Bounds and Code Constructions for Partially Defect Memory Cells

Haider Al Kim[1,2], Sven Puchinger[3], Antonia Wachter-Zeh[1]

[1]Institute for Communications Engineering, Technical University of Munich (TUM), Germany
[2]Electronic and Communications Engineering, University of Kufa (UoK), Iraq
[3]Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Denmark
Email: haider.alkim@tum.de, svepu@dtu.dk, antonia.wachter-zeh@tum.de

Abstract—This paper considers coding for so-called *partially stuck* memory cells. Such memory cells can only store partial information as some of their levels cannot be used due to, e.g., wear out. First, we present a new code construction for masking such partially stuck cells while additionally correcting errors. This construction (for cells with $q > 2$ levels) is achieved by generalizing an existing masking-only construction in [1] (based on binary codes) to correct errors as well. Compared to previous constructions in [2], our new construction achieves larger rates for many sets of parameters. Second, we derive a sphere-packing (any number of u partially stuck cells) and a Gilbert-Varshamov bound ($u < q$ partially stuck cells) for codes that can mask a certain number of partially stuck cells and correct errors additionally. A numerical comparison between the new bounds and our previous construction of PSMCs for the case $u < q$ in [2] shows that our construction lies above the Gilbert–Varshamov-like bound for several code parameters.

Index Terms—flash memories, phase change memories, defect memory, (partially) stuck cells, defective cells error correction, sphere packing bound, Gilbert-Varshamov bound

I. Introduction

The demand for reliable memory solutions and in particular for non-volatile memories such as *phase-change memories* (PCMs) for different applications is steadily increasing. These memories provide permanent storage, rapidly extendable capacity, and multi-levels devices. However, due to increasing the number of levels while decreasing the size of the memory, it is essential to suggest sophisticated coding and signal processing solutions to overcome reliability issues. The key characteristic of PCM cells is that they can switch between two main states: an amorphous state and a crystalline state. PCM cells may become *defect* (also called *stuck*) if they fail in switching their states. This occasionally happens due to the cooling and heating processes of the cells, and therefore cells can only hold a single phase [3]–[6]. The crystalline state consists of multiple substates which motivates *partially stuck* type of defects to happen.

In flash memories, an electric charge might be trapped in the cell at a certain state, and the cell's status cannot be switched to be writable again. The suggested mechanism to deal with these defect memory cells whose charges are trapped is called *masking*. Masking determines a codeword that matches the stuck level of the stuck memory. Therefore, it can be placed properly on the defective memory. In multi-level PCM cells, failure may occur at a position in both extreme states or in the partially programmable states of crystalline. In [1], a cell that

can only hold levels at least a certain reference level $s > 0$ are called *partially stuck*. For multi-level PCMs, the case $s = 1$ is particularly important since this means that a cell can reach all crystalline sub-states, but cannot reach the amorphous state anymore. In flash memories, information is stored by different charge levels. Similar to PCMs, (partial) defects can occur in flash memory cells. In order to write information in a new write, either all current levels are only increased or a whole block has to be erased. Erasing the whole block reduces the lifespan of flash memory devices.

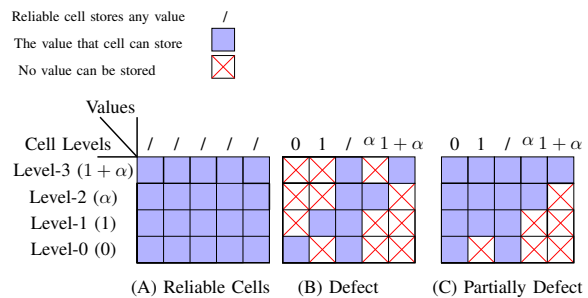Figure 1 shows the general idea of reliable and (partially) defect memory cells.

Figure 1. Difference between reliable and (partially) defect memory cells. In this figure, there are $n = 5$ cells with $q = 4$ possible levels. The cell levels $\in \mathbb{F}_4$ are mapped to (0, 1, α or $1 + \alpha$). Case (A) illustrates only reliable cells that can store any of the four values. In the stuck scenario, as shown in case (B), the defect cells can store only the exact stuck level s. Case (C) is more flexible (partially defect scenario). Partially stuck cells at level $s \geq 1$ can store level s or higher.

A. Related Work

In [7], code constructions for masking stuck memory cells were proposed. In addition to masking the stuck cells, it is possible to correct errors that occur during the storing and reading processes. A generator matrix of a specific form was constructed for this purpose. In [1], improvements on the redundancy necessary for masking *partially* stuck cells are achieved, and lower and upper bounds are derived. However, the paper does not consider error correction in addition to masking.

Combined methods of [7] and [1] to obtain code constructions for joint masking partially stuck cells and error correction are conducted in [2]. These code constructions reduce the redundancy necessary for masking, similar to the results in [1]. In contrast to [1], however, these constructions are able to correct additional random errors.

B. Our Contribution

In this paper, we extend the constructions of [2]. We obtain a code construction for combined error correction and masking

q-ary partially defect cells by means of binary stuck memory cells. Our new construction gives higher rates for several sets of parameters compared to [2, Theorem 4] and it can correct errors compared to [1, Theorem 9]. In this paper, we also analyze how close our constructions are to the sphere packing bound (necessary condition) as an upper bound with the presence of partially defect memory constraints. Further, we introduce a Gilbert-Varshamov-like bounds (sufficient condition) to show the existence of codes with certain parameters that can mask the partially stuck memory and correct errors.

Similar to the main part of [1], this paper deals with partially-stuck-at-1 cells, i.e., $s = 1$, but the results are extendable to arbitrary s similar to [1, Section VII].

II. PRELIMINARIES

A. Notations

For a prime q, let $\mathbb{F}_q$ denote the finite field of order q and $\mathbb{F}_q[x]$ be the set of all univariate polynomial with coefficients in $\mathbb{F}_q$. $\mathbb{F}_{q^\lambda}$ denotes an extension field of $\mathbb{F}_q$ of extension degree λ. Denote $[f] = \{0, 1, \ldots, f - 1\}$ for $f \in \mathbb{Z}_{>0}$.

Throughout this paper, let n be the total number of cells, u be the number of (partially) stuck cells, and t be the number of random errors. Let s_{ϕ_i} denote the (partially) stuck level at position i, where $i \in [u]$, and $\boldsymbol{\phi} = \{\phi_0, \phi_1, \cdots, \phi_{u-1}\} \subseteq [n]$ denotes the positions of the the (partially) stuck cells.

For our construction, let k_1 be the number of information symbols, l be the number of symbols required for masking, and r be the required redundancy for error correction.

B. Definitions

1) Defect and Partially Defect Cells: A cell is called a defect (*stuck at level s*), where $s \in [q]$, if it cannot change its value and always stores the value s. A cell is called partially defect (*partially stuck at level s*), where $s \in [q]$, if it can only store values which are at least s. We fix throughout the paper a total ordering "$\geq$" of the elements of $\mathbb{F}_q$ such that $a \geq 1 \geq 0$ for all $a \in \mathbb{F}_q^*$ (note that such an ordering does not interact well with addition, but this is not relevant here). If a cell is partially stuck at 0, it is a non-defect cell which can store any of the q levels.

2) (u, t)-PSMC: An $(n, M)_q$ (u, t)-*partially-stuck-at-masking code* $\mathcal{C}$ is a coding scheme consisting of an encoder $\mathcal{E}$ and a decoder $\mathcal{D}$. The input of the encoder $\mathcal{E}$ is

- the set of locations of u partially stuck cells $\boldsymbol{\phi} = \{\phi_0, \phi_1, \ldots, \phi_{u-1}\} \subseteq [n]$,
- the partially stuck levels $s_{\phi_0}, s_{\phi_2}, \ldots, s_{\phi_{u-1}} \in [q]$,
- a message $\boldsymbol{m} \in \mathcal{M}$, where $\mathcal{M}$ is a message space of cardinality $|\mathcal{M}|$.

It outputs a vector $\boldsymbol{c} \in \mathbb{F}_q^n$ which fulfills $c_{\phi_i} \geq s_{\phi_i}$ for all $i = 1, \ldots, u$. The decoder is a mapping that takes $\boldsymbol{c} + \boldsymbol{e} \in \mathbb{F}_q^n$ as input and returns the correct message $\boldsymbol{m}$ for all error vectors $\boldsymbol{e}$ of Hamming weight at most t.

C. Code Construction over $\mathbb{F}_{2^\lambda}$

The new code construction works over the finite field $\mathbb{F}_{2^\lambda}$. We denote by $x^0, x^1, \ldots, x^{\lambda-1}$ a basis of $\mathbb{F}_{2^\lambda}$ over $\mathbb{F}_2$. Hence, any element $a \in \mathbb{F}_{2^\lambda}$ can be uniquely represented as $a = \sum_{i=0}^{\lambda-1} a_i x^i$ where $a_i \in \mathbb{F}_2 \forall i$. In particular, $a \in \mathbb{F}_2$ if and only if $a_1 = \cdots = a_{\lambda-1} = 0$. This is a crucial property of $\mathbb{F}_{2^\lambda}$ that we will use in Construction 1.

In the construction, we also use the notation $(\boldsymbol{\Gamma})_{\boldsymbol{\phi}}$ for $\boldsymbol{\Gamma} \in \mathbb{F}_{2^\lambda}^{n+1}$ and $\boldsymbol{\phi} \subseteq \{0, \ldots, n\}$, by which we mean the sub-vector of $\boldsymbol{\Gamma}$ indexed by the entries of $\boldsymbol{\phi}$.

D. Error Models

Assume that the memory has u partially stuck cells at positions $\boldsymbol{\phi} = \{\phi_0, \phi_1, \ldots, \phi_{u-1}\} \subseteq [n]$ and n cells in total.

In the *non-overlapping model*, the t errors can happen only at positions $\boldsymbol{\Psi} = \{\Psi_0, \Psi_1, \ldots, \Psi_{n-u-1}\} = [n] \setminus \boldsymbol{\phi}$.

In the *overlapping model*, on the other hand, we assume that t errors can happen in any cell, i.e., $\boldsymbol{\Psi} \subseteq [n]$. If errors happen in the u partially stuck cells, we assume that the error attains only values such that the corrupted vector still obeys the partially stuck constraints.

III. CODES FOR (PARTIALLY) DEFECT MEMORIES

We propose a new code construction for simultaneous masking and error correction. The new construction is based on the masking-only construction in [1, Section VI], which is able to mask $u \geq q$ partially stuck positions, where q is the field size of the masking code, but cannot correct any errors. We generalize this construction to be able to cope with errors. Compared to [2, Theorem 4], the new construction may lead to larger code dimensions for a given pair (u, t), in a similar fashion as [1, Construction 5] improves compared to [1, Construction 4].

Construction 1. *Let $n, u, t, \lambda, k, k_1, r, l$ be positive integers with $u, t \leq n$, $\lambda > 1$, $k_1 = n - l - r - 1$, and $k = l + k_1 + 1$. Suppose that there are matrices*

- *$\boldsymbol{P} \in \mathbb{F}_{2^\lambda}^{k_1 \times r}$ and*
- *$\boldsymbol{H}_0 := [\boldsymbol{I}_{l \times l} \mid \boldsymbol{R}_{l \times (n-l)}] \in \mathbb{F}_2^{l \times n}$ that is a systematic parity-check matrix of a binary code $\mathcal{C}_0$ with parameters $[n, k_1 + r, d_0 \geq u_0 + 1]_2$, where $u_0 := \lfloor 2u/2^\lambda \rfloor$,*

such that

$$
\boldsymbol{G} = \begin{bmatrix} \boldsymbol{H}_0 & & 0 \\ \boldsymbol{G}_1 & & \vdots \\ & & 0 \\ 1 & \cdots & 1 \end{bmatrix} := \begin{bmatrix} \boldsymbol{I}_{l \times l} & - \boldsymbol{R}_{l \times (n-l)} - & \boldsymbol{0}_{l \times 1} \\ \boldsymbol{0}_{k_1 \times l} & \boldsymbol{I}_{k_1 \times k_1} \quad \boldsymbol{P}_{k_1 \times r} & \boldsymbol{0}_{k_1 \times 1} \\ \hline & \boldsymbol{1}_{1 \times (n+1)} & \end{bmatrix}
$$

is a generator matrix of a $[n+1, k = l + k_1 + 1, d \geq 2t + 1]_{2^\lambda}$ code $\mathcal{C}$.

Based on these definitions, we define a coding scheme in Algorithm 1 and Algorithm 2.

Theorem 1. *The coding scheme in Construction 1 is a (u, t)-PSMC of length $n + 1$ and cardinality*

$$
\mathcal{M}_{u,t} = 2^{\lambda(k_1+l)-l}.
$$

Proof. **Masking:** We first prove that Algorithm 1 outputs a masked vector (i.e., a vector that contains no zeros in the partially stuck-at-($s = 1$) positions $(\phi_0, \ldots, \phi_{u-1})$).

By construction, the vector $(\boldsymbol{a} + \boldsymbol{b})_{\boldsymbol{\phi}}$ has length u and its entries are in $\mathbb{F}_{2^\lambda}$. Consider the partition $\{c, c+1\}$, for all $c \in \mathcal{F}$, of $\mathbb{F}_{2^\lambda}$. These sets are pairwise disjoint for different c and there are $2^{\lambda-1}$ such sets. By the pigeonhole principle, there is one such set, say $\{-z, -z+1\}$, such that the vector $(\boldsymbol{a} + \boldsymbol{b} + (z+1) \cdot \boldsymbol{1}_{1 \times (n+1)})_{\boldsymbol{\phi}}$ contains at most $\lfloor \frac{u}{|\mathcal{F}|} \rfloor = \lfloor 2^{1-\lambda} u \rfloor = u_0$ elements in $\{0, 1\}$.

We need to mask the few remaining entries of $\boldsymbol{w}$ that are 0. First note that since the last entry of $\boldsymbol{a} + \boldsymbol{b}$ is zero and we add $z + 1 \neq 0$ to it, the last entry of $\boldsymbol{w}$ is always non-zero (this is relevant if the last position is partially stuck). We choose a binary vector $\boldsymbol{z}$ such that $[\boldsymbol{z} \cdot \boldsymbol{H}_0 \mid 0]_{\boldsymbol{\phi}}$ contains a 1 (or 0) in positions in which the corresponding entry of $(\boldsymbol{w})_{\boldsymbol{\phi}}$ is 0 (or 1, respectively). Such a vector $\boldsymbol{z}$ always exists since

---

Algorithm 1: Encoding $(\boldsymbol{m}; \boldsymbol{m}'; \boldsymbol{\phi})$

Input:
- Message:
  $\boldsymbol{m} = (m_0, m_1, \ldots, m_{k_1-1}) \in \mathbb{F}_{2^\lambda}^{k_1}$ and
  $\boldsymbol{m}' = (m'_0, m'_1, \ldots, m'_{l-1}) \in \mathcal{F}^l$, where
  $\mathcal{F} := \{\sum_{i=0}^{\lambda-1} a_i x^i \in \mathbb{F}_{2^\lambda} : a_0 = 0\} \subseteq \mathbb{F}_{2^\lambda}$.
- Positions of partially stuck-at-$(s = 1)$ cells:
  $\boldsymbol{\phi} \subseteq \{0, \ldots, n\}$
- Notions introduced in Construction 1.

1   $\boldsymbol{a} \leftarrow \boldsymbol{m}' \cdot [\boldsymbol{H}_0 \mid 0]$
2   $\boldsymbol{b} \leftarrow \boldsymbol{m} \cdot [\boldsymbol{G}_1 \mid 0]$
3   $\boldsymbol{w} \leftarrow \boldsymbol{a} + \boldsymbol{b} + (z+1) \cdot \mathbf{1}_{1 \times (n+1)}$, where $z \in \mathcal{F}$ is
   chosen such that $(\boldsymbol{a} + \boldsymbol{b} + (z+1) \cdot \mathbf{1}_{1 \times(n+1)})_{\boldsymbol{\phi}}$ has at
   most $u_0 = \lfloor 2^{1-\lambda} u \rfloor$ entries in $\mathbb{F}_2$
4   Choose $\boldsymbol{z} \in \mathbb{F}_2^l$ such that, for any $i = 0, \ldots, u-1$,

$$(\boldsymbol{z} \cdot \boldsymbol{H}_0 \mid 0)_{\phi_i} = \begin{cases} 0, & \text{if } (\boldsymbol{w})_{\phi_i} = 1, \\ 1, & \text{if } (\boldsymbol{w})_{\phi_i} = 0, \\ \text{arbitrary}, & \text{if } (\boldsymbol{w})_{\phi_i} \notin \mathbb{F}_2. \end{cases}$$

Output: $\boldsymbol{w} + [\boldsymbol{z} \cdot \boldsymbol{H}_0 \mid 0]$

Algorithm 2: Decoding

Input:
- $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e} \in \mathbb{F}_{2^\lambda}^{n+1}$, where $\boldsymbol{c}$ is a valid output of
  Algorithm 1 and $\boldsymbol{e}$ is an error of Hamming weight at
  most t.
- Notions introduced in Construction 1.

1   $\boldsymbol{v} \leftarrow$ decode $\boldsymbol{y}$ in the code $\mathcal{C}$
2   $\boldsymbol{v}' \leftarrow \boldsymbol{v} - (v_n + 1) \mathbf{1}_{1 \times (n+1)}$
3   $\hat{\boldsymbol{m}}'' \leftarrow [v'_0, \ldots, v'_{l-1}]$
4   $\hat{\boldsymbol{m}}' \leftarrow [\varphi(m''_0), \ldots, \varphi(m''_{l-1})]$, where

$$\varphi : \mathbb{F}_{2^\lambda} \to \mathcal{F}, \quad \sum_{i=0}^{\lambda-1} a_i x^i \mapsto 0x^0 + \sum_{i=1}^{\lambda-1} a_i x^i.$$

5   $\boldsymbol{v}'' \leftarrow \boldsymbol{v}' - \hat{\boldsymbol{m}}'' \cdot [\boldsymbol{I}_{l \times l} \quad \boldsymbol{R}_{l \times (n-l)} \quad \mathbf{0}_{l \times 1}]$
6   $\hat{\boldsymbol{m}} \leftarrow [v''_l, \ldots, v''_{l+k_1-1}]$
Output: $\hat{\boldsymbol{m}}$ and $\hat{\boldsymbol{m}}'$

- the number of $\{0,1\}$-entries in $(\boldsymbol{w})_{\boldsymbol{\phi}}$ is at most u_0,
- $\boldsymbol{H}_0$ is a parity-check matrix of a binary code of minimum distance $\geq u_0 + 1$, which means that any u_0 columns of $\boldsymbol{H}_0$ are linearly independent, and
- the vector $[\boldsymbol{z} \cdot \boldsymbol{H}_0 \mid 0]$ is in $\mathbb{F}_2$ (since $\boldsymbol{z}$ and $\boldsymbol{H}_0$ are binary).

Hence, all entries of $(\boldsymbol{w} + [\boldsymbol{z} \cdot \boldsymbol{H}_0 \mid 0])_{\boldsymbol{\phi}}$ are either 1 or in $\mathbb{F}_{2^\lambda} \setminus \mathbb{F}_2$, i.e., non-zero.

Error correction: Next, we show that the output of Algorithm 1 is a codeword of the code $\mathcal{C}$ as defined in Construction 1, and hence, the first step of Algorithm 2 is able to correct up to t errors in the masked vector.

To prove this, we rewrite

$$\boldsymbol{w} + [\boldsymbol{z} \cdot \boldsymbol{H}_0 \mid 0]$$
$$= \boldsymbol{a} + \boldsymbol{b} + (z+1) \cdot \mathbf{1}_{1 \times (n+1)} + [\boldsymbol{z} \cdot \boldsymbol{H}_0 \mid 0]$$
$$= [\boldsymbol{m}' + \boldsymbol{z} \mid \boldsymbol{m} \mid z+1] \cdot \begin{bmatrix} \boldsymbol{H}_0 & 0 \\ & & \vdots \\ \boldsymbol{G}_1 & 0 \\ 1 & \cdots & 1 \end{bmatrix} \in \mathcal{C} \quad (1)$$

Recovery of the messages: We show that Algorithm 2 uniquely retrieves the messages $\boldsymbol{m}$ and $\boldsymbol{m}'$ from the vector $\boldsymbol{v} := \boldsymbol{w} + [\boldsymbol{z} \cdot \boldsymbol{H}_0 \mid 0]$, which is obtained after the error-correction step.

By (1), we have

$$\boldsymbol{v} = [\boldsymbol{m}' + \boldsymbol{z} \mid \boldsymbol{m} \mid z+1] \begin{bmatrix} \boldsymbol{I}_{l \times l} & -\boldsymbol{R}_{l \times (n-l)} - & \mathbf{0}_{l \times 1} \\ \mathbf{0}_{k_1 \times l} & \boldsymbol{I}_{k_1 \times k_1} & \boldsymbol{P}_{k_1 \times r} & \mathbf{0}_{k_1 \times 1} \\ \hline & \mathbf{1}_{1 \times (n+1)} & \end{bmatrix}$$

Hence, we can retrieve $z+1$ as the last entry of $\boldsymbol{v}$ and subtract it from $\boldsymbol{v}$, i.e.,

$$\boldsymbol{v}' := \boldsymbol{v} - (z+1)\mathbf{1}_{1 \times (n+1)}$$
$$= [\boldsymbol{m}' + \boldsymbol{z} \mid \boldsymbol{m}] \begin{bmatrix} \boldsymbol{I}_{l \times l} & -\boldsymbol{R}_{l \times (n-l)} - & \mathbf{0}_{l \times 1} \\ \mathbf{0}_{k_1 \times l} & \boldsymbol{I}_{k_1 \times k_1} & \boldsymbol{P}_{k_1 \times r} & \mathbf{0}_{k_1 \times 1} \end{bmatrix}$$

Then, the first l positions of $\boldsymbol{v}'$ equals $\hat{\boldsymbol{m}}'' = \boldsymbol{m}' + \boldsymbol{z}$. Thus, we can also subtract

$$\boldsymbol{v}'' := \boldsymbol{v}' - (\boldsymbol{m}' + \boldsymbol{z}) \cdot [\boldsymbol{I}_{l \times l} \quad \boldsymbol{R}_{l \times (n-l)} \quad \mathbf{0}_{l \times 1}]$$
$$= \boldsymbol{m} \cdot [\mathbf{0}_{k_1 \times l} \quad \boldsymbol{I}_{k_1 \times k_1} \quad \boldsymbol{P}_{k_1 \times r} \quad \mathbf{0}_{k_1 \times 1}]$$

and obtain $\boldsymbol{m} = \hat{\boldsymbol{m}}$ as the $l+1$st to $l + k_1$-th position of $\boldsymbol{v}''$.

It remains to prove that we can uniquely retrieve $\boldsymbol{m}' = \hat{\boldsymbol{m}}'$ from $\boldsymbol{m}' + \boldsymbol{z} = \hat{\boldsymbol{m}}''$. Recall that the entries of $\boldsymbol{m}'$ are in $\mathcal{F}$ (i.e., elements are, represented in a polynomial basis x^i of $\mathbb{F}_{2^\lambda}$ over $\mathbb{F}_2$, of the form $\sum_{i=0}^{\lambda-1} a_i x^i$ with $a_0 = 0$). Hence, for $\alpha \in \mathbb{F}_2$ and $\beta = 0x^0 + \sum_{i=1}^{\lambda-1} a_i x^i \in \mathcal{F}$, we have

$$\varphi(\alpha + \beta) = \varphi\left(\alpha x^0 + \sum_{i=1}^{\lambda-1} a_i x^i \in \mathcal{F}\right)$$
$$= 0x^0 + \sum_{i=1}^{\lambda-1} a_i x^i \in \mathcal{F} = \beta.$$

In summary, Algorithm 1 encodes messages $\boldsymbol{m}$ and $\boldsymbol{m}'$ into a codeword of the code $\mathcal{C}$ such that all u partially-stuck positions are non-zero (i.e., masked). Algorithm 2 then corrects up to t errors using the code $\mathcal{C}$ and recovers the message vectors. This means that the coding scheme is a (u, t)-PSMC. The cardinality of the code is given by

$$\mathcal{M}_{u,t} = |\mathbb{F}_{2^\lambda}|^{k_1} |\mathcal{F}|^l = 2^{\lambda(k_1+l)-l},$$

which concludes the proof. $\square$

On the first glance, it is not immediately clear how to construct the matrices $\boldsymbol{H}_0$ and $\boldsymbol{G}_1$ in Theorem 1. An intuition is as follows: we look for a code $\mathcal{C}$ that

- has minimum distance $\geq 2t + 1$ (to correct at least t errors),
- contains the all-one vector, and
- when punctured at the last position, its binary subfield subcode must contain a code (generated by $\boldsymbol{H}_0$) whose dual code has minimum distance at least $u_0 + 1$.

Construction 1 can give higher rates for some chosen codes than using [2, Theorem 4]. Example 1 in the appendix shows a code $\mathcal{C}$ of parameters $[15, 11, 3]_4$ and its generator matrix $\boldsymbol{G}$, $u = 4$ and $t = 1$. Using Theorem 1, we can store $k_1 + l = 6 + 4 = 10$ information symbols. In contrast [2, Theorem 4] can only store $k_1 = 7$ information symbols using $\boldsymbol{G}'$, while in both theorems we can mask $u = q$ cells.

IV. BOUNDS ON THE CARDINALITY AND MINIMUM DISTANCE

In this section, we derive bounds on (u,t)-PSMCs with the goal to evaluate the parameters of code constructions. We derive a sphere-packing-type bound (necessary condition) and a *Gilbert-Varshamov* (GV)-type bound (sufficient condition). Notice that the GV-like bound proves the existence of (u,t)-PSMCs only when $u < q$.

A. Sphere-Packing Bound on PSMCs

The *sphere-packing bound* considers the size of a code by packing spheres around each codeword. We combine the proof of the classical sphere packing bound with constraints from partially stuck cells.

Theorem 2. *Any (u,t)-PSMC over $\mathbb{F}_q$ of cardinality $\mathcal{M}_{u,t}$ has to satisfy:*

- *for non-overlapping errors:*

$$\mathcal{M}_{u,t} \cdot \sum_{j=0}^{t} \binom{n-u}{j}(q-1)^j \leq q^{n-u} \prod_{i=0}^{u-1}(q-s_{\phi_i}), \quad (2)$$

- *for overlapping errors under the assumption that $s_{\phi_i} < q-1$, for $i = 0, \ldots, u-1$:*

$$\mathcal{M}_{u,t} \cdot \sum_{j=0}^{t} \sum_{j_1=0}^{j} \binom{n-u}{j_1}(q-1)^{j_1}\binom{u}{j-j_1} \prod_{i \in \mathcal{J}}(q-1-s_{\phi_i})$$

$$\leq q^{n-u} \prod_{i=0}^{u-1}(q-s_{\phi_i}), \quad (3)$$

where $\mathcal{J}$ denotes the set of cardinality $j - j_1$ of stuck cells that is affected by errors.

Proof. First we prove the **non-overlapping** scenario. [1, Theorem 2] proves that at most $\mathcal{M}_u$ q-ary codewords can be stored in a memory in the presence of u partially stuck cells, where:

$$\mathcal{M}_u \leq q^{n-u} \prod_{i=1}^{u}(q-s_{\phi_i}). \quad (4)$$

We assume that the errors can happen only in the $n - u$ non-stuck cells. A sphere $\mathcal{B}_{t,n-u}(\boldsymbol{a})$ of radius t around a word $\boldsymbol{a} \in \mathbb{F}_q^{n-u}$ is the set of all words in Hamming distance at most t, i.e, $\mathcal{B}_{t,n-u}(\boldsymbol{a}) := \{\boldsymbol{b} \in \mathbb{F}_q^{n-u} : d(\boldsymbol{a},\boldsymbol{b}) \leq t\}$. There are $\binom{n-u}{j}$ words in distance exactly j from a fixed word and $|\mathcal{B}_{t,n-u}(\boldsymbol{a})| = \sum_{j=0}^{t}\binom{n-u}{j}(q-1)^j$. Thus, the total number of words in all decoding spheres (left-hand side (LHS) of (2)) is at most the total number of possible words $\mathcal{M}_u$ (right-hand side (RHS) of (2)).

Second we prove the **overlapping** scenario. Clearly, (4) is still an upper bound on the total number of possible words, i.e., the RHS of the sphere-packing bound. For the LHS in this case, the errors can happen either in the $n - u$ non-stuck cells or in the u stuck cells such that $s_{\phi_i} + e_i \leq q - 1$.

In this case, there are $\sum_{j_1=0}^{j}\binom{n-u}{j_1}\binom{u}{j-j_1}$ possibilities for j erroneous positions (i.e., j_1 errors happen at non-stuck positions and $j - j_1$ errors happen at stuck positions). Therefore, there are $\mathcal{B}_{t,u,n} := \sum_{j=0}^{t}\sum_{j_1=0}^{j}\binom{n-u}{j_1}(q-1)^{j_1}\binom{u}{j-j_1}\prod_{i \in \mathcal{J}}(q-1-s_{\phi_i})$ distinct words that can result from a fixed word when u partially stuck cells and at most t random errors happen, where $\mathcal{J}$ denotes the set of cardinality $j - j_1$ of stuck cells that is affected by errors.

Since the set of these $\mathcal{B}_{t,u,n}$ words around a fixed codeword is disjoint to the corresponding set around another fixed codeword, $(\mathcal{M}_{u,t} \cdot \mathcal{B}_{t,u,n})$ is at most the total number of possibilities and the statement follows. $\square$

Figure 2 illustrates the new sphere packing bounds. They are compared to the amount of storable information symbols for a completely reliable memory (i.e., no stuck cells, no errors) and the upper bound on the cardinality of an only-masking PSMC (only stuck cells, no errors) derived in [1]. The figure also compares the overlapping and non-overlapping error model for $s_{\phi_i} = 1$, for all i. In the overlapping scenario, when the number of errors is small (e.g., $t = 3$), the number of information symbols is bounded by the sphere packing bound and it is slightly better than in non-overlapping case. The more errors happen (e.g., $t = 25$ in the right figure), a smaller number $(n - u)$ of non-stuck cells can be affected by errors, so it is very likely that many of the errors happen at partially stuck positions which affects the amount of storable information less than an error in a non-stuck position.
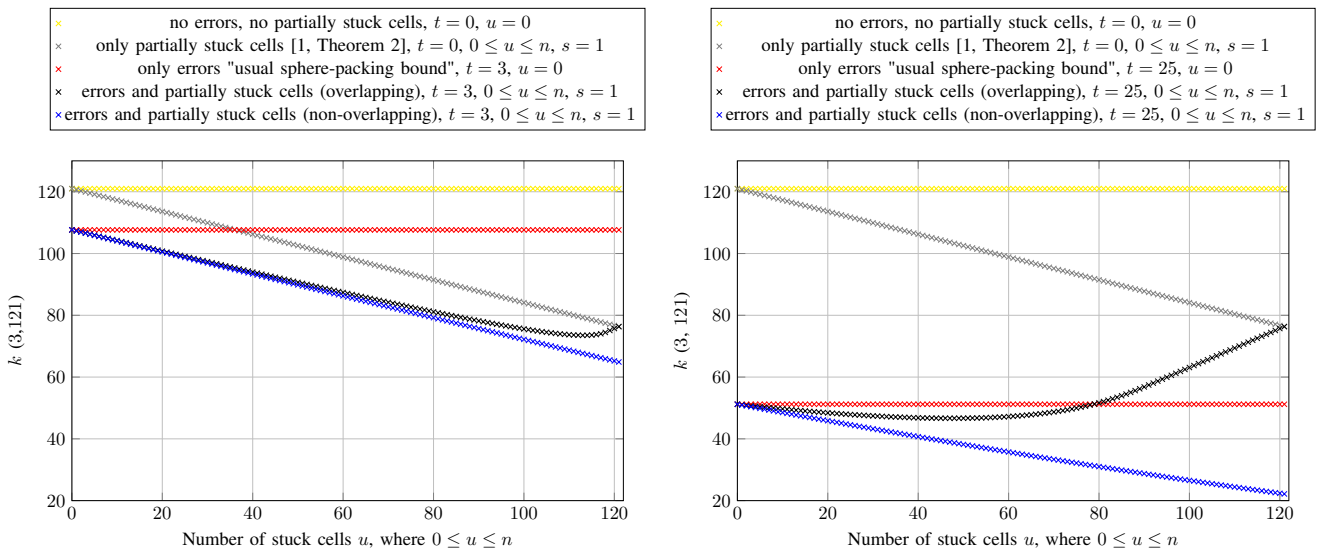

	no errors, no partially stuck cells, $t = 0$, $u = 0$
×	only partially stuck cells [1, Theorem 2], $t = 0$, $0 \leq u \leq n$, $s = 1$
×	only errors "usual sphere-packing bound", $t = 3$, $u = 0$
×	errors and partially stuck cells (overlapping), $t = 3$, $0 \leq u \leq n$, $s = 1$
×	errors and partially stuck cells (non-overlapping), $t = 3$, $0 \leq u \leq n$, $s = 1$

	no errors, no partially stuck cells, $t = 0$, $u = 0$
×	only partially stuck cells [1, Theorem 2], $t = 0$, $0 \leq u \leq n$, $s = 1$
×	only errors "usual sphere-packing bound", $t = 25$, $u = 0$
×	errors and partially stuck cells (overlapping), $t = 25$, $0 \leq u \leq n$, $s = 1$
×	errors and partially stuck cells (non-overlapping), $t = 25$, $0 \leq u \leq n$, $s = 1$

Figure 2. **Sphere-packing bounds**: Comparison for $k(q,n)$ information symbols for the classical sphere-packing bound ("only errors") and our sphere-packing-like bounds ("errors and stuck cells") for non-overlapping and overlapping errors. The chosen parameters are $\lambda = 5$ and $q = 3$, and $n = ((q^\lambda - 1)/(q-1))$.

B. Gilbert–Varshamov Bound on PSMCs

In this section, we derive a sufficient condition for the existence of a code with certain parameters that can mask partially stuck cells and correct errors. To derive this bound, we rely on our previous construction of PSMCs in [2], which for $u < q$ masked cells solely required the existence of an error-correcting code with minimum distance $\geq 2t + 1$, which contains the all-one vector. In the proof, we therefore prove the existence of a code which contains the all-one vector as codeword.

Theorem 3 (Gilbert-Varshamov-like bound). *Let the positive integers n, $k \leq n$, $d \leq n$, q fulfill:*

$$\sum_{i=0}^{d-2} \binom{n-1}{i} (q-1)^i < q^{n-k}. \tag{5}$$

Then, there exists an $[n', k', d]_q$ code that contains the all-one vector, where n', and k' satisfy:

$$n - d + 2 \leq n' \leq n + 1, \qquad k - d + 2 \leq k' \leq k + 1.$$

The parity-check matrix of this $[n', k', d]_q$ code can be constructed as shown in the proof.

Proof. Similar to the proof of the standard Gilbert–Varshamov bound, we construct a systematic parity-check matrix by adding columns $\boldsymbol{h}_l$ for $l = k+1, k+2, \dots$ to a $k \times k$ identity matrix as long as:

$$\sum_{i=0}^{d-2} \binom{l-1}{i} \cdot (q-1)^i < q^{n-k}. \tag{6}$$

Recall from the proof of the Gilbert-Varshamov bound that this condition ensures that there exists a column $\boldsymbol{h}_l$ that is linearly independent of any collection of $d - 2$ other columns.

If (6) is not fulfilled anymore for $l = n + 1$, we append an additional parity-check column $\boldsymbol{p}$ to the previous n columns such that the sum of each row is zero (i.e., the weight is even in the binary case). This matrix is therefore:

$$\boldsymbol{H}_e := \left[\; \underbrace{\left(\boldsymbol{h}_1, \dots, \boldsymbol{h}_n\right)}_{n} \; \middle| \; \boldsymbol{p} \; \right],$$

where

$$\sum_{i=1}^{n} \boldsymbol{h}_i + \boldsymbol{p} = \boldsymbol{0}. \tag{7}$$

However, for $\boldsymbol{H}_e$, we cannot guarantee anymore that any $d-1$ columns are linearly independent (as $\boldsymbol{p}$ might be linearly dependent on a small number of $\boldsymbol{h}_i$'s.). Therefore, in the following, we possibly remove a few columns from $\boldsymbol{H}_e$ to recover this property while still having zero row sums.

If $\boldsymbol{p}$ is linearly independent of any $d - 1$ columns in $\boldsymbol{h}_1, \dots, \boldsymbol{h}_n$, we define $\boldsymbol{H} := \boldsymbol{H}_e$.

Else $\boldsymbol{p}$ is linearly dependent of $\delta \leq d - 2$ columns $\{\boldsymbol{h}_{i_1}, \boldsymbol{h}_{i_2}, \dots, \boldsymbol{h}_{i_\delta}\} \subseteq \{\boldsymbol{h}_1, \boldsymbol{h}_2, \dots, \boldsymbol{h}_n\}$, and $\boldsymbol{p}$ is a linear combination of these δ columns:

$$\boldsymbol{p} = \sum_{j=1}^{\delta} \boldsymbol{h}_{i_j} \cdot a_j, \quad \text{where } a_j \in \{1, 2, \dots, q-1\}.$$

Thus with $1 \leq \delta_1 \leq \dots \leq \delta_{q-1} \leq \delta$ (by assuming w.l.o.g. an ordering on the indices),

$$\boldsymbol{p} = \sum_{j=1}^{\delta_1} \boldsymbol{h}_{i_j} + 2 \cdot \sum_{j=\delta_1+1}^{\delta_2} \boldsymbol{h}_{i_j} + \dots + (q-1) \cdot \sum_{j=\delta_{q-1}+1}^{\delta} \boldsymbol{h}_{i_j}. \tag{8}$$

We can rewrite (7) as:

$$\sum_{i=1\setminus\{i_1,\dots,i_\delta\}}^{n} \boldsymbol{h}_i + \sum_{j=1}^{\delta_1} \boldsymbol{h}_{i_j} + \sum_{j=\delta_1+1}^{\delta_2} \boldsymbol{h}_{i_j} + \dots + \sum_{j=\delta_{q-1}+1}^{\delta} \boldsymbol{h}_{i_j} + \boldsymbol{p} = \boldsymbol{0}.$$

Combining this with (8) yields:

$$\sum_{i=1\setminus\{i_1,\dots,i_\delta\}}^{n} \boldsymbol{h}_i + \sum_{j=1}^{\delta_1} \boldsymbol{h}_{i_j} + \sum_{j=\delta_1+1}^{\delta_2} \boldsymbol{h}_{i_j} + \dots + \sum_{j=\delta_{q-1}+1}^{\delta} \boldsymbol{h}_{i_j} + \boldsymbol{p}$$

$$+ \sum_{j=1}^{\delta_1} \boldsymbol{h}_{i_j} + 2 \cdot \sum_{j=\delta_1+1}^{\delta_2} \boldsymbol{h}_{i_j} + \dots + (q-1) \cdot \sum_{j=\delta_{q-1}+1}^{\delta} \boldsymbol{h}_{i_j} - \boldsymbol{p}.$$

$$= \boldsymbol{0}.$$

Therefore,

$$\sum_{i=1\setminus\{i_1,\dots,i_\delta\}}^{n} \boldsymbol{h}_i + (2 \mod q) \sum_{j=1}^{\delta_1} \boldsymbol{h}_{i_j} + (3 \mod q) \sum_{j=\delta_1+1}^{\delta_2} \boldsymbol{h}_{i_j}$$

$$+ \dots + (q-1 \mod q) \sum_{j=\delta_{q-2}+1}^{\delta_{q-1}} \boldsymbol{h}_{i_j} = \boldsymbol{0}. \tag{9}$$

Therefore, the matrix

$$\boldsymbol{H} :=$$

$$\left(\underbrace{\boldsymbol{h}_1', \dots, \boldsymbol{h}_{n-\delta}'}_{n-\delta} \; \middle| \; \underbrace{2\boldsymbol{h}_{i_1}, \dots, 2\boldsymbol{h}_{i_{\delta_1}}}_{\delta_1} \; \middle| \dots \middle| \; \underbrace{-\boldsymbol{h}_{i_{\delta_{q-2}+1}}, \dots, -\boldsymbol{h}_{i_{\delta_{q-1}}}}_{\delta_{q-1}-\delta_{q-2}} \right)$$

where $\boldsymbol{h}_1', \dots, \boldsymbol{h}_{n-\delta}' = \{\boldsymbol{h}_1, \dots, \boldsymbol{h}_n\} \setminus \{\boldsymbol{h}_{i_1}, \dots, \boldsymbol{h}_{i_\delta}\}$, has sum equal to zero in all rows due to (9) and any $d - 1$ columns are linearly independent since they are all columns (times a non-zero scalar) of the matrix $(\boldsymbol{h}_1, \dots, \boldsymbol{h}_n)$.

The number of columns n' of $\boldsymbol{H}$ is bounded by

$$n - d + 2 \leq n - \delta \leq n' \leq n + 1,$$

where $n' = n + 1$ if $\boldsymbol{p}$ was linearly independent of any $d - 2$ other columns and therefore no columns have to be removed.

Substituting n in l of (6), we obtain:

$$\sum_{i=0}^{d-2} \binom{n-1}{i} (q-1)^i < q^{n-k}. \tag{10}$$

Since $n \leq n' + d - 2$ and since $n - k = n' - k'$ (the number of rows did not change), we get

$$\sum_{i=0}^{d-2} \binom{n'+d-3}{i} (q-1)^i < q^{n'-k'}.$$

Thus, if this is true, there exists an $[n', k', d]_q$ code that contains the all-one vector, where $k' = n' - (n' - k') = n' - (n - k) = n' - n + k \geq n - d + 2 - n + k = k - d + 2$. $\qquad \square$

Corollary 1. *Let $u < q$ and let (5) hold, i.e., such that an $[n', k', d]_q$ code that contains the all-one vector as codeword exists. Then, there is a $(u, \lfloor \frac{d-1}{2} \rfloor)$-PSMC of length n'.*

Proof. In [2, Theorem 1], it was shown that if the all-one vector is a codeword of a code with minimum distance d, then for $u < q$, there is a $(u, \lfloor \frac{d-1}{2} \rfloor)$-PSMC. $\qquad \square$

Figure 3 compares the new GV-like bound to other well-known bounds. To explain how, let us first define the following:

- Let n_f and k_f denote the n and k that is chosen to compute one point in the figure (here $n_f = 127, 121, 124,$ and 120 and $k_f =$ values on the x-axis).
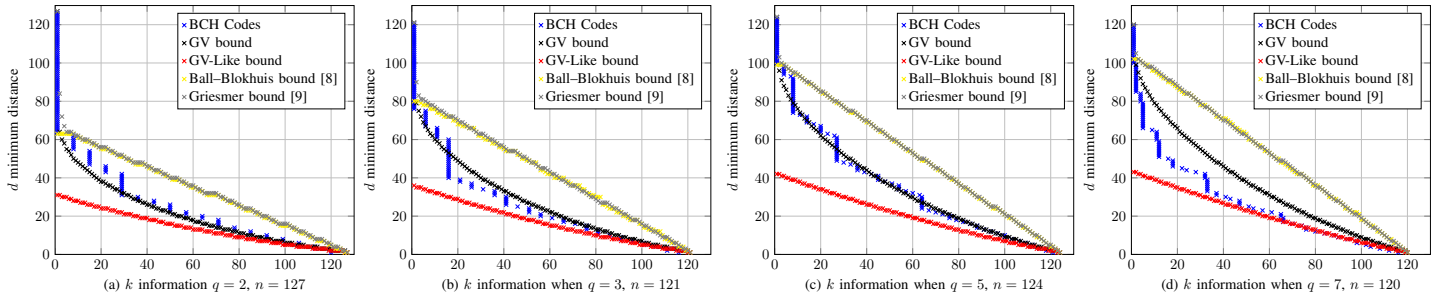
Figure 3. Comparisons of bounds of our new GV-Like-Bound for different n and q. For BCH codes, we use our previous construction (for $u < q$) [2, Theorem 1] since BCH codes have the all-one vector in their generator matrices.

- Let n and k be the *designed* length and dimension in the GV bound, i.e., what is used in the inequality (5).
- Let n' and k' be the actual length and dimension of the existing code.

In the plots, we want to know for a fixed pair $[n_f, k_f]$, what the maximal minimum distance d is that guarantees existence of a code. To have a fair comparison to another lower bound or construction, we chose the *designed* n and k to our largest disadvantage, i.e. $(n_f = n + 1)$ which gives $(n = n_f - 1)$ and $(k_f = k - d + 2)$ which gives $(k = k_f + d - 2)$. Next, we have to obtain d, so we plugged for $d = 1, 2, \ldots$ his n and k (depending on d) into the bound in (5) and computed the maximal d that still satisfies the bound. Then we get a d and Theorem 3 tells us that there is an $[n', k', d]_q$ code with $(n - d + 2 \leq n' \leq n + 1 = n_f)$ and $(k_f = k - d + 2 \leq k' \leq k + 1)$. In particular, the actual code has length $n' \leq n_f$ and dimension $k' \geq k_f$. Hence, a code with exact parameters $[n_f, k_f, \geq d]_q$ exists. However, the values obtained for this d are quite bad (the red curve). For high dimensions k_f and larger q, these values tend to be better as shown in (d), i.e. the red curve matches, and then is above the BCH curve.

V. CONCLUSION

We have proposed a new construction for combined masking of partially stuck-at-1 cells and error correction, by masking only binary classical stuck memory cells as proposed in [1, Thorem 9], with the error correction possibility similar to [2, Theorem 4]. Compared to [1, Thorem 9], the new code construction can correct errors in addition to masking. Furthermore, for specific examples on the code parameters, a higher amount of information symbols $k_1 + l$ when $u \leq q \leq n$ can be stored compared to the code construction for masking and error correction in [2, Theorem 4]. Further, we have derived bounds on the required redundancy for a given number of partially stuck cells to mask and a given number of errors to correct. This includes a sphere-packing and a Gilbert–Varshamov-like bound.

Future work should calculate the capacity of a storage channel in which u cells can be partially stuck at levels s with probability p and the rest $(u - n)$ healthy cells with probability $(1 - p)$. If we assume disjoint case, $(u - n)$ cells have also crossover probability ε, i.e errors occur in $(u - n)$ cells only. Then we should compare it to the code rate of the new construction similar to [1, Section IX].

ACKNOWLEDGEMENT

We would like to thank Ludo Tolhuizen for making us aware of a more accurate curve regarding GV-like bound by commenting on the previous version of this paper.

REFERENCES

[1] A. Wachter-Zeh and E. Yaakobi, "Codes for Partially Stuck-at Memory Cells," *IEEE Transactions on Information Theory*, vol 62, no. 2, pp.639-654, 2016.

[2] H. Al Kim, S. Puchinger, and A. Wachter-Zeh, "Error Correction for Partially Stuck Memory Cells," *in 2019 XVI International Symposium "Problems of Redundancy in Information and Control Systems" (REDUNDANCY)*, 2019, pp. 87–92.

[3] B. Gleixner, F. Pellizzer, and R. Bez, "Reliability Characterization of Phase Change Memory," in *2009 10th Annual Non-Volatile Memory Technology Symposium (NVMTS)*. IEEE, 2009, pp. 7–11.

[4] K. Kim and S. J. Ahn, "Reliability Investigations for Manufacturable High Density PRAM," in *2005 IEEE International Reliability Physics Symposium, 2005. Proceedings. 43rd Annual*. IEEE, 2005, pp. 157–162.

[5] S. Lee, J.-h. Jeong, T. S. Lee, W. M. Kim, and B.-k. Cheong, "A Study on the Failure Mechanism of a Phase-Change Memory in Write/Erase Cycling," *IEEE Electron Device Letters*, vol. 30, no. 5, pp. 448–450, 2009.

[6] A. Pirovano, A. Redaelli, F. Pellizzer, F. Ottogalli, M. Tosi, D. Ielmini, A. L. Lacaita, and R. Bez, "Reliability Study of Phase-Change Nonvolatile Memories," *IEEE Transactions on Device and Materials Reliability*, vol. 4, no. 3, pp. 422–427, 2004.

[7] C. Heegard, "Partitioned Linear Block Codes for Computer Memory with 'Stuck-at' Defects," *IEEE Transactions on Information Theory*, vol. 29, no. 6, pp. 831–842, 1983.

[8] S. Ball and A. Blokhuis, "A Bound for the Maximum Weight of a Linear Code," *SIAM Journal on Discrete Mathematics*, vol. 27, no. 1, pp. 575–583, 2013.

[9] J. H. Griesmer, "A Bound for Error-Correcting Codes," *IBM Journal of Research and Development*, vol. 4, no. 5, pp. 532–542, 1960.

VI. APPENDIX

Example 1. Binary Codes for Masking and Correcting Partially Defect Memory

Let $\boldsymbol{m} = \begin{pmatrix} 1 & 0 & 1 & \alpha & 1 + \alpha & 1 \end{pmatrix} \in \mathbb{F}_4^6$, $\boldsymbol{m}' = \begin{pmatrix} \alpha & 0 & \alpha & 0 \end{pmatrix} \in \mathcal{F}^4 \subseteq \mathbb{F}_{2^\lambda}$. Let $u = 4$ so that $u_0 = 2$. Since we mask u-PSMC by the mean of u_0-SMC, we need a code $\mathcal{C}_0$ of a minimum distance $d_0 = u_0 + 1 = 3$ that its dual code is generated by $\boldsymbol{H}_0$. Let the u stuck positions be $\phi_0 = 1, \phi_1 = 2, \phi_2 = 9$ and $\phi_3 = 14$. Let $\boldsymbol{G}$ over $\mathbb{F}_4$ be a generator matrix of a code $\mathcal{C}$ with parameters $[15, 11, 3]_4$ from Theorem 1:

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{H}_0 & \boldsymbol{0} \\ \boldsymbol{G}_1 & \vdots \\ 1 \ldots & \boldsymbol{0} \\ & 1 \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Encoding follows Algorithm 1 by plugging in the given values and matrices.

$\boldsymbol{G}'$ over $\mathbb{F}_4$ of the code $\mathcal{C}$ of the parameters $[15, 11, 3]_4$ in [2, Theroem 4] is:

$$\boldsymbol{G}' = \begin{bmatrix} \boldsymbol{0}_{k_1 \times l} & \boldsymbol{I}_{k_1} & \boldsymbol{P}_{k_1 \times r} \\ & \boldsymbol{H}_0 & \end{bmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

It is good to mention that if we take the *reduced echelon form* for both $\boldsymbol{G}$ and $\boldsymbol{G}'$, the result is the same matrix $\boldsymbol{G}_e$. However, applying Theorem 1 gives higher rate compare to [2, Theorem 4].