Hamming-Weight Constrained Coded Arrays Based on Covering Codes

Erik Ordentlich
Hewlett–Packard Laboratories
Palo Alto, CA 94304
erik.ordentlich@hp.com

Ron M. Roth
Computer Science Department
Technion, Haifa, Israel
ronny@cs.technion.ac.il

***Abstract*—We present a framework based on covering codes over the set of binary n-words for coding data into $n \times n$ binary arrays with a prescribed upper bound on the Hamming weight (i.e., number of 1's) in each row and column. We obtain previously presented schemes for the case when the upper bound is $n/2$ as special cases of this framework and we study also another potentially practically relevant specialization when the underlying covering code is the first-order Reed–Muller code. Like the previous schemes for the $n/2$ case, the proposed framework and schemes may have applications in improving the performance of a next-generation memory based on programmable resistive devices arranged in a crossbar architecture.**

I. Introduction

Consider the set $\mathcal{A}_{n,r}$ of all $n \times n$ arrays $A = (A_{i,j})_{i,j}$ over the binary alphabet $\mathbb{F} = \{0, 1\}$ such that the (Hamming) weight of each row and each column is at most r for some prescribed $r \le n/2$; i.e., $\mathsf{w}\big((A_{i,s})_{s=1}^n\big) \le r$ and $\mathsf{w}\big((A_{s,j})_{s=1}^n\big) \le r$ for each i and j. We are interested in the problem of efficiently encoding and decoding arbitrary data to and from (a subset of) $\mathcal{A}_{n,r}$. Following the usual formal definitions, a code for this problem consists of a subset $\mathbb{C} \subseteq \mathcal{A}_{n,r}$, an encoder (mapping) $f : \mathcal{M} \to \mathbb{C}$ from a message set $\mathcal{M}$ of size $M = |\mathbb{C}|$ and a decoder $g : \mathbb{C} \to \mathcal{M}$ such that $g(f(u)) = u$ for all $u \in \mathcal{M}$. Of interest are codes for which f and g can be computed with low complexity and M is as large as possible. The gap $n^2 - \log_2 M$ is the redundancy of the code.

In previous work, we presented two efficient schemes for the case of $r = n/2$ [1]. These are the iterative flipping code and the antipodal matching code with respective redundancies of $2n-1$ and $2n$. Both schemes have linear complexity decoding, with the antipodal matching code having also linear complexity encoding. In this work, we present a general framework based on covering codes for constructing schemes that are also applicable to cases when $r < n/2$. The respective schemes of [1] will be seen to essentially be special cases of the framework when the covering code is the trivial repetition code consisting of the all-zero and all-one codewords.

The work of R. M. Roth was carried out in part while visiting Hewlett–Packard Laboratories, Palo Alto, CA. This work was supported in part by Grant No. 2008058 from the United-States–Israel Binational Science Foundation.

Regarding hereafter the alphabet $\mathbb{F}$ as the finite field $\mathrm{GF}(2)$, the proposed framework assumes a linear code over $\mathbb{F}$ with a prescribed covering radius, and yields two constructions that encode into $\mathcal{A}_{n,r}$: an iterative flipping version similar to the iterative scheme in [1] and another version based on antipodal matchings, which were defined in [1]. Recall that the covering radius ρ of a code $\mathcal{C} \subset \mathbb{F}^\ell$ is defined as $\rho = \rho(\mathcal{C}) = \max_{\boldsymbol{x} \in \mathbb{F}^\ell} \min_{\boldsymbol{c} \in \mathcal{C}} \mathsf{d}(\boldsymbol{x}, \boldsymbol{c})$, where $\mathsf{d}(\cdot, \cdot)$ denotes Hamming distance. The iterative flipping encoder uses a linear $[n, k]$ code with covering radius r and has redundancy $2kn - k^2$, while the antipodal matching code uses a linear $[n-1, \kappa]$ code with covering radius $r-1$ and has redundancy $2\kappa n$ (typically κ is slightly larger than k). The advantage of our schemes is seen mainly in the range where r is large (i.e., close to $n/2$), in which case k is small (e.g., logarithmic in n), resulting in an overall redundancy of approximately $2kn$. While the redundancy of $\mathcal{A}_{n,r}$ is generally smaller (yet always grows at least linearly with n—even for $r = n/2$ [2]), selecting a small k allows us to achieve efficient encoding and decoding.

Before presenting the details of our constructions, we review the motivation for row–column weight-limiting codes from [1]. A recently proposed next-generation memory ([3] and references therein) is based on arranging memristive devices in a crossbar architecture comprised of a set of row conductors and a set of column conductors with a memristive device connecting each column conductor to each row conductor. Information may be stored ("written") in such a memory by applying suitably large positive and negative voltages to row and column conductors of a target device, while grounding the remaining conductors, thereby switching the target device between persistent low and high resistance states. One potential problem is that a significant amount of parasitic current may leak through the so-called half-selected devices, which are the devices connected to either the same row or the same column conductors as the targeted device. The use of row–column weight-constrained codes, such as those presented herein, may serve to mitigate this problem. The idea is to store data only after it has been encoded into such constrained arrays, with a 1 (resp. 0) in array location (i, j) indicating that the device spanning the ith row and jth column conductors be put in the low (resp. high) resistance state. If encoded arrays are overwritten in a judicious manner that ensures that all intermediate arrays satisfy the constraint (e.g., by first writing the high resistance states, followed by the low resistance states), it may be possible to significantly reduce the aforementioned parasitic current since the half-selected devices in the low resistance state, which would contribute to the greatest current leakage, would be limited in number. The new framework and, in particular, the instantiation considered in Section IV, expand the range over which the leakage current and redundancy can be traded off relative to the $r = n/2$ constrained schemes of [1].

We next establish some notation that will be used throughout this work. For integers $a \le b$, we denote by $\langle a : b \rangle$ the set $\{a, a+1, a+2, \ldots, b\}$, with $\langle b \rangle$ standing for $\langle 1 : b \rangle$. For an array $A = (A_{i,j})$ and ordered subsets of indices (elements in increasing order) $\mathcal{I}$ and $\mathcal{J}$, we denote by $A_{\mathcal{I},\mathcal{J}}$ the $|\mathcal{I}| \times |\mathcal{J}|$ sub-array of A whose rows (resp. columns) are indexed by $\mathcal{I}$ (resp. $\mathcal{J}$). For the case where $\mathcal{I}$ is a singleton $\{i\}$, we will use the simpler nation $A_{i,\mathcal{J}}$ (and similarly when $\mathcal{J}$ is a singleton); e.g., $A_{\langle n \rangle, j}$ stands for the jth column of an $n \times n$ array A. All arithmetic operations involving binary words and arrays are assumed to be in the binary field $\mathbb{F}$.

II. Iterative covering code framework

Set $\mathcal{C}$ to be a linear $[n, k]$ code over $\mathbb{F}$ with covering radius r, and, without real loss of generality, suppose that $\mathcal{C}$ has a $k \times n$ generator matrix G (over $\mathbb{F}$) of the form

$$G = \big(\boldsymbol{g}_1 \, \boldsymbol{g}_2 \, \ldots \, \boldsymbol{g}_n\big) = \big(\tilde{G} \,|\, I\big) \;, \tag{1}$$

where $\boldsymbol{g}_j$ denotes the jth column of G and I is the $k \times k$ identity matrix. The information symbols are thus assumed to occupy the highest k positions in a codeword.

Our first construction initializes the upper-left $(n-k) \times (n-k)$ portion of an $n \times n$ array with the input data and the remainder with 0's. Then, iteratively, for each row or column with weight greater than r, a codeword is found in $\mathcal{C}$ within Hamming distance r of the respective row or column which is then replaced by its componentwise sum (in $\mathbb{F}$) with the found codeword. The process stops when all rows and columns have weight at most r. This construction thus has a redundancy of $n^2 - (n-k)^2 = 2nk - k^2$, which is the number of boundary positions initialized to 0 and encodes into $\mathcal{A}_{n,r}$. As will be shown below, the encoded entries in the boundary positions can be used, together with the remaining entries, to undo the encoding process and decode the input bits.

The encoding algorithm is detailed in Figure 1. The aforementioned initialization is carried out in Steps 1 and 2, and the row–column iterations are carried out in Steps 3 and 4. Depending on the covering code $\mathcal{C}$, there may be efficient algorithms for finding a codeword at the suitable Hamming distance in these steps. Some (low rate) covering codes, when viewed as error-correcting codes, have efficient maximum-likelihood decoding algorithms for the binary symmetric channel. In these cases, the error-correcting decoding algorithm can be applied to the constraint violating row or column to obtain a codeword within the covering radius. In Section IV, we will specifically consider the case when the covering code is the first-order Reed–Muller code and comment on the complexity of this step.

```
Input: Arbitrary sequence $\boldsymbol{u}$ of $(n-k)^2$ bits.
  1) Arrange $\boldsymbol{u}$ into an $(n-k) \times (n-k)$ array $U$.
  2) Extend $U$ into an $n \times n$ array $A$ by adding $k$ rows and columns
     of 0's.
  3) For $i \in \langle n \rangle$ do:
       If $\mathsf{w}(A_{i,\langle n \rangle}) > r$, do the following:
         a) Find $\boldsymbol{c} \in \mathcal{C}$ such that $\mathsf{d}(A_{i,\langle n \rangle}, \boldsymbol{c}) \le r$.
         b) Set $A_{i,\langle n \rangle} \leftarrow A_{i,\langle n \rangle} + \boldsymbol{c}$.
  4) For $j \in \langle n \rangle$ do:
       If $\mathsf{w}(A_{\langle n \rangle,j}) > r$, do the following:
         a) Find $\boldsymbol{c} \in \mathcal{C}$ such that $\mathsf{d}(A_{\langle n \rangle,j}, \boldsymbol{c}^T) \le r$.
         b) Set $A_{\langle n \rangle,j} \leftarrow A_{\langle n \rangle,j} + \boldsymbol{c}^T$.
  5) If a row or column was replaced in Steps 3 or 4, go to Step 3,
     otherwise terminate and output the final array.
Output: $n \times n$ binary array $A$.
```

Fig. 1. Encoding algorithm for iterative flipping code based on linear covering code.

Notice that the replacement rows and columns in Steps 3 and 4 have r or fewer 1's, since the weights of these replacements coincide precisely with the Hamming distance between the original row (resp. column) and the found codeword. Moreover, the above iterative process is guaranteed to terminate since each row and column replacement in Steps 3 and 4 decreases the corresponding number of 1's in the array by at least 1 (the number of 1's in the affected row or column decreases from $r+1$ or more to r or fewer).

The following proposition establishes that the encoding operations of Figure 1 can be inverted to yield the input sequence $\boldsymbol{u}$ (equivalently, the array U).

Proposition 2.1: Given an array A that is the output of the encoding algorithm of Figure 1, the corresponding input bit $U_{i,j}$ placed into position (i, j) in Step 1 can be recovered by

$$\begin{aligned} U_{i,j} \;\; &= \;\; A_{i,j} + \boldsymbol{g}_i^T A_{\langle n-k+1:n \rangle, j} + A_{i,\langle n-k+1:n \rangle} \boldsymbol{g}_j \\ &\qquad + \boldsymbol{g}_i^T A_{\langle n-k+1:n \rangle, \langle n-k+1:n \rangle} \boldsymbol{g}_j \;, \end{aligned} \tag{2}$$

where $G = (\boldsymbol{g}_j)_{j=1}^n$ is the generator matrix of the form (1).

Proof: We will prove that for each $(i, j) \in \langle n-k \rangle \times \langle n-k \rangle$, the expression

$$\begin{aligned} \tilde{U}_{i,j} \;\; &= \;\; A_{i,j} + \boldsymbol{g}_i^T A_{\langle n-k+1:n \rangle, j} + A_{i,\langle n-k+1:n \rangle} \boldsymbol{g}_j \\ &\qquad + \boldsymbol{g}_i^T A_{\langle n-k+1:n \rangle, \langle n-k+1:n \rangle} \boldsymbol{g}_j \end{aligned} \tag{3}$$

is invariant to any possible row and column modifications in Steps 3 and 4 in Figure 1. The proposition will then follow, since after Step 2, $A_{i,j} = 0$ whenever either $i \ge n-k+1$ or $j \ge n-k+1$, implying that $\tilde{U}_{i,j} = A_{i,j} = U_{i,j}$.

First, note that $\tilde{U}_{i,j}$ is affected only when $A_{i,\langle n \rangle}$, $A_{\langle n \rangle, j}$, or $A_{\langle n-k+1:n \rangle, \langle n-k+1:n \rangle}$ are modified. The latter array components are modified only if one of the highest indexed k columns or rows is modified. Assume first that $A_{i,\langle n \rangle}$ is modified to $A'_{i,\langle n \rangle}$ with

$$A'_{i,\langle n \rangle} = A_{i,\langle n \rangle} + \boldsymbol{c} = A_{i,\langle n \rangle} + \boldsymbol{c}_{\langle n-k+1:n \rangle} G \;, \tag{4}$$

where (4) follows from our assumption (1) on the generator matrix G. Considering only the terms of (3) depending on the updated $A'_{i,\langle n\rangle}$ we get that

$$\begin{aligned} A'_{i,j} + A'_{i,\langle n-k+1:n\rangle}\boldsymbol{g}_j &= A_{i,j} + \boldsymbol{c}_{\langle n-k+1:n\rangle}\boldsymbol{g}_j \\ &\quad + (A_{i,\langle n-k+1:n\rangle} + \boldsymbol{c}_{\langle n-k+1:n\rangle})\boldsymbol{g}_j \\ &= A_{i,j} + A_{i,\langle n-k+1:n\rangle}\boldsymbol{g}_j, \end{aligned}$$

thereby proving that $\tilde{U}_{i,j}$ is invariant to updating $A_{i,\langle n\rangle}$. Invariance with respect to updates to $A_{\langle n\rangle,j}$ can be established similarly.

Next, assume that for some $\ell > n-k$, the row $A_{\ell,\langle n\rangle}$ is modified to $A'_{\ell,\langle n\rangle}$, with a dependence on $\boldsymbol{c}$ analogous to (4). Let A' denote the overall modified array and let $\boldsymbol{e}^{(\ell)}$ denote a column unit vector with its 1 in the ℓth position. Again, considering only the terms of (3) affected by this change, we get that

$$\begin{aligned} &\boldsymbol{g}_i^T A'_{\langle n-k+1:n\rangle,j} + \boldsymbol{g}_i^T A'_{\langle n-k+1:n\rangle,\langle n-k+1:n\rangle}\boldsymbol{g}_j \\ &\quad = \boldsymbol{g}_i^T (A_{\langle n-k+1:n\rangle,j} + \boldsymbol{e}^{(\ell)}\boldsymbol{c}_j) \\ &\qquad + \boldsymbol{g}_i^T (A_{\langle n-k+1:n\rangle,\langle n-k+1:n\rangle} + \boldsymbol{e}^{(\ell)}\boldsymbol{c}_{\langle n-k+1:n\rangle})\boldsymbol{g}_j \\ &\quad = \boldsymbol{g}_i^T (A_{\langle n-k+1:n\rangle,j} + \boldsymbol{e}^{(\ell)}\boldsymbol{c}_{\langle n-k+1:n\rangle}\boldsymbol{g}_j) \\ &\qquad + \boldsymbol{g}_i^T (A_{\langle n-k+1:n\rangle,\langle n-k+1:n\rangle} + \boldsymbol{e}^{(\ell)}\boldsymbol{c}_{\langle n-k+1:n\rangle})\boldsymbol{g}_j \quad (5) \\ &\quad = \boldsymbol{g}_i^T A_{\langle n-k+1:n\rangle,j} + \boldsymbol{g}_i^T A_{\langle n-k+1:n\rangle,\langle n-k+1:n\rangle}\boldsymbol{g}_j \ , \end{aligned}$$

where (5) follows from (4). This establishes the invariance of (3) with respect to changes to rows with indices $n-k+1$ or larger. Invariance with respect to changes to columns with indices in this range is similarly established. ■

III. Antipodal matching covering code framework

In this section, we present the construction based on antipodal matchings that avoids the—at present—unknown and, possibly, large (but still polynomially large) number of non-parallelizable iterations of the iterative flipping encoder of the previous section. We recall from [1] that an antipodal matching is a mapping $\phi : \mathbb{F}^\ell \to \mathbb{F}^\ell$ with the properties:

(i) $\mathsf{w}(\boldsymbol{x}) + \mathsf{w}(\phi(\boldsymbol{x})) = \ell$.
(ii) If $\mathsf{w}(\boldsymbol{x}) \geq \ell/2$ then $\phi(\boldsymbol{x}) \leq \boldsymbol{x}$ componentwise.
(iii) $\phi(\phi(\boldsymbol{x})) = \boldsymbol{x}$.

Thus, from property (ii), no component of $\boldsymbol{x}$ that is a 0 can become a 1 in $\phi(\boldsymbol{x})$.

The antipodal matching weight-limiting code of [1], which addresses the special case $r = n/2$, applies antipodal matchings to weight constraint-violating rows (resp. columns) instead of bit-flipping as in the iterative flipping code, thereby avoiding the creation of new constraint-violating columns (resp. rows) and the resulting additional iterations to fix these. Reserved positions in the array are used to encode the rows and columns to which antipodal matchings are applied, thus allowing the decoder to invert the process.

Here, we show that antipodal matchings can similarly be used to eliminate iterations from the iterative version of covering code framework of the previous section. Referring to Figure 1, one key modification is to the processing of columns in Step 4, where rather than summing in the Hamming distance reducing codeword $\boldsymbol{c}^T$, we instead apply an antipodal matching to the sub-word of positions in the constraint-violating column corresponding to the positions with indices in the set $\langle n-k\rangle$ in which $\boldsymbol{c}$ is 1.

Formally, given a binary word $\boldsymbol{y} = (y_i)$ in $\mathbb{F}^\ell$, let $\mathcal{S}(\boldsymbol{y}) = \{i \in \langle\ell\rangle : y_i = 1\}$ denote the ordered set of indices, from smallest to largest, in which $\boldsymbol{y}$ is 1, and let $\mathcal{S}^c(\boldsymbol{y}) = \langle\ell\rangle \backslash \mathcal{S}(\boldsymbol{y})$. Define the mapping $\psi : \mathbb{F}^\ell \times \mathbb{F}^\ell \to \mathbb{F}^\ell$ as $\boldsymbol{z} = \psi(\boldsymbol{x}, \boldsymbol{y})$ with

$$\boldsymbol{z}_{\mathcal{S}(\boldsymbol{y})} = \phi(\boldsymbol{x}_{\mathcal{S}(\boldsymbol{y})}) \quad \text{and} \quad \boldsymbol{z}_{\mathcal{S}^c(\boldsymbol{y})} = \boldsymbol{x}_{\mathcal{S}^c(\boldsymbol{y})} \tag{6}$$

(recall that, for a generic word $\boldsymbol{v}$, we have defined $\boldsymbol{v}_{\mathcal{I}}$ to be the sub-word that is indexed by $\mathcal{I}$). Note that if $\boldsymbol{y}$ is all-zero then $\mathcal{S}(\boldsymbol{y}) = \emptyset$ and $\psi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}$.

The following proposition states the key properties of ψ that are needed for our construction.

Proposition 3.1: For $\boldsymbol{x} \in \mathbb{F}^\ell$ with $\mathsf{w}(\boldsymbol{x}) > r$, let $\boldsymbol{y}$ be a word satisfying $\mathsf{d}(\boldsymbol{x}, \boldsymbol{y}) \leq r$. Then $\psi(\boldsymbol{x}, \boldsymbol{y}) \leq \boldsymbol{x}$ (componentwise) and $\mathsf{w}\big(\psi(\boldsymbol{x}, \boldsymbol{y})\big) = \mathsf{d}(\boldsymbol{x}, \boldsymbol{y}) \leq r$.

Proof: Note the relations $\mathsf{w}\big(\boldsymbol{x}_{\mathcal{S}(\boldsymbol{y})}\big) + \mathsf{w}\big((\boldsymbol{x}+\boldsymbol{y})_{\mathcal{S}(\boldsymbol{y})}\big) = |\mathcal{S}(\boldsymbol{y})|$ and that $\mathsf{w}\big(\boldsymbol{x}_{\mathcal{S}^c(\boldsymbol{y})}\big) = \mathsf{w}\big((\boldsymbol{x}+\boldsymbol{y})_{\mathcal{S}^c(\boldsymbol{y})}\big)$. Since by assumption $\mathsf{w}(\boldsymbol{x}) > \mathsf{d}(\boldsymbol{x}, \boldsymbol{y}) = \mathsf{w}(\boldsymbol{x}+\boldsymbol{y})$, it follows from the latter relation that $\mathsf{w}\big(\boldsymbol{x}_{\mathcal{S}(\boldsymbol{y})}\big) > \mathsf{w}\big((\boldsymbol{x}+\boldsymbol{y})_{\mathcal{S}(\boldsymbol{y})}\big)$, and, then from the former relation that $\mathsf{w}\big(\boldsymbol{x}_{\mathcal{S}(\boldsymbol{y})}\big) > |\mathcal{S}(\boldsymbol{y})|/2$. We thus have from property (ii) of ϕ and the definition of ψ that $\psi(\boldsymbol{x}, \boldsymbol{y}) \leq \boldsymbol{x}$. From property (i) of ϕ and the definition of ψ we have that

$$\begin{aligned} \mathsf{w}(\psi(\boldsymbol{x}, \boldsymbol{y})) &= |\mathcal{S}(\boldsymbol{y})| - \mathsf{w}\big(\boldsymbol{x}_{\mathcal{S}(\boldsymbol{y})}\big) + \mathsf{w}\big(\boldsymbol{x}_{\mathcal{S}^c(\boldsymbol{y})}\big) \\ &= \mathsf{w}(\boldsymbol{x}+\boldsymbol{y}) = \mathsf{d}(\boldsymbol{x}, \boldsymbol{y}) \leq r \ , \end{aligned} \tag{7}$$

where (7) follows from the first two relations noted at the beginning of the proof. ■

Figures 2 and 3, respectively, present a possible encoder and a corresponding decoder based on this approach. The antipodal matching underlying ψ would ideally be the linear time computable one described in [1]. As in the iterative construction, rows and columns with indices $n-\kappa+1$ or larger are reserved for storing information required to enable decoding of the encoded array. In contrast to the iterative construction, however, we use here a linear $[n-1, \kappa]$ code with covering radius $r-1$; moreover, κ^2 additional reserved positions turn out to be necessary in this case. In Figure 2, it is assumed that $\kappa^2 \leq n - \kappa$, and, therefore, the additional reserved positions can be confined to the diagonal elements ($\mathsf{diag}(A)$) of the (sub)array $A_{\langle n-\kappa\rangle,\langle n-\kappa\rangle}$ (Steps 1 and 2). This assumption holds when κ is sufficiently small and, in particular, when the covering code is the first-order Reed–Muller code, as will be considered in the next section. The main consequence of these reserved positions is that we do need the covering radius to be $r-1$ in order to guarantee the weight constraint of r on the rows and columns of the array.

As can be seen in Steps 4 and 5, the antipodal matching encoder applies ψ in the column processing phase to constraint-violating columns (with the diagonal entries ignored) and again during the processing of the last κ rows. Specifically,

```
Input: Arbitrary sequence $\boldsymbol{u}$ of $(n-\kappa)^2 - \kappa^2$ $(= n^2 - 2\kappa n)$ bits.
  1) Arrange $\boldsymbol{u}$ into positions $\langle n-\kappa\rangle \times \langle n-\kappa\rangle \setminus \{(i,i)\}_{i=n-\kappa-\kappa^2+1}^{n-\kappa}$
     of an $(n-\kappa) \times (n-\kappa)$ binary array $U$.
  2) Extend $U$ into an $n \times n$ array $A$ by adding $\kappa$ rows and columns
     of 0's.
  3) For $i \in \langle n-\kappa\rangle$ do:
      If $\mathsf{w}(A_{i,\langle n\rangle\setminus\{i\}}) \geq r$, do the following:
       a) Find $\boldsymbol{c} \in \mathcal{C}$ such that $\mathsf{d}(A_{i,\langle n\rangle\setminus\{i\}}, \boldsymbol{c}) < r$.
       b) Set $A_{i,\langle n\rangle\setminus\{i\}} \leftarrow A_{i,\langle n\rangle\setminus\{i\}} + \boldsymbol{c}$.
  4) For $j \in \langle n\rangle$ do:
      If $\mathsf{w}(A_{\langle n-\kappa|j\rangle}) \geq r$, do the following:
       a) Find $\boldsymbol{c} \in \mathcal{C}$ such that $\mathsf{d}(A_{\langle n\rangle\setminus\{\min(j,n-\kappa)\},j}, \boldsymbol{c}^T) < r$.
       b) Set $A_{\langle n-\kappa|j\rangle,j} \leftarrow \psi\big(A_{\langle n-\kappa|j\rangle,j}, \boldsymbol{c}^T_{\langle n-\kappa-1\rangle}\big)$.
       c) Set $A_{\langle n-\kappa+1:n\rangle,j} \leftarrow \boldsymbol{c}^T_{\langle n-\kappa:n-1\rangle}$.
  5) For $i \in \langle n-\kappa+1 : n\rangle$ do:
      If $\mathsf{w}(A_{i,\langle n\rangle\setminus\{i\}}) \geq r$, do the following:
       a) Find $\boldsymbol{c} \in \mathcal{C}$ such that $\mathsf{d}(A_{i,\langle n\rangle\setminus\{i\}}, \boldsymbol{c}) < r$.
       b) Set $A_{i,\langle n\rangle\setminus\{i\}} \leftarrow \psi\big(A_{i,\langle n\rangle\setminus\{i\}}, \boldsymbol{c}\big)$.
       c) Set $(\mathsf{diag}(A))_{\langle (i-2)\kappa-n(\kappa-1)+1,(i-1)\kappa-n(\kappa-1)\rangle}$
                                                        $\leftarrow \boldsymbol{c}_{\langle n-\kappa:n-1\rangle}$.
Output: $n \times n$ array $A$
```

Fig. 2. Encoding algorithm for antipodal matching code based on covering codes.

letting $\langle n-\kappa|j\rangle$ stand for the set $\langle n-\kappa\rangle \setminus \{\min(j, n-\kappa)\}$, the $(n-\kappa-1)$-prefix, $A_{\langle n-\kappa|j\rangle,j}$, of each constraint-violating (punctured) column is replaced by $\psi\big(A_{\langle n-\kappa|j\rangle,j}, \boldsymbol{c}^T_{\langle n-\kappa-1\rangle}\big)$, where $\boldsymbol{c}$ is a codeword in the covering code within Hamming distance less than r of the column. The κ-suffix of the column is set as $A_{\langle n-\kappa+1:n\rangle,j} = \boldsymbol{c}^T_{\langle n-\kappa:n-1\rangle}$ in order to allow the decoder to recover $\mathcal{S}(\boldsymbol{c})$, the indices to which the antipodal matching was applied. Note that, since prior to such a step $A_{\langle n-\kappa+1:n\rangle}$ has zero entries, we have $\mathsf{d}\big(A_{\langle n-\kappa|j\rangle,j}, \boldsymbol{c}^T_{\langle n-\kappa-1\rangle}\big) < r - \mathsf{w}(\boldsymbol{c}_{\langle n-\kappa:n-1\rangle})$, and, therefore, by Proposition 3.1, $\mathsf{w}\big(\psi(A_{\langle n-\kappa|j\rangle,j}, \boldsymbol{c}^T_{\langle n-\kappa-1\rangle})\big) + \mathsf{w}\big(\boldsymbol{c}_{\langle n-\kappa:n-1\rangle}\big) < r$, and hence the entire column, after this step, has weight at most r. Moreover, by the monotonicity property of Proposition 3.1, no new constraint violations are created in the first $n-\kappa$ rows when processing the columns.

A similar analysis applies to the processing of the last κ rows in Step 5. In this case, however, the information symbols necessary for recovering $\mathcal{S}(\boldsymbol{c})$ are stored in the previously reserved κ^2 diagonal positions of the array, which can be done for each such row provided that $\kappa^2 \leq n - \kappa$. Note that the setting of each diagonal value might potentially increase by 1 the weights of the containing row and column. If it happens that $\kappa^2 > n-\kappa$, then we would reserve additional off-diagonal elements of the array for this purpose, and, accordingly, $\mathcal{C}$ should then be selected to be (shorter and) with covering radius smaller than $r-1$.

The preceding discussion establishes the following.

Proposition 3.2: Assuming $\kappa^2 \leq n - \kappa$, the output array of the encoder of Figure 2 belongs to $\mathcal{A}_{n,r}$.

The corresponding decoding algorithm detailed in Figure 3 reverses the above encoding steps and recovers the input sequence $\boldsymbol{u}$. In Decoding Step 1, the reserved diagonal positions are partitioned into groups of κ information symbols which are used to reconstruct, via the generator matrix G of the covering code, the sub-words of the last κ rows to which the antipodal matching was applied (if applied at all, as indicated by a non-zero codeword). These antipodal matchings are inverted. Then, in Step 2, the column encoding is inverted by recovering the sub-words to which antipodal matchings were applied from the information symbols stored in the last κ positions of each column. These matchings are then inverted. Finally, the first $n-\kappa$ rows, excepting the diagonal positions, are recovered using information symbols in the last κ columns. We thus have the following.

```
Input: $n \times n$ array $A$.
  1) For $i \in \langle n-\kappa+1 : n\rangle$ do:
       a) Set $\boldsymbol{c} \leftarrow (\mathsf{diag}(A))_{\langle (i-2)\kappa-n(\kappa-1)+1,(i-1)\kappa-n(\kappa-1)\rangle} G$.
       b) Set $A_{i,\langle n\rangle\setminus\{i\}} \leftarrow \psi\big(A_{i,\langle n\rangle\setminus\{i\}}, \boldsymbol{c}\big)$.
  2) For $j \in \langle n\rangle$ do:
       a) Set $\boldsymbol{c} \leftarrow (A_{\langle n-\kappa+1:n\rangle,j})^T G$.
       b) Set $A_{\langle n-\kappa|j\rangle,j} \leftarrow \psi\big(A_{\langle n-\kappa|j\rangle,j}, \boldsymbol{c}^T_{\langle n-\kappa-1\rangle}\big)$.
  3) For $i \in \langle n-\kappa\rangle$ do:
       a) Set $\boldsymbol{c} \leftarrow A_{i,\langle n-\kappa+1:n\rangle} G$.
       b) Set $A_{i,\langle n-\kappa|i\rangle} \leftarrow A_{i,\langle n-\kappa|i\rangle} + \boldsymbol{c}_{\langle n-\kappa-1\rangle}$.
Output: $n^2 - 2\kappa n$ bits in positions of $A$ that are indexed by $\langle n-\kappa\rangle \times$
$\langle n-\kappa\rangle \setminus \{(i,i)\}_{i=n-\kappa-\kappa^2+1}^{n-\kappa}$.
```

Fig. 3. Decoding algorithm for antipodal matching code.

Proposition 3.3: The $n^2 - 2\kappa n$ information symbols in A in positions $\langle n-\kappa\rangle \times \langle n-\kappa\rangle \setminus \{(i,i)\}_{i=n-\kappa-\kappa^2+1}^{n-\kappa}$, after Decoding Step 3 in Figure 3, coincide with the corresponding input bit array entries created in Encoding Step 1 in Figure 2.

An optimality property. As an aside, we note the following optimality property of the antipodal matching in the context of the covering codes framework.

Proposition 3.4: Suppose that n, k, and r are such that there exists a linear $[n, k, 2r+1]$ perfect code over $\mathbb{F}$ (with covering radius r), and let $\mathcal{C}$ be such a code. Then the minimum of

$$\sum_{\boldsymbol{x} \in \mathbb{F}^{n-k}} \mathsf{d}(\boldsymbol{x}, f(\boldsymbol{x})_{\langle n-k\rangle})$$

over all one-to-one mappings f from $\mathbb{F}^{n-k}$ to the set $\{\boldsymbol{z} \in \mathbb{F}^n : \mathsf{w}(\boldsymbol{z}) \leq r\}$ is attained by $f^*(\boldsymbol{x}) = \psi(\boldsymbol{x}, \boldsymbol{c})$, where $\boldsymbol{c} = \arg\min_{\boldsymbol{z} \in \mathcal{C}} \mathsf{d}\big((\boldsymbol{x}\, 0 \ldots 0), \boldsymbol{z}\big)$, with ψ defined in (6).

Thus, in such an ideal scenario, the antipodal matching based mapping ψ, as applied via f^*, requires the fewest symbol changes between the input and the first $n-k$ bits of the output, among all one-to-one mappings between the above sets. The proposition is proved by noting that the antipodal matching only changes 1's to 0's, and thus f^* attains a pointwise lower bound on the number of symbol changes between two words given the respective weights of the words.

IV. Practical covering codes

As mentioned above, we can essentially obtain the iterative flipping and antipodal matching codes of [1] by instantiating the above framework with the simple $[n,1]$ repetition code consisting of the all-zero and all-one codewords. This covering code has a covering radius of $\lfloor n/2 \rfloor$ and the overall scheme encodes into $\mathcal{A}_{n,\lfloor n/2 \rfloor}$ in the case of the iterative scheme and into $\mathcal{A}_{n,\lfloor n/2 \rfloor+1}$ in the case of the antipodal matching scheme. For the antipodal matching case, we do not quite obtain the corresponding scheme from [1], since that scheme is able meet the weight constraint of $n/2$ for all n, even with the extra reserved diagonal bit relative to the iterative scheme. We do not yet know if such an optimization is possible in general.

Another potentially practical instantiation of the framework is with the $[n{=}2^{k-1}, k]$ first-order Reed–Muller code as the covering code. This code is known to have a covering radius of at most $(n-\sqrt{n})/2$ [6, p. 242], and also to have an efficient maximum likelihood decoder (based on a fast Hadamard transform) with a complexity of $O(n \log^2 n)$ bit operations [7, §14.4]. Thus, the iterative and antipodal matching Reed–Muller code instantiated frameworks could be used to encode into $\mathcal{A}_{n,\lfloor (n-\sqrt{n})/2 \rfloor}$, and $\mathcal{A}_{n,\lfloor (n-\sqrt{n})/2 \rfloor+1}$, respectively, with a redundancy on the order of $2n \log_2 n$. Note that the Reed–Muller code satisfies the constraint that $k^2 \leq (n-k)$ for $k \geq 7$, which allows the antipodal matching version to store the additional reserved bits along the main diagonal.

From Figure 2, we see that encoding according to the antipodal matching construction requires $2n$ computations of covering codewords, and at most $n+k$ applications of the antipodal matching. Since the latter can be done in $O(n)$ register operations, or $O(n \log n)$ bit operations, the overall encoding complexity is dominated by the covering codeword computation. Thus, for the first-order Reed–Muller code case, the overall encoding complexity is $O(n^2 \log^2 n)$ bit operations. Decoding, on the other hand, for both the iterative and antipodal matching versions of the scheme, is less complex and can be carried out in $O(n^2 \log n)$ bit operations. Note that for the iterative case, the term $\boldsymbol{g}_i^T A_{\langle n-k+1:n \rangle, \langle n-k+1:n \rangle} \boldsymbol{g}_j$ appearing in the decoding rule (2) can be efficiently determined for all i, j by saving and reusing the computation of, say, $\boldsymbol{g}_i^T A_{\langle n-k+1:n \rangle, \langle n-k+1:n \rangle}$. Note also that all inner products involve vectors over $\mathbb{F}$ of dimension $O(\log n)$.

To get an idea of the potential advantages of the Reed–Muller instantiation over known approaches, we can fix the row–column weight constraint r to that induced by a Reed–Muller code (where $r \leq (n-\sqrt{n})/2$) and compare the redundancy and complexity of the known approaches for such a constraint. We are aware of two other potentially applicable approaches, respectively, from [4] and [5]. The latter approach encodes into arrays with constant row and column weights and is based on a two-dimensional approximate enumerative encoding–decoding scheme that is polynomial-time computable, but nonetheless substantially more complex than the Reed–Muller covering code approach. On the other hand, the redundancy of this approach for the Reed–Muller induced constraint can be shown to improve to $n \log_2 n$ plus lower order terms.

The approach of [4] involves first encoding input data into weight constraint satisfying rows and then permuting the rows using a recursive scheme to also obtain constraint satisfying columns. One is free to use any scheme for the row encoding, and, in particular, in one comparison it makes sense to assume that the Reed–Muller covering code approach is used for this step, on a row-by-row basis. The resulting encoding and decoding complexity can then both be shown to be $O(n^2 \log^2 n)$ bit operations. The $O(n^2 \log^2 n)$ complexity for decoding arises from the inversion of the row permutations carried out during encoding, with the subsequent decoding of the rows requiring only an additional $O(n^2 \log n)$ bit operations. Thus, both the iterative and antipodal matching Reed–Muller based approaches would have lower decoding complexity, and roughly similar encoding complexity. The redundancy of the approach of [4] would also be $2n \log_2 n$ plus lower order terms, and thus comparable to the proposed approach. We do note that the redundancies of the antipodal matching version of the proposed approach and the approach of [4] can both be improved by using, say an enumerative coding scheme for Step 3 in Figure 2, at the expense of comparable increases in complexity for both.

Other potential practical advantages of the proposed approach over the previously known approaches are that, in the case of the iterative scheme, the decoding can be parallelized, and to decode one (or a few) bit(s) does not require reading in the entire array, as would be necessary for the schemes of [4] and [5], but requires only the desired entry and $\log_2^2 n + 2\log_2 n$ boundary entries. The antipodal matching approach does require reading in the entire array, but in this case, both the encoding and decoding operations applied to different rows and columns can be done in parallel. In contrast, the critical steps of the algorithms of [4] and [5] are highly sequential and do not seem to be amenable to parallelization.

We end this section by stating the following result, the proof of which we omit due to space limitations.

Proposition 4.1: For $r = n/2 - O(\sqrt{n})$, the redundancy of $\mathcal{A}_{n,r}$ grows (at most) linearly with n.

Yet, we do not know of an encoding–decoding algorithm which is both efficient and has redundancy which is linear in n.

References

[1] E. Ordentlich and R.M. Roth, *Low complexity two-dimensional weight-constraint codes, IEEE Trans. Inf. Theory,* 58 (2012), 3892–3899.

[2] E. Ordentlich, F. Parvaresh, and R.M. Roth, *Asymptotic enumeration of binary matrices with bounded row and column sums, SIAM J. Disc. Math.,* 26 (2012), 1550–1575.

[3] D.B. Strukov and R.S. Williams, *Four-dimensional address topology for circuits with stacked multilayer crossbar arrays, Proc. Nat'l. Acad. Sci.*, 106 (2009), 20155–20158.

[4] R. Talyansky, T. Etzion, and R.M. Roth, *Efficient code construction for certain two-dimensional constraints, IEEE Trans. Inf. Theory,* 45 (1999), 794–799.

[5] E. Ordentlich and R.M. Roth, *Two-dimensional weight-constrained codes through enumeration bounds, IEEE Trans. Inf. Theory,* 46 (2000), 1292–1301.

[6] G. Cohen, I. Honkala, S. Litsyn, and A. Lobstein, *Covering Codes,* North-Holland, Amsterdam, 1997.

[7] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error-Correcting Codes,* North-Holland, Amsterdam, 1977.