with an (n, k) code $\mathcal{C}^*$, then the overall rate of the scheme will be $R = \frac{2k}{2n+3}$.

In the figures we also compare the results of the previous exhaustive decoding algorithm (from [19]) with the fast syndrome-based decoding presented here.

[6]This is an example of a code with a shorter user word length than the maximum attainable, which according to Table IV is 42.

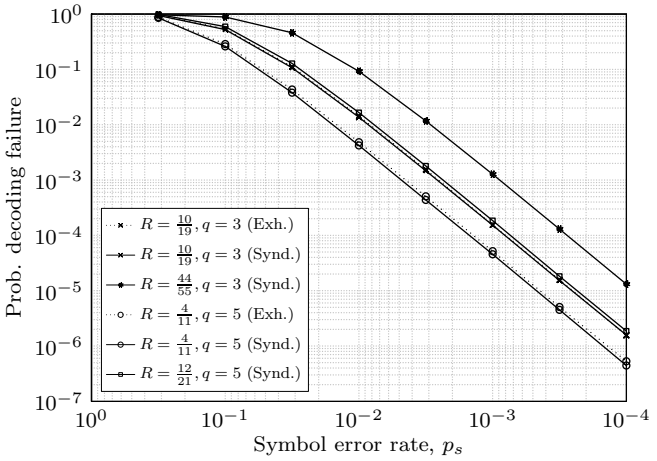

Fig. 6. Probability of decoding failure

Fig. 6 shows the probability that a decoding failure occurred. This can occur when:

- the imbalance indicates that possible multiple errors occurred, i.e. the imbalance size is greater than the maximum symbol size or an imbalance occurred in both the codeword and the check symbols, or
- checks during the decoding indicated that decoding was unsuccessful, i.e. if the syndrome indicated invalid positions, correction would have resulted in an invalid channel symbol or correction proceeded, but the syndromes are still non-zero.

Fig. 7 shows the symbol error rate after decoding for these four codes. If a decoding failure occurred, the information was discarded, and was not taken into account in the symbol error rate calculations. Typically this would be applicable in an ARQ system where the information would be requested again.

These figures show that the same performance is attained by the fast syndrome-based decoding method compared to the previous exhaustive decoding method presented in [19]. However, it should be noted that the small difference in performance between the two decoders is because in the exhaustive method, decoding was performed even though the check symbols could not determine the even or odd position of the channel error, by iterating through all positions. In some instances this resulted in two possible positions where the error could be corrected, one correct and one incorrect. In these cases the first possible position was used for correction. However, this situation only occurs when more than one

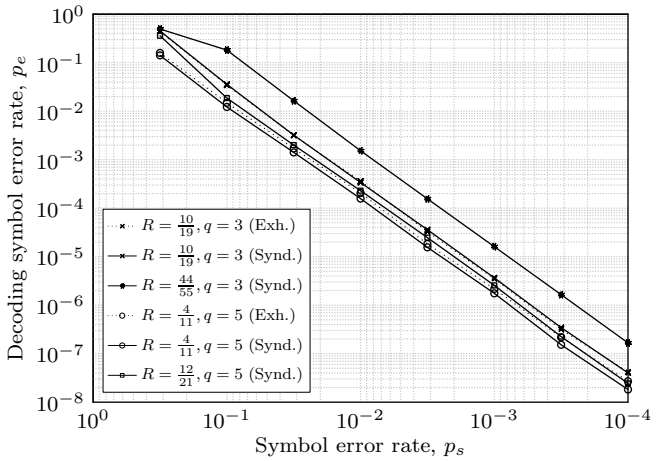

Fig. 7. Decoding symbol error rate

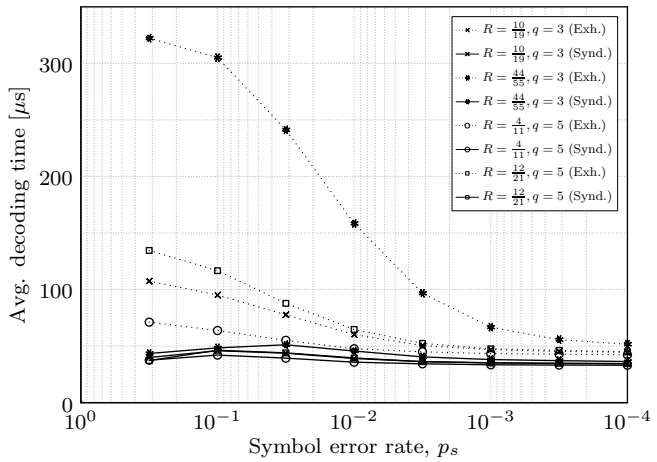

Fig. 8. Average time to perform one decoding operation.

channel error occurred, and is regarded as a decoding failure in the fast syndrome-based decoder. If the exhaustive decoder is changed to do the same, the exact same performance is attained by both decoders.

Finally, we look at the time it takes for the algorithms to decode. These results should be seen as comparative, as different times would be obtained when the same simulations are run on different computer hardware or using a different programming language. Fig. 8 shows the average time that the algorithms spend on a decoding operation (specifically Steps 4 to 8 in the syndrome decoding algorithm above, and the corresponding steps in the exhaustive algorithm). It is clear that for the fast syndrome-based algorithm the average decoding time is approximately the same, regardless of the symbol error rate and the length of the code. As one would expect, longer lengths cause the exhaustive algorithm to spend considerable more time on decoding.

VI. Discussions

Although our new balancing scheme does not improve in terms of redundancy or complexity compared to other balancing schemes that do not need lookup tables (such as the two schemes in [11]), it provides us with an error correcting framework than can be extended to include error correction of channel errors, as was done in Section IV for single errors.

Further improvements are possible if some of the known tEC-AUED codes are employed, provided that one can adhere to the parameter constraints (e.g. length of code, alphabet size of code, etc.) of the chosen code. An immediate extension would be to employ some of the known tEC-AUED codes with $t > 1$, to be able to correct more than one channel error. Alternatively, a tEC-AUED code with $t = 2$ could be used to avoid the use of interleaving.

The most promising and flexible option appears to be [24], where the proposed codes can correct t_1 asymmetric errors of maximum magnitude l_1 and t_2 asymmetric errors of maximum magnitude l_2 with $l_1 < l_2$. The authors state that the "model can be naturally generalized to a wider range of magnitudes as well as for errors in both directions". Provided that the code is generalized for errors in both directions for the larger magnitude errors, our code in Section IV can be replaced by such a code with $t_1 = t_2 = 1$, $l_1 = 1$ and $l_2 = q - 1$. Again, to avoid interleaving we can use a similar code with $t_1 = 1$, $t_2 = 2$, $l_1 = 1$ and $l_2 = q - 1$. Either of these options can be extended to correct multiple channel errors by increasing the value of t_2.

An easy method to obtain lower redundancies for the error-correcting balancing scheme described can be attained by adding more columns to the check matrix, where the first non-zero element from the bottom is one, e.g.

$$\boldsymbol{H}^*_{3,3} = \begin{bmatrix} 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix},$$

where the last four columns were added. However, this will increase the complexity of the decoding algorithm slightly, as the error states cannot be determined simply by looking at s_{r^*} and s'_{r^*}.

VII. Conclusions

We have presented a simple method for balancing q-ary codewords, where look-up tables for coding and decoding the prefix can be avoided by making use of an error correction technique. The redundancy of the new construction is comparable to other constructions for certain parameters, but a factor away in other cases.

The method was expanded to include error correction capabilities to correct single channel errors by simply extending the already used error correction code, introducing interleaving and adding a further three redundant symbols. A fast syndrome-based decoding algorithm was presented that can correct single channel errors more quickly than the prior art exhaustive decoding algorithm.

Although simultaneous balancing and error correction have been investigated before, this is the first attempt to closely tie the two operations together. By doing this we have established a balancing scheme within an error correction framework that can easily be extended in future to account for multiple channel errors.

ACKNOWLEDGMENTS

The authors would like to extend their gratitude towards the anonymous reviewers for their insightful comments and critiques that improved this paper.

REFERENCES

[1] K. A. S. Immink, "Coding methods for high-density optical recording," *Philips J. Res.*, vol. 41, pp. 410–430, 1986.
[2] K. W. Cattermole, "Principles of digital line coding," *Int. J. Electron.*, vol. 55, pp. 3–33, July 1983.
[3] H. Zhou, A. Jiang, and J. Bruck, "Balanced modulation for nonvolatile memories," *IEEE Trans. Inform. Theory*, submitted for publication. Available: http://arxiv.org/abs/1209.0744
[4] D. E. Knuth, "Efficient balanced codes," *IEEE Trans. Inform. Theory*, vol. 32, no. 1, pp. 51–53, Jan. 1986.
[5] S. Al-Bassam and B. Bose, "On balanced codes," *IEEE Trans. Inform. Theory*, vol. 36, no. 2, pp. 406–408, Mar. 1990.
[6] L. G. Tallini, R. M. Capocelli, and B. Bose, "Design of some new efficient balanced codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 3, pp. 790–802, May 1996.
[7] J. H. Weber and K. A. S. Immink, "Knuth's balanced codes revisited," *IEEE Trans. Inform. Theory*, vol. 56, no. 4, pp. 1673–1679, Apr. 2010.
[8] H. van Tilborg and M. Blaum, "On error-correcting balanced codes," *IEEE Trans. Inf. Theory*, vol. 35, no. 5, pp. 1091–1095, Sep. 1989.
[9] S. Al-Bassam and B. Bose, "Design of efficient error-correcting balanced codes," *IEEE Trans. Comput.*, vol. 42, no. 10, pp. 1261–1266, Oct. 1993.
[10] J. H. Weber, K. A. S. Immink, and H. C. Ferreira, "Error-correcting balanced Knuth codes," *IEEE Trans. Inform. Theory*, vol. 58, no. 1, pp. 82–89, Jan. 2012.
[11] R. M. Capocelli. L. Gargano, and U. Vaccaro, "Efficient q-ary immutable codes," *Discrete Appl. Math.*, vol. 33, pp. 25–41, 1991.
[12] L. G. Tallini and U. Vaccaro, "Efficient m-ary balanced codes," *Discrete Appl. Math.*, vol. 92, pp. 17–56, 1999.
[13] S. Al-Bassam, "Balanced codes," Ph.D. dissertation, Oregan State University, USA, 1990.
[14] T. G. Swart and J. H. Weber, "Efficient balancing of q-ary sequences with parallel decoding," in *Proc. IEEE Intl. Symp. Inform. Theory*, Seoul, South Korea, Jun. 29–Jul. 3, 2009, pp. 1564–1568.
[15] D. Pelusi, S. Elmougy, L. G. Tallini and B. Bose, "m-ary balanced codes with parallel decoding," *IEEE Trans. Inform. Theory*, vol. 61, no. 6, pp. 3251–3264, May 2015.
[16] A. Baliga and S. Boztaş, "Balancing sets of non-binary vectors," in *Proc. IEEE Intl. Symp. Inform. Theory*, Lausanna, Switzerland, Jun. 30–Jul. 5, 2002, p. 300.
[17] R. Mascella, L. G. Tallini, S. Al-Bassam and B. Bose, "On efficient balanced codes over the mth roots of unity," *IEEE Trans. Inform. Theory*, vol. 52, no. 5, pp. 2214–2217, May 2006.
[18] R. Mascella and L. G. Tallini, "Efficient m-ary balanced codes which are invariant under symbol permutation," *IEEE Trans. Comput.*, vol. 55, no. 8, pp. 929–946, Aug. 2006.
[19] T. G. Swart and K. A. S. Immink, "Prefixless q-ary balanced codes with ECC," in *Proc. IEEE Inform. Theory Workshop*, Seville, Spain, Sep. 9–13, pp. 1–5.
[20] R. W. Hamming, *Coding and Information Theory*, Prentice-Hall, Englewood Cliffs, 1986.
[21] F.-W. Fu, S. Ling and C. Xing, "Constructions for nonbinary codes correcting t symmetric errors and detecting all unidirectional errors: Magnitude error criterion," *Progress in Comput. Sci. and Applied Logic*, vol. 23, pp. 139–152, 2004.
[22] I. Naydenova and T. Kløve, "Some optimal binary and ternary t-EC-AUED codes," *IEEE Trans. Inform. Theory*, vol. 55, no. 11, pp. 4898–4904, Nov. 2009.
[23] T. Kløve, B. Bose and N. Elarief, "Systematic, single limited magnitude error correcting codes for flash memories," *IEEE Trans. Inform. Theory*, vol. 57, no. 7, pp. 4477–4487, Jun. 2011.
[24] E. Yaakobi, P. H. Siegel, A. Vardy and J. K. Wolf, "On codes that correct asymmetric errors with graded magnitude distribution," in *Proc. IEEE Intl. Symp. Inform. Theory*, Saint-Petersburg, Russia, Jul. 31–Aug. 5, 2011, pp. 1056–1060.
[25] N. J. A. Sloane, ed., "The on-line encyclopedia of integer sequences," 2005. [Online]. Available: https://oeis.org
[26] Z. Star, "An asymptotic formula in the theory of compositions," *Aequationes Mathematicae*, vol. 13, pp. 279–284, 1975.

Theo G. Swart (M'05-SM'14) received the B.Eng. and M.Eng. degrees in electrical and electronic engineering from the Rand Afrikaans University, South Africa, in 1999 and 2001, respectively, and the D.Eng. degree from the University of Johannesburg, South Africa, in 2006.

He is an associate professor in the Department of Electrical and Electronic Engineering Science and a member of the UJ Center for Telecommunications. He is the chair of the IEEE South Africa Chapter on Information Theory. His research interests include digital communications, error-correction coding, constrained coding and power-line communications.

Jos H. Weber (S'87-M'90-SM'00) was born in Schiedam, The Netherlands, in 1961. He received the M.Sc. (in mathematics, with honors), Ph.D., and MBT (Master of Business Telecommunications) degrees from Delft University of Technology, Delft, The Netherlands, in 1985, 1989, and 1996, respectively.

Since 1985 he has been with the Faculty of Electrical Engineering, Mathematics, and Computer Science of Delft University of Technology. Currently, he is an associate professor in the Department of Applied Mathematics. He is the chairman of the WIC (Werkgemeenschap voor Informatie- en Communicatietheorie in the Benelux) and the secretary of the IEEE Benelux Chapter on Information Theory. He was a Visiting Researcher at the University of California at Davis, USA, the University of Johannesburg, South Africa, the Tokyo Institute of Technology, Japan, and EPFL, Switzerland. His main research interests are in the area of channel coding.

Kees A. Schouhamer Immink (M'81-SM'86-F'90) received his Ph.D. degree from the Eindhoven University of Technology. He was from 1994 till 2014 an adjunct professor at the Institute for Experimental Mathematics, Essen, Germany. In 1998, he founded Turing Machines Inc., an innovative start-up focused on novel signal processing for hard disk drives and solid-state (Flash) memories.

He received the Golden Jubilee Award for Technological Innovation by the IEEE Information Theory Society in 1998. He received the 2017 IEEE Medal of Honor, a knighthood in 2000, a personal Emmy award in 2004, and the 2015 IET Faraday Medal. He was elected into the (US) National Academy of Engineering, and he received an honorary doctorate from the University of Johannesburg in 2014.