Patterned Cells for Phase Change Memories

Anxiao (Andrew) Jiang
Computer Sci. and Eng.
Texas A&M University
College Station, TX 77843
ajiang@cse.tamu.edu

Hongchao Zhou
Electrical Engineering
Caltech
Pasadena, CA 91125
hzhou@caltech.edu

Zhiying Wang
Electrical Engineering
Caltech
Pasadena, CA 91125
zhiying@caltech.edu

Jehoshua Bruck
Electrical Engineering
Caltech
Pasadena, CA 91125
bruck@caltech.edu

***Abstract*—Phase-change memory (PCM) is an emerging nonvolatile memory technology that promises very high performance. It currently uses discrete cell levels to represent data, controlled by a single amorphous/crystalline domain in a cell. To improve data density, more levels per cell are needed. There exist a number of challenges, including cell programming noise, drifting of cell levels, and the high power requirement for cell programming.**

In this paper, we present a new cell structure called *patterned cell*, and explore its data representation schemes. Multiple domains per cell are used, and their connectivity is used to store data. We analyze its storage capacity, and study its error-correction capability and the construction of error-control codes.

I. Introduction

Phase-change memory (PCM) is an important emerging nonvolatile memory (NVM) technology that promises high performance. It uses chalcogenide glass as cells, which has two stable states: amorphous and crystalline [2]. The amorphous state has very high electrical resistance, and the crystalline state has low resistance. Intermediate states, called partially crystalline states, can also exist. High temperatures induced by electrical currents are used to switch the state of a portion of the cell, which is called a *domain*. By quantizing cell resistance into multiple discrete levels, one or more bits per cell can be stored. Currently, four-level cells have been developed. To improve data density, more levels are needed [2].

The current multi-level cell (MLC) approach faces a number of challenges, including cell-programming noise, cell-level drifting, and high power consumption [2], [3]. It is difficult to program cell levels accurately due to cell heterogeneity and noise. The cell levels can drift away significantly after they are programmed, making it even harder to control their accuracy. And the high power requirement for cell programming is hindering PCM's application in mobile devices [3].

In this paper, we explore a new cell structure and its data representation scheme. In the new structure, called *patterned cells*, multiple domains per cell are used. An example is shown in Fig. 1, where two or four domains exist in a cell, whose states are independently controlled by their respective bottom electrodes. (The state of a domain is switched by the current between the bottom and top electrodes. We assume that the PCM layer is sufficiently thin such that changing a domain to the crystalline state, which is called the SET operation and requires a lower temperature/current, will not affect its neighboring domains.) The base of a cell is in the amorphous state, while every domain can be switched to the crystalline state. (To change domains back to amorphous, called the RESET operation, we can RESET them together to avoid interference.) We call this model the *crystalline-domain model*, because the domains have a different state from the cell base when they are crystalline. The *amorphous-domain model*, where the cell base is crystalline and the domains can be amorphous, can also be defined. Due to the space limitation, we omit its details, and focus on the crystalline-domain model.

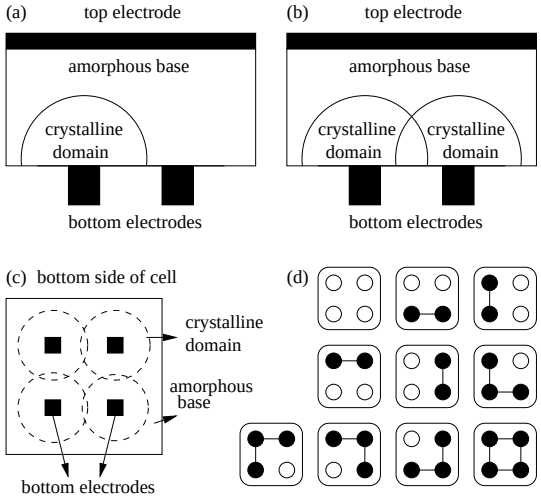


Fig. 1. Patterned cell with the crystalline-domain model. (a) A PCM cell with two bottom electrodes and one crystalline domain. The two bottom electrodes are not connected (i.e., there is high resistance between them). (b) The two bottom electrodes are connected by two overlapping crystalline domains. (c) The bottom-side view of a cell with $n = 4$ potential crystalline domains. (d) The 10 different connectivity patterns for the 2×2 rectangular array of domains shown in (c). The black vertices are crystalline domains (called "on" vertices); the white vertices are not crystalline (called "off" vertices). The edges between vertices denote their connectivity.

We let every domain have two basic states: *on* (crystalline) or *off* (amorphous). If two neighboring domains are both on, they overlap and become electrically connected (i.e., low resistance). The connectivity of domains can be detected by measuring the resistance between their bottom electrodes, which uses low reading voltage and does not change the state of the domains. We use the connectivity patterns of domains to represent data. As an example, the connectivity patterns of the four domains in Fig. 1 (c) are illustrated in Fig. 1 (d).

Pattern cell is a new approach to store data using the internal structure of domains in PCM cells. The two basic states of its domains may eliminate the high precision and power requirements imposed by programming cell levels. The data

representation scheme is a new type of code defined based on graph connectivity. In this paper, we explore this new scheme, analyze its storage capacity, and study its error-correction capability and the construction of error-control codes.

The rest of the paper is organized as follows. In Section II, we study the storage capacity of patterned cell. In Section III, we study error correction and detection for patterned cell. In Section IV, we present concluding remarks.

II. Storage Capacity of Patterned Cell

In this section, we present the graph model for connectivity-based data representation. Then we analyze the storage capacity of domains that form one or two dimensional arrays.

A. Graph Model for Connectivity-based Data Representation

Let $G = (V, E)$ be a connected undirected graph, whose vertices V represent the domains in a cell. An edge (u, v) exists if the two domains are adjacent (which means they overlap if they are both *on*). Let $S : V \to \{0, 1\}$ denote the states of vertices: $\forall\ v \in V$, $S(v) = 1$ if v is *on*, and $S(v) = 0$ if v is *off*. Denote the $|V|$ vertices by $v_1, v_2, \cdots, v_{|V|}$. We call $\big(S(v_1), S(v_2), \cdots, S(v_{|V|})\big)$ a *configuration* of G. Let $\tilde{U} = \{0, 1\}^{|V|}$ denote the set of all configurations. Since in the crystalline-domain model, the purpose of making a domain crystalline is to connect it to at least one crystalline neighbor, we focus on configurations denoted by $\mathcal{U}$ that satisfy this property: "For any $v \in V$ that is on, at least one of its neighbors is also on." That is, $\mathcal{U} = \{\big(S(v_1), S(v_2), \cdots, S(v_{|V|})\big) \in \tilde{U} |\ \forall 1 \le i \le |V|\,,$ if $S(v_i) = 1,$ then $\exists v_j \in V$ such that $(v_i, v_j) \in E$ and $S(v_j) = 1\}$. We call $\mathcal{U}$ the set of *valid* configurations.

Let $C : V \times V \to \{0, 1\}$ denote the connectivity between vertices: "$\forall\ w_1 \neq w_2 \in V$, $C(w_1, w_2) = 1$ if there exists a sequence of vertices $(w_1 = u_1, u_2, \cdots, u_k = w_2)$ such that $(u_i, u_{i+1}) \in E$ and $S(u_i) = S(u_{i+1}) = 1$ for $i = 1, 2, \cdots, k-1$; otherwise, $C(w_1, w_1) = 0$. And for any $w \in V$, we set $C(w, w) = 1$ by default." Two vertices w_1, w_2 are *connected* if $C(w_1, w_2) = 1$. The vector $(C(v_1, v_1), C(v_1, v_2), \cdots, C(v_1, v_{|V|}); C(v_2, v_1), C(v_2, v_2), \cdots, C(v_2, v_{|V|}); \cdots\cdots; C(v_{|V|}, v_1), C(v_{|V|}, v_2), \cdots, C(v_{|V|}, v_{|V|}))$ is called the *connectivity pattern* of G. Clearly, not all vectors in $\{0, 1\}^{|V|\times|V|}$ are connectivity patterns that correspond to valid configurations (or even just configurations). So to be specific, let $f : \mathcal{U} \to \{0, 1\}^{|V|\times|V|}$ be the function that maps a valid configuration to its connectivity pattern. Let $\mathcal{C} = \{f(\vec{u}) \mid \vec{u} \in \mathcal{U}\}$, and we call $\mathcal{C}$ the set of *valid connectivity patterns*.

Lemma 1. *The mapping $f : \mathcal{U} \to \mathcal{C}$ is a bijection.*

Proof: Given a connectivity pattern $\vec{c} \in \mathcal{C}$, we see that a vertex $v \in V$ is *on* if and only if it is connected to at least one neighbor. So the configuration is determined by $\vec{c}$. ■

A PCM can read the connectivity pattern. We store data by mapping elements in $\mathcal{C}$ to symbols. The rate of graph G is $\frac{\log_2|\mathcal{C}|}{|V|} = \frac{\log_2|\mathcal{U}|}{|V|}$ bits per vertex (i.e., domain).

B. Capacity of One-dimensional Array

It is not difficult to compute the rate of G when $|V|$ is small. In this paper, we focus on large $|V|$ (especially for $|V| \to \infty$), which corresponds to using numerous domains in a large PCM layer. Let $n = |V|$, and define $N(n) \triangleq |\mathcal{C}| = |\mathcal{U}|$. We define the *capacity* of G as $cap = \lim_{n\to\infty} \frac{log_2 N(n)}{n}$.

We first consider the case where the domains form a one-dimensional array. That is, in graph $G = (V, E)$, we have $V = \{v_1, v_2, \cdots, v_n\}$ and $E = \{(v_1, v_2), (v_2, v_3), \cdots, (v_{n-1}, v_n)\}$. We denote the *capacity* of the one-dimensional array by cap_{1D}.

Theorem 2. *Let* $\lambda^* = \frac{1}{6}\left(100 + 12\times\sqrt{69}\right)^{1/3} + \frac{2}{3\left(100+12\times\sqrt{69}\right)^{1/3}} + \frac{2}{3} \approx 1.7549$. *We have*

$$cap_{1D} = \log_2 \lambda^* \approx 0.8114.$$

Proof: The valid configuration of a one-dimensional array is a constrained system, where every run of 1s (i.e., "on" vertices) needs to have length at least two. The Shannon cover of the system is shown in Fig. 2. Its adjacency matrix is $A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$. By solving $|A - \lambda I| = -(\lambda^3 - 2\lambda^2 + \lambda - 1) = 0$, we find that for matrix A, its eigenvalue of the greatest absolute value is $\lambda^* \approx 1.7549$. It is known that the capacity of the constrained system is $\log_2 \lambda^*$. ■

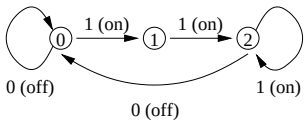


Fig. 2. Shannon cover for one-dimensional array.

We further present the number of valid configurations for a one-dimensional array with n vertices.

Theorem 3. *Let $\alpha_1, \alpha_2, \alpha_3$ be the three solutions to x for the equation $x^3 - 2x^2 + x - 1 = 0$, and let μ_1, μ_2, μ_3 be the numbers that satisfy the linear equation set*

$$\begin{cases} \mu_1\alpha_1 + \mu_2\alpha_2 + \mu_3\alpha_3 = 1 \\ \mu_1\alpha_1^2 + \mu_2\alpha_2^2 + \mu_3\alpha_3^2 = 2 \\ \mu_1\alpha_1^3 + \mu_2\alpha_2^3 + \mu_3\alpha_3^3 = 4 \end{cases}$$

(We get $\alpha_1 = \frac{1}{6}\cdot(100 + 12\sqrt{69})^{\frac{1}{3}} + \frac{2}{3}\cdot(100 + 12\sqrt{69})^{-\frac{1}{3}} + \frac{2}{3} \approx 1.7549$, $\alpha_2 = -\frac{1}{12}\cdot(100 + 12\sqrt{69})^{\frac{1}{3}} - \frac{1}{3}\cdot(100 + 12\sqrt{69})^{-\frac{1}{3}} + \frac{2}{3} + i\cdot(\frac{\sqrt{3}}{12}\cdot(100 + 12\sqrt{69})^{\frac{1}{3}} - \frac{\sqrt{3}}{3}\cdot(100 + 12\sqrt{69})^{-\frac{1}{3}}) \approx 0.1226 + 0.7449i$, $\alpha_3 = -\frac{1}{12}\cdot(100 + 12\sqrt{69})^{\frac{1}{3}} - \frac{1}{3}\cdot(100 + 12\sqrt{69})^{-\frac{1}{3}} + \frac{2}{3} - i\cdot(\frac{\sqrt{3}}{12}\cdot(100 + 12\sqrt{69})^{\frac{1}{3}} - \frac{\sqrt{3}}{3}\cdot(100 + 12\sqrt{69})^{-\frac{1}{3}}) \approx 0.1226 - 0.7449i$, $\mu_1 \approx 0.7221$, $\mu_2 \approx 0.1389 + 0.2023i$, *and* $\mu_3 \approx 0.1389 - 0.2023i$.*) Then for a one-dimensional array with n vertices, we have* $N(n) = |\mathcal{C}| = |\mathcal{U}| = \mu_1\alpha_1^n + \mu_2\alpha_2^n + \mu_3\alpha_3^n$.

Proof: We derive the value of $N(n)$ by recursive functions. Define $g(n)$ to be the set of valid configurations for a linear array with n vertices given that the first vertex is "on". That is, $g(n) = \{(s_1, s_2, \cdots, s_n) \in \mathcal{U} \mid s_1 = 1\}$.

To compute $g(n)$, we notice that for a valid configuration $(s_1, s_2, \cdots, s_n) \in \mathcal{U}$, if $s_1 = 1$, then $s_2 = 1$ and we also have the following properties:

- If $s_3 = 0$, the states of the last $n-3$ vertices can be any configuration for a one-dimensional array with $n-3$ vertices. There are $N(n-3)$ such configurations.
- If $s_3 = 1$, the states of the last $n-1$ vertices can be any configuration in $g(n-1)$. There are $|g(n-1)|$ such configurations.

So we get $|g(n)| = N(n-3) + |g(n-1)|$.

To compute $N(n)$, we notice that for a valid configuration $(s_1, s_2, \cdots, s_n) \in \mathcal{U}$:

- If $s_1 = 0$, the states of the last $n-1$ vertices can be any configuration for a one-dimensional array with $n-1$ vertices. There are $N(n-1)$ such configurations.
- If $s_1 = 1$, the states of the n vertices can be any configuration in $g(n)$. There are $|g(n)|$ such configurations.

So we get $N(n) = N(n-1) + |g(n)|$.

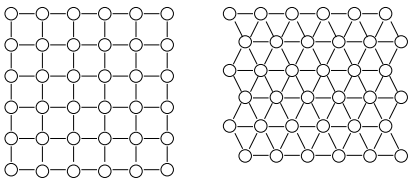
Combing the above two equations, we get the recursive function

$$N(n) = 2N(n-1) - N(n-2) + N(n-3).$$

By solving the recursive function and using the boundary conditions that $N(1) = 1$, $N(2) = 2$, $N(3) = 4$, we get the conclusion. ■

C. Capacity of Two-dimensional Arrays

We now consider the case where the domains form a two-dimensional array. Specifically, we study two types: the *rectangular array* and the *triangular array*, illustrated in Fig. 3. We denote the capacity of the two-dimensional array by cap. [1] Some existing techniques based on convex/concave programming, including tiling, bit-stuffing, *etc.*, can be applied here to obtain the upper and lower bounds of the capacity. We summarize the bounds in Table I. It is interesting that the capacity is really high (close to 1) for both arrays. In the rest of this section, we will discuss the bounds in detail.



(a) Rectangular array (b) Triangular array

Fig. 3. Two types of two-dimensional arrays.

[1]It will be clear from the context which array it refers to. And we comment that compared to the rectangular array, it is possible to pack domains more compactly in the triangular array.

	Lower (Tiling)	Lower (Bit-Stuffing)	Upper Bound
Rectangular	0.959338	0.961196	0.963109
Triangular	0.987829	0.987218	0.990029

TABLE I
UPPER AND LOWER BOUNDS FOR TWO-DIMENSIONAL ARRAY'S CAPACITY.

1) Lower Bound based on Tiling: If we consider a distribution θ on the valid configuration set $\mathcal{U}$, then the rate of G is

$$R(\theta) = \frac{H(\theta)}{n}.$$

So another expression for capacity is

$$cap = \max_{\theta} \lim_{n\to\infty} R(\theta).$$

For any distribution θ, $\lim_{n\to\infty} R(\theta)$ is a lower bound for cap. Different ways of constructing θ lead us to different methods.

In [4], Tiling was proposed as a variable-length encoding technique for two-dimensional (2-D) constraints, such as runlength-limited (RLL) constraints and no isolated bits (n.i.b.) constraints. The idea of tiling is that we can divide all the 2-D plane using shifted copies of two certain shapes, referred as 'W' and 'B' tiles. Here, we say that a set of vertices A is a shift or shifted copy of another set B if and only if their vertices are one-to-one mapped and the position movement (vector) between each vertex in A and its corresponding vertex in B is fixed. For these two types of tiles – 'W' tiles and 'B' tiles, – they have the following properties:

1) The 'W' tiles are freely configurable. That means given any configuration for all the 'W' tiles, we can always find a configuration for all the 'B' tiles such that they satisfy the 2-D constraints.
2) Given any configuration for all the 'W' tiles, the configurations for the 'B' tiles are independent with each other.

According to these properties, we can first set 'W' tiles independently based on a predetermined distribution π, and then configure the 'B' tiles uniformly and independently (given the 'W' tiles). Finally, the maximal information rate $\max_\pi R(\pi)$ is a lower bound of the array's capacity.

As discussed previously, our constraint for a valid configuration is that each "on" vertex has at least one "on" neighbor. For the rectangular/triangular arrays, we can use the tiling schemes in Fig. 4.

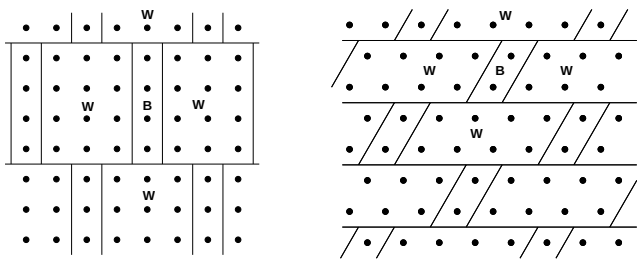


Fig. 4. Tiling schemes for the rectangular (left) and triangular (right) arrays.

According to Theorem 3.1 in [4], we have

$$cap \geq \max_{\pi} R(\pi) = \max_{\pi} \frac{H(\pi) + \sum_{\phi} P_{\pi}(\phi)|S(\phi)|}{|W| + |B|}.$$

Here, $|W|$ (or $|B|$) is the size of each 'W' ('B') tile, e.g., $|W| = 12$ in the left-side tiling of Fig. 4 and $|B| = 2$ in the right-side tiling of Fig. 4; $H(\pi)$ is the entropy corresponding to distribution π; ϕ is the configuration of the 'W' blocks around a 'B' block (four blocks in Fig. 4), whose distribution is a function of π, denoted as $P_{\pi}(\phi)$; $|S(\phi)|$ is the number of available distinct configurations for a 'B' blocks given the 'W' blocks around it. Based on this formula, we are able to get the lower bounds in the first column of Table I using convex programming with linear constraints.

2) Lower Bound based on Bit-Stuffing: Another way to obtain the lower bounds for the capacities of 2-D constraint codes is based on bit-stuffing [5]. In bit-stuffing, let ∂ denote the vertices near the left and top boundaries, called boundary vertices. Assume we know the state configuration of ∂; then we can programm the remaining vertices one by one such that the ith vertex depends on a set of programmed vertices near it, denoted by D_i. In this scheme, for different i, j, we have that the set $D_i \bigcup i$ is a shift of the set $D_j \bigcup j$, and for all i, the conditional distribution $P(x_i|x(D_i))$ is fixed, denoted by γ, where $x(D_i)$ is the configuration of D_i.

Let θ denote the probability distribution of the configuration on all the vertices V, and let δ denote the probability distribution of the configuration on the boundary islands ∂. Then we see that θ is uniquely determined by δ and the conditional distribution γ. It is not hard to prove that for any conditional distribution γ, when the 2-D array is infinitely large, there exists a distribution δ such that θ is stationary. That means for any subset $A \subset V$ and its arbitrary shift $\sigma(A) \subset V$, A and $\sigma(A)$ have the same configuration distribution, namely,

$$P_{\theta}(x(A) = a) = P_{\theta}(x(\sigma(A)) = a)$$

for any state configuration a. Note that this equation is true only when the block is infinity large; otherwise, θ is quasi-stationary [5].

Given this stationary distribution θ, we would like to calculate the relative entropy R_i of the ith vertex given the states of the vertices programmed before it. (Here the ith vertex is not a boundary vertex). Assume the state distribution on D_i is ϕ; then according to the definition of bit-stuffing

$$R_i = \sum_{y \in \{0,1\}, z \in \{0,1\}^{|D_i|}} \phi(z) H(\gamma(y|z))$$

where $|D_i|$ is the same for different i, so we can also write it as $|D|$. It is not easy to get the exact value of R_i because ϕ is unknown (it depends on γ) and there are too many constraints to guarantee that θ is stationary. By relaxing the constraints, we get a set of distributions on D_i, denoted as $\{\phi'\}$, such that θ is stationary near the ith vertex (limited in a fixed area T near the ith vertex). Therefore,

$$R_i \geq \min_{\phi'} \sum_{y \in \{0,1\}, z \in \{0,1\}^{|D|}} \phi'(z) H(\gamma(y|z))$$

such that (1) the configuration distribution on T is stationary, and (2) given some $z \in \{0,1\}^{|D|}$, we have $\gamma(0|z) = 0$ to guarantee that each "on" vertex has at least one "on" neighbor.

Since the inequality above holds for all the vertices except the boundary vertices, a lower bound of the capacity can be written as

$$\max_{\gamma} \min_{\phi'} \sum_{z} \phi'(z) H(\gamma(y|z))$$

under the constraints. (For more discussions, please see [5].)

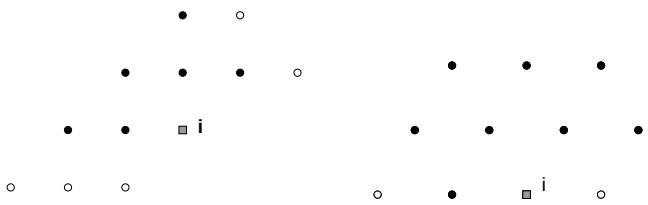


Fig. 5. The bit-stuffing schemes for the rectangular and triangular arrays.

Fig. 5 shows the bit-stuffing schemes that we use to calculate the lower bounds of the 2-D arrays' capacities. In this figure, the vertex i is marked as a gray square; D_i is indicated by the black vertices that the vertex i depends on; the stationary constraint is applied to the region T that includes all the vertices plotted. Based on these schemes, we get the lower bounds for the capacities, which are given in the second column in Table I.

3) Upper Bound based on Convex Programming: In [6], convex programming was used as a method for calculating an upper bound on the capacity of 2-D constraints. The idea is based on the observations that there exists an optimal distribution θ^* such that θ^* is stationary and symmetric when the array is sufficiently large. The stationary property implies that for any set of vertices A, – let $\sigma(A)$ be an arbitrary shift of A, – A and $\sigma(A)$ have the same state (configuration) distribution. The symmetric property depends on the type of the array. For a rectangular array, if two sets of vertices A and B are reflection symmetric about a horizontal/vertical line or a 45-degree line, then they have the same state (configuration) distribution. Note that the reflection symmetry about a 45-degree line is also called transposition invariance in [6]. For a triangular array, there are more symmetries: if two sets of vertices A and B are reflection symmetric about a horizontal/vertical line or a 30/60-degree line, then they have the same state (configuration) distribution.

Now let us consider the distribution over a small region T for both arrays, as shown in Fig. 6. For example, in the rectangular array, assume the distribution on T (the 12 vertices) is ϕ; then given the first ten vertices, the relative entropy of the next vertex is a function of ϕ, denoted by $R(\phi)$. Let's index all the vertices by $1, 2, 3, ..., n$ from left to right and then from top to bottom and let $R_i = H(x_i|x_1, x_2, ..., x_{i-1})$.

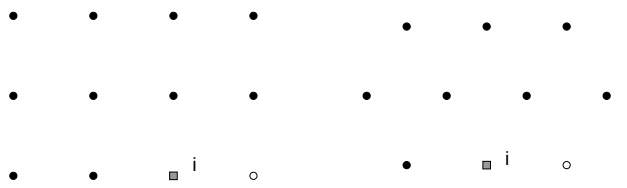


Fig. 6. The schemes for calculating the upper bounds of the capacities.

It is easy to see that if a vertex i is not on the boundary, then

$$R_i \leq H(x_i|\{x_1, x_2, ..., x_{i-1}\} \bigcap T) = R(\phi).$$

That implies that $R(\phi)$ is an upper bound for

$$cap = \lim_{n\to\infty} \max_{\theta} \frac{\sum_{i=1}^{n} R_i}{n}$$

So our work is to maximize $R(\phi)$ such that ϕ is stationary and symmetric on T. Thus we get the upper bounds for the capacity of the rectangular array in Table I. The same method also applies to the triangular array.

III. Error Correction and Detection

In this section, we study error correction/detection for patterned cells. We focus on one-dimensional arrays and two-dimensional rectangular arrays. When programming domains, a common error is to make a domain too large such that it changes the connectivity pattern unintentionally. Two types of such errors are shown in Fig. 7, where in (a) two diagonal "on" domains overlap, and in (b) an "on" domain touches its neighboring "off" domain's bottom electrode. It can be proved that the former type of errors can always be corrected, because the two concerned domains' states can be correctly determined by checking if they are connected to one of their four neighbors. So in this paper, we focus on the latter type of error, which is important and less trivial. We call the latter error an *overreach error*, which happens only between an "on" vertex and a neighboring "off" vertex, and the error makes them become connected. We assume that between every pair of neighboring "on" and "off" vertices, the overreach error happens independently with probability p_e. Given p_e, we define the *capacity* as the maximum number of bits that can be stored per vertex such that the data can be decoded correctly with high probability (which approaches one as the array's size approaches infinity).

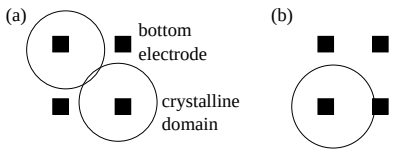


Fig. 7. Error models. (a) Two diagonal domains overlap. (b) Overreach error.

A. One-dimensional Array

Let $G = (V, E)$ be a one-dimensional array of n vertices: $v_1, v_2, \cdots, v_n$. When $n \to \infty$ and given the overreach error probability p_e, let $cap_1(p_e)$ denote its capacity.

Theorem 4. *For one-dimensional array,* $cap_1(p_e) \geq$

$$\max\{0.5, \max_{x\in[0,0.4]} x(1 - H(p_e)) + \frac{2-x}{4} H\left(\frac{4x}{2-x}\right)\}.$$

Proof: We prove the theorem constructively by presenting error-correcting codes for one-dimensional arrays.

To see that $cap_1(p_e) \geq 0.5$, consider n to be even. Partition the n vertices into pairs: $(v_1, v_2), (v_3, v_4), \cdots, (v_{n-1}, v_n)$. Store one bit in every pair (v_{2i-1}, v_{2i}) (for $i = 1, 2, \cdots, \frac{n}{2}$) this way: if the bit is 0, set both vertices as "off"; if the bit is 1, set both vertices as "on". Clearly, the code can correct *all* overreach errors. And its rate is 0.5 bit per vertex. So $cap_1(p_e) \geq 0.5$. In the following, we need to prove that

$$cap_1(p_e) \geq \max_{x\in[0,0.4]} x(1 - H(p_e)) + \frac{2-x}{4} H\left(\frac{4x}{2-x}\right).$$

Given a valid configuration $\vec{s} = (s_1, s_2, \cdots, s_n) \in \mathcal{U} \subseteq \{0, 1\}^n$, a 1-run (respectively, 0-run) is a maximal segment in the vector $\vec{s}$ whose elements are all 1s (respectively, all 0s). Let m be a positive integer. Define $\mathcal{U}_{m,1} \subseteq \mathcal{U}$ to be the set of valid configurations that satisfy the following constraints: "The configuration has exactly $m + 1$ 1-runs and 0-runs in total. Every 1-run or 0-run has at least two elements. The first run (i.e., the left-most run) is a 1-run." Define $\mathcal{U}_{m,0}$ in the same way except that a configuration in $\mathcal{U}_{m,0}$ starts with a 0-run (instead of a 1-run). The analysis below is very similar for both $\mathcal{U}_{m,1}$ and $\mathcal{U}_{m,0}$. So to be succinct, we sometimes only present the analysis for $\mathcal{U}_{m,1}$.

For a configuration $\vec{s} = (s_1, s_2, \cdots, s_n)$ in $\mathcal{U}_{m,1}$ (or $\mathcal{U}_{m,0}$), let $L_1, L_2, \cdots, L_{m+1}$ denote the lengths of its $m + 1$ 1-runs and 0-runs. (Clearly, $\sum_{i=1}^{m+1} L_i = n$.) We define the *signature of* $\vec{s}$, denoted by $sig(\vec{s})$, as

$$sig(\vec{s}) = (L_1 \mod 2,\ \sum_{i=1}^{2} L_i \mod 2,\ \sum_{i=1}^{3} L_i \mod 2,\ \cdots,\ \sum_{i=1}^{m} L_i \mod 2).$$

$sig(\vec{s})$ is a binary vector of length m.

Given a binary vector $\vec{d} = (d_1, d_2, \cdots, d_m)$, we define its *difference vector* $\Delta(\vec{d})$ as

$$\Delta(\vec{d}) = (d_1,\ d_2 + d_1 \mod 2,\ d_3 + d_2 \mod 2,\ \cdots,\ d_m + d_{m-1} \mod 2).$$

$\Delta(\vec{d})$ is also a binary vector of length m. Given any binary vector $\vec{y}$, let $w(\vec{y})$ denote its Hamming weight.

We first prove the following property:

- Property ♣: Let $\vec{d} = (d_1, d_2, \cdots, d_m)$ be a binary vector of length m. Let $n \geq 2m + w(\Delta(\vec{d})) + 2$, and let $n - w(\Delta(\vec{d}))$ be even. Then we have

$$\left|\{\vec{s} \in \mathcal{U}_{m,1} \mid sig(\vec{s}) = \vec{d}\}\right| = \left|\{\vec{s} \in \mathcal{U}_{m,0} \mid sig(\vec{s}) = \vec{d}\}\right|$$

$$= \binom{\frac{n-w(\Delta(\vec{d}))}{2}-1}{m}.$$

Due to the symmetry between $\mathcal{U}_{m,1}$ and $\mathcal{U}_{m,0}$ (just replace 1-runs with 0-runs and vice versa), we have $\left|\{\vec{s}\in\mathcal{U}_{m,1} \mid sig(\vec{s})=\vec{d}\}\right| = \left|\{\vec{s}\in\mathcal{U}_{m,0} \mid sig(\vec{s})=\vec{d}\}\right|$. So we just need to show that $\left|\{\vec{s}\in\mathcal{U}_{m,1} \mid sig(\vec{s})=\vec{d}\}\right| = \binom{\frac{n-w(\Delta(\vec{d}))}{2}-1}{m}$. To prove that, consider a configuration $\vec{s} = (s_1, s_2, \cdots, s_n) \in \mathcal{U}_{m,1}$ whose signature $sig(\vec{s}) = \vec{d}$. Let $L_1, L_2, \cdots, L_{m+1}$ denote its $m+1$ 1-runs and 0-runs, from left to right. It is not hard to see that if the ith element in the vector $\Delta(\vec{d})$ is 0, then $L_i \geq 2$ and L_i is even; if the ith element in $\Delta(\vec{d})$ is 1, then $L_i \geq 3$ and L_i is odd.

Let us obtain a new binary vector $\vec{y} = \left(y_1, y_2, \cdots, y_{\frac{n-w(\Delta(\vec{d}))}{2}}\right)$ this way: first, for $i = 1, 2, \cdots, m$, if the ith element in $\Delta(\vec{d})$ is 1, decrease the length of the ith 1-run or 0-run in $\vec{s}$ by one; then, for $i = 1, 2, \cdots, m+1$, reduce the length of the ith 1-run or 0-run by half. Clearly, $\vec{y}$ is a binary vector of length $\frac{n-w(\Delta(\vec{d}))}{2}$ that has $m+1$ 1-runs and 0-runs (without any limitation on the lengths of the 1-runs and 0-runs), and there is a one-to-one mapping between configurations in $\mathcal{U}_{m,1}$ of signature $\vec{d}$ and such $\vec{y}$ vectors. There are $\binom{\frac{n-w(\Delta(\vec{d}))}{2}-1}{m}$ such vectors $\vec{y}$. So Property ♣ is true.

We now consider $m \to \infty$, let m be even, and let ϵ be an arbitrarily small constant. Define $\tilde{K} \triangleq \{\vec{y} \in \{0,1\}^m \mid w(\Delta(\vec{y})) = \frac{m}{2}\}$. Note that for a random binary vector $\vec{y} \in \{0,1\}^m$ whose elements are i.i.d. and equally likely to be 0 and 1, with high probability we have $\lim_{m\to\infty} \frac{w(\Delta(\vec{y}))}{m} = \frac{1}{2}$. So $\lim_{m\to\infty, m \text{ is even}} \frac{\log_2|\tilde{K}|}{m} = 1$. Let $K \subset \tilde{K}$ be a set whose elements are uniformly randomly chosen from $\tilde{K}$ such that $\lim_{m\to\infty, m \text{ is even}} \frac{\log_2|K|}{m} = 1 - H(p_e) - \epsilon$. It is not difficult to see that K is an error-correcting code of length m (with $m \to \infty$), rate $1 - H(p_e)$ (we make $\epsilon \to 0$) that can correct binary symmetric errors with error probability p_e.

Let $n \geq \frac{5}{2}m + 2$, and let $n - \frac{m}{2}$ be even. By Property ♣, for every vector $\vec{y} \in K$, there are $\binom{\frac{n}{2}-\frac{m}{4}-1}{m}$ configurations in $\mathcal{U}_{m,1}$ (and in $\mathcal{U}_{m,0}$) of signature $\vec{y}$. Define $x \triangleq \frac{m}{n}$, $\mathcal{D}_1 \triangleq \{\vec{s} \in \mathcal{U}_{m,1} \mid \exists \vec{y} \in K \text{ such that } sig(\vec{s}) = \vec{y}\}$, and $\mathcal{D}_0 \triangleq \{\vec{s} \in \mathcal{U}_{m,0} \mid \exists \vec{y} \in K \text{ such that } sig(\vec{s}) = \vec{y}\}$. Since $\lim_{n,m\to\infty} \frac{\log_2\binom{\frac{n}{2}-\frac{m}{4}-1}{m}}{\frac{n}{2}-\frac{m}{4}-1} = \lim_{n,m\to\infty} H(\frac{m}{\frac{n}{2}-\frac{m}{4}-1}) = H(\frac{4x}{2-x})$, we can encode $1 + \lfloor nx(1 - H(p_e) - \epsilon)\rfloor + \lfloor n(\frac{1}{2} - \frac{x}{4} - \frac{1}{n})H(\frac{4x}{2-x})\rfloor$ information bits into the configurations in $\mathcal{D}_1 \cup \mathcal{D}_0$ as follows:

1) If the 1st information bit is 1, the codeword will be a configuration in $\mathcal{U}_{m,1}$; otherwise, it will be a configuration in $\mathcal{U}_{m,0}$.
2) The next $\lfloor nx(1 - H(p_e) - \epsilon)\rfloor$ information bits are mapped to one of the vectors in K, where the mapping is injective. Let $\vec{y}$ denote the corresponding vector in K.
3) The last $\lfloor n(\frac{1}{2} - \frac{x}{4} - \frac{1}{n})H(\frac{4x}{2-x})\rfloor$ information bits are mapping to one of the configurations in $\mathcal{U}_{m,1}$ or $\mathcal{U}_{m,0}$ (depending on if the 1st information bit is 1 or 0) whose signatures equal $\vec{y}$, where the mapping is injective.

We now show how to decode the codewords (i.e., configurations) in $\mathcal{D}_1 \cup \mathcal{D}_0$ to recover the information bits, where the codewords can contain overreach errors (with error probability p_e).

Let $\vec{s} = (s_1, s_2, \cdots, s_n) \in \mathcal{D}_1 \cup \mathcal{D}_0$ denote the codeword (configuration) that is stored, and let $\vec{b}$ denote the information bits encoded into codeword $\vec{s}$. After $\vec{s}$ is stored, overreach errors happen and change the connectivity pattern. Let $\vec{\beta} = (\beta_1, \beta_2, \cdots, \beta_n) \in \{0,1,2,3\}^n$ denote the connectivity pattern detected after overreach errors happen, defined as follows: "For $i = 1, 2, \cdots, n$, if the ith vertex is not connected to any other vertex, then $\beta_i = 0$; if $i > 1$ and it is only connected to the $(i-1)$th vertex, then $\beta_i = 1$; if $i < n$ and it is only connected to the $(i+1)$th vertex, then $\beta_i = 2$; if $1 < i < n$ and it is connected to both the $(i-1)$th vertex and the $(i+1)$th vertex, then $\beta_i = 3$." (For example, if $\vec{s} = (1,1,0,0,1,1,1,0,0,1,1)$ and two overreach errors connect the 2nd vertex with the 3rd vertex and connect the 4th vertex with the 5th vertex, then $\vec{\beta} = (2,3,1,2,3,3,1,0,0,2,1)$.) Based on $\vec{\beta}$, the decoding algorithm will recover both the codeword $\vec{s}$ and the information bits $\vec{b}$.

Since every 1-run or 0-run in $\vec{s}$ has length at least two, if $\beta_1 = 0$, then $\vec{s} \in \mathcal{U}_{m,0}$ and the first information bit in $\vec{b}$ is 0; otherwise, $\vec{s} \in \mathcal{U}_{m,1}$ and the first information bit in $\vec{b}$ is 1. In the following, without loss of generality (w.l.o.g.), we assume that $\beta_1 = 1$ and present the corresponding decoding method.

Let $\hat{L}_1, \hat{L}_2, \cdots, \hat{L}_{m+1}$ be our estimation of $L_1, L_2, \cdots, L_{m+1}$ (the lengths of 1-runs and 0-runs in $\vec{s}$), computed as follows. For $i = 1, 2, \cdots, \frac{m}{2}+1$, let $\hat{L}_{2i-1}$ be the length of the ith segment in $\vec{\beta}$ of the form $(2, 3, \cdots, 3, 1)$, and let $\alpha_{2i-1}, \alpha'_{2i-1} \in \{1, 2, \cdots, n\}$ denote the first and last positions of that segment, respectively. (That is, the segment begins with 2, ends with 1, and has zero or more 3's in between.) For $i = 1, 2, \cdots, \frac{m}{2}$, let $\hat{L}_{2i} = \alpha_{2i+1} - \alpha'_{2i-1} - 1$. Define the signature of $\vec{\beta}$ as $sig(\vec{\beta}) = (\eta_1, \eta_2, \cdots, \eta_m) \triangleq (\hat{L}_1 \mod 2, \sum_{i=1}^{2}\hat{L}_i \mod 2, \sum_{i=1}^{3}\hat{L}_i \mod 2, \cdots, \sum_{i=1}^{m}\hat{L}_i \mod 2)$. Let $sig(\vec{s}) = (\mu_1, \mu_2, \cdots, \mu_m)$ be the signature of $\vec{s}$. It is not hard to see we have the following property:

- For $i = 1, 3, 5, \cdots, m-1$, if there is no overreach error between the last vertex of the ith run (which is a 1-run) and the first vertex of the $(i+1)$th run (which is a 0-run) in $\vec{s}$, then $\sum_{j=1}^{i}\hat{L}_j = \sum_{j=1}^{i}L_j$ and therefore $\eta_i = \mu_i$; otherwise, $\sum_{j=1}^{i}\hat{L}_j = \sum_{j=1}^{i}L_j + 1$ and therefore $\eta_i = \mu_i + 1 \mod 2$.
- For $i = 2, 4, 6, \cdots, m$, if there is no overreach error between the last vertex of the ith run (which is a 0-run) and the first vertex of the $(i+1)$th run (which is a 1-run) in $\vec{s}$, then $\sum_{j=1}^{i}\hat{L}_j = \sum_{j=1}^{i}L_j$ and therefore $\eta_i = \mu_i$; otherwise, $\sum_{j=1}^{i}\hat{L}_j = \sum_{j=1}^{i}L_j - 1$ and therefore $\eta_i = \mu_i + 1 \mod 2$.

So the overreach errors have a one-to-one mapping to the 1's in the vector $(\mu_1+\eta_1 \mod 2, \mu_2+\eta_2 \mod 2, \cdots, \mu_m+\eta_m \mod 2)$. Since $sig(\vec{s})$ is a codeword in K, and the code K

can correct binary symmetric errors with error probability p_e, we can decode $sig(\vec{\beta})$ to recover the correct value of $sig(\vec{s})$ (with probability one as $m \to \infty$). Then based on $sig(\vec{s})$ and $\hat{L}_1, \cdots, \hat{L}_m$, we can recover the values of $L_1, \cdots, L_m$ and therefore the codeword (configuration) $\vec{s} \in \mathcal{U}_{m,1}$. Based on $sig(\vec{s}) \in K$, we can recover the $\lfloor nx(1 - H(p_e) - \epsilon) \rfloor$ information bits that follow the first information bit. Then based on $\vec{s}$, we can recover the last $\lfloor n(\frac{1}{2} - \frac{x}{4} - \frac{1}{n})H(\frac{4x}{2-x}) \rfloor$ information bits. That concludes the decoding algorithm.

We now analyze the rate R of the above code. When $n, m \to \infty$, we have $R = x(1-H(p_e)) + \frac{2-x}{4}H(\frac{4x}{2-x})$. Since $n \geq \frac{5}{2}m + 2$, $x = \frac{m}{n} \in [0, 0.4]$. That leads to the conclusion. ■

It is noticeable that the overreach error is a type of asymmetric error for graph connectivity. In the following, we present an error-detecting code that can detect *all* overreach errors. Its underlying idea is closely related to the well-known Berger code [1] for asymmetric errors.

The framework of the code construction is as follows. We use m information vertices and r redundant vertices, which form a one-dimensional array of $n = m + r$ vertices. (The redundant vertices follow the information vertices in the array.) Let the constants $\alpha_1, \alpha_2, \alpha_3, \mu_1, \mu_2, \mu_3$ be as specified in Theorem 3. The m information vertices store data from an alphabet of size $N(m) = \mu_1\alpha_1^m + \mu_2\alpha_2^m + \mu_3\alpha_3^m$. When m is large, the m information vertices store about $0.8114m$ information bits, and $r \approx \log_{1.7549} m$. (So the redundancy is logarithmic in the codeword length.) Let χ denote the number of connected components in the subgraph induced by the information vertices, which overreach errors can only decrease. We use the redundant vertices to record the value of χ, and the mapping is constructed such that the recorded value can only be increased by overreach errors. This way, the mismatch between information vertices and redundant vertices can be used to detect all overreach errors.

We now present details of the code. Let $v_1, v_2, \cdots, v_m$ denote the m information vertices. A *connected component* among them is a maximal segment of vertices $(v_i, v_{i+1}, \cdots, v_j)$ such that their corresponding bottom electrodes are all electrically connected. Let χ and $\hat{\chi}$ denote the number of connected components among the information vertices before and after overreach errors happen (if any), respectively. Clearly, $1 \leq \hat{\chi} \leq \chi \leq m$. If there is one or more overreach errors among the m information vertices, then $\hat{\chi} < \chi$; otherwise, $\hat{\chi} = \chi$.

Let $u_1, u_2, \cdots, u_r$ denote the r redundant vertices, and let $\mathcal{U}_r \subseteq \{0,1\}^r$ denote the set of valid configurations for them. For every $\vec{s} = (s_1, s_2, \cdots, s_r) \in \mathcal{U}_r$, let $B(\vec{s}) \triangleq \sum_{i=1}^{r} s_i \cdot 2^{r-i}$. We have $|\mathcal{U}_r| = N(r) = \mu_1\alpha_1^r + \mu_2\alpha_2^r + \mu_3\alpha_3^r$. We build a bijective function

$$F \ : \ \mathcal{U}_r \to \{1, 2, \cdots, N(r)\}$$

with the following property:

- For any two valid configurations $\vec{s}, \vec{t} \in \mathcal{U}_r$, $F(\vec{s}) < F(\vec{t})$ if and only if $B(\vec{s}) < B(\vec{t})$.

That is, the function F sorts the valid configurations of the redundant vertices based on their lexical order. Let F^{-1} denote the inverse function of F. We will introduce the specific computations used by F and F^{-1} at the end of the subsection.

We now introduce how to encode the value of χ using the configuration of the r redundant vertices. We choose r to be the smallest positive integer such that $N(r) \geq m$. Let $\vec{\theta} \in \mathcal{U}_r$ denote the programmed configuration of the r redundant vertices. Then as the the encoding algorithm, we choose $\vec{\theta}$ such that

$$F(\vec{\theta}) = \chi.$$

We introduce details of the decoding (i.e., error detection) process. Let $\vec{x} = (x_1, x_2, \cdots, x_m) \in \{0,1\}^m$ denote our estimated configuration of the information vertices, defined as follows:

- We measure the connectivity between the information vertices. For $i = 1, 2, \cdots, m$, if vertex v_i is connected to at least one other information vertex, then $x_i = 1$; otherwise, $x_i = 0$.

Similarly, let $\vec{y} = (y_1, y_2, \cdots, y_r) \in \{0,1\}^r$ denote our estimated configuration of the redundant vertices, defined as follows:

- We measure the connectivity between the redundant vertices. For $i = 1, 2, \cdots, r$, if vertex u_i is connected to at least one other redundant vertex, then $y_i = 1$; otherwise, $y_i = 0$.

The decoding (i.e., error detection) algorithm is as follows:

1) Let $\hat{\chi}$ be the number of connected components among the information vertices derived from (i.e., computed based on) the estimated configuration $\vec{x}$. If $F(\vec{y}) > \hat{\chi}$, then either one or more overreach errors exist.
2) If the two vertices v_m and u_1 are connected but either "$x_m = 1, y_1 = 0$" or "$x_m = 0, y_1 = 1$", then there is an overreach error between v_m and u_1.

Theorem 5. *The above code can detect all overreach errors.*

Proof: If overreach errors happen among the information vertices, we will have $\hat{\chi} < \chi$. Let overreach errors happen among the redundant vertices, some "off" redundant vertices will be incorrectly estimated to be "on", so we will have $F(\vec{y}) > F(\vec{\theta})$. Since $F(\vec{\theta}) = \chi$, if overreach errors happen among information vertices or among redundant vertices (or both), we will have $F(\vec{y}) > \hat{\chi}$, and the errors will be detected.

The only remaining case is that no overreach error happens among the information vertices or among the redundant vertices, however there is an overreach error between the two segments (namely, between v_m and u_1). In this case, x_m and y_1 will be the true states of the two vertices, and the second step of the algorithm will detect the error. ■

Theorem 6. *Let $m \geq 2$ be an integer. Let r be the smallest positive integer such that $\mu_1\alpha_1^r + \mu_2\alpha_2^r + \mu_3\alpha_3^r \geq m$. (The*

constants $\alpha_1, \alpha_2, \alpha_3, \mu_1, \mu_2, \mu_3$ are specified in Theorem 3.) Then, there is an error-detecting code of length $m+r$ and rate

$$\frac{\log_2(\mu_1\alpha_1^m + \mu_2\alpha_2^m + \mu_3\alpha_3^m)}{m+r}$$

bits per vertex that can detect all overreach errors. When $m \to \infty$, we have $r = \log_{\alpha_1} m \approx \log_{1.7549} m$, and the rate of the code is $cap_{1D} = \log_2 \alpha_1 \approx 0.8114$, which reaches the capacity of one-dimensional arrays.

We now introduce how the function $F : \mathcal{U}_r \to \{1, 2, \cdots, N(r)\}$ maps configurations to integers, and how its inverse function $F^{-1} : \{1, 2, \cdots, N(r)\} \to \mathcal{U}_r$ maps integers to configurations.

We first show that given any valid configuration $\vec{s} = (s_1, s_2, \cdots, s_r) \in \mathcal{U}_r$, how to compute $F(\vec{s})$. If $\vec{s} = (0, 0, \cdots, 0)$, then $F(\vec{s}) = 1$. So in the following we assume $\vec{s} \neq (0, 0, \cdots, 0)$. Let

$$i = \min\{k \in \{1, 2, \cdots, r\} \mid s_k = 1\}.$$

Let $j \in \{i+1, i+2, \cdots, r\}$ be defined as follows: if $s_i = s_{i+1} = \cdots = s_r = 1$, then $j = n$; otherwise, let j be the integer such that $s_i = s_{i+1} = \cdots = s_j = 1$ and $s_{j+1} = 0$. For any two configurations $\vec{t_1}, \vec{t_2} \in \mathcal{U}_r$, we say $\vec{t_1}$ is smaller than $\vec{t_2}$ if $F(\vec{t_1}) < F(\vec{t_2})$. Namely, $\vec{t_1}$ is smaller than $\vec{t_2}$ if $\vec{t_1}$ is lexically smaller than $\vec{t_2}$. We have the following observation:

- The smallest $N(r-i)$ configurations $(a_1, a_2, \cdots, a_r) \in \mathcal{U}_r$ are those with $a_1 = a_2 = \cdots = a_i = 0$; the next $N(r-i-2)$ smallest configurations are those with $a_1 = \cdots = a_{i-1} = 0$, $a_i = a_{i+1} = 1$ and $a_{i+2} = 0$; the next $N(r-i-3)$ smallest configurations are those with $a_1 = \cdots = a_{i-1} = 0$, $a_i = a_{i+1} = a_{i+2} = 1$ and $a_{i+3} = 0$; and so on.

Consequently, we obtain the following formula:

$$\begin{aligned} F(\vec{s}) = \quad & N(r-i) + \textstyle\sum_{k=i+1}^{j-1} N(r-k-1) \\ & + F((0, \cdots, 0, s_{j+2}, s_{j+3}, \cdots, s_r)). \end{aligned}$$

(By default, let $N(0) = 1$; and if $j \geq r-1$, let $F((0, \cdots, 0, s_{j+2}, s_{j+3}, \cdots, s_r)) = 1$.) The above recursion can be easily used to compute $F(\vec{s})$.

Next, we show that given an integer $z \in \{1, 2, \cdots, N(r)\}$, how to compute $F^{-1}(z) = (s_1, s_2, \cdots, s_r) \in \mathcal{U}_r$. If $z = 1$, then $F^{-1}(z) = (0, 0, \cdots, 0)$. In the following we assume $z > 1$. Let i be the greatest integer such that $N(r-i+1) \geq z$; then we have

$$s_1 = s_2 = \cdots = s_{i-1} = 0 \quad \text{and} \quad s_i = 1.$$

Let j be the smallest integer such that

$$N(r-i) + \sum_{k=i+1}^{j} N(r-k-1) \geq z.$$

(By default, let $N(0) = N(-1) = 1$.) Then we have

$$s_i = s_{i+1} = \cdots = s_j = 1.$$

If $j = n-1$, we have $s_n = 0$. If $j \leq n-2$, we have $s_{j+1} = 0$ and

$$\begin{aligned} & (0, \cdots, 0, s_{j+2}, s_{j+3}, \cdots, s_r) \\ = \quad & F^{-1}\left(z - N(r-i) - \textstyle\sum_{k=i+1}^{j-1} N(r-k-1)\right). \end{aligned}$$

With the above recursion, we can easily determine $F^{-1}(z)$.

B. Two-dimensional Array

We now focus on the capacity of two-dimensional rectangular array when i.i.d. overreach errors happen with probability p_e between neighboring *on* and *off* vertices. Let $G = (V, E)$ be an $m \times m$ two-dimensional rectangular array, where $m \to \infty$. Let $cap_2(p_e)$ denote its capacity.

Theorem 7. *For any $q \in [0, 1/2]$, let $\eta(q, p_e) = (1-q^3)(p_e + (1-p_e)(1-(1-(1-q)p_e)^3))$. Then for two-dimensional rectangular array,*

$$cap_2(p_e) \geq \frac{4}{5} \max_{q \in [0, 0.5]} H(1 - q + q\eta(q, p_e)) - qH(\eta(q, p_e)).$$

Proof: The proof is constructive. First, consider a tile of 5 vertices as in Fig. 8 (a), where the 5 vertices are denoted by a, b, c, d, e, respectively. Let $q \in [0, \frac{1}{2}]$ be a parameter we will optimize. Let the on/off states of the four vertices a, b, c, d be i.i.d., where a (or b, c, d) is *on* with probability $1-q$ and *off* with probability q. We set the state of vertex e – the vertex in the middle – this way: "If a, b, c, d are all *off*, then e is *off*; otherwise, e is *on*." Clearly, the above approach guarantees that every *on* vertex has at least one neighboring vertex that is also *on*. Let $S(a), S(b), S(c), S(d) \in \{0, 1\}$ denote the states of the vertices a, b, c, d, respectively. We let each of the four vertices a, b, c, d store a bit, which equals $S(a), S(b), S(c), S(d)$, respectively.

It is well known that the small tiles can be packed perfectly to fill the two-dimensional space. It is illustrated in Fig. 8 (b). To differentiate the vertices in different small tiles, for $i = 1, 2, 3 \cdots$, the five vertices in the ith tile are denoted by a_i, b_i, c_i, d_i, e_i, respectively.

Let us focus on the stored bit $S(a_1)$. (The analysis applies to the other stored bits in the same way.) After overreach errors happen, let $S'(a_1)$ denote our estimation of the bit $S(a_1)$. We determine $S'(a_1)$ this way:

- If vertex a_1 is connected to e_1 (the central vertex in its small tile), then $S'(a_1) = 1$; otherwise, $S'(a_1) = 0$.

We can see that if $S(a_1) = 1$, there will be no decoding error for this bit because we will have $S'(a_1) = 1$. If $S(a_1) = 0$, with a certain probability P (which we will analyze later) the overreach errors will make $S'(a_1)$ be 1. So the channel for the stored bits is asymmetric, similar to the Z-channel but not memoryless. We first show the following property:

- Property ♣:

$$P \leq (1-q^3)(p_e + (1-p_e)(1-(1-(1-q)p_e)^3)).$$

To prove Property ♣, assume $S(a_1) = 0$. If $S'(a_1) = 1$, then $S(e_1) = 1$, and there must be an overreach error that connects

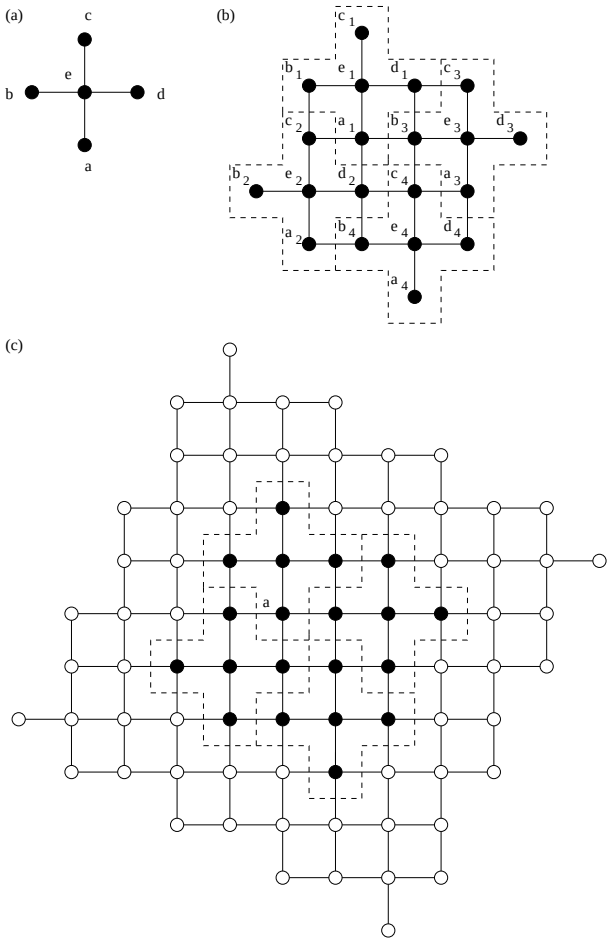


Fig. 8. Tiling and coding in two-dimensional rectangular array. (a) A small tile of 5 vertices. (b) Packing the small tiles in (a) to fill the two-dimensional space. (c) Separating large tiles using *off* vertices. Here the black vertices form a large tile. The white vertices are buffer vertices and are always *off*, and they separate the large tiles in the two-dimensional space.

a_1 to a neighbor that is *on*. We have $Pr\{S(e_1) = 1|S(a_1) = 0\} = Pr\{S(b_1) = 1, \text{ or } S(c_1) = 1, \text{ or } S(d_1) = 1\} = 1 - q^3$. Given $S(e_1) = 1$, the probability that an overreach error connects a_1 to either e_1 or one of the *on* vertices among $\{b_3, c_2, d_2\}$ – see Fig. 8 (b) – equals $p_e + (1 - p_e)(1 - (1 - (1 - q)p_e)^3)$. So Property ♣ is true.

We now use N small tiles to form a large tile, and use infinitely many such large tiles to fill the two-dimensional space with the following special arrangement: These large tiles are separated by buffer vertices that are always set as *off*, and for any two vertices in two different large tiles, there are at least two consecutive buffer vertices separating them on any path between them. (We illustrate it in Fig. 8 (c), where one large tile and the buffer vertices surrounding it are shown. Note that for easy illustration, in the figure a large tile consists of only $N = 4$ small tiles. However, for our proof on capacity, we will make N sufficiently large such that the buffer vertices have a negligible impact on the capacity.) Clearly, due to the existence buffer vertices and the fact that overreach errors cannot affect two vertices separated by two consecutive *off* vertices, the decoding errors for two different large tiles are independent.

Build a sub-channel as follows: Take one vertex from each large tile (which is either an a_i, b_i, c_i or d_i vertex, but not an e_i vertex), and let each vertex store one bit as described before (i.e., the vertex stores bit 0 with probability q and bit 1 with probability $1 - q$). (For example, we can take the vertex a shown in Fig. 8 (c) in each large tile.) Overall, the large tiles contain $4N$ such sub-channels. Consider one sub-channel, whose capacity is clearly a lower bound of the capacity of the aggregation of the $4N$ sub-channels. The errors for the different vertices in the sub-channel are independent and asymmetric (like a Z-channel); and due to the existence of the buffer vertices, the probability that its stored bit 0 is correctly decoded as 1 (i.e., the cross-over probability in the Z-channel) is *at most* P. Let $X, Y \in \{0, 1\}$ denote the input and output bit of the channel, respectively. Then we get

$$\begin{aligned} & I(X;Y) \\ = \; & H(Y) - H(Y|X) \\ = \; & H(Y) - \textstyle\sum_{x\in\{0,1\}} Pr\{X = x\}H(Y|X = x) \\ \geq \; & H(1 - q + qP) - qH(P) \\ \geq \; & H(1 - q + q\eta(q, p_e)) - qH(\eta(q, p_e)) \end{aligned}$$

Since in every small tile, 4 out of the 5 vertices are used to store bits, we get the conclusion. ■

It can be seen that when $p_e \to 0$, the low bound in the above theorem approaches $4/5$.

IV. Conclusion

In this paper, a new cell structure named patterned cell is introduced for phase-change memories. It has a new data representation scheme based on graph connectivity. The storage capacity of the scheme is analyzed, and its error correction and detection performance is studied.

References

[1] J. M. Berger, "A note on an error detection code for asymmetric channels," *Information and Control*, vol. 4, pp. 68–73, March 1961.

[2] G. W. Burr *et al.*, "Phase change memory technology," *Journal of Vacuum Science and Technology*, vol. 28, no. 2, pp. 223–262, March 2010.

[3] D. Lammers, "Resistive RAM gains ground," in *IEEE Spectrum*, pp. 14, September 2010.

[4] A. Sharov and R. M. Roth, "Two-Dimensional constrained coding based on tiling", *IEEE Transactions on Information Theory*, vol. 56, no. 4, pp. 1800–1807, 2010.

[5] I. Tal and R. M. Roth, "Bounds on the rate of 2-D bit-stuffing encoders", *IEEE Trans. on Information Theory*, vol. 56, no. 6, pp 2561-2567, 2010.

[6] I. Tal and R. M. Roth, "Convex programming upper bounds on the capacity of 2-D constraints", *IEEE Transactions on Information Theory*, vol. 57, no. 1, pp 381–391, 2011.