Efficient Encoding/Decoding of Capacity-Achieving Constant-Composition ICI-Free Codes

Yeow Meng Chee, Chrisnata Johan, Han Mao Kiah, San Ling, Tuan Thanh Nguyen, and Van Khu Vu
School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
email: {ymchee, jchrisnata, hmkiah, lingsan, tuan3, vankhu001}@ntu.edu.sg

***Abstract*—We give the first known efficient encoder/decoder for q-ary constant-composition ICI-free codes achieving ICI channel capacity, for all q. Previously, the best result known is an efficient encoder/decoder for binary constant-weight ICI-free codes with more than 2% loss over ICI channel capacity.**

I. Introduction

Flash memories have become a popular nonvolatile storage of information owing to its advantage of high speed, low noise, low power consumption, compact form factor, and good physical reliability. The basic information storage element of a flash memory is called a *cell*, which consists of a floating-gate (FG) transistor. The amount of charge on an FG transistor is discretized into *charge levels* as a way to store information. The operation of injecting charge into an FG transistor to a desired level is called *programming*. In a single level cell (SLC) flash memory, each cell has two charge levels (corresponding to a charged or uncharged FG transistor), and hence can store one bit per cell. More recent multi-level cell (MLC[1]) flash memories have cells with $q > 2$ charge levels, with the ability to store $\log_2 q$ bits per cell. More specifically, we use qLC to refer to cells with q charge levels. The cells of a flash memory are further organized into *blocks*, each containing a constant number of cells. Hence, a block in a qLC flash memory stores a q-ary word (where symbol i is used to represent charge level i of a cell), and such a flash memory stores a collection of q-ary words.

MLC technology increases the storage density of flash memories. However, very precise programming is needed. There are two main challenges to reliable programming and storage:

(i) Intercell interference (ICI) caused by parasitic capacitance coupling between adjacent cells [1]. Such interference occurs when there are three adjacent cells c_1, c_2, c_3 and we want to increase the charge levels of the left-most and right-most cells, c_1 and c_3, while maintaining the charge level of the center cell c_2. Parasitic capacitance coupling can cause the charge level of the (victim) cell c_2 to increase when we increase the charge levels of its neighbouring cells c_1 and c_3.

(ii) Charge leakage [2]. The charge in an FG transistor leaks away over time as a result of trap-assisted tunneling effect. This results in charge levels of cells drifting downwards over time, giving rise to asymmetric errors.

[1]MLC is commonly used to refer to the specific technology that allows four charge levels per cell. For lack of a better notion, we extend the use of "MLC" here to refer to technology allowing three or more charge levels per cell.

Different techniques have been explored to mitigate ICI. Physical methods, such as using low-κ dielectric material to reduce capacitative coupling [3], and programming methods such as proportional programming [4], have been investigated but the approach that is most effective has been the constrained coding method of Berman and Birk [5]–[7]. In their approach, certain words are forbidden to be stored, since the programming required to store such a word is highly unreliable, owing to ICI. For example, the quaternary word of length eight $(1, 2, 1, 3, 0, 3, 2, 0)$ should be avoided as the charge level of the fifth cell can be increased unintentionally during the programming of the fourth and sixth cells. More generally, Taranalli et al. [8] performed a comprehensive series of program/erase (P/E) cycling experiments recently to quantify ICI effects, and concluded that the words permitted for storage on a qLC flash memory should avoid containing any $(q-1, \sigma, q-1)$ as a substring, where $\sigma \in \{0, 1, \dots, q-2\}$.

To mitigate the effect of charge leakage, a straightforward way is to adopt asymmetric error-correcting codes [9], [10]. Dynamic threshold techniques, introduced by Zhou et al. [11] for SLC and extended to MLC by Sala et al. [12], have been shown to be not only highly effective against asymmetric errors caused by charge leakage but also offer some protection against over-programming. In error-correcting schemes with dynamic threshold, the codes have constant composition, and in particular, the case when the codes have both constant composition and balanced (where the number of times a symbol appears in a codeword is as close as possible) was studied in detail by Zhou et al. and Sala et al. [11], [12].

Recent approaches have combined constrained coding and dynamic threshold techniques [13]. Before we give an account of these results, we introduce some necessary notations and terminology.

A. Notations and Terminology

For n a positive integer, the set $\{1, 2, \dots, n\}$ is denoted $[\![n]\!]$. Let $\Sigma = \{0, 1, \dots, q-1\}$ be an alphabet of $q \geq 2$ symbols. A q-ary *word* of length n over Σ is an element $\mathsf{u} \in \Sigma^n$. The ith coordinate of u is denoted u_i, so that $\mathsf{u} = (\mathsf{u}_1, \mathsf{u}_2, \dots, \mathsf{u}_n)$. The word of all ones, $(1, 1, \dots, 1) \in \Sigma^n$, is denoted j, and the word with a "1" in position m and "0" everywhere else is denoted e_m. There is a natural correspondence between the data represented by the charge levels of a block of n cells in a qLC flash memory and a q-ary word $\mathsf{u} \in \Sigma^n$: u_i is the charge level of the ith cell in the block, for all $i \in [\![n]\!]$.

For a positive integer n, a *composition of n into q parts* is an (ordered) q-tuple, $\overline{w}(n) = [w_0, w_1, \dots, w_{q-1}]$ of non-negative integers such that $\sum_{i=0}^{q-1} w_i = n$. We normally write

$\overline{w}$ for $\overline{w}(n)$, unless we want to emphasize a composition's dependence on n. A q-ary word $\mathsf{u} \in \Sigma^n$ is said to have *composition* $[w_0, w_1, \ldots, w_{q-1}]$ if the frequency of symbol $\sigma \in \Sigma$ in u is w_σ. The *weight* of a word $\mathsf{u} \in \Sigma^n$ with composition $[w_0, w_1, \ldots, w_{q-1}]$ is $w = \sum_{\sigma=1}^{q-1} w_\sigma$. A word $\mathsf{u} \in \Sigma^n$ is said to be *balanced* if it has composition $[w_0, w_1, \ldots, w_{q-1}]$ such that $w_\sigma \in \{\lfloor n/q \rfloor, \lceil n/q \rceil\}$, for all $\sigma \in \Sigma$. Hence, in a balanced word, every symbol is as evenly distributed as possible.

A *substring* of a word $\mathsf{u} = (\mathsf{u}_1, \mathsf{u}_2, \ldots, \mathsf{u}_n) \in \Sigma^n$ is a word $(\mathsf{u}_{i+1}, \mathsf{u}_{i+2}, \ldots, \mathsf{u}_{i+m}) \in \Sigma^m$, where $i \geq 0$ and $i + m \leq n$. Let $\mathcal{F}$ be a set of words over Σ. A word $\mathsf{u} \in \Sigma^n$ is said to *avoid* $\mathcal{F}$ if no words in $\mathcal{F}$ is a substring of u.

A q-ary *code* of length n is a nonemtpy subset $\mathcal{C} \subseteq \Sigma^n$. Elements of $\mathcal{C}$ are called *codewords*. The *size* of $\mathcal{C}$ is the number of codewords in $\mathcal{C}$. A code $\mathcal{C}$ is said to have

(i) *constant weight* w, if every codeword in $\mathcal{C}$ has weight w; and
(ii) *constant composition* $\overline{w}$, if every codeword in $\mathcal{C}$ has composition $\overline{w}$.

Note that a constant-composition code is also constant-weight, though the reverse need not hold. However, for the case of binary codes ($q = 2$), the notions of constant-composition and constant-weight are equivalent. A code is *balanced* if each of its codewords is balanced. Let $\mathcal{F}$ be a set of words over Σ. A code $\mathcal{C} \subseteq \Sigma^n$ is said to *avoid* $\mathcal{F}$ if every codeword in $\mathcal{C}$ avoids $\mathcal{F}$.

The *rate* of a code $\mathcal{C} \subseteq \Sigma^n$ is $R = \log_2 |\mathcal{C}|/n$.

Define $\mathcal{I}(q) = \{(q-1, \sigma, q-1) : 0 \leq \sigma \leq q-2\}$.

Example 1. $\mathcal{I}(2) = \{(1,0,1)\}$ and $\mathcal{I}(4) = \{(3,0,3), (3,1,3), (3,2,3)\}$.

Observe that $\mathcal{I}(q)$ is the set of charge levels of three adjacent cells in a qLC flash memory that we should avoid since they give rise to ICI effects. This motivates the following definition.

Definition 1 (ICI-Free Code)**.** A q-ary code $\mathcal{C} \subseteq \Sigma^n$ is *ICI-free*[2] if it avoids $\mathcal{I}(q)$.

An *ICI channel* is a channel whose input codewords are ICI-free. The *(Shannon) capacity* of a q-ary ICI channel is

$$C_{\mathrm{ICI}}(q) = \lim_{n\to\infty} \frac{\log_2 |\mathcal{C}_n|}{n},$$

where $\mathcal{C}_n$ is the set of all ICI-free words of length n over Σ.

As mentioned earlier, recent approaches combine constrained coding and dynamic threshold techniques, leading to the consideration of codes that are both ICI-free and constant-composition. The set of all q-ary ICI-free words of length n and constant composition $\overline{w}$ is denoted $\mathcal{S}(n, \overline{w})$. Note that q, the size of the alphabet, is determined by the composition $\overline{w}$. In the case $q = 2$, we further abbreviate $\mathcal{S}(n, [w_0, w_1])$ to $\mathcal{S}(n, w_1)$. The size of $\mathcal{S}(n, \overline{w})$ is denoted by $A_{\mathrm{ICI}}(n, \overline{w})$.

Let $\overline{w}(n) = [w_0, w_1, \ldots, w_{q-1}]$. Define $\omega_i = \lim_{n\to\infty} w_i/n$. Note that we necessarily have $\sum_{i=0}^{q-1} \omega_i = 1$.

[2]Qin et al. [13] used "ICI-free" to mean codes that only avoid $\{(q-1, 0, q-1)\}$, but Taranalli et al. [8] have shown that avoiding $\{(q-1, 0, q-1)\}$ is not enough to mitigate ICI effects, and that the entire set $\mathcal{I}(q)$ should be avoided.

The *asymptotic information rate* of ICI-free codes of constant composition $\overline{w}(n)$ is defined as

$$R(\omega_0, \omega_1, \ldots, \omega_{q-1}) = \limsup_{n\to\infty} \frac{\log_2 A_{\mathrm{ICI}}(n, \overline{w}(n))}{n}.$$

The interest is to seek such codes of large size, or of high rate, with the ultimate goal of constructing constant-composition ICI-free codes whose rate meets the ICI channel capacity, and whose encoding and decoding can be performed with low complexity.

B. Previous Work

The capacity of binary ICI channels has been determined by Kayser and Siegel [14].

Proposition 1 (Kayser and Siegel [14])**.** *$C_{\mathrm{ICI}}(2) = \log_2(z) \approx 0.81137$, where z is the unique real root of $z^3 - 2z^2 + z - 1$.*

Binary ICI-free constant-weight codes were considered by Qin et al. [13], with the focus on balanced binary ICI-free codes of even length (making them also constant-weight). The intuition behind the criterion of balance is that

(i) these codes have the largest size among all constant-weight codes; and
(ii) these codes have simple encoders and decoders.

In particular, they showed:

Proposition 2 (Qin et al. [13])**.** *$R(1/2, 1/2) = (\log_2 3)/2 \approx 0.79248$.*

These balanced ICI-free codes of Qin et al. [13] have rates that fall short of over 2% of the ICI channel capacity.

Let $0 < p < 1$. Kayser and Siegel [14] constructed a family $\{\mathcal{C}_n\}_{n\geq 1}$ of binary constant-weight ICI-free codes, parametrized by a positive integer m, such that each $\mathcal{C}_n$ is an ICI-free code of constant composition $[(1-p)n, pn]$, and showed that there exists a p such that

$$\lim_{m\to\infty} \limsup_{n\to\infty} \frac{\log_2 |\mathcal{C}_n|}{n} = C_{\mathrm{ICI}}(2).$$

Unfortunately, for the encoder/decoder pair to work, an auxiliary codebook $\mathcal{C}_a$ of length $n_a < n$ is required. Here, $|\mathcal{C}_a|$ is exponential in n_a. In order to approach capacity, both m and n_a are required to be sufficiently large. Since the encoding and decoding complexity grows in terms of m and $|\mathcal{C}_a|$, we have that the encoding and decoding complexity is exponential in term of n_a (see [14, Remark 1] for more details).

For $q > 2$, no such concrete results are even known.

Indeed, the problem of constructing efficiently encodable and decodable constant-composition ICI-free codes that achieves the ICI channel capacity is wide open. Even the question of whether there exist constant-composition ICI-free codes achieving ICI channel capacity is not answered till recently by Chee et al. [15].

Theorem 1 (Chee et al. [15])**.**

$$A_{\mathrm{ICI}}(n, [w_0, w_1, \ldots, w_{q-1}]) = \sum_{i=0}^{w_{q-1}-1} \binom{w_{q-1}-1}{i} \binom{n - w_{q-1} - i + 1}{n - w_{q-1} - 2i} \frac{(n - w_{q-1})!}{\prod_{i=0}^{q-2} w_i!}.$$

Using Theorem 1, Chee et al. [15] proved that for every q,

$$\lim_{n\to\infty} \max_{\overline{w}(n) \text{a } q\text{-part composition of } n} \frac{\log_2 A_{\rm ICI}(n,\overline{w}(n))}{n} = C_{\rm ICI}(q).$$

C. Our Contribution

The main contributions of this paper are efficient encoding and decoding algorithms for binary constant-weight ICI-free codes and a special class of q-ary constant-composition ICI-free codes. Paired with the results of Chee et al. [15], this gives the first efficient encoding and decoding of constant-composition ICI-free codes achieving ICI channel capacity.

II. A Recursive Construction for (Binary) $\mathcal{S}(n,w)$

Let $n \geq w \geq 2$ and define the map

$$\phi : \bigcup_{k\in[\![n-w+1]\!]\setminus\{2\}} \mathcal{S}(n-k,w-1) \to \mathcal{S}(n,w),$$

such that

$$\phi : (\mathsf{u}_1,\mathsf{u}_2,\ldots,\mathsf{u}_{n-k}) \mapsto$$
$$(\mathsf{u}_1,\mathsf{u}_2,\ldots,\mathsf{u}_{r(\mathsf{u})},\underbrace{0,0,\ldots,0}_{k-1 \text{ 0's}},1,\mathsf{u}_{r(\mathsf{u})+1},\mathsf{u}_{r(\mathsf{u})+2},\ldots,\mathsf{u}_{n-k}),$$

where $r(\mathsf{u})$ is the position of the rightmost "1" in $\mathsf{u} = (\mathsf{u}_1,\mathsf{u}_2,\ldots,\mathsf{u}_{n-k})$, that is, $r(\mathsf{u}) = \max\{i \in [\![n-k]\!] : \mathsf{u}_i = 1 \text{ and } \mathsf{u}_j = 0 \text{ for all } j \geq i\}$.

Theorem 2. *The map ϕ is a bijection.*

Proof. We first show injectivity of ϕ. If u and v are distinct elements of $\mathcal{S}(n-k,w-1)$ for some $k \in [\![n-w+1]\!]\setminus\{2\}$, then

- when $r(\mathsf{u}) = r(\mathsf{v})$, $\phi(\mathsf{u})$ and $\phi(\mathsf{v})$ must differ in some of their first $r(\mathsf{u})$ positions;
- when $r(\mathsf{u}) \neq r(\mathsf{v})$, $\phi(\mathsf{u})$ and $\phi(\mathsf{v})$ must differ in some of their last $n-k-r(\mathsf{u})$ positions.

If $\mathsf{u} \in \mathcal{S}(n-k_1,w-1)$ and $\mathsf{v} \in \mathcal{S}(n-k_2,w-1)$ for some $k_1,k_2 \in [\![n-w+1]\!]\setminus\{2\}$, where $k_1 \neq k_2$, then the rightmost occurrence of a substring of the form $(1,0,0,\ldots,0,1)$ in $\phi(\mathsf{u})$ and $\phi(\mathsf{v})$ has lengths k_1+1 and k_2+2, respectively.

To prove surjectivity of ϕ, consider $\mathsf{u} \in \mathcal{S}(n,w)$. Let $s(\mathsf{u})$ be the starting position of the rightmost substring in u of the form $(1,\underbrace{0,0,\ldots,0}_{k-1 \text{ 0's}},1)$, where $k \geq 1$. Deleting the substring $(\mathsf{u}_{s(\mathsf{u})+1},\mathsf{u}_{s(\mathsf{u})+2},\ldots,\mathsf{u}_{s(\mathsf{u})+k})$ from u gives an element $\mathsf{v} \in \mathcal{S}(n-k,w-1)$ such that $\phi(\mathsf{v}) = \mathsf{u}$. ∎

Corollary 1.

$$A_{\rm ICI}(n,w) = \sum_{k\in[\![n-w+1]\!]\setminus\{2\}} A_{\rm ICI}(n-k,w-1).$$

Theorem 2 and Corollary 1, together with Proposition 3 below, give recurrence for the construction of $\mathcal{S}(n,w)$ and the determination of $A_{\rm ICI}(n,w)$.

Proposition 3. *For $n \geq 1$,*

(i) $\mathcal{S}(n,1)$ is the set of all words of weight one in Σ^n, and hence $A_{\rm ICI}(n,1) = n$;

(ii) $\mathcal{S}(n,n) = \{(1,1,\ldots,1)\}$, and hence $A_{\rm ICI}(n,n) = 1$.

Example 2. We can construct $\mathcal{S}(5,3)$ from $\mathcal{S}(4,2)$ and $\mathcal{S}(2,2)$. $\mathcal{S}(4,2)$ can in turn be constructed from $\mathcal{S}(3,1)$ and $\mathcal{S}(1,1)$. Hence, with the trivial codes $\mathcal{S}(3,1) = \{100,010,001\}$, $\mathcal{S}(1,1) = \{1\}$, and $\mathcal{S}(2,2) = \{11\}$, we obtain

$$\begin{aligned}\mathcal{S}(4,2) &= \phi(\mathcal{S}(3,1)) \cup \phi(\mathcal{S}(1,1)) \\ &= \{1100,0110,0011\} \cup \{1001\},\end{aligned}$$

which in turn gives

$$\begin{aligned}\mathcal{S}(5,3) &= \phi(\mathcal{S}(4,2)) \cup \phi(\mathcal{S}(2,2)) \\ &= \{11100,01110,00111,10011\} \cup \{11001\}.\end{aligned}$$

Similarly,

$$\begin{aligned}A_{\rm ICI}(5,3) &= A_{\rm ICI}(4,2) + A_{\rm ICI}(2,2) \\ &= A_{\rm ICI}(3,1) + A_{\rm ICI}(1,1) + A_{\rm ICI}(2,2) \\ &= 5.\end{aligned}$$

In fact, Corollary 1 gives rise to a polynomial time algorithm, via dynamic programming, for computing the value of $A_{\rm ICI}(n,w)$, for any given n and w. Let A be the $n \times w$ matrix whose (i,j)-th entry, $\mathsf{A}(i,j) = A_{\rm ICI}(i,j)$. Prefill the first column so that $\mathsf{A}(i,1) = i$ for all $i \in [\![n]\!]$, and the "diagonal" entries so that $\mathsf{A}(i,i) = 1$ for all $i \in [\![w]\!]$. Now fill the remaining entries $\mathsf{A}(i,j)$, where $i > j$, column wise from left to right (that is, by increasing value of j), and within each column j from top to bottom (that is, by increasing value of i), until we fill in the entry $\mathsf{A}(n,w)$, which gives the value of $A_{\rm ICI}(n,w)$.

The building up of codewords in $\mathcal{S}(n,w)$ from shorter codewords in $\mathcal{S}(n-k,w-1)$ via ϕ leads also to an efficient ranking/unranking algorithm for codewords in $\mathcal{S}(n,w)$. We describe this next.

III. Ranking and Unranking $\mathcal{S}(n,w)$

A *ranking function* for a finite set S of cardinality N is a bijection

$$\text{rank} : S \to [\![N]\!].$$

There is a unique *unranking function* associated with the function rank:

$$\text{unrank} : [\![N]\!] \to S,$$

so that $\text{rank}(s) = i$ if and only if $\text{unrank}(i) = s$ for all $s \in S$ and $i \in [\![N]\!]$. In this section, we present an algorithm for ranking and unranking $\mathcal{S}(n,w)$.

The basis of our ranking and unranking algorithms is the unfolding of the recurrence

$$\mathcal{S}(n,w) = \bigcup_{k\in[\![n-w+1]\!]\setminus\{2\}} \phi(\mathcal{S}(n-k,w-1)) \qquad (1)$$

implied by Theorem 2, which yields a natural total ordering of codewords in $\mathcal{S}(n,w)$, given a total ordering of codewords in $\mathcal{S}(n,1)$ and $\mathcal{S}(n,n)$. Throughout this paper, the reverse lexicographic order is used as a total ordering on $\mathcal{S}(n,1)$, so that the rank of $\mathsf{u} \in \mathcal{S}(n,1)$ is the position of the symbol "1" in u. Note that the total ordering on $\mathcal{S}(n,n)$ is trivial since it contains only one element. Let us first illustrate the idea behind the unranking algorithm through an example.

Example 3. Consider $\mathcal{S}(7,3)$. This code has size 18. Suppose we want to compute $\text{unrank}(13)$. First, (1) gives

$$\mathcal{S}(7,3) = \phi(\mathcal{S}(6,2)) \cup \phi(\mathcal{S}(4,2)) \cup \phi(\mathcal{S}(3,2)) \cup \phi(\mathcal{S}(2,2)),$$

where the codes in the union on the right hand side are ordered in decreasing length. Now, $|\mathcal{S}(6,2)| = 11$, $|\mathcal{S}(4,2)| = 4$, $|\mathcal{S}(3,2)| = 2$, and $|\mathcal{S}(2,2)| = 1$. We are interested in the 13th element of $\mathcal{S}(7,3)$. Since $|\mathcal{S}(6,2)| < 13 \le |\mathcal{S}(6,2)|+|\mathcal{S}(4,2)|$, the 13th element of $\mathcal{S}(7,3)$ is the $13-|\mathcal{S}(6,2)| = 2$-nd element of $\phi(\mathcal{S}(4,2))$, which can be obtained from the 2nd element of $\mathcal{S}(4,2)$. Recursing gives

$$\mathcal{S}(4,2) = \phi(\mathcal{S}(3,1)) \cup \phi(\mathcal{S}(1,1)),$$

where $|\mathcal{S}(3,1)| = 3$ and $|\mathcal{S}(1,1)| = 1$.

Hence the 2nd element of $\mathcal{S}(4,2)$ is the 2nd element of $\phi(\mathcal{S}(3,1))$, which can be obtained from the 2nd element of $\mathcal{S}(3,1)$, namely 010. This gives

$$\begin{aligned}\mathrm{unrank}(13) &= \phi^2(010)\\ &= \phi(0110)\\ &= 0110010.\end{aligned}$$

The formal unranking algorithm is described in Algorithm 1 below.

Algorithm 1 $\texttt{unrank}(n, w, m)$

```
Input: Integers $n \ge w \ge 1$ and $1 \le m \le A_{\mathrm{ICI}}(n,w)$.
Output: $(\mathsf{u}, r)$, where $\mathsf{u}$ is the codeword of rank $m$ in $\mathcal{S}(n,w)$, and $r$ is the position
  of the rightmost “1” in $\mathsf{u}$.
  if $w = n$ then
      return $(\mathsf{j}, n)$;
  if $w = 1$ then
      return $(\mathsf{e}_m, m)$;
  let $k \ge 1$ be such that
      $L = \sum_{i\in[\![k-1]\!]\setminus\{2\}} A_{\mathrm{ICI}}(n-i, w-1) < m \le \sum_{i\in[\![k]\!]\setminus\{2\}} A_{\mathrm{ICI}}(n-i, w-1)$;
  $(\mathsf{u}, r) = \texttt{unrank}(n-k, w-1, m-L)$;
  return $((\mathsf{u}_1, \mathsf{u}_2, \ldots, \mathsf{u}_r, \underbrace{0, 0, \ldots, 0}_{k\,-\,1\text{ 0's}}, 1, \mathsf{u}_{r+1}, \mathsf{u}_{r+2}, \ldots, \mathsf{u}_{n-k}), r+k)$;
```

The values of $A_{\mathrm{ICI}}(n,w)$ required in Algorithm 1 can be pre-computed using the dynamic programming method described at the end of the previous section.

The corresponding ranking algorithm for $\mathcal{S}(n,w)$ has a similar recursive structure and is described in Algorithm 2.

Algorithm 2 $\texttt{rank}(n, w, \mathsf{u})$

```
Input: Integers $n \ge w \ge 1$ and $\mathsf{u} \in \mathcal{S}(n,w)$.
Output: $m$, where $m = \mathrm{rank}(\mathsf{u})$.
  if $w = n$ then
      return 1;
  if $w = 1$ then
      return $m$, where $m$ is the position of the rightmost “1” in $\mathsf{u}$;
  let $r$ be the starting position of the rightmost substring in $\mathsf{u}$ of the form
  $(1, \underbrace{0, 0, \ldots, 0}_{k\,-\,1\text{ 0's}}, 1)$, where $k \ge 1$;
  $\mathsf{v} \leftarrow (\mathsf{u}_1, \mathsf{u}_2, \ldots, \mathsf{u}_r, \mathsf{u}_{r+k+1}, \mathsf{u}_{r+k+2}, \ldots, \mathsf{u}_n)$;
  return $\texttt{rank}(n-k, w-1, \mathsf{v}) + \sum_{i\in[\![k-1]\!]\setminus\{2\}} A_{\mathrm{ICI}}(n-i, w-1)$;
```

Example 4. Consider $\mathcal{S}(7,3)$ again. Suppose we want to compute $\mathrm{rank}(7,3,0110010)$. First, we look for the rightmost “1” in 0110010 and set $k-1$ to be the number of zeroes preceding it. In other words, $k = 3$ and so,

$$\begin{aligned}\mathrm{rank}(7,3,0110010) &= \mathrm{rank}(4,2,0110) + A_{\mathrm{ICI}}(6,2)\\ &= \mathrm{rank}(4,2,0110) + 11.\end{aligned}$$

To compute $\mathrm{rank}(4,2,0110)$, we observe that $k = 1$ and we have

$$\mathrm{rank}(4,2,0110) = \mathrm{rank}(3,1,010).$$

Finally, since the weight of 010 is one, we have that $\mathrm{rank}(3,1,010) = 2$. Therefore, $\mathrm{rank}(7,3,0110010) = 2 + 11 = 13$, and we recover the rank of 0110010 given in Example 3.

Algorithms 1 and 2 run in polynomial time. We focus on explaining the ideas behind the design of our ranking/unranking algorithms and defer the optimization and detailed running time analysis of these algorithms to the full paper. We may treat this pair of unranking/ranking algorithms as an encoder/decoder pair for constant-weight ICI-free codes. Combined with results of Chee et al. [15], this gives the first known efficient encoder/decoder for constant-weight ICI-free codes achieving ICI channel capacity.

IV. Extension to $q > 2$

Let $\mathcal{C} \subseteq \Sigma^n$ and $\Omega \subseteq \Sigma$. Let $f : \Sigma \to \Omega$. By canonical extension, we have $f : \Sigma^n \to \Omega^n$, so that

$$\Sigma^n \ni (\mathsf{u}_1, \mathsf{u}_2, \ldots, \mathsf{u}_n) \overset{f}{\mapsto} (f(\mathsf{u}_1), f(\mathsf{u}_2), \ldots, f(\mathsf{u}_n)) \in \Omega^n.$$

The *restriction* of $\mathcal{C}$ by f is the code $f(\mathcal{C}) \subseteq \Omega^n$.

The idea behind the extension of our results in the previous section for binary codes to q-ary codes is based on the simple observation that if a q-ary code is ICI-free, then its restriction by $f : \Sigma \to \{0,1\}$, where

$$f(\sigma) = \begin{cases}1, & \text{if } \sigma = q-1\\ 0, & \text{otherwise,}\end{cases}$$

is a binary ICI-free code. Hence, a binary ICI-free code $\mathcal{C} \subseteq \{0,1\}^n$ can be used as a template to construct a q-ary ICI-free code $\mathcal{C}' \subseteq \Sigma^n$: for each codeword $\mathsf{u} \in \mathcal{C}$, replace a coordinate with symbol “1” by $q-1$ and replace a coordinate with symbol “0” by all possible symbols from $\Sigma \setminus \{q-1\}$. Therefore, a binary codeword of weight w in $\mathcal{C}$ generates $(q-1)^{n-w}$ codewords in $\mathcal{C}'$.

We are concerned here with q-ary ICI-free codes of constant composition $[w_0, w_1, \ldots, w_{q-1}]$, where $w_0 = w_1 = \cdots = w_{q-2}$. We call codes of such composition *almost balanced*. The intuition behind this condition is that an ICI-free code avoids substrings of the form $q-1, \sigma, q-1$, for all $\sigma \in \Sigma \setminus \{q-1\}$, and so the symbol $q-1$ has a special status. Therefore, if we were to look for a constant-composition ICI-free code of maximum size, it would be a good strategy to look within almost balanced codes. Indeed, Chee et al. [15] has found ICI channel capacity-achieving ICI-free codes that are almost balanced.

To construct an almost balanced ICI-free code $\mathcal{C} \subseteq \Sigma^n$ of constant composition $[w_0, w_1, \ldots, w_{q-1}]$, we can start with $\mathcal{S}(n, w_{q-1})$ as a template and replace every occurrence of symbol “1” in each codeword $\mathsf{u} \in \mathcal{S}(n, w_{q-1})$ by $q-1$. However, instead of replacing the remaining $n - w_{q-1}$ “0”s in u with all possible words in $(\Sigma \setminus \{q-1\})^{n-w_{q-1}}$, we replace them with codewords from a balanced $(q-1)$-ary code of length $n - w_{q-1}$ over $\Sigma \setminus \{q-1\}$.

Efficient encoder/decoder pairs for capacity-achieving balanced q-ary codes are known [16], [17]. We can combine the encoder/decoder for $\mathcal{S}(n, w)$ and that for a capacity-achieving balanced $(q-1)$-ary code $\mathcal{B}$ of length $n-w$ to give an efficient encoder/decoder for an almost balanced q-ary ICI-free code $\mathcal{C}$. The encoding algorithm is described in Algorithm 3.

Algorithm 3 $\texttt{encode}(m)$

```
Input: $0 \le m < |\mathcal{S}(n,w)| \cdot |\mathcal{B}|$.
Output: $\mathsf{u}$, where $\mathsf{u}$ is an encoding of $m$ as a codeword in $\mathcal{C}$.
    let $m = s \cdot |\mathcal{B}| + t$, where $0 \le t < |\mathcal{B}|$;
    $\mathsf{u} \leftarrow$ encoding of $s$ as a codeword in $\mathcal{S}(n,w)$;
    $\mathsf{v} \leftarrow$ encoding of $t$ as a codeword in $\mathcal{B}$;
    $\mathsf{w}$ = word obtained by replacing each occurrence of symbol "1" in $\mathsf{u}$ by $q-1$ and all the other $n-w$ "0"s in $\mathsf{u}$ by the word $\mathsf{v}$;
    return $\mathsf{w}$;
```

The corresponding decoding algorithm is given in Algorithm 4.

Algorithm 4 $\texttt{decode}(\mathsf{u})$

```
Input: $\mathsf{u} \in \mathcal{C}$.
Output: $m$, where $\mathsf{u} = \texttt{encode}(m)$.
    $\mathsf{v} \leftarrow$ word obtained from $\mathsf{u}$ by deleting occurrences of symbol $q-1$;
    $t \leftarrow$ decoding of $\mathsf{v} \in \mathcal{B}$;
    $\mathsf{w} \leftarrow$ word obtained from $\mathsf{u}$ by replacing each occurrence of symbol $q-1$ in $\mathsf{u}$ by "1" and all the other symbols by "0";
    $s \leftarrow$ decoding of $\mathsf{w} \in \mathcal{S}(n,w)$;
    return $s \cdot |\mathcal{B}| + t$;
```

A. Application to the Case $q = 4$

Using Perron-Frobenius theory, the capacity of q-ary ICI channels can be determined to be $\log_2 \lambda$, where λ is the largest root of $x^3 - qx^2 + (q-1)x - (q-1)^2$ (see, for example, [18], [19]). This gives $C_{\rm ICI}(4) \approx 1.9374$. Taranalli et al. [8] gave an encoding/decoding algorithm for quaternary ICI-free codes that has rate 1.6942.

Chee et al. [15] constructed almost balanced quaternary ICI-free codes of composition $[\alpha n, \alpha n, \alpha n, \beta n]$, where $\alpha \approx 0.268582$ and $\beta \approx 0.194254$, and showed them to be capacity-achieving (having rate 1.9374). These codes can be encoded and decoded with the algorithms described earlier in this section. Hence, we now have efficient encoding/decoding algorithms for quaternary constant-composition ICI-free codes that are capacity-achieving.

V. Conclusion

We have proposed efficient encoder/decoder for q-ary constant-composition ICI-free codes that are capacity-achieving. The structure of the encoders and decoders are simple and yields to easy implementation.

Acknowledgement

The authors thank the anonymous reviewers and the TPC member for their constructive comments.

References

[1] J.-D. Lee, S.-H. Hur, and J.-D. Choi, "Effects of floating-gate interference on NAND flash memory cell operation," *IEEE Electron. Device Lett.*, vol. 23, no. 5, pp. 264–266, 2002.

[2] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error patterns in MLC NAND flash memory: measurement, characterization, and analysis," in *DATE 2012 – Proceedings of the Conference on Design, Automation and Test in Europe*, 2012, pp. 521–526.

[3] D. Kang, H. Shin, S. Chang, and J. An, "The air spacer technology for improving the cell distribution in 1 giga bit NAND flash memory," in *NVSMW 2006 – Proceedings of the 21st IEEE Non-Volatile Semiconductor Memory Workshop*, 2006, pp. 36–37.

[4] R. Fastow and S. Park, "Minimization of FG-FG coupling in flash memory," US Patent 6996004 B1, February 2006.

[5] A. Berman and Y. Birk, "Mitigating inter-cell coupling effects in MLC NAND flash via constrained coding," in *Proceedings of the 2010 Flash Memory Summit*, 2010.

[6] ——, "Error correction scheme for constrained inter-cell interference in flash memory," in *NVMW 2011 – 2nd Annual Non-Volatile Memories Workshop*, 2011.

[7] ——, "Constrained flash memory programming," in *ISIT 2011 – Proceedings of the 2011 IEEE International Symposium on Information Theory*, 2011, pp. 2128–2132.

[8] V. Taranalli, H. Uchikawa, and P. H. Siegel, "Error analysis and inter-cell interference mitigation in multi-level cell flash memories," in *ICC 2015 – Proceedings of the IEEE International Conference on Communications*, 2015, pp. 271–276.

[9] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck, "Codes for asymmetric limited-magnitude errors with application to multi-level flash memories," *IEEE Trans. Inform. Theory*, vol. 56, no. 4, pp. 1582–1596, 2010.

[10] E. Yaakobi, P. H. Siegel, A. Vardy, and J. K. Wolf, "On codes that correct asymmetric errors with graded magnitude distribution," in *ISIT 2011 – Proceedings of the 2011 IEEE International Symposium on Information Theory*, 2011, pp. 1056–1060.

[11] H. Zhou, A. Jiang, and J. Bruck, "Error-correcting schemes with dynamic thresholds in nonvolatile memories," in *ISIT 2011 – Proceedings of the 2011 IEEE International Symposium on Information Theory*, 2011, pp. 2109–2113.

[12] F. Sala, R. Gabrys, and L. Dolecek, "Dynamic threshold schemes for multi-level non-volatile memories," *IEEE Trans. Commun.*, vol. 61, no. 7, pp. 2624–2634, 2013.

[13] M. Qin, E. Yaakobi, and P. H. Siegel, "Constrained codes that mitigate inter-cell interference in read/write cycles for flash memories," *IEEE J. Select. Areas Commun.*, vol. 32, no. 5, pp. 836–846, 2014.

[14] S. Kayser and P. H. Siegel, "Constructions for constant-weight ICI-free codes," in *ISIT 2014 – Proceedings of the 2014 IEEE International Symposium on Information Theory*, 2014, pp. 1431–1435.

[15] Y. M. Chee, C. Johan, H. M. Kiah, S. Ling, T. T. Nguyen, and V. K. Vu, "Rates of Constant-Composition Codes that Mitigate Intercell Interference," 2016, preprint.

[16] T. G. Swart and J. H. Weber, "Efficient balancing of q-ary sequences with parallel decoding," in *ISIT 2009 – Proceedings of the 2009 IEEE International Symposium on Information Theory*, 2009, pp. 1564–1568.

[17] K. A. S. Immink, "Prefixless q-ary balanced codes," in *AEW8 – Proceedings of the Eighth Asian-European Workshop on Information Theory*, 2013.

[18] ——, *Codes for Mass Data Storage Systems*, 2nd ed. Eindhoven, The Netherlands: Shannon Foundation Publishers, 1999.

[19] B. Marcus, R. M. Roth, and P. H. Siegel, *An Introduction to Coding for Constrained Systems*, 5th ed., 2001.